Routers with a Single Stage of Buffering*

Sigcomm Paper Number: 342, Total Pages: 14

Abstract -- **Most high performance routers today use combined input and output queueing (CIOQ). The CIOQ router is also frequently used as an abstract model for routers: at one extreme is input queueing, at the other extreme is output queueing, and in-between there is a continuum of performance as the speedup is increased from 1 to N (where N is the number of linecards). The model includes architectures in which a switch fabric is sandwiched between two stages of buffering. There is a rich and growing theory for CIOQ routers, including algorithms, throughput results and conditions under which delays can be guaranteed.**

But there is a broad class of architectures that are not captured by the CIOQ model, including routers with centralized shared memory, and load-balanced routers. In this paper we propose an abstract model called Single-Buffered (SB) routers that includes these architectures. We describe a method called Constraint Sets to analyze a number of SB router architectures.

The model helped identify previously unstudied architectures, in particular the Distributed Shared Memory router. Although commercially deployed, its performance is not widely known. We find conditions under which it can emulate an ideal shared memory router, and believe it to be a promising architecture. Questions remain about its complexity, but we find that the memory bandwidth, and potentially the power consumption of the router is lower than for a CIOQ router.

Keywords--**Packet Switching, Model, Single Stage Buffering**

I. INTRODUCTION

A. Background

The first Internet routers consisted of linecards connected to a shared backplane. Arriving packets were written into a central pool of shared buffer memory where they waited their turn to depart. The reasons for using a shared memory architecture are well known. First, the router's throughput is maximized: Like an output queued switch, a shared memory router is work-conserving and so achieves 100% throughput and minimizes the average queueing delay of packets. Network operators prefer routers that can guarantee 100% throughput so that they can maximize the utilization of their expensive long-haul links. Second, a shared memory router can control the rate given to each flow and the delay of individual packets using weighted fair queueing [1] and its variants [2][3][4]. In a shared memory router, the shared buffer memory must have sufficient bandwidth to accept packets from and write packets to all of the linecards at the same time. In other words, for a router with N linecards each connected to a line at rate R, the shared memory must have a bandwidth of $2NR$.

Since the first routers were introduced, the capacity of commercial routers[1] has increased by about 2.2 times every 18 months (slightly faster than Moore's Law). Routers can continue to use centralized shared memory only if memory bandwidth keeps up with the increased capacity of the router. Unfortunately, this is not the case. Router buffers are built from commercial DRAMs, which are optimized for size rather than speed, and the random access time to commercial DRAMs has increased by only about 1.1 times every 18 months (slower than Moore's Law) [5].[2] By the mid 1990s, router capacity grew to a point where central shared memory could no longer be used, and it became popular to use input queueing instead. The linecards were connected to a non-blocking crossbar switch which was configured by a centralized scheduling algorithm. From a practical point of view, input queueing allows the memory to be distributed to each linecard, where it can be added incrementally. More importantly, each memory need only run at a rate $2R$ (instead of $2NR$) enabling higher capacity routers to be built. Theoretical results showed that: (1) With virtual output queues (VOQs) and a maximum weight matching algorithm an input queued router can achieve 100% throughput [6], (2) With a speedup of two, and with combined input and output queueing (CIOQ), the router can emulate an ideal shared memory router [7], and (3) with a speedup greater than two, and WFQ schedulers at both inputs and outputs, the router can provide delay and bandwidth guarantees [8][9].

Table 1 summarizes some well-known results for CIOQ routers. While the results in Table 1 might be appealing to the router architect, the algorithms required by the theoretical results are not practical at high speed because of the complexity of the scheduling algorithms. And so the theoretical results have not made

* Research grant information to be provided later.

1. We define the capacity of a router to be the sum of the maximum data rates of its linecards, NR. For example, we will say that a router with 16 OC192c linecards has a capacity of approximately 160Gb/s.

2. It is interesting to ask whether SRAM — which tracks the speed of Moore's Law — could be used instead. Unfortunately, SRAM is not dense enough. The largest commercial SRAM today is approximately 16Mbits. A router needs to store about $RTT \times R$ bits for TCP congestion control to work efficiently. A 160Gb/s router with an RTT of 0.25seconds, requires 40Gbits of buffering, or 2,500 SRAM devices! Given that router capacity roughly tracks SRAM density, SRAM will continue to be impractical for shared memory routers.

TABLE 1 The CIOQ model for switch architectures

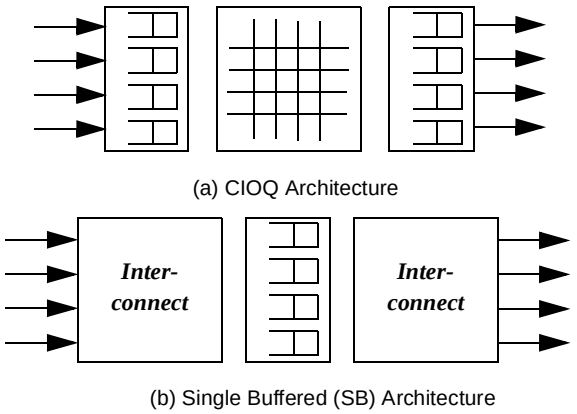
Type	Number of memories	BW of each memory	Total BW of memories	Crossbar Speed (if applicable)	Comment
Input Queued	N	$2R$	$2NR$	NR	100% throughput with maximum weight matching [6], or randomized algorithms [12].
Output Queued	N	$(N+1)R$	$N(N+1)R$		Work conserving, 100% throughput, delay guarantees.
CIOQ Speedup of two	$2N$	$3R$	$6NR$	$2NR$	With maxim*al* size matching: 100% throughput [13].
					With CCF: emulates OQ with WFQ [7].

much difference to the way routers are built. Instead, most routers use a heuristic scheduling algorithm such as *i*SLIP [10] or WFA [11], and a speedup of approximately two. Performance studies are limited to simulations suggesting that most of the queueing takes place at the output, so WFQ schedulers are usually placed on the egress linecards to provide differentiated qualities of service. While this might be a sensible engineering compromise, the resulting system has unpredictable performance. There are no throughput, fairness or delay guarantees, and the worst case is not known.

In summary, CIOQ has emerged as a common router architecture, but the performance of practical CIOQ routers is difficult to predict. This is not very satisfactory given that routers make up such a large fraction of the Internet infrastructure.

Our goal is to find more tractable and practical router architectures, and this leads us to consider a different model, one that we call the Single Buffered (SB) router.

Whereas a CIOQ router 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. Figure 1 illustrates both architectures. A key distinguishing 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. In an SB router, the two interconnects are defined more generally. In particular, the two interconnects in an SB router are not necessarily the same, and the operation of one might constrain the operation of the other. For example, we will explore one architecture in which both interconnects are built from a single crossbar switch.



(a) CIOQ Architecture

(b) Single Buffered (SB) Architecture

Figure 1: A comparison of the CIOQ and SB router architectures.

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 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, then find general tools for analyzing their performance. We divide the SB 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 shared memory router and/or can provide delay guarantees for packets. We call these Deterministic SB routers.

For example, the well-known Washington University ATM Switch [14] — which is essentially a buffered Clos network, with buffering only in the center stage — is an example of a Randomized SB architecture. Arriving packets are randomly load-balanced over the single stage of buffering. Similarly, the Parallel Packet Switch (PPS) [15] is an example of a Deterministic SB architecture, in which arriving packets are carefully and deterministically distributed by the first stage over a single stage of buffering in the central stage, then recombined in the 3rd stage. In this paper, we will be considering only the Deterministic SB router.

In the SB model, we allow — where needed — the introduction of additional (usually small) coordination buffers, so long as they are not used because of congestion. For example, in the Washington University ATM Switch, resequencing buffers are

TABLE 2 : Routers according to the Single Buffered architecture.

Type	# of memories	BW of memory	total BW	crossbar BW (if applicable)	Comment
Input Queued	N	$2R$	$2NR$	NR	100% throughput with MWM.
Output Queued	N	$(N+1)R$	$N(N+1)R$		Gives best theoretical performance.
Parallel Packet Switch [15][16]	kN	$2R(N+1)/k$	$2N(N+1)R$		Emulates FCFS OQ.
	kN	$3R(N+1)/k$	$3N(N+1)R$		Emulates OQ with WFQ.
Buffered Parallel Packet Switch	kN	$R(N+1)/k$	$N(N+1)R$		Emulates FCFS OQ.
Two-Stage (Chang [17])	N	$2R$	$2NR$		100% throughput with mis-sequencing.
Two-Stage (Keslassy [18])	$2N$	$2R$	$4NR$	$2NR$	100% throughput, delay guarantees, no mis-sequencing.
Shared Memory	1	$2NR$	$2NR$		Gives best theoretical performance.
Parallel Shared Memory or Bus-based Distributed Shared Memory (Section II and III)	k	$3NR/k$	$3NR$		Emulates FCFS OQ.
	k	$4NR/k$	$4NR$		Emulates OQ with WFQ.
Crossbar-based Distributed Shared Memory (Section IV)	N	$3R$	$3NR$	$4NR$	Emulates FCFS OQ, but crossbar schedule complex.
				$6NR$	Emulates FCFS OQ, with simple crossbar schedule.
	N	$4R$	$4NR$	$5NR$	Emulates OQ with WFQ, but crossbar schedule complex.
				$8NR$	Emulates OQ with WFQ, with simple crossbar schedule.
	N	$4R$	$4NR$	$4NR$	Emulates FCFS OQ.
	N	$6R$	$6NR$	$6NR$	Emulates OQ with WFQ.
FCFS Crossbar-based Parallel and Distributed Shared Memory (Section V)	$(2h-1)xN$	R/h	$(2h-1)xNR/h$	yNR	FCFS Crossbar-based DSM switch with memories slower than R, where xNR and yNR are memory and crossbar speeds of the DSM.
PIFO Crossbar-based Parallel and Distributed Shared Memory (Section V)	$(3h-2)xN$	R/h	$(3h-2)xNR/h$	yNR	PIFO Crossbar-based DSM switch with memories slower than R, where xNR and yNR are memory and crossbar speeds of the DSM.

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 [16].

Other examples of the SB architecture include the interesting load-balancing switch proposed by Chang [17] (which is a Randomized SB and achieves 100% throughput, but mis-sequences packets), and the Deterministic SB variant by Keslassy [18] (which has delay guarantees and doesn't mis-sequence packets, but requires an additional coordination buffer). Table 2 shows a collection of results for different SB routers, some of which are proved later in this paper.

We believe that many (but not all) interesting SB architectures can be analyzed the same way. For example, the Randomized SB routers that we have examined appear to be variants of the Chang load-balancing switch. Therefore, they can be shown to have 100% throughput using the differential equation-based fluid models described in [17] or [19].

Likewise, the Deterministic SB routers can be analyzed using Constraint Sets to find conditions under which they can emulate ideal shared memory routers. Constraint Sets are introduced in Section II and are simply a generalization of the way that Charles Clos used the pigeon-hole principle to find conditions under which a Clos network is strictly non-blocking [20]. This method was introduced for analyzing the parallel packet switch [15][16]. We shall use and extend the above technique to analyze more general Deterministic SB routers.

In the remainder of this paper we'll only be considering Deterministic SB routers. In what follows, we describe two Deterministic SB architectures that seem practically interesting, but have been overlooked in the academic literature. As we will see, Constraint Sets can be used to find conditions under which both router architectures can emulate an ideal shared memory router. We call the first architecture the Parallel Shared Memory (PSM) router, which has a centralized shared memory that is decomposed into a number of parallel memories. The second architecture we call the Distributed Shared Memory (DSM)

router, in which memory is distributed to each linecard. At first glance, the DSM router looks like an input queued router, because each linecard contains buffers, and there is no central shared memory. However, the buffers do not necessarily hold packets that arrived from, or are destined to the linecard. All buffers are a shared and distributed resource available to all linecards. From a practical viewpoint, the DSM router has the appealing characteristic that buffering is added incrementally with each linecard. This architecture is particularly interesting as it appears to have been used in the widely deployed Juniper M40 router. A patent describes the general approach [21], although the router's performance has not been published. The DSM router described here differs from the architecture described in [21]. In the DSM router, the address lookup (and hence the determination of the output port) is performed before the packet is buffered, whereas in [21] the address lookup is performed afterwards. This leads us to assume that the M40 router does not use the outgoing port number, or departure order, when choosing which linecard will buffer the packet. Presumably, the M40 is an example of a Randomized SB router, and so is not covered by the analysis described here.

Perhaps the most interesting outcome of this paper is the comparison between two routers that emulate an ideal shared memory router that performs weighted fair queueing (WFQ). The CIOQ router requires $2N$ memories, each running at a speed of $3R$, for a total memory bandwidth of $6NR$. Ignoring the complexity of the WFQ scheduler, the centralized scheduling algorithm has a complexity of at least $O(C^2)$, where C is the number of packets in the system. In Section IV we show that the DSM router requires N memories running at a speed of $4R$, for a total memory bandwidth of $4NR$, with simpler scheduling algorithms. In Section VI we consider the implementation complexity of different DSM routers.

B. Performance metrics

Throughout this paper we will be using memory bandwidth as a means to compare different router architectures. It serves as a good metric for two reasons: (1) Routers are, and will continue to be, limited by the bandwidth of commercially available memories. All else being equal, a router with small overall memory bandwidth can have a higher capacity, and (2) A router with higher memory bandwidth will, in general, consume more power. Core routers are frequently limited by the power that they consume (because they are backed up by batteries) and dissipate (because they must use forced air cooling). The total memory bandwidth indicates the total bandwidth of the high-speed serial links that connect the memories to control logic. In current systems, the power dissipated by high speed serial links often accounts for over 50% of the router's power.

We will not be using the commonly used metric known as "speedup". The term speedup is used differently by different authors, and there is no accepted standard definition. For example, the input queues in a CIOQ router with a "speedup" of two perform two read operations for every write. Do they have a speedup of one and a half, or two? On the other hand, the crossbar in the router operates at twice the speed that it would in a purely input queued switch. So instead of the term speedup, we will use the term "bandwidth". In our example above, the input queues have a memory bandwidth of 3*R*, and the crossbar has a bandwidth of 2*R*.

II. The Parallel Shared Memory Router

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 just use lots of memories in parallel, as shown in Figure 2? This is not as simple as it first seems. 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 (about 40bytes), 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). For example, a 160Gb/s shared memory router built from memories with a random access time of 50ns requires the data bus to be at least 16,000 bits wide (50 minimum length packets) to meet the bandwidth requirement.

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) different memories. We call such a router a Parallel Shared Memory Router. $k \geq 2NR/B$ physical memories are arranged in parallel, where B is the bandwidth of one memory device. We are interested in the conditions under which the Parallel Shared Memory router behaves identically to a shared memory router. More precisely, if we apply the same traffic to a Parallel Shared Memory router and to an ideal shared memory router, we would like to find the conditions under which identical packets will depart from both at the same time.[3] This is equivalent to asking if we can always find a memory that is free for writing when a packet arrives, and

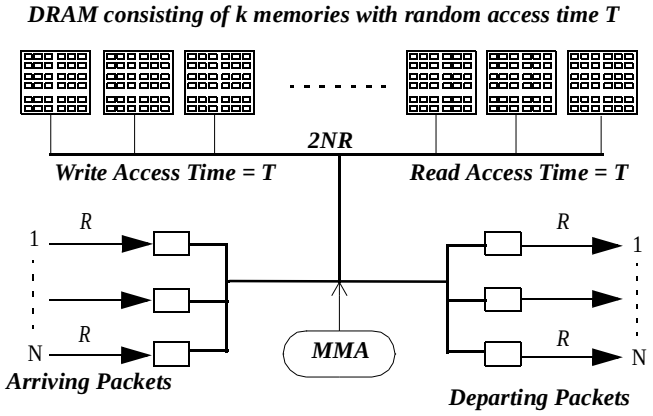


Figure 2: Memory hierarchy of the Parallel Shared Memory switch, showing a large DRAM memory. The DRAM memory has a total bandwidth of at least 2NR. The logical DRAM memory consists of multiple separate DRAMs each of which run at a slower rate.

will be also be free for reading when the packet needs to depart. We will, shortly, show how, but first we'll describe a simple technique, called Constraint Sets, that we will use repeatedly to analyze Deterministic SB routers.

A. Constraint Sets

1) Pigeons and pigeon holes

Consider M pigeon holes, where each hole may contain several pigeons. Each time slot up to N pigeons arrive, which must immediately be placed into a pigeon hole. Likewise, each time slot up to N pigeons depart. Now suppose we constrain the pigeon hole so that in any one time slot at most one pigeon may arrive to it, or at most one pigeon may depart from it. We do not allow a pigeon to enter a pigeon hole while another one is departing.

Now we ask the question: How many pigeon holes do we need 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, $D(t)$. 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. Hence by the well-known pigeon-hole principle, if $M \geq 3N-1$ our pigeon can find a hole.

2) Using Constraint Sets

In a Deterministic SB router, the arriving (departing) packets are written to (read from) memories that are constrained to either read or write in any one time slot. We can use the pigeon hole technique to determine how many memories are needed, and to design an algorithm to decide which memory each arriving packet is written into. We use the following three steps:

Step 1. Determine packet's departure time, $D(t)$: If packets 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. If the packets are scheduled to depart in a more complicated way, for example using WFQ, then it is harder to determine its departure time. We'll consider this in more detail in Section C. For now, we'll assume that the $D(t)$ is known for each packet.

Step 2. Define the Constraint Sets: Identify the constraints on the resource (such as buffer, switch fabric, etc.) for each incoming packet.

Step 3. Apply the Pigeon hole principle: Add up all the constraints, and apply the pigeon-hole principle.

Overall, the technique of using constraint sets is a generalization of the approach used by Charles Clos in his classic analysis of the conditions under which a 3-stage circuit switch is strictly non-blocking [20].

[3] We shall ignore time differences due to propagation delays, pipelining etc. and consider only queueing delays in this comparison.

B. A Parallel Shared Memory router can emulate an FCFS Shared Memory router

From our pigeon hole example, it is easy to see how many memories are needed for the Parallel Shared Memory router to emulate an ideal shared memory router.

Theorem 1: (Sufficiency) A total memory bandwidth of $3NR$ is sufficient for a Parallel Shared Memory Router to emulate an ideal shared memory router.

Proof: *(Using Constraint Sets)* See Appendix A. ❐

The algorithm described in the Appendix sequentially searches the linecards to find a non-conflicting location for an arriving packet. Hence the complexity of the algorithm is $O(N)$. Also the algorithm needs to know the location of every packet buffered in the router. While this appears expensive, we will explore ways to reduce the complexity in Section VI.

C. QoS in a Parallel Shared Memory router

Many routers provide weighted fairness among flows, or delay guarantees using WFQ or GPS [1][22]. We will now find the conditions under which a Parallel Shared Memory Router can emulate an ideal shared memory router that implements WFQ. We will use the generalization of WFQ known as a "Push-in First-out Queue" (PIFO) [7]. A PIFO is defined as follows:

1. Arriving packets are placed at (or, "pushed-in" to) an arbitrary location in the departure queue.
2. The relative ordering of packets in the queue does not change once packets are in the queue.
3. Packets depart from the head of line.

PIFO queues include strict priority queues, and a variety of work-conserving QoS disciplines such as WFQ. In what follows, we will show that a Parallel Shared Memory router can emulate a shared memory router with PIFO queues.

1) Additional memory constraints for PIFO queues

First, let's look at an example of a PIFO schedule for a Parallel Shared Memory router. Let the Parallel Shared Memory router have three ports, *a*, *b*, and *c*. We will assume that packets all have fixed size, and denote each by its output port and its departure order at the output: Packet $(b3)$ is the third packet to depart from output b, and so on. Figure 3(a) shows an example of some packets waiting to depart.

Now assume that a packet arrives for output port a. Since, each output port follows a PIFO scheduling policy, assume that the new packet is inserted after $(a1)$ but before $(a2)$. We'll call the new arrival $(a2')$. When it's inserted into the queue, $(a2')$ delays the departure time of all the packets behind it destined to output *a*. The new departure order is shown in Figure

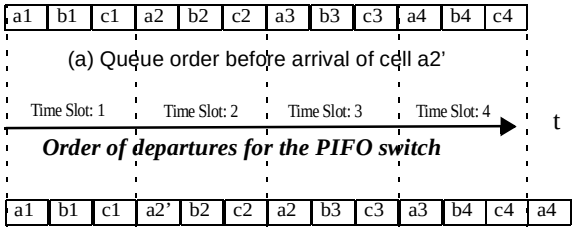


Figure 3: An arriving cell a2' is pushed in between cells a1 and a-2 destined for output a. The departure order for the whole switch after insertion is no longer PIFO despite each output having a PIFO scheduling policy.

3(b). Consider any packet destined to output port a, which is queued to depart later than $(a2')$. In particular consider $(a3)$ which (in order to avoid memory conflicts when it departs) was written into a different memory from $(b3)$ and $(c3)$. However, when $(a2')$ arrives, it changes the departure order so that $(a3)$ now departs in the same time slot as $(b4)$ and $(c4)$. And so to avoid memory conflicts, we need to make sure that $(a3)$ is in a different memory from $(b4)$ and $(c4)$ too. This creates additional potential memory conflicts for cell $(a3)$.

2) Modifying the departure order to reduce memory conflicts

We can reduce the number of memory conflicts by permuting the departure order of the packets. For example, consider a router with n ports and k shared memories. If the departure order was:

$$\Pi = (a1, b1, \ldots, n1), (a2, b2, \ldots, n2), \ldots, (ak, bk, \ldots, nk)$$

i.e., in each time slot a packet departs from each output port, we can permute it to give:

$$\Pi' = (a1, a2, \ldots, ak), (b1, b2, \ldots, bk), \ldots, (n1, n2, \ldots, nk)$$

i.e., exactly k packets are scheduled to depart each output during the k time slots, and each output can simply read from the k shared memories without conflicting with the other outputs. When an output completes 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.

After we apply the conflict-free permutation, the departure time of a packet changes by up to $k-1$ time slots; and so to ensure that it departs at exactly the right time, we need a small coordination buffer at each output to hold up to k packets. This also means that the packet might depart at most $k-1$ time slots later than planned.

We can now see how a Parallel Shared Memory can emulate a shared memory router with PIFO queues. 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. Note that the emulation is not quite as precise as before: the Parallel Shared Memory router can lag the shared memory router by up to $k-1$ time slots.

Theorem 2: (Sufficiency) With a total memory bandwidth of $4NR$ a Parallel Shared Memory router can emulate a PIFO shared memory router, within $k-1$ time slots.

Proof: *(Using Constraint Sets)*. See Appendix A. ❐

III. Distributed Shared Memory Routers

Up until now we have considered only the Parallel Shared Memory router shown in Figure 2. 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 linecard. This is particularly important in high-end routers, where the total amount of memory is huge, and linecards are expensive.

And so for practical reasons, we now turn our attention to the Distributed Shared Memory router illustrated in Figure 4. We assume that the router is physically packaged as shown in Figure 4(a) — each linecard contains some memory buffers as in an input queued router. But the memories on a linecard don't necessarily hold packets that arrived at or will depart from that linecard. In fact, the N different memories (one on each linecard) 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 linecard. When it is time for the packet to depart, it is read from the memory and passed across the switch fabric again, and sent through its outgoing linecard directly onto the output line.

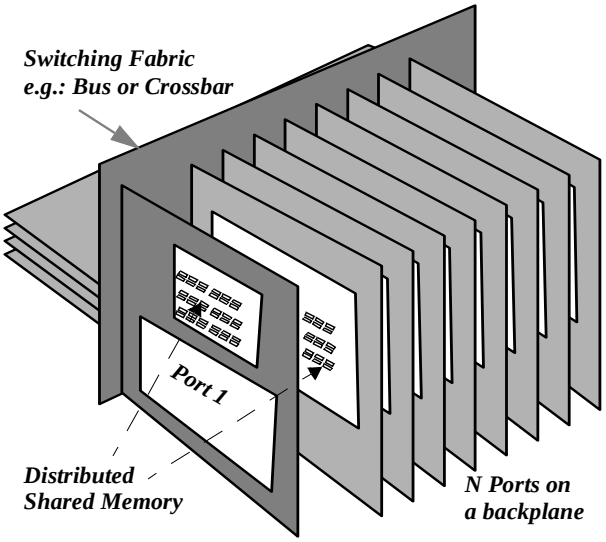
Notice that each packet is buffered in exactly one memory, and so the router is an example of a Single-Buffered router. The Distributed Shared Memory router is logically equivalent to a Parallel Shared Memory as long as the shared bus has sufficient capacity. Instead of all the memories being placed centrally, they are just moved out to the linecards. As a result, we can immediately conclude that the theorems for the Parallel Shared Memory router also apply to the Distributed Shared Memory router.

While these results are interesting, the bus bandwidth is too large to be practical in a high capacity router. For example, a 160Gb/s router would require a shared multidrop broadcast bus with a capacity of 480Gb/s (or 640Gb/s). This is not practical with today's serial link and connector technology.

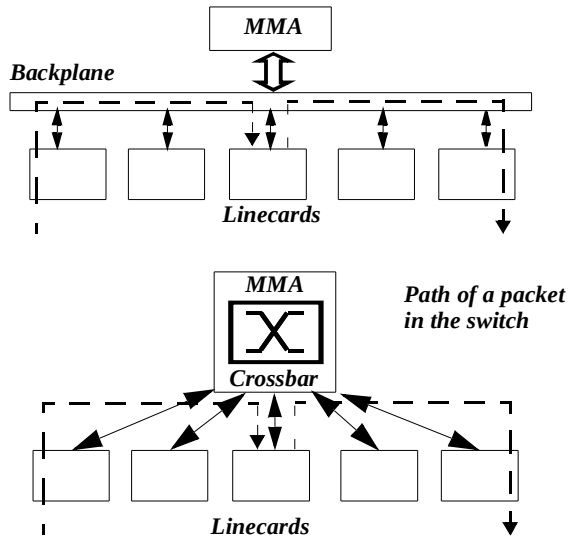
IV. Crossbar-based DSM Router

We can replace the shared broadcast bus with an $N \times N$ crossbar switch, then connect each linecard to the crossbar switch using a short point-to-point link. This is similar to the way input queued routers are built today, although in a Distributed Shared Memory router every packet traverses the crossbar switch twice, even though the packet is buffered only once.

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. Note that 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-



(a) Physical view of the Distributed Shared Memory switch. The switch fabric can be either a backplane or a crossbar. Each linecard contains a part of the shared memory. The memory on a single linecard can be shared by packets arriving from other linecards.



(b) Logical view of the Distributed Shared Memory switch. An arriving packet can be buffered in the memory of any linecard, say x. It is later read by the output port from the intermediate linecard x.

Figure 4: The Distributed Shared Memory router.

cards.

In what follows we'll see that there are several ways to schedule the crossbar switch, each with its pros and cons. We will find different algorithms; and for each, we will find the speed that the memories and crossbar need to run at.

We will define the bandwidth of a crossbar to be the speed of the connection from a linecard to the switch, and will assume that the link bandwidth is the same in both directions. So for example, just to carry every packet across the crossbar fabric twice, we know that each link needs a bandwidth of at least $2R$. We find that, in general, we need a higher bandwidth than this in order to emulate a shared memory router. The additional bandwidth serves three purposes: (1) It provides additional bandwidth to write into (read from) the memories on the linecards to overcome the memory constraints, (2) It relaxes the requirements on the scheduling algorithm that configures the crossbar, and (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. This suggests that more efficient crossbars could be built with half-duplex, rather than simplex serial links.

A. A Crossbar-based DSM router can emulate an FCFS shared memory router

We start by showing trivially sufficient conditions under which a Crossbar-based DSM router can emulate an FCFS shared memory router. We will follow this with some tighter results which show how the crossbar bandwidth can be reduced at the cost of either increased memory bandwidth, or a more complex crossbar scheduling algorithm.

Lemma 1: A Crossbar-based DSM router can emulate an FCFS shared memory router with a total memory bandwidth of $3NR$ and a crossbar speed of $6R$.

Proof: Imagine that we operate the crossbar in two phases: first, we read all departing packets from memory and transfer them across the crossbar. From Theorem 1, this requires at most three transfers per linecard per time slot. In the second phase, we write all arriving packets to memory, requiring at most three more transfers per linecard per time slot. This corresponds to running the link connecting the linecard to the crossbar at a speed of $6R$. ❐

Lemma 2: A Crossbar-based DSM router can emulate a PIFO shared memory router with a total memory bandwidth of $4NR$ and a crossbar speed of $8R$ within a relative delay of $2N-1$ time slots.

Proof: This will follow directly from Theorem 2 and the proof of Lemma 1. How we schedule the crossbar is covered in detail in the proof of Theorem 4. ❐

B. Minimizing the bandwidth of the crossbar switch

We can represent the set of memory operations in a time slot using a bipartite graph with $2N$ vertices, as shown in Figure 5(a). An edge connecting input i and output j represents a (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-

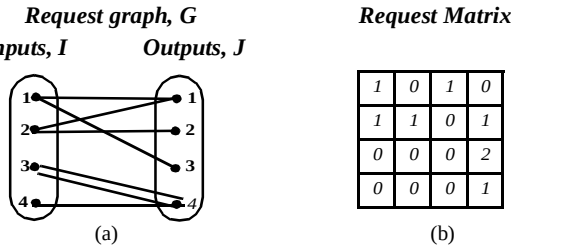


Figure 5: A request graph and a request matrix resulting from the MMA for an $N \times N$ switch.

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: (Sufficiency) A Crossbar-based DSM router can emulate an FCFS shared memory router with a total memory bandwidth of $3NR$ and a crossbar speed of $4R$.

Proof: From the above discussion, the degree of the bipartite request graph is at most 4. From previous results on edge-coloring [23] and Theorem 1, a total memory bandwidth of $3NR$ and a crossbar speed of $4R$ is sufficient. ❐

Theorem 4: (Sufficiency) A Crossbar-based DSM router with a total memory bandwidth of $4NR$ and a crossbar speed of $5R$, can emulate a PIFO shared memory 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 switch, 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 (MMA) 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' (spread over N time slots), is no more than $4N + N = 5N$. As in Theorem 3, König's method shows that there exists a schedule to switch the packets in Π', with a crossbar bandwidth of $5R$.

Now we will prove 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 exited the switch. 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}), i.e. not later than time slot a_{2N}. Hence the maximum relative delay that a cell can incur is $2N-1$ time slots. From Theorem 2, the memory bandwidth required is no more than $4NR$. ❐

C. A tradeoff between crossbar bandwidth and scheduler complexity

Theorem 3 is the lowest bound that we have found for the bandwidth of the crossbar ($4R$) and we suspect that it is a necessary condition to emulate an FCFS shared memory router. Unfortunately, edge-coloring has complexity $O(N\log\Delta)$ [24], and is too complex to implement at high speed. We now explore a more practical algorithm which also needs a crossbar bandwidth of $4R$, but requires the memory bandwidth to be increased to $4NR$.

We schedule the crossbar switch in two phases: 1) *Write-Phase:* Arriving packets are transferred across the crossbar switch to memory on a linecard, and 2) *Read-Phase:* Departing packets are transferred across the crossbar from memory to the egress linecard.

Theorem 5: (Sufficiency) A Crossbar-based DSM router can emulate an FCFS shared memory router with a total memory bandwidth of $4NR$ and a crossbar speed of $4R$.

Proof: *(Using Constraint Sets).* See Appendix B. ❐

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

Proof: *(Using Constraint Sets)* See Appendix B. ❐

In summary, we have described three different results. Let's 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 $6R$ (Lemma 1). With a more complex scheduling algorithm, we can schedule the crossbar with a memory bandwidth of $4NR$ and a crossbar bandwidth of $4R$ (Theorem 5). But our first results suggest that although possible, it is very complicated to schedule the crossbar when the memory bandwidth is $3NR$ and the crossbar bandwidth is $4R$. 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. $\sum_i \sum_j R_{ij} \le 2N$. This is also true for any subset of vertices; if I and J are subsets of indices $\{1, 2, \ldots, N\}$, then

$\sum_{i \in I} \sum_{j \in J} R_{ij} \le |I| + |J|$.

We find it convenient to *complete* the request graph by adding requests so that it has exactly $2N$ edges.

2. In a complete graph, the degree of each vertex is at least one, and the degree is bounded by four. i.e. for a complete matrix, $1 \le \sum_i R_{ij} \le 4$ and $1 \le \sum_j R_{ij} \le 4$.
3. The maximum number of edges between an input and an output is 2, i.e. $R_{ij} \le 2$. We call such a pair of edges a *double edge*.
4. Each vertex can have at most one double edge, i.e., if $R_{ij} = 2$, then $R_{ik} < 2 (k \ne j)$ and $R_{kj} < 2 (k \ne i)$.
5. 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, in a complete request matrix, if $\sum_j R_{mj} = R_{mn} = 1$, then $\sum_i R_{in} \ge 2$; if $\sum_i R_{in} = R_{mn} = 1$, then, $\sum_j R_{mj} \ge 2$. To see why this is, suppose an edge connects input i, which has degree one, and output j. This edge represents a packet arriving at i being stored in j. But j has a departure which initiates another request, thus the degree of j is greater than one. By symmetry, the same is true for an edge connecting an output of degree one.

Our goal is to exploit these properties so as to find a crossbar scheduling algorithm that can be implemented on a wave-front arbiter (WFA [11]). The WFA is widely used to find maximal size matches in a crossbar switch. It has the property of being readily pipelined and can be decomposed over multiple chips [XXX will add references to Tamir's other paper on decomposition].

Definition 1: ***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 $v_1 \ge v_2$ iff $i_1 \le i_2$, and $v_1 = v_2$ iff $i_1 = i_2$.*

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

Proof: See Appendix C. ❐

Theorem 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 [11], can find a conflict-free schedule.

Proof: See Appendix C. ❐

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

V. Routers with Parallel and Distributed Shared Memory

The DSM router architecture assumes that there is one memory device on each linecard. For line-rates up to about 10Gb/s, this seems reasonable today using a single commercially available DRAM on each linecard. For line-rates above 10Gb/s, we need multiple memories on each linecard to achieve the bandwidth we need. In other words, each linecard is now similar to the Parallel Shared Memory router in Section II. It is interesting to note that in the resulting router, each memory device can have a memory bandwidth lower than the line-rate.

Once again, we can use Constraint Sets to determine how many memory devices are needed.

Theorem 8: A set of $2h - 1$ memories of rate R/h running in parallel can emulate a memory of rate R in an FCFS DSM router.

Proof: The analysis is similar to Theorem 1. However the read and write constraints at the current time collapse into a single constraint, resulting in requiring only $2h - 1$ memories. ❐

Theorem 9: A set of $3h - 2$ memories of rate R/h running in parallel can emulate a memory of rate R in a PIFO DSM router.

Proof: Similar to Theorem 8. ❐

VI. Practical Considerations

In this section, we investigate whether or not we could actually build a DSM router that emulates a shared memory router. As always, we'll find that the architecture has limits to its scalability, and these arise for the usual reasons when the system is big, such as algorithms that take too long to complete, buses that are too wide, connectors and devices that require too many pins, or an overall system power that is impractical to cool. Many of these constraints are imposed by currently available technology, and so even if our assessment is accurate today, it might be meaningless in the future. And so wherever possible, we will make relative comments, such as "Architecture A has half the memory bandwidth of Architecture B" to allow comparisons independent of the technology.

We'll pose a series of questions about the feasibility, and try to answer each one in turn.

1. A PIFO DSM router requires a lot of memory devices. Is it feasible to build a system with so many memories?

   We can answer this question relative to a CIOQ router with a speedup of two. The CIOQ router has an aggregate memory bandwidth of $6NR$ and requires $2N$ physical memory devices (although we could use one memory per linecard with twice the bandwidth). The PIFO DSM router has a memory bandwidth of $4NR$ and requires at least N physical memories. It seems clear that we can build a PIFO DSM router with at least the same capacity as a CIOQ router. The fastest single-rack CIOQ router in development today has a capacity of approximately 1Tb/s (although the speedup is probably slightly less than two, and the scheduling algorithm is a heuristic). This suggests that considering only the number of memories and their bandwidth, it is possible to build a 1Tb/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 linecard and the switch will increase (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. While this is true, it might be possible to exploit the fact that we know that the links will always be used asymmetrically. For example, we know that the total number of transactions between a linecard and the crossbar switch is limited to five per time slot. This can be compared with four transactions per time slot for the CIOQ router. The difference is that in the CIOQ router, the link utilization is symmetric. 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 between the linecards and the crossbar switch.

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 emulate a shared memory router.* In the CIOQ router, the scheduling algorithm needs to know the departure time so as to ensure 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 when 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 each time slot depending on the number of new arrivals. All else being equal, we would prefer a distributed mechanism, as ultimately the maintenance of a global counter will limit scalability. However, the communication and processing requirements are probably smaller than for the scheduling algorithm itself (which we 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 a shared memory 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 then 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 probably much 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 distributed algorithm, 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 scheduling frames instead of cells. Essentially, this is equivalent to increasing the size of each "cell". The input linecard 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 requirements) 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 an example, consider a router with 16 OC768c linecards (i.e. a total capacity of 640Gb/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 bit-

maps, and access them every 8ns. If instead we use 48kilobyte frames, the bitmaps are reduced in size by a factor of more than 1,000 and can be stored on-chip in fast SRAM. Furthermore, the bitmap interection algorithm need run only once every $9.6\mu s$, which is readily implemented in hardware. The appropriate frame size to use will depend on the capacity of the router, the number of linecards 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 linecards are sized so as to allow TCP to perform well, and the buffers on the ingress linecard 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 linecard 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 linecards of a router can be bottlenecked at the same time. As a consequence, statistical sharing reduces the required size of the buffers for TCP to work well. While we don't yet have a definitive answer on how much buffering is saved, preliminary investigations suggest that the buffer size can be approximately halved. This not only reduces the system cost, but also reduces board area and system power. A point to note is that as a consequence of the scheduling algorithm, the buffers in the DSM router may not be equally filled. We have not evaluated this effect as yet. Also, in the event of a failure of a linecard some packets which arrived from other linecards that were stored on the failed linecard, may be lost. On the other hand packets which arrived at the failed line card (and were buffered somewhere else) will not be dropped.

Our conclusion is that a PIFO DSM router is less complex than a PIFO CIOQ router (has lower memory bandwidth, fewer memories, a simpler scheduling algorithm, but slightly higher crossbar bandwidth). However, it seems that the PIFO DSM router has two main problems: (1) The departure times of each packet must be determined centrally 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 in turn. Although we have not solved either problem, we present them in the hope that others might overcome them (or find good heuristics), and make the PIFO DSM router more practical.

On the other hand, departure times are much easier to calculate in an FCFS DSM router. In fact, any service policy in which the departure time is readily determined upon arrival is much easier to implement than a general PIFO policy.

APPENDIX A

A. *Proof of Theorem 1*

Assume that the aggregate memory bandwidth of the k memories is SNR. S is a measure of how much we must increase the memory bandwidth over and above NR. We can think of the access time T as equivalent to $\lceil k/S \rceil$ decision slots.[4] We will now find the minimum value of S needed for the switch to emulate an FCFS shared memory router. It will help if we consider all packets to be of the same size, C. This doesn't mean they are all the same size when they arrive; just that they are all segmented into C bytes before being written to memory, and reassembled into variable length packets before they depart. Segmenting packets into fixed length "cells" is common in routers, so as to simplify memory management and scheduling.

In what follows, we define two constraint sets; one set for when cells are written to memory and another for when they are read.

Definition 3: ***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 which are busy at time t due to cells being written, and therefore cannot accept a new cell. Thus, $BWS(t)$ is the set of memories which have started a new write operation in the previous $\lceil k/S \rceil - 1$ decision slots. Clearly $|BWS(t)| \leq \lceil k/S \rceil - 1$.*

Definition 4: ***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 which have started a read operation in the previous $\lceil k/S \rceil - 1$ decision slots. Clearly $|BRS(t)| \leq \lceil k/S \rceil - 1$.*

Theorem 1: (Sufficiency) A total memory bandwidth of $3NR$ is sufficient for a Parallel Shared Memory Router to emulate an ideal shared memory router.

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

1. Memory l must not be busy writing or reading a cell at time t. Hence $l \notin BWS(t)$, and $l \notin BRS(t)$.

[4] We shall denote N decision slots to comprise a time slot.

2. 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:

$$\begin{array}{l} l \notin BWS(t) \wedge \\ l \notin BRS(t) \wedge \\ l \notin BRS(DT(t, j)) \end{array} \quad (1)$$

A sufficient condition to satisfy this is:

$$\begin{array}{l} k - \\ |BWS(t)| - \\ |BRS(t)| - \\ |BRS(DT(t, j))| > 0 \end{array} \quad (2)$$

From Definitions 3 and 4, we know that Equation (2) is true if: $k - 3(\lceil k/S \rceil - 1) > 0$, i.e., $S \geq 3$. By definition, this corresponds to a total memory bandwidth of $3NR$. ❐

Remark: 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.

B. Proof of Theorem 2

Theorem 2: (Sufficiency) With a total memory bandwidth of $4NR$ *a Parallel Shared Memory router can emulate a PIFO shared memory router, within* $k - 1$ *time slots.*

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

1. Memory l must not be busy writing or reading a cell at time t. This gives two memory constraints.
2. 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.
3. 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, and thus, a total memory bandwidth of $4NR$ is sufficient for the PSM router to emulate a PIFO shared memory router. ❐

REFERENCES

[1] A. Demers, S. Keshav, and S. Shenker, "Analysis and simulation of a fair queuing algorithm," *ACM Computer Communication Review* (*SIGCOMM'89*), pp. 3-12, 1989.

[2] L. Zhang, "Virtual clock: A new traffic control algorithm for packet switching networks," *ACM Transactions on Computer Systems*, vol.9 no.2, pp.101-124, 1990.

[3] J. Bennett and H. Zhang, "WF2Q: Worst-case fair weighted fair queueing," *Proc. of IEEE INFOCOM '96*, pp. 120--128, San Francisco, CA, March 1996.

[4] M. Shreedhar and G. Varghese, "Efficient fair queueing using deficit round robin," in *Proc. ACM Sigcomm*, Sep 1995, pp. 231-242.

[5] R. R. Schaller, "Moore's law: Past, present and future," *IEEE Spectrum*, vol. 34, no. 6, June 1997, pp. 52-59.

[6] N. McKeown, V. Anantharam, J. Walrand, "Achieving 100% throughput in an input-queued switch," *Proceedings of IEEE Infocom '96*, vol. 1, pp. 296-302, March 1996

[7] S. Chuang, A. Goel, N. McKeown, B. Prabhakar, "Matching output queueing with a combined input/output-queued switch," *IEEE J.Sel. Areas in Communications*, Vol. 17, no. 6, pp. 1030-1039, June 1999.

[8] A. Charny, P. Krishna, N. S. Patel, R. Simcoe, "Algorithms for providing bandwidth and delay guarantees in input-buffered crossbars with speedup", *6th International Workshop on Quality of Service (IWQoS 98)*, Napa, CA, May 1998, pp.235-244.

[9] A. Hung, G. Kesidis and N. Mckeown, "ATM input-buffered switches with guaranteed-rate property," *Proc. IEEE ISCC'98*, Athens, pp. 331-335.

[10] N. McKeown, "iSLIP: A Scheduling algorithm for input-queued switches," *IEEE Transactions on Networking*, vol 7, No. 2, April 1999.

[11] Y. Tamir and H. C. Chi, "Symmetric crossbar arbiters for VLSI communication switches", *IEEE Transactions on Parallel and Distributed Systems*, vol. 4, No. 1, pp. 13-27, Jan. 1993.

[12] I. Keslassy, N. McKeown, "Analysis of scheduling algorithms that provide 100% throughput in input-queued switches," *Proceedings of the 39th Annual Allerton Conference on Communication, Control and Computer*, October 2001.

[13] E. Leonardi, M. Mellia, M. Marsan, and F. Neri, "Stability of maximal size matching in input-queued cell switches," *Proceedings of the International Conference on Communications*, June 2000.

[14] T. Chaney, J. A. Fingerhut, M. Flucke, J. Turner, "Design of a gigabit ATM switching system," Technical Report WUCS-96-07, Computer Science Department, Washington University, St. Louis, Missouri, Feb. 1996.

[15] S. Iyer, A. Awadallah, N. McKeown, "Analysis of a packet switch with memories running slower than the line rate," in *Proc. IEEE Infocom '00*, pp.529-537.

[16] S. Iyer, N. McKeown, "Making parallel packet switches practical," in *Proc. IEEE INFOCOM '01*, vol.3, pp. 1680-1687.

[17] C.S. Chang, D.S. Lee, Y.S. Jou, "Load balanced Birkhoff-von Neumann switches, part I: one-stage buffering," *IEEE HPSR Conference*, Dallas, May 2001 [www.ee.nthu.edu.tw/~cschang/PartI.ps].

[18] I. Keslassy, N. McKeown, "Maintaining packet order in two-stage switches," *Proceedings of the IEEE Infocom*, June 2002

[19] J.G. Dai, B. Prabhakar, "The throughput of data switches with and without speedup," *Proceedings of the IEEE Infocom*, Tel Aviv, Israel, 2000.

[20] C. Clos, "A study of non-blocking switching networks," *The Bell System Technical Journal*, vol.32, pp. 406-424, 1953.

[21] P. S. Sindhu, R. K. Anand, D. C. Ferguson, B. O. Liencres, "High speed switching device," United States Patent No. 5905725, May 1999.

[22] A. K. Parekh and R. G. Gallager, "A generalized processor sharing approach to flow control in integrated services networks: The single node case," *IEEE/ACM Transaction on Networking*, Vol. 1, No. 3, pp. 344-357, June 1993.

[23] D. König, "Über Graphen und ihre Anwendung auf Determinantentheorie und Mengenlehre," *Math. Ann.*, 77 (1916), pp. 453-465 (in German).

[24] R. Cole, K. Ost and S. Schirra, "Edge-coloring bipartite multigraphs in $O(E \log D)$ time," *Combinatorica* Vol. 21, Issue 1, pp. 5-12, 2001.

APPENDIX B

A. Proof of Theorem 5

As before, we shall define the constraint sets as shown below and apply the pigeon-hole principle to derive the minimum memory bandwidth required for emulating an FCFS shared

memory switch.

Definition 5: ***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 $|BVWS(t)| \le \lfloor (N-1)/S_w \rfloor$.*

Definition 6: ***Busy Vertex Read Set (BVRS)*** *- Similarly, BVRS(t) is the set of ports busy at t due to cells being read, and therefore cannot accept a new cell. Since, for a given output no more than $N-1$ other departures occur during that time slot, clearly $|BVRS(t)| \le \lfloor (N-1)/S_R \rfloor$.*

CONSTRAINT SETS FOR A CROSSBAR-BASED DISTRIBUTED SHARED MEMORY ROUTER

Theorem 5: (Sufficiency) A Crossbar-based DSM router can emulate an FCFS shared memory router with a total memory bandwidth of $4NR$ and a crossbar speed of $4R$.

Proof: *(Using Constraint Sets).* Consider a cell c that arrives to the Crossbar-based Distributed Shared Memory switch at time slot t and destined for output port j, with departure time, $DT(t,j)$. We can now apply the constraint set method. Our choice of an 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 port x must meet the following constraints:

$$x \notin BVWS(t) \wedge \quad x \notin BVRS(DT(t,j)) \tag{3}$$

This is true if $S_R, S_W \ge 2$. Hence, we need a crossbar speed of $S_C R = (S_R + S_W)R = 4R$. Similarly, the only operations which occur at the memory of any port in a given time slot can be two reads and two writes. This corresponds to a total memory bandwidth of $4NR$. ❐

B. Proof for Theorem 6

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

Proof: *(Using Constraint Sets).* Similar to Theorem 5, we consider a cell c arriving at time slot t, destined for output port j and with departure time, $DT(t,j)$. We note that this departure time is based on the conflict-free permutation departure order Π'. We now apply the Constraint Set method. Our choice of an 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.
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 immediately in front of it.
4. Similarly, 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. All these constraints can be satisfied if $S_R, S_W \ge 3$. Hence, we need a crossbar speed of $S_C R = (S_R + S_W)R = 6R$. Similarly, the only operations which occur at the memory of any port in a given time slot can be three reads and three writes. This corresponds to a total memory bandwidth of $6NR$.

Note that $S_R = 2, S_W = 4$ will also satisfy Theorem 6. ❐

APPENDIX C

Lemma 3: A request matrix can be ordered in no more than $2N-1$ alternating row and column permutations.

Proof: We shall perform the ordering in a iterative way. The first iteration consists of one ordering permutation of rows or columns, and the subsequent iterations consist of two permutations, one of rows and one of columns. We will prove the theorem by induction.

1. After the first permutation, either by row or by column, the entry at $(1,1)$ is non-zero, thus this entry will never be moved to another position in the future. We can define submatrices of S as follows:

$$A_n = \{S_{ij} | 1 \le i, j \le n\}$$
$$B_n = \{S_{ij} | (1 \le i \le n, n < j \le N)\}$$
$$C_n = \{S_{ij} | n < i \le N, 1 \le j \le n\}$$

2. If a sub-matrix of S is ordered and will not change in future permutations, we call it *optimal*. Suppose A_n is optimal after the nth iteration. We want to prove that after another iteration, the sub-matrix A_{n+1} becomes optimal. Without loss of generality, suppose a row permutation was last performed, then in this iteration, we'll do a column permutation followed by a row permutation. There are four cases:

a. The entries of B_n and C_n are all zeros. Then $S_{n+1,\,n+1} > 0$ after just one permutation, so the sub-matrix A_{n+1} is optimal.

b. The entries of B_n are all zeros but those of C_n are not. After the column permutation, suppose $S_{m,\,n+1} (m > n)$ is the first positive entry in column $n+1$, then the first m rows of S are ordered and will remain so. Thus, column $n+1$ will remain the biggest column in B_n, and A_{n+1} is optimal.

c. The entries of C_n are all zeros but those of B_n are not. This case is similar to case b.

d. The sub-matrices B_n and C_n both have positive entries. The column permutation will not change row $n+1$ such that it becomes smaller than the rows below it. Similarly, the row permutation following will not change column $n+1$ such that it becomes smaller than the columns on its right. So A_{n+1} is optimal.

This means, after N iterations, or a total of $2N-1$ permutations, the request matrix becomes ordered. In most cases, the matrix becomes ordered after fewer than $2N-1$ permutations. ❐

Theorem 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 [11], can find a conflict-free schedule.

Proof: By contradiction. Suppose the scheduling algorithm cannot find a conflict free time slot for request (m,n). This means

$$\sum_{j=1}^{n-1} S_{mj} + \sum_{i=1}^{m-1} S_{in} \geq 4 .$$

Now consider the sub-matrix S', consisting of the first m rows and the first n columns of S. Let's look at the set of the first non-zero entries of each row, L_r, and the set of the first non-zero entries of each column, L_c. Without loss of generality, suppose S'_{11} is the only entry belonging to both sets. (If this is not true, and S'_{kl}, where $k \neq 1$ or $l \neq 1$, also belongs to both L_r and L_c, then we can remove the first $k-1$ rows and the first $l-1$ columns of S' to obtain a new matrix. Repeat until L_r and L_c only have one common entry.) Then $\left| L_r \bigcup L_c \right| = m + n - 1$. At most two of the entries in the mth row and those in the nth column are in $L_r \bigcup L_c$, so the sum of all the entries satisfies

$$\sum_i \sum_j S_{ij} \geq (\left| L_r \bigcup L_c \right| - 2) + S_{mn} + \left(\sum_{j=1}^{n-1} S_{mj} + \sum_{i=1}^{m-1} S_{in} \right)$$

$$\geq m + n + 2$$

which conflicts with property 1. ❐