Chapter 3: Analyzing Routers with Distributed Slower Memories

Jan 2008, Half Moon Bay, 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 distributed shared memory (DSM) router introduced in this chapter is an example of a single-buffered router (See Section 2.2), and the general load balancing technique to analyze this router is described in Section 2.3. The load balancing technique is also used to analyze other router architectures in Chapters 2, 4, 6, and 10.

Table: *List of Symbols.*

Δ	Vertex Degree of a Graph
c, C	Cell
DT	Departure Time
N	Number of Ports of a Router
R	Line Rate
T	Time Slot

Table: *List of Abbreviations.*

CIOQ	Combined Input Output Queued
DRAM	Dynamic Random Access Memory
DSM	Distributed Shared Memory
FCFS	First Come First Serve (Same as FIFO)
OQ	Output Queued
PIFO	Push In First Out
PDSM	Parallel Distributed Shared Memory
QoS	Quality of Service
RTT	Round Trip Time
SB	Single-buffered
TCP	Transmission Control Protocol
WFQ	Weighted Fair Queueing

"A distributed system is one in which the failure of a computer you didn't even know existed can render your own computer unusable".

— Leslie Lamport[†]

3

Analyzing Routers with Distributed Slower Memories

3.1 Introduction

In Chapter 2, we considered the parallel shared memory router. While this router architecture is interesting, it has the drawback that all k memories are in a central location. In a commercial router, we would prefer to add memories only as needed, along with each new line card. And so we now turn our attention to the distributed shared memory router shown in Figure 3.1. We assume that the router is physically packaged as shown in the figure, and that each line card contains some memory buffers.

At first glance, the DSM router looks like an input queued router, because each line card contains buffers, and there is no central shared memory. But the memories on a line card don't necessarily hold packets that have arrived to or will depart from that line card. In fact, the N different memories (one on each line card) can be thought of as collectively forming one large shared memory. When a packet arrives, it is transferred across the switch fabric (which could be a shared bus backplane, a crossbar switch, or some other kind of switch fabric) to the memory in another line card. This is shown in Figure 3.2. When it is time for the packet to depart, it is read

[†]Leslie Lamport (1941 –), Computer Scientist.

from the memory, passed across the switch fabric again, and sent through its outgoing line card directly to the output line. Notice that each packet is buffered in exactly one memory, and so the router is an example of a single-buffered router.

3.1.1 Why Are Distributed Shared Memory Routers Interesting?

From a practical viewpoint, the DSM router has the appealing characteristic that buffering is added incrementally with each line card. This architecture is similar to that employed by Juniper Networks in a commercial router [56], although analysis of the router's performance has not been published.[1]

☞**Observation 3.1.** A DSM router has a peculiar caveat that is inherent in all distributed systems. If a line card fails, it may have packets destined for it from multiple (if not all) line cards. Thus, a failure in one line card can cause packet drops to flows that arrive from and are destined to line cards that have nothing to do with the failed line card! However, this is an ephemeral problem, and only packets that are temporarily buffered in the failed line card are lost. The system can continue operating (albeit at a slower rate) in the absence of the failed line card. Alternatively, an additional line card can be provided that is automatically utilized in case of failure, allowing the system to operate at full line rates. (This is typically referred to as $N + 1$ resiliency.)

The DSM router's resiliency to failures, its graceful degradation, and the fault-tolerant nature of the system make it particularly appealing.

[1]The Juniper router appears to be a Randomized SB router. In the DSM router, the address lookup (and hence the determination of the output port) is performed before the packet is buffered, whereas in [56] the address lookup is performed afterward, suggesting that the Juniper router does not use the outgoing port number, or departure order, when choosing which line card will buffer the packet.

3.1.2 Goal

Our goal is to find the conditions under which a high-speed DSM router can give deterministic performance guarantees, as described in Section 1.4. In particular, this means we want to derive the conditions under which a DSM router can emulate an FCFS-OQ router and a PIFO-OQ router. We will consider two switch fabrics in a DSM router — (1) a bus, and (2) a crossbar.

3.2 Bus-based Distributed Shared Memory Router

In a bus-based DSM router, the switch fabric is a bus that carries all arriving and departing packets. This router is logically equivalent to a parallel shared memory router as long as the shared bus has sufficient capacity, *i.e.*, a switching bandwidth $2NR$. Rather than placing all the memories centrally, they are moved to the line cards. Therefore, the theorems for the PSM router derived in Chapter 2 also apply directly to the distributed shared memory router. While these results may be interesting, the bus bandwidth is too large.

✎**Example 3.1.** Consider a bus-based DSM router with $N = 32$ ports and $R = 40$ Gb/s, *i.e.*, a router with a capacity of 1.28 Tb/s. The switch fabric would have to switch twice the bandwidth, and would require a shared multidrop broadcast bus with a capacity of 2.56 Tb/s. This is not practical with today's serial link and connector technologies, as it would require over 400 links!

3.3 Crossbar-based DSM Router

We can replace the shared broadcast bus with an $N \times N$ crossbar switch, then connect each line card to the crossbar switch using a short point-to-point link. This is similar

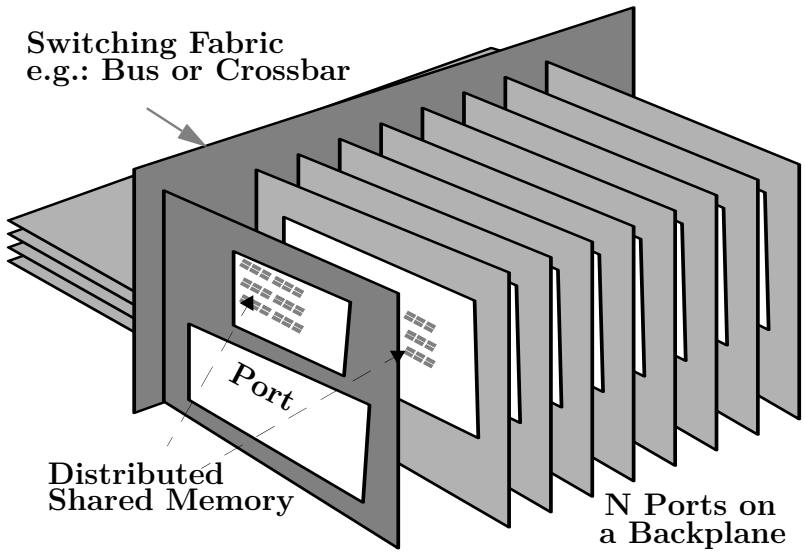
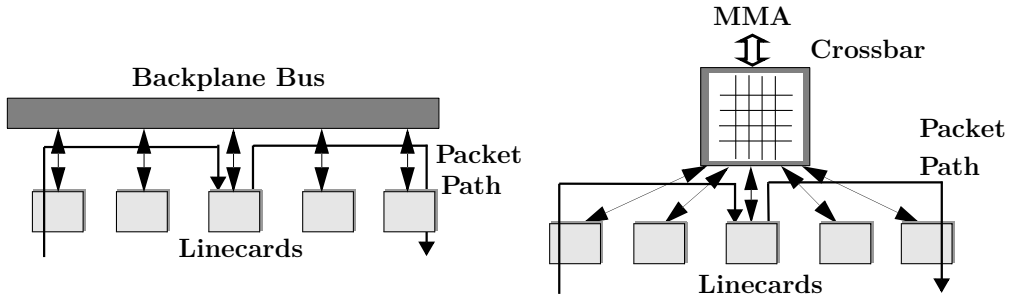


Figure 3.1: *Physical view of the DSM router. The switch fabric can be either a backplane or a crossbar. The memory on a single line card can be shared by packets arriving from other line cards.*



(a) Bus Interconnect (b) Crossbar Interconnect

Figure 3.2: *Logical view of the DSM router. An arriving packet can be buffered in the memory of any line card, say x. It is later read by the output port from the intermediate line card x.*

to the way input queued routers are built today, although in a distributed shared memory router every packet traverses the crossbar switch twice.

The crossbar switch needs to be configured each time packets are transferred, and so we need a scheduling algorithm that will pick each switch configuration. (Before, when we used a broadcast bus, we didn't need to pick the configuration, as there was sufficient capacity to broadcast packets to the line cards.) We will see that there are several ways to schedule the crossbar switch, and that each has its pros and cons. We will find different algorithms, and for each algorithm we will find the speed at which the memories and crossbar need to run.

We will define the bandwidth of a crossbar to be the speed of the connection from a line card to the switch, and we will assume that the link bandwidth is the same in both directions. So, for example, we know that each link needs a bandwidth of at least $2R$ just to carry every packet across the crossbar fabric twice. In general, we find that we need a higher bandwidth than this in order to emulate an output queued router. The additional bandwidth serves three purposes:

1. It provides additional bandwidth to write into (read from) the memories on the line cards to overcome the memory constraints.
2. It relaxes the requirements on the scheduling algorithm that configures the crossbar.
3. Because the link bandwidth is the same in both directions, it allocates a bandwidth for the peak transfer rate in one direction, even though we don't usually need the peak transfer rate in both directions at the same time.

3.4 A Crossbar-based DSM router Can Emulate an FCFS Output Queued Router

We start by showing trivially sufficient conditions for a Crossbar-based DSM router to emulate an FCFS output queued router. We then follow with tighter results that show how the crossbar bandwidth can be reduced at the cost of either increased memory bandwidth, or a more complex crossbar scheduling algorithm.

Theorem 3.1. *(Sufficiency) A Crossbar-based DSM router can emulate an FCFS output queued router with a total memory bandwidth of* $3NR$ *and a crossbar bandwidth of* $6NR$.

Proof. Consider operating the crossbar in two phases: first, read all departing packets from memory and transfer them across the crossbar. From Theorem 2.2, this requires at most three transfers per line card per time slot. In the second phase, write all arriving packets to memory, requiring at most three more transfers per line card per time slot. This corresponds to running the link connecting the line card to the crossbar at a speed of $6R$, and a crossbar bandwidth of $6NR$. ❐

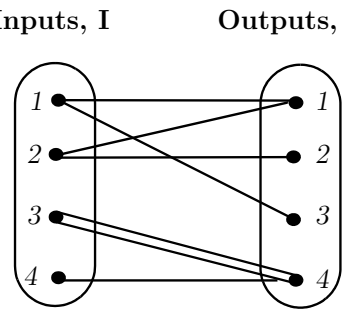
Theorem 3.2. *(Sufficiency) A Crossbar-based DSM router can emulate a PIFO output queued router with a total memory bandwidth of* $4NR$ *and a crossbar bandwidth of* $8NR$ *within a relative delay of* $2N-1$ *time slots.*

Proof. This will follow directly from Theorem 2.3 and the proof of Theorem 3.1. How the crossbar is scheduled is described in the proof of Theorem 3.4. ❐

3.5 Minimizing the Crossbar Bandwidth

We can represent the set of memory operations in a time slot using a bipartite graph with $2N$ vertices, as shown in Figure 3.3. An edge connecting input i to output j represents an (arriving or departing) packet that needs to be transferred from i to j. In the case of an arrival, the output incurs a memory write; and in the case of a departure, the input incurs a memory read. The degree of each vertex is limited by the number of packets that enter (leave) the crossbar from (to) an input (output) line card. Recall that for an FCFS router, there are no more than three memory operations at any given input or output. Given that each input (output) vertex can also have an arrival (departure), the maximum degree of any vertex is four.

Theorem 3.3. *(Sufficiency) A Crossbar-based DSM router can emulate an FCFS output queued router with a total memory bandwidth of* $3NR$ *and a crossbar bandwidth of* $4NR$.



(a) Request Graph, G

Matrix

1	0	1	0
1	1	0	1
0	0	0	2
0	0	0	1

(b) Request Matrix, G

Figure 3.3: *A request graph and a request matrix for an $N \times N$ switch.*

Proof. From the above discussion, the degree of the bipartite request graph is at most 4. From [57, 58] and Theorem 2.2, a total memory bandwidth of $3NR$, a crossbar link speed of $4R$, and a crossbar bandwidth of $4NR$ is sufficient. ❒

Theorem 3.4. *(Sufficiency) A Crossbar-based DSM router with a total memory bandwidth of $4NR$ and a crossbar bandwidth of $5NR$ can emulate a PIFO output queued router within a relative delay of $2N-1$ time slots.*

Proof. The proof is in two parts. First we shall prove that a conflict-free permutation schedule Π' over N time slots can be scheduled with a crossbar bandwidth $5R$. Unlike the crossbar-based distributed shared memory router, the modified conflict-free permutation schedule Π' cannot be directly scheduled on the crossbar, because the conflict-free permutation schedules N cells to each output per time slot. However, we know that the memory management algorithm schedules no more than 4 memory accesses to any port per time slot. Since each input (output) port can have no more than N arrivals (departures) in the N time slots, the total out-degree per port in the request graph for Π' (over N time slots), is no more than $4N + N = 5N$. From König's method [57, 58], there exists a schedule to switch the packets in Π', with a crossbar link speed of $5R$. This corresponds to a crossbar bandwidth of $5NR$.

Now we show that a packet may incur a relative delay of no more than $2N-1$ time slots when the conflict-free permutation Π' is scheduled on a crossbar. Assume that

the crossbar is configured to schedule cells departing between time slots (a_1, a_N) (and these configurations are now final), and that other cells prior to that have departed. The earliest departure time of a newly arriving packet is time slot a_1. However, a newly arriving cell cannot be granted a departure time between (a_1, a_N), since the crossbar is already being configured for that time interval. Hence, Π' will give the cell a departure time between (a_{N+1}, a_{2N}), and the cell will leave the switch sometime between time slots (a_{N+1}, a_{2N}). Hence the maximum relative delay that a cell can incur is $2N - 1$ time slots. From Theorem 2.3, the memory bandwidth required is no more than $4NR$. ❐

3.6 A Tradeoff between Crossbar Bandwidth and Scheduler Complexity

Theorem 3.3 is the lowest bound that we have found for the crossbar bandwidth ($4NR$), and we suspect that it is a necessary condition to emulate an FCFS output queued router. Unfortunately, edge-coloring has complexity $O(N \log \Delta)$ [31] (where Δ is the vertex degree of the graph), and is too complex to implement at high speed. We now explore a more practical algorithm which also needs a crossbar bandwidth of $4NR$, but requires the memory bandwidth to be increased to $4NR$. The crossbar is scheduled in two phases:

1. **Write-Phase**: Arriving packets are transferred across the crossbar switch to memory on a line card, and
2. **Read-Phase**: Departing packets are transferred across the crossbar from memory to the egress line card.

Theorem 3.5. *(Sufficiency) A Crossbar-based DSM router can emulate an FCFS output queued router with a total memory bandwidth of $4NR$ and a crossbar bandwidth of $4NR$.*

Proof. (Using constraint sets). We will need the following definitions to prove the theorem.

ℵ**Definition 3.1. Busy Vertex Write Set (BVWS):** When a cell is written into an intermediate port x during a crossbar schedule, port x is no longer available during that schedule. $BVWS(t)$ is the set of ports busy at t due to cells being written, and therefore cannot accept a new cell. Since, for a given input, no more than $N-1$ other arrivals occur during that time slot, clearly $|BVWS(t)| \leqslant \lfloor (N-1)/S_W \rfloor$).

ℵ**Definition 3.2. Busy Vertex Read Set (BVRS):** Similarly, $BVWS(t)$ is the set of ports busy at t due to cells being read, and that therefore cannot accept a new cell. Since, for a given output, no more than $N-1$ other arrivals occur during that time slot, clearly $|BVRS(t)| \leqslant \lfloor (N-1)/S_R \rfloor$).

Consider cell c that arrives to the crossbar-based distributed shared memory router at time t destined for output j, with departure time $DT(t,j)$. Applying the constraint set method, our choice of intermediate port x to write c into must meet these constraints:

1. Port x must be free to be written to during at least one of the s_W crossbar schedules reserved for writing cells at time t. Hence, $x \notin BVWS(t)$.
2. Port x must not conflict with the reads occurring at time t. However, since the write and read schedules of the crossbar are distinct, this will never happen.
3. Port x must be free to be read from during at least one of the S_R crossbar schedules reserved for reading cells at time $D(t)$. Hence, $x \notin BVRS(D(t))$.

Hence our choice of memory must meet the following constraints:

$$x \notin BVWS(t) \wedge x \notin BVRS(DT(t,j)) \tag{3.1}$$

This is true if $S_R, S_W \geqslant 2$. Hence, we need a crossbar link speed of $S_C R = (S_R + S_W)R = 4R$. Because a memory requires just two reads and two writes per time slot, the total memory bandwidth is $4NR$. ❒

Theorem 3.6. *(Sufficiency) A Crossbar-based DSM router can emulate a PIFO output queued router within a relative delay of $N-1$ time slots, with a total memory bandwidth of $6NR$ and a crossbar bandwidth of $6NR$.*

Proof. (Using constraint sets). Similar to Theorem 3.5, we consider a cell c arriving at time t, destined for output j and with departure time $DT(t,j)$ (which is based on the conflict-free permutation departure order Π'). Applying the constraint set method, our choice of x to write c into meets these constraints:

1. Port x must be free to be written to during at least one of the S_W crossbar schedules reserved for writing cells at time t.
2. Port x must not conflict with the reads occurring at time t. However, since the write and read schedules of the crossbar are distinct, this will never happen.
3. Port x must not have stored the $\lceil N/S_R \rceil - 1$ 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 $\lceil N/S_R \rceil - 1$ cells in front of it.
4. Port x must not have stored the $\lceil N/S_R \rceil - 1$ cells immediately after cell c in the PIFO queue for output j.

Hence our choice of port x must meet one write constraint and two read constraints, which can be satisfied if $S_R, S_W \geqslant 3$. Hence, we need a crossbar link speed of $S_C R = (S_R + S_W)R = 6R$. A memory can have three reads and three writes per time slot, corresponding to a total memory bandwidth of $6NR$. Note that $S_R = 2, S_W = 4$ will also satisfy the above theorem. ❐

In summary, we have described three different results. Let's now compare them based on memory bandwidth, crossbar bandwidth, and the complexity of scheduling the crossbar switch when the router is emulating an ideal FCFS shared memory router. First, we can trivially schedule the crossbar with a memory bandwidth of $3NR$ and a crossbar bandwidth of $6NR$ (Theorem 3.1). With a more complex scheduling algorithm, we can schedule the crossbar with a memory bandwidth of $4NR$ and a

crossbar bandwidth of $4NR$ (Theorem 3.5). But our results suggest that, although possible, it is complicated to schedule the crossbar when the memory bandwidth is $3NR$ and the crossbar bandwidth is $4NR$. We now describe a scheduling algorithm for this case, although we suspect there is a simpler algorithm that we have been unable to find.

The bipartite request graph used to schedule the crossbar has several properties that we can try to exploit:

1. The total number of edges in the graph cannot exceed $2N$, *i.e.*, $\sum_i \sum_j R_{ij} \leqslant 2N$. This is also true for any subset of vertices; if I and J are subsets of indices $\{1, 2, \ldots, N\}$, then $\sum_{i \in I} \sum_{j \in J} R_{ij} \leqslant |I| + |J|$. We complete the request graph by adding requests so that it has exactly $2N$ edges.
2. In the complete graph, the degree of each vertex is at least one, and is bounded by four, *i.e.*, $1 \leqslant \sum_i R_{ij} \leqslant 4$ and $1 \leqslant \sum_j R_{ij} \leqslant 4$.
3. In the complete graph, $\sum_j R_{ij} + \sum_j R_{ji} \leqslant 5$. This is because each vertex can have at most three operations to memory, and one new arrival (which may get buffered on a different port), and one departure (which may be streamed out from a memory on a different port).[2]
4. The maximum number of edges between an input and an output is 2, *i.e.*, $R_{ij} \leqslant 2$. We call such a pair of edges a *double edge*.
5. Each vertex can have at most one double edge, *i.e.*, if $R_{ij} = 2$, then $R_{ik} < 2 (k \neq j)$ and $R_{kj} < 2 (k \neq i)$.
6. In a complete request graph, if an edge connects to a vertex with degree one, the other vertex it connects to must have a degree greater than one. This means, if $\sum_j R_{mj} = R_{mn} = 1$, then $\sum_i R_{in} \geqslant 2$; if $\sum_i R_{in} = R_{mn} = 1$, then, $\sum_j R_{mj} \geqslant 2$. To see why this is so, suppose an edge connects input i, which has degree one, and output j. This edge represents a packet arriving at i and stored at j. But j has a departure that initiates another request, thus the degree of j is greater

[2] This constraint was missed in an earlier publication [59].

than one. By symmetry, the same is true for an edge connecting an output of degree one.

3.6.1 Implementation Considerations for the Scheduling Algorithm

Our goal is to exploit these properties to find a crossbar scheduling algorithm that can be implemented on a wave-front arbiter (WFA [15]). The WFA is widely used to find maximal size matches in a crossbar switch. It can be readily pipelined and decomposed over multiple chips [60].

ℵ**Definition 3.3. Inequalities of vectors** – v_1 and v_2 are vectors of the same dimension. The index of the first non-zero entry in v_1 (v_2) is i_1 (i_2). We will say that $v_1 \geqslant v2$ iff $i_1 \leqslant i_2$, and $v_1 = v_2$ iff $i_1 = i_2$.

ℵ**Definition 3.4. Ordered** - The row (column) vectors of a matrix are said to be ordered if they do not increase with the row (column) index. A matrix is ordered if both its row and column vectors are ordered.

Lemma 3.1. *A request matrix can be ordered in no more than $2N - 1$ alternating row and column permutations.*

Proof. See Appendix C.1. ❒

Theorem 3.7. *If a request matrix S is ordered, then any maximal matching algorithm that gives strict priority to entries with lower indices, such as the WFA [15], can find a conflict-free schedule.*

Proof. See Appendix C.2. ❒

This algorithm is arguably simpler than edge-coloring, although it depends on the method used to perform the $2N - 1$ row and column permutations.

3.7 The Parallel Distributed Shared Memory Router

The DSM router architecture assumes that each memory allows one read or one write access per time slot. This requires the scheduler to keep track of $3N-1$ ($4N-2$) memories in order to emulate an FCFS (WFQ) OQ router. For line rates up to 10 Gb/s, it seems reasonable today to use a single commercially available DRAM on each line card to run at that rate. For line rates above 10 Gb/s, we need even more memories operating in parallel. If there are too many memories for the scheduler to keep track of, it may become a bottleneck. So we will explore ways to alleviate this bottleneck, as motivated by the following idea —

> **Idea.** *"Local to each line card, we could allocate packets to parallel memories running at a slower rate (*i.e.*, $< R$), even though the scheduler believes that there is one memory which operates at rate R".*

So we have to find a way in which a memory that runs at rate R can be *emulated* locally on each line card by multiple memories running at a slower rate. This is done as follows — The scheduler allocates packets to memories assuming that they run at rate R. It assumes that there are a total of $3N-1$ ($4N-2$) memories in the router, so that it can emulate an FCFS (WFQ) OQ router. However, in reality, when the scheduler informs a line card to access a memory at rate R, the memory accesses will be load-balanced locally by the line card over multiple memories operating at a slower rate R/h. In other words, the scheduler operates identically to the DSM router described in Section 3.3, while each line card operates like the parallel shared memory router described in Chapter 2. We call this router the parallel distributed shared memory (PDSM) router. We will use constraint sets to determine how many memory devices are needed locally to perform this emulation.

Theorem 3.8. *A set of $2h-1$ memories of rate R/h running in parallel can emulate a memory of rate R in an FCFS PDSM router.*

Proof. Consider the memory that runs at rate R that is going to be emulated by using load balancing. Since the scheduler has already ensured that at any give time, a packet is either written to or read from this memory (and not both), the read and write constraints at the current time collapse into a single constraint. The rest of the analysis is similar to Theorem 2.2. This results in requiring only $2h-1$ memories running at rate R/h. ❐

Theorem 3.9. *A set of $3h-2$ memories of rate R/h running in parallel can emulate a memory of rate R in a PIFO PDSM router.*

Proof. The analysis is similar to Theorem 3.8. As assumed in that proof, we will modify the departure time of packets leaving the output (see Section 2.6.3) so that packets from different outputs do not conflict with each other, and a PIFO order of departures is maintained. Also, since the scheduler has already ensured that, at any given time, a packet is either written to or read from the memory (and not both) that will be emulated, the read and write constraints at the current time collapse into a single constraint. This results in only three constraints, and requires only $3h-2$ memories running at rate R/h. ❐

We can apply the above emulation technique shown in Theorem 3.8 and Theorem 3.9 for all the results obtained for the DSM router. Table 3.1 summarizes these results.

3.8 Practical Considerations

In this section, we investigate whether we could actually build a DSM router that emulates an output queued router. As invariably happens, we will find that the architecture is limited in its scalability, and that the limits arise for the usual reasons when a system is big: algorithms that take too long to complete, buses that are too wide, connectors and devices that require too many pins, or overall system power that is impractical to cool. Many of these constraints are imposed by technologies available today, and may disappear in future. We will therefore, when possible, phrase our comments to allow technology-independent comparisons, *e.g.*, "Architecture A has half the memory bandwidth of Architecture B".

Table 3.1: *Comparison between the DSM and PDSM router architectures.*

Name	Type	Row	# of mem	Mem. Access Rate	Total Mem. BW	Switch BW	Comment
Crossbar-based Distributed Shared Memory Router (Section 3.3)	IV.D	1a	$3N$	R	$3NR$	$4NR$	Emulates FCFS-OQ, complex crossbar schedule.
		1b	$3N$	R	$3NR$	$6NR$	Emulates FCFS-OQ, trivial crossbar schedule.
		1c	$4N$	R	$4NR$	$5NR$	Emulates OQ with WFQ, complex crossbar schedule.
		1d	$4N$	R	$4NR$	$8NR$	Emulates OQ with WFQ, trivial crossbar schedule.
		1e	$4N$	R	$4NR$	$4NR$	Emulates FCFS-OQ, simple crossbar schedule.
		1f	$6N$	R	$6NR$	$6NR$	Emulates OQ with WFQ, simple crossbar schedule.
Crossbar-based Parallel Distributed Shared Memory Router (Section 3.7)	V.D	2a	$3N(2h-1)$	R/h	$\approx 6NR$	$4NR$	Theorem 3.8 applied to Row 1a.
		2b	$3N(2h-1)$	R/h	$\approx 6NR$	$6NR$	Theorem 3.8 applied to Row 1b.
		2c	$4N(3h-2)$	R/h	$\approx 12NR$	$5NR$	Theorem 3.9 applied to Row 1c.
		2d	$4N(3h-2)$	R/h	$\approx 12NR$	$8NR$	Theorem 3.9 applied to Row 1d.
		2e	$4N(2h-1)$	R/h	$\approx 8NR$	$4NR$	Theorem 3.8 applied to Row 1e.
		2f	$6N(3h-2)$	R/h	$\approx 18NR$	$6NR$	Theorem 3.9 applied to Row 1f.

We will pose a series of questions about feasibility, and attempt to answer each in turn.

1. A PIFO DSM router requires lots of memory devices. Is it feasible to build a system with so many memories? We can answer this question relative to a CIOQ router (since it's the most common router architecture today) that emulates an OQ router. From [13], it is known that a CIOQ router with a crossbar bandwidth of $2NR$ can emulate a WFQ OQ router with $2N$ physical memory devices running at rate $3R$ for an aggregate memory bandwidth of $6NR$. The PIFO DSM router requires N physical devices running at rate $4R$ for an aggregate memory bandwidth of $4NR$. So the access rates of the individual memories are comparable, and the total memory bandwidth is less than that

needed for a CIOQ router. It seems clear, from a memory perspective, that we can build a PIFO DSM router with at least the same capacity as a CIOQ router. This suggests that, considering only the number of memories and their bandwidth, it is possible to build a 1 Tb/s single-rack DSM router.

2. A crossbar-based PIFO DSM router requires a crossbar switch with links operating at least as fast as $5R$. A CIOQ router requires links operating at only $2R$. What are the consequences of the additional bandwidth for the DSM router? Increasing the bandwidth between the line card and the switch will more than double the number of wires and/or their data rate, and place more requirements on the packaging, board layout, and connectors. It will also increase the power dissipated by the serial links on the crossbar chips in proportion to the increased bandwidth. But it may be possible to exploit the fact that the links are used asymmetrically.

   ☞**Observation 3.2.** For example, we know that the total number of transactions between a line card and the crossbar switch is limited to five per time slot. If each link in the DSM router was half-duplex rather than simplex, then the increase in serial links, power, and size of connectors is only 25%. Even if we can't use half-duplex links, the power can be reduced by observing that many of the links will be unused at any one time, and therefore need not have transitions. But overall, in the best case it seems that the DSM router requires at least 25% more bandwidth.

3. In order to choose which memory to write a packet into, we need to know the packet's departure time as soon as it arrives. This is a problem for both a DSM router and a CIOQ router that emulates an output queued router. In the CIOQ router, the scheduling algorithm needs to know the departure time to ensure that the packet traverses the crossbar in time. While we can argue that the DSM router is no worse, this is no consolation when the CIOQ router itself is impractical! Let's first consider the simpler case, where a DSM router

is emulating an FCFS shared memory router. Given that the system is work-conserving, the departure time of a packet is simply equal to the sum of the data in the packets ahead of it. In principle, a global counter can be kept for each output, and updated at each time slot depending on the number of new arrivals. All else being equal, we would prefer a distributed mechanism, since the maintenance of a global counter will ultimately limit scalability. However, the communication and processing requirements are probably smaller than for the scheduling algorithm itself (which we will consider next).

4. How complex is the algorithm that decides which memory each arriving packet is written into? There are several aspects to this question:

   - *Space requirements*: In order to make its decision, the algorithm needs to consider k different memory addresses, one for each packet that can contribute to a conflict. How complex the operation is, depends on where the information is stored. If, as currently seems necessary, the algorithm is run centrally, then it must have global knowledge of all packets. While this is also true in a CIOQ router that emulates an output queued router, it is not necessary in a purely input or output queued router.
   - *Memory accesses*: For an FCFS DSM router, we must read, update, and write bitmaps representing which memories are busy at each future departure time. This requires two additional memory operations in the control structure. For a PIFO DSM router, the cost is greater, as the control structures are most likely arranged as linked lists, rather than arrays. Finding the bitmaps is harder, and we don't currently have a good solution to this problem.
   - *Time*: We have not found a simple distributed algorithm that does not use any additional memory bandwidth, and so currently we believe it to be sequential, requiring $O(N)$ operations to schedule at most N new arrivals. However, it should be noted that each operation is a simple comparison of three bitmaps to find a conflict-free memory.

- *Communication*: The algorithm needs to know the destination of each arriving packet, which is the minimum needed by any centralized scheduling algorithm.

5. We can reduce the complexity by aggregating packets at each input into frames of size F, and then schedule frames instead of packets. This is called *frame scheduling* (a technique in wide use in some Cisco Ethernet switches and Enterprise routers). Essentially, this is equivalent to increasing the size of each "cell". The input line card maintains one frame of storage for each output, and a frame is scheduled only when F bits have arrived for a given output, or until a timeout expires. There are several advantages to scheduling large frames rather than small cells. First, as the size of frame increases, the scheduler needs to keep track of fewer entities (one entry in a bitmap per frame, rather than per cell), and so the size of the bitmaps (and hence the storage requirement) falls linearly with the frame size. Second, because frames are scheduled less often than cells, the frequency of memory access to read and write bitmaps is reduced, as is the communication complexity, and the complexity of scheduling as shown in the following example.

   ✎**Example 3.2.** Consider a DSM router with 16 OC768c line cards (*i.e.*, a total capacity of 640 Gb/s). If the scheduler were to run every time we scheduled a 40-byte cell, it would have to use off-chip DRAM to store the bitmaps, and access them every 8 ns. If, instead, we use 48 kB frames, the bitmaps are reduced more than 1,000-fold and can be stored on-chip in fast SRAM. Furthermore, the bitmap interaction algorithm needs to run only once every 9.6 μs, which is readily implemented in hardware.

   The appropriate frame size to use will depend on the capacity of the router, the number of line cards, and the technology used for scheduling. This technique can be extended to support a small number of priorities in a PIFO DSM router, by aggregating frames at an input for every priority queue for every output. One disadvantage of this approach is that the strict FCFS order among all

inputs is no longer maintained. However, FCFS order is maintained between any input-output pair, which is all that is usually required in practice.

6. Which requires larger buffers, a DSM router or a CIOQ router? In a CIOQ router, packets between a given input and output pass through a fixed pair of buffers. The buffers on the egress line cards are sized so as to allow TCP to perform well, and the buffers on the ingress line card are sized to hold packets while they are waiting to traverse the crossbar switch. So the total buffer size for the router is at least $NR \times RTT$ because any one egress line card can be a bottleneck for the flows that pass through it. On the other hand, in a DSM router we can't predict which buffer a packet will reside in; the buffers are shared more or less equally among all the flows. It is interesting to note that if the link data rates are symmetrical, not all of the egress line cards of a router can be bottlenecked at the same time. As a consequence, statistical sharing reduces the required size of the buffers. This reduces system cost, board area, and power. As a consequence of the scheduling algorithm, the buffers in the DSM router may not be equally filled. We have not yet evaluated this effect.

3.9 Conclusions

It seems that the PIFO DSM router has two main problems: (1) The departure times of each packet must be determined centrally (or at least by each output separately) with global knowledge of the state of the queues in the system, and (2) A sequential scheduler must find an available memory for each packet (similar to that faced in a PSM router).

The *frame scheduling* approach described in Section 3.8 can help alleviate the former problem by reducing the rate at which departure times need to be calculated. In subsequent work, the authors [54, 55] propose parallel or randomized bi-partite matching algorithms to allocate packets to memories for the PSM router. These algorithms can also be applied to alleviate the latter problem by eliminating the need for a sequential scheduler for the DSM router. This requires us to extend the algorithms in [54, 55], and take into account the additional link access required for

the arriving (departing) packet as described in Section 3.5. In some cases this requires additional memory bandwidth. Also, if the memories operate slower than the line rate, the PDSM router can be used, and can help reduce the number of memories that the centralized scheduler needs to keep track of. This simplifies the scheduler's complexity, but comes at the expense of additional memory bandwidth.

Our conclusion is that a PIFO DSM router is less complex (see Table 2.1 for a comparison) than a PIFO CIOQ router (has lower memory bandwidth, fewer memories, a simpler scheduling algorithm, but slightly higher crossbar bandwidth).

Summary

1. In this chapter, we analyze the distributed shared memory (DSM) router and the parallel distributed shared memory (PDSM) router.
2. When a packet arrives to a DSM router, it is transferred across the switch fabric to the memory in another line card, where it is held until it is time for the packet to depart.
3. DSM routers have the appealing characteristic that buffering is added incrementally with each line card.
4. Although commercially deployed, the performance of a DSM router is not widely known.
5. We use the constraint set technique (introduced in the previous chapter) to analyze the conditions under which a DSM router can emulate an OQ router. We consider two fabrics: (1) a bus and (2) a crossbar.
6. We show that a parallel shared memory router can emulate an FCFS output queue router with a memory bandwidth of $3NR$ (Theorem 3.1), and can emulate an OQ router that supports qualities of service with a memory bandwidth of $4NR$ (Theorem 3.2).
7. The results for the memory bandwidth are the same as those described for the PSM router, since from a memory perspective the two routers are identical.
8. The interesting problem for DSM routers is: how do we schedule packets (which are allocated to memories based on the pigeonhole principle) across a crossbar-based fabric?
9. We show that there is a tradeoff between the memory bandwidth, crossbar bandwidth, and the complexity of the scheduling algorithm. We derive a number of results that take advantage of these tradeoffs (Table 3.1).
10. The central scheduler for the DSM router must keep track of all the memories in the

router. If the memories operate at a very low rate, it would require more memories to keep track of. This can limit the scalability of the scheduler.

11. So, we explore ways in which the central scheduler believes that it load-balances over a smaller number of memories. However, when the central scheduler allocates these packets to memories, the line card that receives these packets load-balances them locally over a larger number of slower memories operating in parallel. These slower memories are not visible to the scheduler.
12. We call this the parallel distributed shared memory (PDSM) router, because the router appears as a standard DSM router from the central scheduler's perspective, while it appears as a PSM router from the line card's perspective.
13. We derive bounds on the memory bandwidth required for a PDSM router to emulate an FCFS-OQ router (Theorem 3.8) and emulate an OQ router that supports qualities of service (Theorem 3.9).
14. Based on the results derived in this chapter, we believe that the DSM and PDSM router architectures are promising. However, questions remain about their complexity. But we find that the memory bandwidth, and potentially the power consumption of the router, are lower than for a CIOQ router.