Chapter 2: Analyzing Routers with Parallel Slower Memories

Dec 2007, Monterey, 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 to alleviate memory access time problems for routers.

Additional Readings

- **Related Chapters:** The load balancing technique described here is also used to analyze routers in Chapters 3, 4, 6, and 10.

Table: *List of Symbols.*

c, C	Cell
$D(t)$	Departure Time
M	Total Number of Memories
N	Number of Ports of a Router
R	Line Rate
SNR	Aggregate Memory Bandwidth
T	Time Slot

Table: *List of Abbreviations.*

CIOQ	Combined Input Output Queued
DSM	Distributed Shared Memory
FCFS	First Come First Serve (Same as FIFO)
OQ	Output Queued
PIFO	Push In First Out
PPS	Parallel Packet Switch
PSM	Parallel Shared Memory
PDSM	Parallel Distributed Shared Memory
SB	Single-buffered
QoS	Quality of Service
WFQ	Weighted Fair Queueing

"Unlike some network technologies, communication by pigeons is not limited to line-of-sight distance".

— David Waitzman[†]

2

Analyzing Routers with Parallel Slower Memories

2.1 The Pigeonhole Principle

Chapter 1 described the history of high-speed router architectures and introduced the memory access time problem. In this chapter we will analyze and bring under a common umbrella a variety of router architectures. But first we will answer the question posed by the conversation in Chapter 1, since this will further our understanding of the memory access time problem in routers. We want to discover the number of pigeon holes that will allow a pigeon to enter or leave a pigeon hole within a given time slot, so that the N departing pigeons are guaranteed to be able to leave, and the N arriving pigeons are guaranteed a pigeon hole.

Consider a pigeon arriving at time t that will depart at some future time, $D(t)$. Let M be the total number of memories. We need to find a pigeon hole, H, that meets the following three constraints: (1) No other pigeon is arriving to H at time t; (2) No pigeon is departing from H at time t; and (3) No other pigeon in H wants to depart at time $D(t)$. Put another way, the pigeon is barred from no more than $3N-2$ pigeon holes by $N-1$ other arrivals, N departures, and $N-1$ other future departures. In

[†]David Waitzman, RFC 1149: Standard for the Transmission of IP Datagrams on Avian Carriers, 1st April 1990.

keeping with the well-known pigeonhole principle (see Theorem 2.1), if $M \geqslant 3N - 1$, our pigeon can find a hole.

2.1.1 The Constraint Set Technique

Algorithm 2.1: The constraint set technique for emulation of OQ routers.

1 **input** : Any Router Architecture.
2 **output**: A bound on the number of memories, total memory, and switching bandwidth required to emulate an OQ router.

3 **for** *each cell* c **do**
4 **Determine packet's departure time**, $D(t)$: If cells depart in FCFS order to a given output, and if the router is work-conserving, the departure time is simply one more than the departure time of the previous packet to the same output. If the cells are scheduled to depart in a more complicated way, for example using WFQ, then it is harder to determine a cells's departure time. We will consider this in more detail in Section 2.6. For now, we'll assume that the $D(t)$ is known for each cell.
5 **Identify constraints:** Identify all the constraints in the router architecture. This usually includes constraints on the memory bandwidth, memory access time, speed of the switch fabric, *etc.* The constraints themselves vary based on the router architecture being considered.
6 **Define the memory constraint sets:** Identify and set up the memory constraints for both the arriving and departing cells.
7 **Apply the pigeonhole principle:** Add up all the memory constraints, and apply the pigeonhole principle. This will identify the minimum number of memories and minimum memory bandwidth required to write and read all cells without any conflicts.
8 **Solve the fabric constraints:** Ensure that the switch fabric has sufficient bandwidth to read and write cells from memory, as derived in the previous step. Identify a switching algorithm to transport these cells. In some cases, if the router has a simple switching fabric (say, a broadcast bus), this step is not necessary.

"Constraint sets" are a simple way of formalizing the pigeonhole principle so that we can repeatedly apply it to a broad class of routers. In the routers that we will

consider, the arriving (departing) packets are written to (read from) memories that are constrained. In some cases, they may only allow either a read or a write operation in any one time slot. In other cases they may even operate slower than the line rate. We can use the constraint set technique to determine how many memories are needed (based on the speed at which the memory operates), and to design an algorithm to decide which memory each arriving packet is written into. The technique is described in Algorithm 2.1.

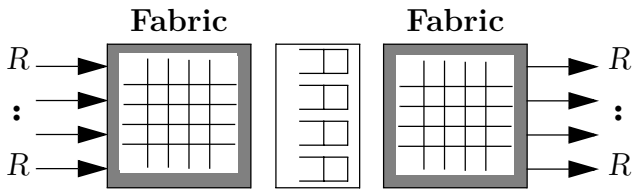
2.2 Single-buffered Routers

We now introduce a new class of routers called "single-buffered" routers. In contrast to the classical CIOQ router, which has two stages of buffering that "sandwich" a central switch fabric (with purely input queued and purely output queued routers as special cases), a SB router has only one stage of buffering sandwiched between two interconnects.

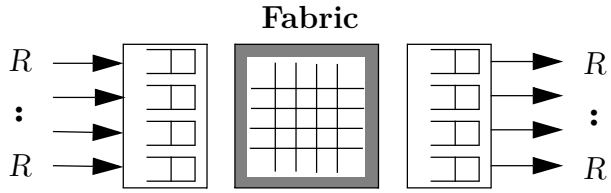
☞**Observation 2.1.** Figure 2.1 illustrates both architectures. A key feature of the SB architecture is that it has only one stage of buffering. Another difference is in the way that the switch fabric operates. In a CIOQ router, the switch fabric is a non-blocking crossbar switch, while in an SB router, the two interconnects are defined more generally. For example, the two interconnects in an SB router are not necessarily the same, and the operation of one may constrain the operation of the other.

We will explore one architecture in which both interconnects are built from a single crossbar switch. In another case we will explore an architecture in which the interconnect is a Clos network.

A number of existing router architectures fall into the SB model, such as the input queued router (in which the first stage interconnect is a fixed permutation, and the second stage is a non-blocking crossbar switch), the output queued router (in



(a) Single-buffered (SB) Architecture



(b) CIOQ Architecture

Figure 2.1: *A comparison of the CIOQ and SB router architectures.*

which the first stage interconnect is a broadcast bus, and the second stage is a fixed permutation), and the shared memory router (in which both stages are independent broadcast buses).

It is our goal to include as many architectures under the umbrella of the SB model as possible, then find tools to analyze their performance. We divide SB routers into two classes: (1) Routers with randomized switching or load balancing, for which we can at best determine statistical performance metrics, such as the conditions under which they achieve 100% throughput. We call these Randomized SB routers; and (2) Routers with deterministically scheduled switching, for which we can hope to find conditions under which they emulate a conventional output queued router and/or can provide delay guarantees for packets. We call these Deterministic SB routers.

In this thesis we will only study Deterministic SB routers. But for completeness, we will describe here some examples of both Randomized and Deterministic SB routers. For example, the well-known Washington University ATM Switch [34] – which is

essentially a buffered Clos network with buffering in the center stage – is an example of a Randomized SB architecture. The Parallel Packet Switch (PPS) [35] (which is further described in Chapter 6) is an example of a Deterministic SB architecture, in which arriving packets are deterministically distributed by the first stage over buffers in the central stage, and then recombined in the third stage.

In the SB model, we allow – where needed – the introduction of additional coordination buffers. This buffer is much smaller (and is usually placed on chip), compared to the large buffers used for congestion buffering. It can be used in either randomized or deterministic SB routers. For example, in the Washington University ATM Switch, resequencing buffers are used at the output because of the randomized load balancing at the first stage. In one version of the PPS, fixed-size coordination buffers are used at the input and output stages [36].

Other examples of the SB architecture include the load balancing switch proposed by Chang [37] (which is a Randomized SB and achieves 100% throughput, but mis-sequences packets), and the Deterministic SB variant by Keslassy [38] (which has delay guarantees and doesn't mis-sequence packets, but requires an additional coordination buffer).

☞**Observation 2.2.** Table 2.1 shows a collection of results for different deterministic and randomized routers. The routers are organized into eight types (shown in roman numerals) based on the architecture being used. Within each type the deterministic routers are denoted by D, and the randomized routers are denoted by R.

We've found that within each class of SB routers (Deterministic and Randomized), performance can be analyzed in a similar way. For example, Randomized SB routers are usually variants of the Chang load balancing switch, so they can be shown to have 100% throughput using Lyapunov functions and the standard Loynes construction [37, 12]. In a previous paper [43] we use this technique to analyze the randomized buffered

Table 2.1: *Unification of the theory of router architectures.*

Name	Type	# of memories	Memory Access Rate	Total Memory BW	Switch BW	Comment
Output Queued	I.D	N	$(N+1)R$	$N(N+1)R$	NR	OQ Router.
Centralized Shared Memory	I.D	1	$2NR$	$2NR$	$2NR$	OQ Router built with shared memory.
Input Queued [11, 12, 39]	II.D	N	$2R$	$2NR$	NR	100% throughput with MWM. Can't emulate OQ Router.
Parallel Shared Memory (PSM) (Section 2.4)	III.D	k	$3NR/k$	$3NR$	$2NR$	Emulates FCFS-OQ.
	III.D	k	$4NR/k$	$4NR$	$2NR$	Emulates WFQ OQ.
PSM (SMS - Prakash) [40]	III.D	$2N$	$2R$	$4NR$	$2NR$	Emulates FCFS-OQ
Distributed Shared Memory (Section 3.2 & 3.3)	IV.D	N	$3R$	$3NR$	$2NR$	Emulates FCFS-OQ; Bus Based.
	IV.D	N	$4R$	$4NR$	$2NR$	Emulates FCFS-OQ; Bus Based.
	IV.D	N	$3R$	$3NR$	$4NR$	Emulates FCFS-OQ; Xbar schedule complex.
	IV.D	N	$3R$	$3NR$	$6NR$	Emulates FCFS-OQ; trivial Xbar schedule.
	IV.D	N	$4R$	$4NR$	$5NR$	Emulates WFQ OQ; Xbar schedule complex.
	IV.D	N	$4R$	$4NR$	$8NR$	Emulates WFQ OQ ; trivial Xbar schedule.
	IV.D	N	$4R$	$4NR$	$4NR$	Emulates FCFS-OQ; simple Xbar schedule.
	IV.D	N	$6R$	$6NR$	$6NR$	Emulates WFQ OQ; simple Xbar schedule.
DSM (Two stage - Chang) [37]	IV.R	N	$2R$	$2NR$	$2NR$	100% throughput with mis-sequencing.
DSM (Two stage -Keslassy) [38]	IV.D	$2N$	$2R$	$4NR$	$2NR$	100% throughput, delay guarantees, no mis-sequencing.
Parallel Distributed Shared Memory (PDSM) (Section 3.7)	V.D	$(2h-1)xN$	R/h	$\frac{(2h-1)xNR}{h}$	yNR	FCFS Crossbar-based PDSM router with memories slower than R, where x and y are variables (See Table 3.1).
Parallel Distributed Shared Memory (PDSM) (Section 3.7)	V.D	$(3h-2)xN$	R/h	$\frac{(3h-2)xNR}{h}$	yNR	PIFO Crossbar-based DSM router with memories slower than R, where x and y are variables (See Table 3.1).
CIOQ Router (Box 4.2, [41])	VI.D	$2N$	$5R$	$10NR$	$4NR$	Emulates WFQ OQ with forced stable marriage.
CIOQ Router (Section 4.9, [13])	VI.D	$2N$	$3R$	$6NR$	$2NR$	Emulates WFQ OQ with complex preferred stable marriage.
CIOQ Router (Section 4.2)	VI.D	$2N$	$(3+\epsilon)R$	$(6+2\epsilon)NR$	$(2+\epsilon)NR$	Emulates constrained FCFS-OQ; trivial Xbar schedule.
Buffered CIOQ Router (Box 5.2)	VII.R	$2N$	$3R$	$6NR$	$2NR$	100% throughput with trivial input and output policy.
Buffered CIOQ Router [42]	VII.D	$2N$	$3R$	$6NR$	$2NR$	Emulates FCFS-OQ.
Buffered CIOQ Router (Section 5.5)	VII.D	$2N$	$3R$	$6NR$	$3NR$	Emulates WFQ OQ.
	VII.D	$2N$	$3R$	$6NR$	$2NR$	Emulates WFQ OQ with modified crossbar.
Parallel Packet Switch (PPS) (Section 6.5 & 6.6)	VIII.D	kN	$2(N+1)R/k$	$2N(N+1)R$	$4NR$	Emulates FCFS-OQ; centralized algorithm.
	VIII.D	kN	$3(N+1)R/k$	$3N(N+1)R$	$6NR$	Emulates WFQ OQ; centralized algorithm.
Buffered PPS (Section 6.7)	VIII.D	kN	$R(N+1)/k$	$N(N+1)R$	$2NR$	Emulates FCFS-OQ; distributed algorithm

CIOQ router (type VII.R in Table 2.1[1]) to prove the conditions under which it can give 100% throughput.

Likewise, the Deterministic SB routers that we have examined can be analyzed using constraint sets (described in Section 2.1.1) to find conditions under which they can emulate output queued routers. By construction, constraint sets also provide switch scheduling algorithms. We derive new results, switching algorithms, and bounds on the memory, and switching bandwidth for routers in each type (except the input queue router (type II) for which it is known that it cannot emulate an OQ router).

2.3 Unification of the Theory of Router Architectures

Constraint sets help us derive a number of new results, and more important, vastly simplify our understanding of high-speed router analysis and architectures. And so, we will show by the analysis of various router architectures that the "constraint set" technique unifies several results on router architectures under a common theoretical framework.

☞**Observation 2.3.** Essentially, constraint sets are simply a way to capture the load balancing constraints on memory and thus can be applied extensively to memory sub-systems. Table 2.1 summarizes the different router architectures that are analyzed in this thesis using the constraint set technique.

In this thesis, we describe two versions of the constraint set technique to analyze deterministic routers, as follows –

1. **Apply the basic constraint set technique (Algorithm 2.1) on deterministic SB routers:** We describe four Deterministic SB architectures that

[1]Henceforth when mentioning the router type, we drop the reference to the table number since it is clear from the context.

seem of practical interest but have been overlooked in the academic literature: (1) the parallel shared memory (PSM) router — a router built from a large pool of central memories (type III, analyzed in this chapter), (2) the distributed shared memory (DSM) router — a router built from a large but distributed pool of memories (type IV, Chapter 3), (3) the parallel distributed shared memory (PDSM) Router — a router that combines the techniques of the PSM and DSM router architectures (type IV, Section 3.7), and (4) the parallel packet switch (PPS) — a router built from a large pool of parallel routers running at a lower line rate (type VIII, Chapter 6). As we will see, we can use the basic constraint set technique to identify the conditions under which the above single-buffered router architectures can emulate an output queued router.

2. **Apply the basic constraint set technique (Algorithm 2.1) on deterministic CIOQ Routers:** We also apply the basic constraint set technique to analyze the CIOQ router (type IV, Chapter 4), which is *not* an SB router. In a CIOQ router, each packet gets buffered twice, as shown in Figure 1.3(b). We will consider two switch fabrics: (1) the classical unbuffered crossbar, and (2) the buffered crossbar, which is a crossbar with a small number of buffers. We will see that constraint sets can simplify our understanding of previously known results, and help derive new bounds to emulate an FCFS-OQ router.

3. **Apply the extended constraint set technique (Algorithm 4.2) on deterministic CIOQ routers:** In [13], the authors present a counting technique to analyze crossbar based CIOQ routers. In hindsight, the technique can be viewed as an application of the pigeonhole principle. So we will extend the basic constraint set technique and restate the methods in [13] in terms of the pigeonhole principle in Section 4.2. This helps to bring under a common umbrella the analysis techniques used to analyze deterministic routers. As we will see, the extended constraint set technique leads to tighter bounds and broader results for the CIOQ router as compared to the basic constraint set technique. We then apply this technique to a buffered CIOQ router to find the conditions under which a buffered CIOQ router can emulate an output queued router.

✂Box 2.1: The Pigeonhole Principle✂

The pigeonhole principle states the following —

Theorem 2.1. *Given two natural numbers p and h with $p > h$, if p items (pigeons) are put into h pigeonholes, then at least one pigeonhole must contain more than one item (pigeon).*

Figure 2.2: *The pigeonhole principle*

An equivalent way of stating this (used most often in this thesis and shown in Figure 2.2) [44] is as follows — if the number of pigeons is less than the number of pigeonholes, then there's at least one pigeonhole that must be empty.

The principle was first mentioned by Dirichlet in 1834, using the name Schubfachprinzip ("drawer principle"). It is rumored that the term *pigeonhole* itself was introduced to prevent any further discussion of Dirichlet's drawers! The principle itself is obvious, and humans have been aware of it historically. For example, the old game of musical chairs is a simple application of the pigeonhole principle.

☞**Observation 2.4.** The pigeonhole principle is powerful and has many uses in computer science and mathematics, *e.g.*, it can be used to prove that any hashing algorithm (where the domain is greater than the number of hash indices) cannot prevent collision, or that a lossless compression algorithm cannot compress the size of all its inputs. Surprisingly, it is also used to find good fractional approximations for irrational numbers.

In terms of set theory, pigeons and pigeonholes are simply two *finite* sets, say P and H; and the pigeonhole principle can be applied if $|P| > |H|$. It is interesting to note that the pigeonhole principle can also be applied to infinite sets, but only if it can still be shown that the cardinality of P is greater than H.[a]

Here's an interesting magic trick based on applying the pigeonhole principle — *A magician asks an audience member to randomly choose five cards from a card deck, then asks her to show them to an understudy. The understudy shows four of the five cards to the magician, one by one. The magician then guesses the fifth one.*

[a] And so, the principle can't be applied to the interesting mathematical paradox known as *Hotel Infinity* [45], where new guests can always be accommodated in a hotel that is full.

As we will show later in this thesis, we also use constraint sets to analyze two router data path tasks that use load balancing — (1) state maintenance (Chapter 10) and multicasting (Appendix J).

In the rest of the thesis, we will repeatedly use the following lemmas in our analysis using constraint sets. They are both direct consequences of the pigeonhole principle.

Lemma 2.1. *Let A_1, A_2 ... A_n be n sets, such that $\forall i \in \{1, 2, \ldots, n\}, |A_i| \leqslant k$, where k is the size of the universal set. If $|A_1| + |A_2| + \cdots + |A_n| > (n-1)k$, then there exists at least one element which is common to all sets,* i.e., *$\exists e, e \in A_1 \cap A_2 \cap \ldots A_n$.*

Lemma 2.2. *Let A_1, A_2 ... A_n be n sets, such that $\forall i \in \{1, 2, \ldots, n\}, |A_i| \leqslant k$, where $nk+1$ is the size of the universal set. Then there exists at least one element that does not belong to any of the sets,* i.e., *$\exists e, e \notin A_1 \cup A_2 \cup \ldots A_n$.*

2.4 The Parallel Shared Memory Router

2.4.1 Architecture

A shared memory router is characterized as follows: *When packets arrive at different input ports of the router, they are written into a centralized shared buffer memory. When the time arrives for these packets to depart, they are read from this shared buffer memory and sent to the egress line.* The shared memory architecture is the simplest technique for building an OQ router.

✎**Example 2.1.** Early examples of commercial implementations of shared memory routers include the SBMS switching element from Hitachi [46], the RACE chipset from Siemens [47], the SMBS chipset from NTT [48], the PRELUDE switch from CNET [49, 50], and the ISE chipset by Alcatel [51]. A recent example is the ATMS2000 chipset from MMC Networks [52].

In general, however, there have been fewer commercially available shared memory routers than those using other architectures. This is because (as described in Chapter 1)

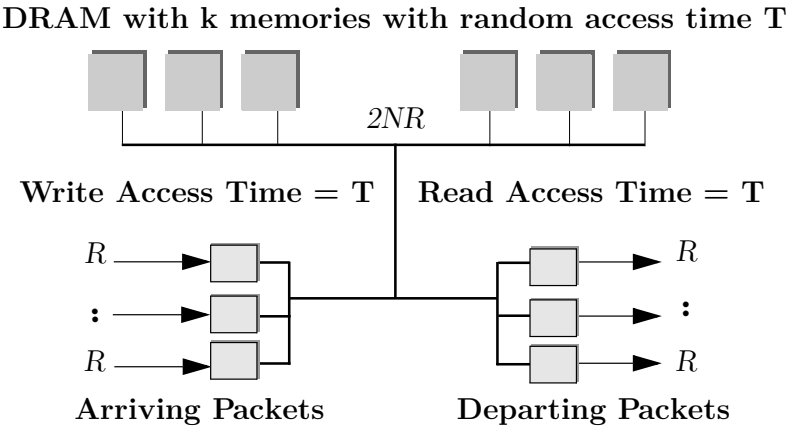


Figure 2.3: *The parallel shared memory router.*

it is difficult to scale the capacity of shared memory switches to the aggregate capacity required today.

An obvious question to ask is: *If the capacity of a shared memory router is larger than the bandwidth of a single memory device, why don't we simply use lots of memories in parallel?* However, this is not as simple as it first seems.

☞**Observation 2.5.** If the width of the memory data bus equals a minimum length packet (about 40 bytes), then each packet can be (possibly segmented and) written into memory. But if the width of the memory is wider than a minimum length packet,[2] it is not obvious how to utilize the increased memory bandwidth.

We cannot simply write (read) multiple packets to (from) the same memory location, as they generally belong to different queues. The shared memory contains multiple queues (at least one queue per output; usually more).

But we can control the memories individually, and supply each device with a separate address. In this way, we can write (read) multiple packets in parallel to (from)

[2]For example, a 160 Gb/s shared memory router built from memories with a random access time of 50 ns requires the data bus to be at least 16,000 bits wide (50 minimum length packets).

different memories. We call such a router a parallel shared memory router, as shown in Figure 2.3. $k \geqslant 2NR/B$ physical memories are arranged in parallel, where B is the bandwidth of one memory device. A central bus transports arriving (departing) packets to (from) different line cards. The bus carries all the $2N$ packets across it and has a bandwidth of $2NR$.

2.4.2 Why is a PSM Router Interesting?

The main appeal of the PSM router is its simplicity – the router is simply a bunch of memories. Also, it is scalable in terms of memory access time. If we had only one memory, as in the centralized shared memory router described in Chapter 1, a PSM router would not be scalable. However, in a PSM router, the access rate of the individual memories can be arbitrarily small, in which case we will simply need more memories.

2.4.3 Is the PSM Router Work-conserving?

We are interested in the conditions under which the parallel shared memory router can emulate an FCFS output queued router. This is equivalent to asking if we can always find a memory that is free for writing when a packet arrives, and that will also be free for reading when the packet needs to depart. Consider the following example.

✎**Example 2.2.** Figure 2.4 shows a PSM router with $N = 3$ ports. The outputs are numbered a, b, and c. Packets $(a4)$, $(a5)$, and $(c3)$ arrive to the router, where $(a4)$ denotes a packet destined to output a with departure time $DT = 4$. Packets $(a1)$, $(b1)$, and $(c1)$ depart from the PSM router in the current time slot, with $DT = 1$. Some packets that have arrived earlier, such as $(a3)$ and $(b3)$, are shown buffered in some of the memories of the PSM router. If arriving packets $(a4)$ and $(a5)$ are sent to the first two memories, then packet $c3$ is forced to be buffered in the memory numbered 5.

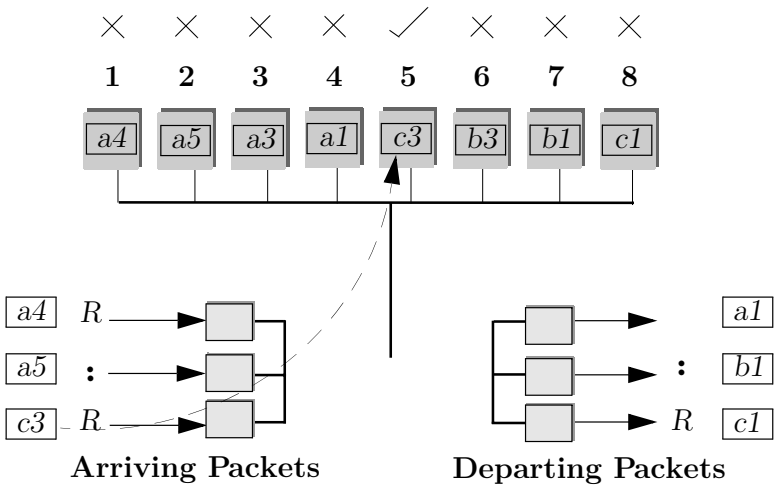


Figure 2.4: *An example where $k = 7$ memories are not enough, but $k = 8$ memories suffice.*

Clearly, if the packets were allocated as shown, and the memory numbered 5 was not present, the PSM router would not be work-conserving for this allocation policy. We are interested in knowing whether the PSM router is work-conserving in the general case, *i.e.*, for any value of N and k, irrespective of the arriving traffic pattern.

2.5 Emulating an FCFS Shared Memory Router

Using constraint sets, it is easy to see how many memories are needed for the parallel shared memory router to emulate an FCFS output queued router.

Theorem 2.2. *(Sufficiency) A total memory bandwidth of $3NR$ is sufficient for a parallel shared memory router to emulate an FCFS output queued router.*

Proof. (*Using constraint sets*) Assume that the aggregate memory bandwidth of the k memories is SNR, where $S > 1$. We can think of the access time T of a memory, as equivalent to k/S *decision slots.*[3] We will now find the minimum value of S needed for the switch to emulate an FCFS output queued router. Assume that all packets are segmented into cells of size C, and reassembled into variable-length packets before

[3]There are N arriving packets for which we need to make a decision in any time slot. So we shall denote N decision slots to comprise a time slot.

they depart. As follows, we define two constraint sets: one set for when cells are written to memory, and another for when they are read.

ℵ**Definition 2.1. Busy Write Set (BWS):** When a cell is written into a memory, the memory is busy for $\lceil k/S \rceil$ decision slots. $BWS(t)$ is the set of memories that are busy at the time, due to cells being written, and therefore cannot accept a new cell. Thus, $BWS(t)$ is the set of memories that have started a new write operation in the previous $\lceil k/S \rceil - 1$ decision slots. Clearly $|BWS(t)| \leqslant \lceil k/S \rceil - 1$.

ℵ**Definition 2.2. Busy Read Set (BRS):** Likewise, the $BRS(t)$ is the set of memories busy reading cells at time t. It is the set of memories that have started a read operation in the previous $\lceil k/S \rceil - 1$ decision slots. Clearly $|BRS(t)| \leqslant \lceil k/S \rceil - 1$.

Consider a cell c that arrives to the shared memory switch at time t destined for output port j. If c's departure time is $DT(t, j)$ and we apply the constraint set technique, then the memory l that c is written into must meet these constraints:

1. Memory l must not be busy writing a cell at time t. Hence $l \notin BWS(t)$.
2. Memory l must not be busy reading a cell at time t. Hence $l \notin BRS(t)$.
3. We must pick a memory that is not busy when the cell departs from the switch at $DT(t, j)$: Memory l must not be busy reading another cell when c is ready to depart: *i.e.*, $l \notin BRS(DT(t, j))$.

Hence our choice of memory l must meet the following constraints:

$$l \notin BWS(t) \wedge l \notin BRS(t) \wedge BRS(DT(t, j)). \tag{2.1}$$

A sufficient condition to satisfy this is:

$$k - |BWS(t)| - |BRS(t)| - |BRS(DT(t, j))| > 0 \tag{2.2}$$

From Definitions 2.1 and 2.2, we know that Equation 2.2 is true if:

$$k - 3(\lceil k/S \rceil - 1) > 0. \tag{2.3}$$

This is satisfied if $S \geqslant 3$, and corresponds to a total memory bandwidth of $3NR$.❐

Note that it is possible that an arriving cell must depart before it can be written to the memory, *i.e.*, $DT(t, j) < t + T$. In that case, the cell is immediately transferred to the output port j, bypassing the shared memory buffer. The algorithm described here sequentially searches the line cards to find a non-conflicting location for an arriving packet. Hence the complexity of the algorithm is $\Theta(N)$. Also, the algorithm needs to know the location of every packet buffered in the router. The bus connecting the line cards to the memory must run at rate $2NR$. While this appears expensive, in Chapter 3 we will explore ways to reduce the complexity using a crossbar switch fabric.

2.6 QoS in a Parallel Shared Memory Router

We now consider routers that provide weighted fairness among flows, or delay guarantees using WFQ [17] or GPS [18]. We find the conditions under which a parallel shared memory router can emulate an output queued router that implements WFQ. We will use the generalization of WFQ known as a "Push-in First-out" (PIFO) queue as described in Chapter 1. As follows, we will explore how a PSM router can emulate a PIFO output queued router that maintains N separate PIFO queues, one for each output.

2.6.1 Constraint Sets and PIFO queues in a Parallel Shared Memory Router

We saw above that if we know a packet's departure time when it arrives – which we do for FCFS – we can immediately identify the memory constraints to ensure the packet can depart at the right time. But in a router with PIFO queues, the departure time of a packet can change as new packets arrive and push in ahead of it. This complicates

the constraints; but as we will see, we can introduce an extra constraint set so as to choose a memory to write the arriving packet into. First, we'll explain how this works by way of an example; the general principle follows easily.

✎**Example 2.3.** Consider a parallel shared memory router with three ports, and assume that all packets are of fixed size. We will denote each packet by its initial departure order: Packet $(a3)$ is the third packet to depart, packet $(a4)$ is the fourth packet to depart, and so on. Figure 2.5(a) shows a sequence of departures, assuming that all the packets in the router are stored in a single PIFO queue. Since the router has three ports, three packets leave the router from the head of the PIFO queue in every time slot. Suppose packet $(a3')$ arrives and is inserted between $(a2)$ and $(a3)$, as shown in Figure 2.5(b). If no new packets push in, packet $(a3')$ will depart at time slot 1, along with packets $(a1)$ and $(a2)$ (which arrived earlier and are already in memory). To be able to depart at the same time, packets $(a1)$, $(a2)$, and $(a3')$ must be in different memories. Therefore, they must be written into a different memory.

Things get worse when we consider what happens when a packet is pushed from one time slot to the next. Figure 2.5(c) shows $(a1')$ arriving and pushing $(a3')$ into time slot 2. Packet $(a3')$ now conflicts with packets $(a3)$ and $(a4)$, which were in memory $(a3')$ when they arrived, and are also scheduled to depart in time slot 2. To be able to depart at the same time, packets $(a3')$, $(a3)$, and $(a4)$ must be in different memories.

In summary, when $(a3')$ arrives and is inserted into the PIFO queue, there are only four packets already in the queue that it could conflict with: $(a1)$ and $(a2)$ ahead of it, and $(a3)$ and $(a4)$ behind it. Therefore, we only need to make sure that $(a3')$ is written into a different memory than these four packets. Of course, new packets that arrive and are pushed in among these four packets will be constrained and must pick different memories, but these four packets will be unaffected.

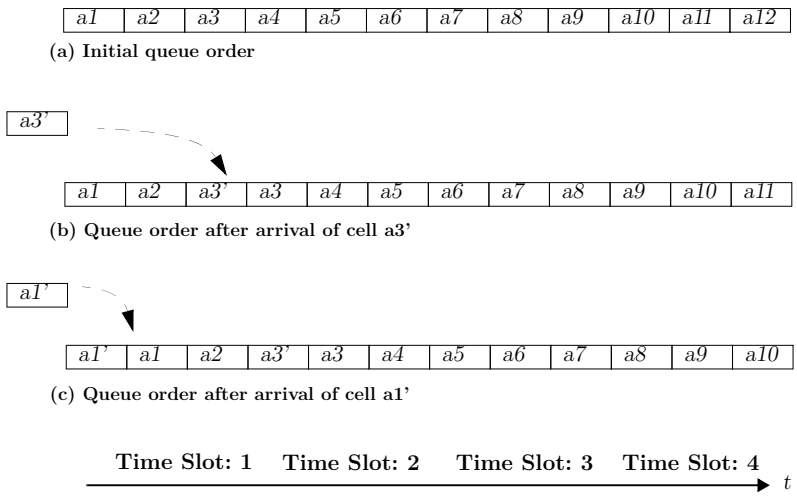


Figure 2.5: *Maintaining a single PIFO queue in a parallel shared memory router. The order of departures changes as new cells arrive. However, the relative order of departures between any two packets remains unchanged.*

In general, we can see that when a packet arrives to a PIFO queue, it should not use the memories used by the $N-1$ packets scheduled to depart immediately before or after it. This constrains the packet not to use $2(N-1)$ memories.

2.6.2 Complications When There Are N PIFO Queues

The example above is not quite complete. A PSM router holds N independent PIFO queues in one large pool of shared memory (not one PIFO queue, as the previous example suggests). When a memory contains multiple PIFO queues, the memory as a whole does not operate as a single PIFO queue, and so the constraints are more complicated. We'll explain by way of another example.

✎**Example 2.4.** Consider the same parallel shared memory router with three ports: a, b, and c. We'll denote each packet by its output port and its departure order at that output: packets $(b3)$ and $(c3)$ are the third packets to depart from outputs b and c, and so on. Figure 2.6(a) shows packets waiting to depart – one packet is scheduled to depart from each output during each time slot.

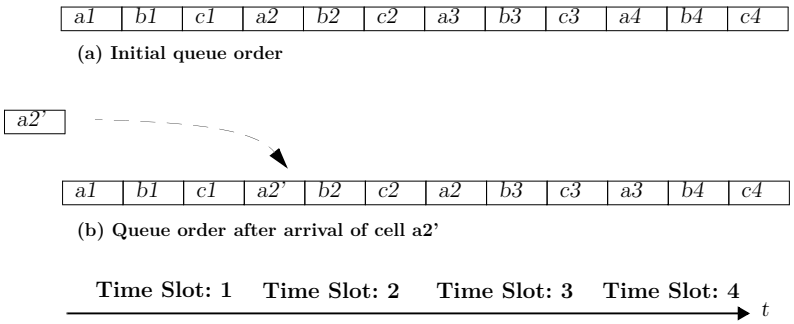


Figure 2.6: *Maintaining N PIFO queues in a parallel shared memory router. The relative order of departure of cells that belong to the same output remains the same. However, the relative order of departure of cells among different outputs can change due to the arrival of new cells.*

Assume that packet $(a2')$ arrives to output port a and is inserted between $(a1)$ and $(a2)$ (two packets scheduled to depart consecutively from port a). $(a2')$ delays the departure time of all the packets behind it destined to output a, but leaves unchanged the departure time of packets destined to other outputs. The new departure order is shown in Figure 2.6(b).

Taken as a whole, the memory (which consists of N PIFO queues) does not behave as one large PIFO queue. This is illustrated by packet $(a3)$, which is pushed back to time slot 4, and is now scheduled to leave after $(b3)$. The relative order of $(a3)$ and $(b3)$ has changed after they were in memory, so by definition of a PIFO, the queue is not a PIFO.

The main problem in the above example is that the number of potential memory conflicts is unbounded. This could happen if a new packet for output a was inserted between $(a2)$ and $(a3)$. Beforehand, $(a3)$ conflicted with $(b4)$ and $(c4)$; afterward, it conflicts with $(b5)$ and $(c5)$, both of which might already have been present in memory when $(a3)$ arrived. This argument can be continued. Thus, when packet $(a3)$ arrives, there is no way to bound the number of memory conflicts that it might have with

packets already present. In general, the arrivals of packets create new conflicts between packets already in memory.

2.6.3 Modifying the Departure Order to Prevent Memory Conflicts

We can prevent packets destined to different outputs from conflicting with each other by slightly modifying the departure order, as described in the following idea —

> **Idea.** *"Instead of sending one packet to each output per time slot, we can transmit several packets to one output, then cycle through each output in turn".*

More formally, consider a router with n ports and k shared memories. Let Π be the original departure order. This is described in Equation 2.4. In each time slot, a packet is read from memory for each output port. We will permute Π to give a new departure order Π', as shown in Equation 2.5. Exactly k packets are scheduled to depart each output during the k time slots. Each output receives service for k/n consecutive time slots, and k packets leave per output consecutively. The outputs are serviced in a round robin manner.

$$\begin{aligned}\Pi = (&a1, b1, \ldots, n1),\\ &(a2, b2, \ldots, n2),\\ &\ldots,\\ &(ak, bk, \ldots, nk).\end{aligned} \tag{2.4}$$

$$\begin{aligned}\Pi' = (&a1, a2, \ldots, ak),\\ &(b1, b2, \ldots, bk),\\ &\ldots,\\ &(n1, n2, \ldots, nk).\end{aligned} \tag{2.5}$$

Note that Π' allows each output to simply read from the k shared memories without conflicting with the other outputs. So, when an output finishes reading the k packets, all the memories are now available for the next output to read from. This resulting conflict-free permutation prevents memory conflicts between outputs.

The conflict-free permutation Π' changes the departure time of a packet by at most $k-1$ time slots. To ensure that packets depart at the right time, we need a small coordination buffer at each output to hold up to k packets. Packets may now depart at, at most, $k-1$ time slots later than planned.

We can now see how a parallel shared memory router can emulate a PIFO output queued router. First, we modify the departure schedule using the conflict-free permutation above. Next, we apply constraint sets to the modified schedule to find the memory bandwidth needed for emulation using the new constraints. The emulation is not quite as precise as before: the parallel shared memory router can lag the output queued router by up to $k-1$ time slots.

Theorem 2.3. *(Sufficiency) With a total memory bandwidth of $4NR$, a parallel shared memory router can emulate a PIFO output queued router within $k-1$ time slots.*

Proof. (*Using constraint sets*) Consider a cell c that arrives to the shared memory router at time t destined for output j, with departure time $DT(t,j)$ based on the conflict-free permutation. The memory l that c is written into must meet the following four constraints:

1. Memory l must not be busy writing a cell at time t. Hence $l \notin BWS(t)$.
2. Memory l must not be busy reading a cell at time t. Hence $l \notin BRS(t)$.
   The above constraints are similar to the conditions derived for an FCFS PSM router in Theorem 2.2.
3. Memory l must not have stored the $\lceil k/S \rceil - 1$ cells immediately in front of cell c in the PIFO queue for output j, because it is possible for cell c to be read out in the same time slot as some or all of the $\lceil k/S \rceil - 1$ cells immediately in front of it.
4. Similarly, memory l must not have stored the $\lceil k/S \rceil - 1$ cells immediately after cell c in the PIFO queue for output j.

Hence our choice of memory l must meet four constraints. Unlike Theorem 2.2, there is an additional memory constraint to satisfy, and so this requires a total memory bandwidth of $4NR$ for the PSM router to emulate a PIFO output queued router. ❐

2.7 Related Work

Our results on the use of the pigeonhole principle for parallel shared memory routers were first described in [53]. In work done independently, Prakash et al. [40] also analyze the parallel shared memory router. They prove that $2N - 1$ dual-ported memories (*i.e.*, where each memory can perform a read *and* a write operation per time slot) is enough to emulate an FCFS-OQ router (the authors do not consider WFQ policies). This requires a total memory bandwidth of $(4N - 2)R$. The bound obtained in our work is tighter than in [40], because single-ported memories allow for higher utilization of memory. We have not considered the implementation complexity of our switching algorithm, which allocates packets to memories. The proof techniques derived above implicitly assume $\Theta(N)$ time complexity. In [40], the authors take a different approach, demonstrating a parallel switching algorithm with time complexity $O(log^2 N)$. However, the algorithm utilizes additional memory bandwidth up to $\Theta(6NR)$ to efficiently compute the packet allocation, and requires a parallel machine with $O(N^2)$ independent processors.

2.7.1 Subsequent Work

In subsequent work [54, 55], the authors describe a randomized algorithm with low time complexity that can match packets to memories with high probability.

2.8 Conclusions

It is widely assumed that a shared memory router, although providing guaranteed 100% throughput and minimum delay, does not scale to very high capacities. Our results show that this is not the case. While our results indicate that a parallel shared memory router requires more overall memory bandwidth as compared to a

centralized shared memory router, the bandwidth of any individual memory can be made arbitrarily small by using more physical devices. The bounds we derive for the PSM router are upper bounds on the number of memories required to emulate an output queued router. Clearly, $3N - 1$ memories are sufficient, and at least $2N$ memories are necessary. However, we have been unable to show a tighter bound in the general case, and this remains an interesting open problem.

Summary

1. In Chapter 1, we introduced a conversation about pigeons and pigeonholes. We posed a question and showed its analogy to the design of high-speed routers.
2. We solve the analogy by the use of the pigeonhole principle.
3. We introduce a technique called "constraint sets" to help us formalize the pigeonhole principle, so that we can apply it to a broad class of router architectures.
4. The constraint set technique will help us determine the conditions under which a given router architecture can provide deterministic performance guarantees — the central goal of this thesis. Specifically, it will help us identify the conditions under which a router can emulate an FCFS-OQ router and a router that supports qualities of service.
5. We would like to bring under one umbrella a broad class of router architectures. So we propose an abstract model for routers, called single-buffered (SB) routers, where packets are buffered only once.
6. Single-buffered routers include a number of router architectures, such as centralized shared memory, parallel shared memory, and distributed shared memory, as well as input queued and output queued routers.
7. In Chapters 2, 3, and 7, we apply the pigeonhole principle to analyze single-buffered routers.
8. In Chapters 4, 5, we analyze combined input output queued (CIOQ) routers that have two stages of buffering.
9. The results of our analysis are summarized in Table 2.1.
10. The first router that we analyze is the parallel shared memory (PSM) router. The PSM router is appealing because of its simplicity; the router is built simply, from k parallel memories, each running at the line rate R.

11. A parallel shared memory router that does not have a sufficient number of memories, or that allocates packets to memories incorrectly, may not be work-conserving.
12. We show that a parallel shared memory router can emulate an FCFS output queue router with a memory bandwidth of $3NR$ (Theorem 2.2).
13. We also show that a PSM router can emulate an OQ router that supports qualities of service with a memory bandwidth of $4NR$ (Theorem 2.3).
14. It was widely believed that a shared memory router, although providing guaranteed 100% throughput and minimum delay, does not scale to very high capacities. Our results show that this is not the case.