A Near-Optimal Scheduler for Switch-Memory-Switch Routers

Adnan Aziz **Amit Prakash** **Vijaya Ramachandran**
Electrical and Computer Engineering Computer Sciences
The University of Texas at Austin

TR-03-32

August 6, 2003

Abstract

We present a simple and near optimal randomized parallel scheduling algorithm for scheduling packets in routers based on the *Switch-Memory-Switch (SMS)* architecture, which emulates output queuing by using a collection of small memories within the switch to buffer packets, and which forms the basis of the fastest routers in use today. For a router with N inputs and N outputs, our algorithm computes the schedule in $O(\log^* N)$ rounds, where a round is a communication of a few bits between input ports and memory together with simple local computation at the inputs and memory. Furthermore, by using an $O(\log^* N)$ deep pipeline at each input, our algorithm computes the schedule in a constant number of rounds. Our pipelined algorithm is quite simple and achieves optimal (i.e., constant) throughput with a tiny $O(\log^* N)$ delay.

We show that the total amount of buffer memory required by our algorithm is close to the minimum required. We also show that the number of buffer memories is within an ϵN additive term of $2N-1$, for any positive constant $\epsilon > 0$ (and is within an additive term of $o(N)$ for the basic scheduler), where $2N-1$ is the minimum number of memories needed under adversarial placement of packets. Furthermore we show that the number of extra memories that we use over the minimum of N that is required in the offline version, is within a constant factor of the minimum required by any on-line scheduler, even if that scheduler is allowed to fail occasionally.

Our scheduling algorithm is randomized and works with high probability in N. We also prove that it is self-stabilizing, i.e., it resumes its normal behavior if occasional lapses occur due to the probabilistic nature of the algorithm.

A preliminary version of these results appeared in the Proceedings of the 15th ACM Symposium on Parallelism in Algorithms and Architectures, June 2003. The main new contribution in this Technical Report is an improved pipelined scheduler that requires at most one message from each memory bank to an input in each communication step. A companion manuscript presents simulation results that show that the constant factors in our algorithms are quite small, indicating that our algorithms are likely to be quite practical.

1 Introduction

Routers play a critical role in modern computing of all forms including wide-area networks, multiprocessor servers, and data storage systems [16, 25, 11, 4, 9, 12, 30] (see also [13, Chapters 7.12, 8.12]). Modern routers achieve high performance by solving computationally intensive tasks using custom hardware. One of the most challenging problems in designing a high-end router is scheduling the transfer of packets from inputs to outputs.

A router used to be nothing more than a general purpose computer connected via a standard bus to hardware for transmitting and receiving packets over links. This was because the link bandwidth was low enough for a general purpose processor to implement the entire router functionality. With the advent of high-speed fiber optic technology [27, 28], the situation has reversed, and in many networks today routers are the bottleneck in moving data.

Given that the cost of deploying and maintaining links far exceeds the cost of router hardware [16, Page 203] the trend has been to use quite extensive hardware in the router. Some of the tasks performed by routers can be accelerated using brute-force solutions, e.g., by demultiplexing high-speed links and using replicated hardware. However the task of quickly transferring packets from inputs to outputs has not been solved satisfactorily so far, largely because of the complex co-ordination problem that is associated with it.

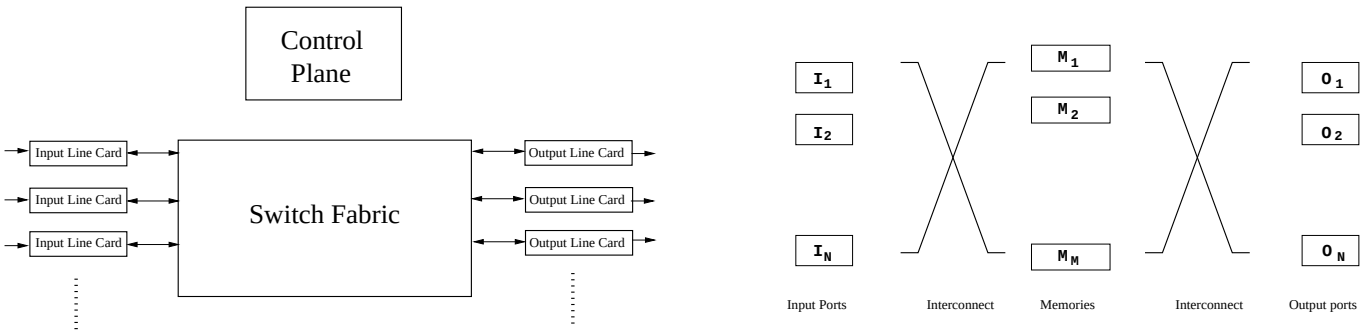
Figure 1(a) shows the block-level architecture of a router. Packets are assumed to be of a fixed size. (IP network packets can be variable sized; this is dealt with by segmenting them into fixed size packets at the input port, and reassembling them at the output port [25, Page 203].) *Input line cards* (or *input ports*) take packets from incoming links, and compute the output link to which the packet is to be forwarded. (It is assumed that the output link is determined by the final destination of the packet, and is not within the control of the scheduler.) The *switch fabric* transfers packets to the *output ports*, which transmit the packets on outgoing links. Peterson and Davie [25, Chapter 3] and Keshav and Sharma [17] survey router architectures.

Logically, the router operates in *cycles*: in each cycle, at most one packet may arrive at an input port. The *cycle time* is defined to be the amount of time between cycles; ideally it is equal to the link bandwidth divided by the packet size, unless the router requires large cycle time to be able to perform all the tasks that it needs to do on every packet, which is currently the case.

We restrict our attention to routers that have N input ports and N output ports, with all links having the same bandwidth. At the beginning of every cycle, the router receives at most one packet at each input and transmits at most one packet on each output. The *arrival time* of a packet p is the cycle in which p arrived at the input of the router; the *departure time* of p is the cycle in which p is transmitted from the output. The difference between departure and arrival times of a packet is called its *latency*.

Two (or more) packets destined for the same output port can arrive at different input ports in the same cycle. Consequently, one of the two packets will have to be *buffered* [25, 16, 14]. This buffering can be performed at the input ports, within the switch fabric, and at the output ports. Because of contention for a shared output link, a link may become congested; when the number of packets waiting for the link exceeds the buffer capacity, packets will be dropped [16, Chapter 8.5].

At any given time, a router may have a large number of packets, enqueued in different queues, waiting to be transmitted through different outputs. In a single cycle only a subset of these queues can be advanced based on the constraints imposed by the architecture of the router. Routers need to make scheduling decisions about which queues get advanced in each cycle. The average latency that packets observe at the router as well as the number of packets that get dropped by the router because of buffer overflow greatly depend upon the scheduling decisions made by the router. Thus it is essential to have an efficient scheduler. In a router with a large number of input and output



(a.) A generic router

(b.) The SMS architecture

Figure 1: *(a.)* Architecture of a generic router. *(b.)* The Switch-Memory-Switch (SMS) architecture.

ports, the scheduling algorithm often takes more time to compute the schedule than the router takes to transmit the packets. This paper introduces a fast scheduling algorithm; we are motivated by the fact that the schedule must be computed within the cycle time.

A router is said to be *output-queued* if packets are buffered solely at the outputs. Output queuing is strongly preferred for a number of reasons [24]. For example, it minimizes the average queuing delay faced by packets. It also guarantees that the relative ordering of packets is preserved. However, buffering packets solely at the output ports requires very high-speed memories and switch fabrics. Specifically, in an N input router, N packets for the same output can arrive in a cycle; consequently, the memory at the output port should be able to support N writes in a single cycle.

In an *input-queued* router, packets are buffered solely at the inputs. The advantage of an input-queued architecture is that the buffer memory need only to be able to support one read and one write in a cycle. However, it is extremely difficult to schedule packets for departure across the switch fabric in such an architecture — naive approaches result in high drop rates [14], and more sophisticated approaches are too complex to run within the cycle time [21].

The *switch-memory-switch (SMS)* architecture buffers packets in small memories placed between the input and output ports. In this architecture, the output ports have buffers that need to hold just one packet, and the input ports have buffers of small size. Thus the main buffers in this architecture are the small memories placed between inputs and outputs, which operate together. This is the architecture used by the fastest routers available today, the M160 and T640 Internet core routers from Juniper Networks [23]. (The power of this architecture can be seen in the fact that within three years of its inception Juniper Networks took over from Cisco as the leading provider of routers for the Internet core.)

There are three main advantages to using an SMS architecture over other architectures:
(1) The average delay can be minimized (as in output queuing),
(2) The buffer memories need to support only one read and one write per cycle (as in input queuing),
(3) With a good scheduling algorithm, the packets can be distributed almost equally among the buffer memories to make sure that a packet gets dropped only if all the buffers are full (thus the same packet drop rate can be achieved with smaller memories as compared to an output-queued or input-queued switch).

In this paper we present a near optimal scheduler for the SMS architecture. The scheduler is described in Section 4 and its memory requirements, which are also close to optimal, are analyzed in Section 5. In recent simulation results [3] we show that the constant factors in our algorithms

are small, indicating that our algorithms are likely to be quite practical.

1.1 Prior Work on Router Scheduling

Early routers used sequential algorithms; however, this is not an option with modern link speeds. Broadly speaking, recent parallel algorithms for scheduling have one or both of the following shortcomings: 1.) they are *ad hoc*, working well on some cases and very badly in others [20, 5, 6], or 2.) they involve pointer-manipulating algorithms that are unacceptably complicated even in the context of a large budget for dedicated hardware [26]. McKeown *et al.* [20] describe a heuristic parallel algorithm for scheduling in input-queued switches. However, its performance depends greatly on the incoming traffic, and there are natural traffic patterns for which it has an unacceptably high drop probability [6]. Prakash *et al.* [26] proposed an $O(\log^2 N)$ parallel algorithm based on pointer jumping for scheduling packets in the SMS architecture; as in [7], this router emulates an output-queued router. However, the algorithm is impractical to implement, since it uses the NC algorithm in [18] to edge-color bipartite graphs.

Subsequent to the work in [26], Iyer *et al.* [15] showed that an SMS router[1] with $3N$ packet memories running at the line rate could emulate an N input, N output output-queued switch—this was a lemma in [26], and the proof method was identical, namely applying the pigeonhole principle in a straightforward manner. Iyer *et al.* did not consider the implementation of the scheduler, and the scheduler arising from their proof has time complexity $\Omega(N)$.

Chuang *et al.* [7] have shown that a router with buffering at both the input and output ports can emulate an output-queued router by performing 2 reads and 2 writes on the input and output buffers, respectively, and running the switch fabric twice in a cycle. Their approach hinges on a sophisticated scheduling algorithm which solves an instance of the stable marriage problem, which is again impractical to implement in hardware.

2 The SMS architecture

Since we use the switch-memory-switch (SMS) architecture presented in [26], we review the architecture and key results in that paper. We defer a discussion of the details of the model of computation to Section 3.

Figure 1(b) depicts the SMS architecture. The set of input ports is connected via an $N \times M$ interconnect to M memories; these M memories are connected to the set of output ports through another interconnect. Each of these memories are of size K; we assume $K \geq N$ (in practice, $K \gg N$). In every cycle one packet can be read from and s packets can be written to each memory. Not surprisingly, we will show that as s increases the requirement on M goes down. Thus if memory bandwidth is the bottleneck in the system then it would be desirable to use $s = 1$ but otherwise one can boost s as much as possible to reduce M. One can also consider the case of using memory banks that supports s reads and s writes every cycle. But that would be equivalent to using $s \cdot M$ memory banks that support one read and one write every cycle in our scheme. Since this case is already captured in the analysis we do not consider it as a separate case.

2.1 Emulating output queuing

Since output-queuing is highly desirable (cf. Section 1), our goal is to emulate the behavior of an $N \times N$ output-queued switch that has buffer memory space for L packets at each output using an

[1] They refer to their architecture as a Parallel Shared Memory Router, but it is isomorphic to SMS.

SMS architecture. By emulation, we mean that for any arrival sequence **(1)** a packet is dropped by the SMS router iff it will be dropped by the output-queued router, and **(2)** if a packet is not dropped then the cycle in which it departs the SMS router must be same as the cycle in which it would have departed the output-queued router.

The cycle in which a packet would have departed an output-queued router is referred to as its *time-stamp*. When a packet arrives at an input of an SMS router, its time-stamp is computed as described in section 2.4. In each cycle, packets at the inputs are written to a subset of memories through the first interconnect, and packets whose time-stamp is equal to the current time are read from the memories and transferred to the outputs through the second interconnect.

2.2 Conflicts

In the SMS architecture each memory can support one read and s writes per cycle. Hence packets cannot be arbitrarily placed in the memories. A packet faces two kinds of conflicts. More than s packets that arrive at the same time cannot be written to the same memory; this is referred to as an *arrival conflict*. Since there are N input ports, the maximum number of arrival conflicts a packet can have is $\lceil (N-1)/s \rceil$. *Departure conflicts* occur if multiple packets in the same memory need to depart simultaneously through different outputs. Since there are N outputs, a packet can have departure conflicts with at most $N-1$ memories. Hence if the number of memories $M \geq \lceil (N-1)/s \rceil + N$ there will always be a conflict-free memory for each packet. A conflict-free memory for an input is said to be *compatible* with that input.

2.3 Scheduler tasks

In order to construct a conflict-free schedule for transfer of packets the scheduler has three tasks to perform in every cycle.

Task 1 Compute the time-stamp of all the newly arrived packets.

Task 2 Match the newly arrived packets to memories such that there are no departure and arrival conflicts.

Task 3 Read packets whose time-stamp is equal to the current time and transfer them to the output.

Since the time-stamp of a packet is known when it is written to a memory, Task 3 is simple. We briefly describe how Tasks 1 and 2 are performed. Task 2 is the most complex step and is the focus of this paper.

2.4 Task 1: Time-stamp computation

An array $E[1 \ldots N]$ stores the earliest available time-slot for each output.

Let P_1^o through $P_{c^o}^o$ be the packets destined for output port o that arrived in the cycle T and let them be ordered according to the id of the input port they arrived. Then time-stamp of packet P_i^o is set to $(E[o]+i)$ and $E[o]$ is set to $\max((E[o]+c^o, T)$. This time-stamp assignment is consistent with the requirement of emulating an output-queued router, and can be efficiently computed by simple circuitry.

If the difference of time-stamp of a packet and current time is greater than L then it is dropped. This behavior is consistent with the behavior of an output-queued router with buffer of size L at each output.

2.5 Task 2: Scheduling using graph matching

For routers that are relatively small and slow, the SMS architecture can emulate output-queuing by using a straightforward greedy sequential algorithm to compute an assignment of incoming packets to compatible memories. However for routers with many ports operating at high speeds, the sequential algorithm is not fast enough to compute the assignment. The only known parallel algorithm for computing the assignment is that of Prakash *et al.* [26]; however, it has the disadvantages mentioned in Section 1.1.

3 Computational Model

In Section 4 we describe simple and fast algorithms for Task 2. In this section we describe the main features of the abstract model of the interface between the input ports and the memory banks in the SMS architecture.

- There are N input ports, each with a buffer that can hold I packets. At each input port, the *current packet* is the packet at the head of that input buffer. In our basic algorithm I is a constant; in the pipelined version $I = O(\log^* N)$. There are N output ports, which need to buffer only one packet each.
- There are $M \geq N$ memory banks, and each can hold up to K packets. Our schedulers work for $M = (1+(1/s)+\epsilon)N$, where ϵ is either an arbitrarily small constant or is $o(1)$ as described later.
- There is simple hardware at the input ports as described in [26] (and summarized in Section 2.4 of this paper) that computes the departure time stamp for each current packet at the start of each cycle, based on the packet's output port.
- Each input port and memory bank has $O(\log N)$ depth circuitry of size $\tilde{O}(N)$. Note that as routers become larger, distributing the hardware for computing the schedule across the input ports and the memory banks is preferable to having a separate centralized processing unit.
- There is a dedicated wire connecting every (input port, memory bank) pair. This investment in hardware is not considered excessive if the wire needs to support transfer of only a few bits per cycle (see, e.g., [2, page 6], [22]). With this hardware support, each input port can send a short message to each memory bank (and vice versa) in one communication step. At the receiving end the identity of the transmitting node can be determined by examining the wire along which the message arrives. We will refer to such a communication step as a *transmit step*.

Under current technology, the time taken by a transmit step dominates the cost of $O(\log N)$ time computation in hardware at a single input port or memory bank. However, it is considerably faster than the time taken to transfer a packet through the crossbar, since a packet is typically hundreds of bits long.

4 The Scheduling Algorithms

In section 4.1 we describe a basic randomized scheduling strategy for matching input packets to compatible memory banks. We measure performance in terms of *rounds*, where a round is a transmit

step together with $O(\log N)$ time computation at each input port and each memory bank. Our basic scheduler runs in $O(\log^* N)$ rounds.

In section 4.2 we present a pipelined version of our basic scheduler with a latency of $O(\log^* N)$ rounds, but with the improved performance of constant throughput. Thus in this scheme the lag between successive transfers of of packets from input ports to memory banks is a constant number of rounds. Since in many networks, the limiting feature for the cycle time is the router and not the link speed, this will have the desirable effect of reducing the cycle time, thus improving the bandwidth.

4.1 The Basic Matching Algorithm

In this discussion each input is identified with the packet that just arrived at that input. Recall that an input i is *compatible* with a memory m if the packet that just arrived at i can be stored in memory m without *arrival* and *departure conflicts* (see Section 2.2).

Anderson *et al.* [2] proposed an algorithm somewhat similar to ours which they called "Parallel Iterative Matching (PIM)." PIM was developed for a completely different architecture, namely a crossbar-based input-queued router with "virtual output queues." In their case they need to compute a maximal matching in an arbitrary bipartite graph, and they prove that the expected number of rounds for their algorithm is $O(\log N)$.

At the beginning of a cycle, the time-stamp of each input port is broadcast to each memory and memories construct a list of inputs that are compatible with the memory. The algorithm then works in rounds according to the 'Basic Matching Process' given below. Initially all the memory banks are unmatched.

Basic Matching Process:

1. In parallel each unmatched memory sends a message to a random compatible input port.
2. In parallel each input port i picks a memory bank j that sent it a message and assigns its current packet to that memory bank. It then broadcasts a bit to all memory banks to inform them that it is no longer available to be matched (the bit sent to memory bank j is a 1 and the bit sent to all other processors is 0).
3. In parallel each memory bank that receives a 1-bit from its matched input decrements a counter initially set to s. If the counter goes down to zero, the processor declares itself matched.

4.1.1 Analysis of the Basic Matching Algorithm

In this section we establish that if $M = (N + \lceil N/s \rceil + \epsilon N)$, for any $\epsilon > 1/2^{\log^* N}$, then w.h.p. in N, the number of rounds needed to match every input to a compatible memory bank is $O(\log^* N)$. The analysis views the computation in the 'balls-in-bins' framework, and the slight excess in the number of available memory banks over the bound of $(N + \lceil (N-1)/s \rceil)$ given in section 2.2 allows for the acceleration in the matching process in successive rounds leading to the $O(\log^* N)$ bound. Randomized strategies with $O(\log^* N)$ complexity are known in the literature for other scenarios, e.g., in the context of highly-parallel algorithms for the CRCW PRAM [19] and in emulating shared-memory on distributed memory (see, e.g., [8]), and our strategy is similar to these in terms of accelerating progress in successive rounds. However, our framework and analysis are different. Our main theorem is proved through a sequence of lemmas.

Lemma 4.1 *If there are k unmatched inputs at a beginning of a round then there must be $(\epsilon N + \lceil k/s \rceil)$ unmatched compatible memory banks for each input.*

Proof: A memory bank could be unavailable for a given input because of two reasons, either because there is already a packet in that memory (either stored in previous cycles or matched to that memory for current cycle) that has the same time-stamp or because s other inputs have been already matched to that memory. There could be at most $N-1$ packets with same time stamp, that could eliminate $N-1$ memories as potential match. Since $N-k$ inputs have been matched to memories, there could be at most $\lfloor (N-k)/s \rfloor$ memories that have been matched to s inputs. This could further eliminate at most $\lfloor (N-k)/s \rfloor$ memories as a potential match. Thus we will have at least $M-(N-1)-\lfloor (N-k)/s \rfloor > \epsilon N + \lceil k/s \rceil$ memories that are compatible with a given input. ■

Define a round that starts with k unmatched inputs to be *successful* if it ends with at most $ke^{-(1/s+\epsilon N/k)} + \sqrt{2M \log M}$ unmatched inputs. In the following lemma we prove that w.h.p. a round is successful.

Lemma 4.2 *If there are k unmatched inputs and M memories at the beginning of a round and each input can be matched to at least $\epsilon N + \lceil k/s \rceil$ memories, then the expected number of unmatched inputs at the end of that round is at most $ke^{-(1/s+\epsilon N/k)}$. Furthermore the probability that the number of unmatched inputs exceeds its mean by more than $\sqrt{2M \log M}$ is at most $\frac{1}{M}$.*

Proof: First we bound the expectation. Let $\nu(m)$ be the set of unmatched inputs that can be matched to memory m and let $\eta(i)$ be the set of unmatched memories that can be matched to input i. Clearly $|\nu(m)| \leq k$ and $|\eta(i)| \geq \epsilon N + k/s$.

Let C_m be the index of the input to which memory i sends a request. Thus $\Pr[C_m = j] = 1/|\nu(m)|$ if $j \in \nu(m)$ and 0 otherwise. Let $\mathbf{C} = (C_1, C_2 \ldots, C_M)$ and define the random variable $X_i(\mathbf{C})$ to be 1 if $\forall j.\,(C_j \neq i)$ and 0 otherwise. Informally $X_i(\mathbf{C})$ indicates that input i did not get a request from any of the memories. Since an input is matched if and only if it gets a request from at least one of the memories, $X_i(\mathbf{C}) = 1$ implies input i did not get a match in that round. Let $X(\mathbf{C}) = \sum_i X_i(\mathbf{C})$ be the total number of unmatched inputs at the end of the round. Then,

$$\begin{aligned} \mathrm{E}(X(\mathbf{C})) &= k(1-1/k)^{(\epsilon N + \lceil k/s \rceil)} \\ &\leq ke^{-(1/s+\epsilon N/k)}. \end{aligned}$$

We now use Azuma's inequality [1] to bound the probability of deviation. Let us define a sequence of random variables Y_0 through Y_M as follows

$$Y_m(\mathbf{C}) = \mathrm{E}(X(\mathbf{C}) | C_1, C_2, \ldots, C_{m-1}).$$

In particular, $Y_0(\mathbf{C})$ is equal to the constant $\mathrm{E}(X(\mathbf{C}))$ and $Y_M(\mathbf{C})$ is identical to $X(\mathbf{C})$. Since $\mathrm{E}(Y_m | Y_{m-1}) = Y_{m-1}$ the sequence of random variables Y_m is a martingale. Furthermore if $\mathbf{C}$ and $\mathbf{C}'$ differ in choice of only one memory then that memory could choose at most one input that was not chosen by any other memory. Thus the difference in number of unmatched inputs can be at most one. Hence by Azuma's inequality we have $\Pr\left[X(\mathbf{C}) > \mathrm{E}(X(\mathbf{C})) + \sqrt{2M \log M}\right] < \frac{1}{M}$. ■

Since $1/M \leq 1/N$, the first $O(\log^* N)$ rounds are successful w.h.p. in N. The following discussion assumes that they are successful.

Let k_r be the number of unmatched inputs at the beginning of round r. We know that $k_0 = N$ and k_r decreases in successive rounds. Let R be the last round for which $k_R \geq W\sqrt{2M \log M}$,

where W is a constant chosen to ensure that $k_{r+1} \leq (k_r/\alpha)e^{-\frac{\epsilon N}{k_r}}$ for $r < R$, where $1 < \alpha < e^{1/s}$. We will prove that $R = O(\log^* N)$. (Note that by Lemma 4.1 and a Chernoff bound, w.h.p. in N all inputs are matched in round $R+1$.)

For $a > 1$ and integer $i \geq 0$ we define $a \uparrow\uparrow i = g(a,i)$, where $g(a,0) = 1$ and $g(a,i) = a^{g(a,i-1)}$ for $i > 0$.

Lemma 4.3 *For every constant $c > 0$ there exists a constant $b = e^{\epsilon/c}$ such that if there are k unmatched packets at the beginning of a round $r < R$ and for some positive integer i we have $k \leq \frac{cN}{b\uparrow\uparrow i}$ then the number of unmatched inputs at the end of that round is at most $\frac{k}{\alpha(b\uparrow\uparrow(i+1))}$, w.h.p. in N.*

Proof: The number of unmatched inputs at the end of round is at most $\frac{k}{\alpha e^{\epsilon N/k}} \leq \frac{k}{\alpha e^{\epsilon(b\uparrow\uparrow i)/c}} = \frac{k}{\alpha b^{(b\uparrow\uparrow i)}}$. ■

From Lemma 4.3 it trivially follows that $k_{r+1} \leq k_r/\alpha$. Let $A = \lceil \log_\alpha \frac{\ln 2}{\epsilon} \rceil$. Hence after A initial rounds we have $k_A \leq N\epsilon/\ln 2$. Now substituting $c = \epsilon/\ln 2$ in Lemma 4.3 we have $b = 2$, and hence $k_r \leq \frac{N\epsilon}{(\ln 2)(2\uparrow\uparrow i)}$ implies $k_{r+1} \leq \frac{k_r}{\alpha(2\uparrow\uparrow(i+1))} \leq \frac{N}{(2\uparrow\uparrow(i+1))}$.

Since $k_A \leq N\epsilon/\ln 2$, applying the above inequality repeatedly we obtain $k_{r+A} \leq \frac{N}{\alpha(2\uparrow\uparrow r)}$. Thus at the end of $A + \log^* N$ rounds we cannot have more than $W\sqrt{2M\log M}$ unmatched inputs. Since $W\sqrt{2M\log M}$ inputs can be matched in a single round w.h.p. in N, we can match all the inputs in $A + \log^* N + 1 = O(\log^* N)$ rounds, if $\epsilon = \Omega(1/2^{(1/s)\log^* N})$. This gives us the following theorem.

Theorem 4.4 *If the router can transfer s packets to each memory in a cycle, then if $M = N + \lceil N/s \rceil + \Omega(\frac{N}{2^{(1/s)\log^* N}})$, repeated applications of the basic matching process will match all inputs to memories in $O(\log^* N)$ rounds with high probability in N.*

4.2 Pipelined Randomized Scheduler

The scheduling algorithm described in the previous section uses $O(\log^* N)$ rounds of the *basic matching procedure*. Thus the cycle time must be sufficiently long to be able to complete these $O(\log^* N)$ rounds, and as N increases the cycle time must increase resulting in a drop of throughput. In this section we address this drawback by presenting a pipelined scheduler that executes each cycle in a constant number of rounds.

The pipelined scheduler uses multiple cycles to construct a matching for each set of packets that arrive together. However matchings are constructed for multiple sets of packets simultaneously in a pipelined fashion. Consequently, the amount of computation per cycle reduces but packets wait for D cycles at the inputs before they are transferred to the memories. The value D is the *latency* of the pipelined scheduler (we will show later that $D = O(\log^* N)$). The input buffer size I equals D, and packets are stored FIFO.

Let P_1^o through $P_{c^o}^o$ be the packets destined for output port o that arrived during cycle T and let them be ordered according to the id of the input port they arrived. We maintain an array $earliest[1 \cdots N]$ to keep track of earliest time-stamp available for any output, after taking latency into account. The time-stamp of packet P_i^o is then set to $earliest[o] + i + D$ and $earliest[o]$ is updated to $\max(earliest[o] + c^o, T)$.

In cycle T the packets that arrived between cycles $T - D$ and T are in the input buffers and at the end of cycle T the packets that arrived at cycle $T - D$ that are matched are transferred to the memories. Each input port will have an initial sequence of packets in its buffer that have been matched to some memory by the scheduling algorithm in earlier iterations, and the remaining packets are not yet matched by the scheduling algorithm. At any point in the scheduling algorithm,

the first unmatched packet in each buffer is the *active* packet for the step, and the basic matching process will be applied to the set of active packets.

Let the current cycle be T. A *stage* of the pipeline executes the three steps in the following pipelined matching procedure ω times, where ω is an integer constant to be defined later in the analysis.

Pipelined Matching Procedure

(a) The input ports perform a transmit step in which each input port broadcasts to all the memories the time-stamp of its active packet (as in the first scheduling algorithm) together with its arrival time mod D.

(b) In parallel, each memory bank picks an index i between 0 and D, and matches itself to a random compatible input with exactly i unmatched inputs. The index i is chosen with probability p_i, where $p_i = 1/2^{i+1}$ if $i < D$ and $p_D = 1/2^D$.

(c) Each matched active packet is replaced by the first unmatched packet in its buffer.

Finally, all matched packets that arrived in cycle $T - D$ are transferred to the memory banks, and this concludes the stage. Any unmatched packet that arrived in cycle $T - D$ is dropped.

We show below that w.h.p. every packet that arrived in cycle $T - D$ will be matched at the end of this stage. Note that the pipelined scheduling algorithm performs a constant number of rounds per stage.

4.2.1 Analysis

Our analysis assumes that $M = (1 + (1/s) + \epsilon)N$, where ϵ is an arbitrarily small positive constant. The complete analysis is in the Appendix. Here we present a simplified analysis for the case when ϵ and s are both 1.

We start by analyzing a variant of the basic matching process in which only a random sample of the memory banks attempt to match themselves to the inputs. The rounds start with round $i = 0$ to facilitate relating this process to the rounds in the pipelined matching process. In the ith round of this 'sampled matching process' each memory bank attempts to match itself with probability $1/2^{i+1}$, for $i \geq 0$. We now describe this algorithm and we establish that it computes a perfect matching in $O(\log^* N)$ rounds. The base used for the logarithm for the $\log^* N$ analysis is not 2, but a value b, which is less than 2 but greater than $e^{1/e}$. (We note that $b \uparrow\uparrow i$ remains less than e for all i if $b < e^{1/e}$.) The more detailed analysis in the appendix, which works for any $\epsilon > 0$, establishes the result using the traditional base 2.

Sampled Matching Process:
for $i = 0, 1, \cdots$ in parallel

1. each unmatched memory sends a message to a random compatible input port with probability $1/2^{i+1}$ and does nothing with probability $1 - 1/2^{i+1}$.

2. each input port i picks a memory bank j that sent it a message and assigns its current packet to that memory bank. It then broadcasts a bit to all memory banks to inform them that it is no longer available to be matched (the bit sent to memory bank j is a 1 and the bit sent to all other processors is 0).

3. In parallel each memory bank that receives a 1-bit from its matched input decrements a counter initially set to s. If the counter goes down to zero, the processor declares itself matched.

Let $b = e^{(1/2-\delta)}$ where $0 < \delta < 1/2 - 1/e$, and let $z_i = \frac{N}{2^{i+1} \cdot b \uparrow\uparrow i}$.

Lemma 4.5 *After the ith iteration of the Sampled Matching Process, the number of unmatched inputs is* $\leq max\{\sqrt{N}, z_i\}$ *w.h.p. in* N, *where* $z_i = \frac{N}{2^{i+1} \cdot b \uparrow\uparrow i}$.

Proof: We observe that in iteration i for any given unmatched input port p, the expected number of processors compatible with p that send a message to some compatible input is $\geq (N+N\epsilon)/2^{i+1} = N/2^i$. Using a Chernoff bound we can show that with very high probability, for any constant $c > 0$, at least $(1-c) \cdot N/2^i$ of the processors that are compatible with a given unmatched input port do actually send a message in that round.

For $i \geq 0$, let x_i denote the number of unmatched inputs that remain after the ith iteration of the sampled matching process. For the base case of the lemma we note that $E[x_0] \leq N \cdot (1 - 1/N)^{(N+N\epsilon)\cdot(1-\delta)/2} \leq N/e^{(1-\delta)}$. Hence by applying Azuma's inequality as in the proof of Lemma 4.2 we have that $x_0 \leq N/b$ w.h.p. in N.

Assume inductively that the result holds for x_{i-1} for some $i > 0$, and consider x_i. We have

$$\begin{aligned} E[x_i] &\leq \frac{x_{i-1}}{e^{\frac{(x_{i-1}+N\epsilon)\cdot(1-\delta)}{x_{i-1}\cdot 2^{i+1}}}} \\ &\leq \frac{N}{2^i \cdot 2 \uparrow\uparrow (i-1) \cdot e^{\frac{N\cdot(1-\delta)\cdot 2^i \cdot b\uparrow\uparrow(i-1)}{2^{i+1}\cdot N}}} \\ &\leq \frac{N}{2^{i+1} \cdot e^{(1/2-\delta/2)\cdot b\uparrow\uparrow(i-1)}} \\ &\leq \frac{N}{2^{i+1} \cdot b \uparrow\uparrow i} \end{aligned}$$

If $E[x_i] \geq \sqrt{N}$ by Azuma's inequality we have that $x_i \leq \frac{N}{2^{i+1} b \uparrow\uparrow i}$ w.h.p. in N.

■

Let us now return to the analysis of the pipelined matching process, and let $D = \log_b^* N$.

Let $Q_i(T)$ be the set of input ports that have i unmatched packets at the start of cycle T, and let $q_i(T) = |Q_i(T)|$. Let $s_i(T) = \sum_{k=i}^{D} q_k(T)$. We define a predicate $\Lambda_0(T)$ to be true iff for all $i \leq D$, $s_i(T) \leq z_i$.

Theorem 4.6 *If* $\Lambda_0(T-1)$ *is true then w.h.p. in* N, $\Lambda_0(T)$ *is true.*

Proof: Consider the start of cycle T. Note that for any input port with i unmatched packets, the number of packets that can be matched at that port during cycle $T-1$ is 0, 1, or 2 (since we have assumed that $\omega = 2$). Let $r_i(T-1)$ be the number of inputs that had i or $i-1$ unmatched packets at the start of cycle $T-1$ and have at least $i-1$ unmatched packets at the end of cycle $T-1$. Since one new packet arrives at each input port at the start of cycle T, we have

$$\begin{aligned} s_i(T) = \sum_{k=i}^{D} q_k(T) &\leq \sum_{k=i+1}^{D} q_k(T-1) + r_i(T-1) \\ &\leq s_{i+1}(T-1) + 3z_{i+1} \end{aligned}$$

The last equation above uses the inequality $r_i(T-1) \leq 3z_{i+1}$. We can establish this as follows:

Let n_1 be the number of active inputs in $Q_i(T-1)$ that are unmatched after the first iteration of stage $T-1$, let X be the set of inputs that have $i-1$ unmatched packets after the first iteration of stage $T-1$, and let n_2 be the number of inputs in X that are unmatched after the second iteration of stage $T-1$. Then $r_i(T-1) = n_1 + n_2$.

Since $q_i(T-1) \leq s_i(T-1) \leq z_i$ (by the induction assumption), we have $n_1 \leq z_{i+1}$ by Lemma 4.5.

For n_2 we note that $|X| = x_1 + x_2$, where x_1 is the number of inputs that had i unmatched packets at the start of cycle $T-1$, and have $i-1$ unmatched packets after the first iteration, and x_2 is the number of inputs that had $i-1$ unmatched packets at the start of cycle $T-1$ and continue to have $i-1$ unmatched packets after the first iteration. Clearly, $x_1 \leq q_i(T-1)$, and $x_2 \leq z_i$ by the behavior of the sampled matching process on inputs that had $i-1$ unmatched packets at the start of cycle $T-1$. Hence, $|X| \leq q_i(T-1) + z_i \leq z_i + z_i \leq 2z_i$. In the second iteration of stage $T-1$ of the pipelined matching procedure each compatible processor chooses an active packet in X with probability $1/2^i$ since each input in X has exactly $(i-1)$ unmatched packets. Hence

$$n_2 \leq \frac{2z_i}{e^{\frac{(2z_i+N)\cdot(1-\delta)}{2z_i \cdot 2^i}}} \leq 2 \cdot z_{i+1}$$

Hence $r_i \leq 3z_{i+1}$. So we have

$$s_i(T) \leq s_{i+1}(T-1) + 3z_{i+1} \leq 4z_{i+1} \leq z_i$$

■

Corollary 4.7 W.h.p. in N, all packets that arrived in cycle $T-D$ have been matched by end of cycle T.

Proof: From the theorem, $q_D(T-1) = s_D(T-1) \leq \max(p_D, \sqrt{N}) = \sqrt{N}$. During the first iteration of cycle T, the basic matching procedure is applied to these $\sqrt{N}$ inputs. Hence w.h.p. in N all packets that arrived in cycle $T-D$ are matched after this step, and certainly by the end of cycle T. ■

Since $\Lambda_0(0)$ is trivially true, by Theorem 4.6 we can argue inductively that $\lambda_0(T)$ is true when $T = O(N)$. However as T grows large, the probability that $\lambda_0(T)$ will continue to be true becomes small and then we can no longer guarantee that all the packets that arrived in cycle $T-D$ will be matched at the end of cycle T. However our algorithm has a "self-stabilizing" property, i.e., if $\Lambda_0(T)$ becomes false for some T, within $O(\log N)$ cycles the input queues get back to a state where the predicate Λ_0 is true.

Define a series of predicates $\Lambda_j(T)$ such that $\Lambda_j(T)$ is true iff for all i, $s_i(T) \leq (\phi)^j p_i$ for some constant $\phi > 1$. Note that $\Lambda_j(T)$ implies $\Lambda_k(T)$ if $k \geq j$.

Theorem 4.8 If $j > 0$ and $\Lambda_j(T-1)$ is true then, w.h.p, $\Lambda_{j-1}(T)$.

Proof: (Sketch.) Recall that in the proof of Theorem 4.6 we proved that $s_i(T) \leq 3z_i$. Using a similar argument here we can prove if that $\Lambda_j(T-1)$ is true then $s_i(T) \leq 3\phi^j z_i$. Now for $c > 3\phi$ we get $s_i(T) \leq c\phi^{j-1} z_i$. If $j > 0$ then $s_0 \leq p_0$ trivially. Hence $\Lambda_{j-1}(T)$. ■

Now since $\Lambda_{\log_\phi N}(T)$ is always true, in $\log_\phi N$ steps we get back to a state where $\Lambda_0(T)$ is true. This establishes the self-stabilizing feature of our pipelined algorithm.

5 Memory Requirements

The memories used to buffer packets contribute significantly to the total cost of a router. Thus it is important to minimize both the number of memories used, and the size of each memory.

Routers need a large amount of memory in order to achieve low drop rates. Studies of Eckberg *et al.* [10] reveal that packet drop probability significantly decreases if memories can be shared across the queues for different outputs. Eckberg *et al.* show that for a Poisson packet arrival process, the amount of buffer required to achieve a certain drop probability when the arrival rate of packets is more than 90% of the total capacity of the router, reduces by a factor of 4 if a shared memory is used.

In this section we establish that our schedulers make very effective use of memory. In section 5.1 we show that the total memory used by our schedulers is very close to the minimum needed. In section 5.2 we show that the number of memory banks used by our schedulers is also close to the best possible.

5.1 Load Balance

One of the features of our algorithms is that they distribute packets evenly across the memory banks. This enables us to achieve the effect of a pure shared memory. This is independent of any assumptions on the packet arrival process, as shown in the theorem below.

Theorem 5.1 *Consider an SMS switch with N input and output ports, M memory banks, each of size K, and with each shared buffer supporting s writes per cycle. Let Q be given as an upper bound on the total number of packets in the memories in any cycle. If $K \geq Q/M + \sqrt{2cs Z \log M}$, with $c > 1$, then w.h.p. in M both of our SMS schedulers can buffer packets for up to Z cycles without dropping any packets.*

Proof: The result follows through the use of Azuma's inequality on the martingale that considers the number of packets in any given memory bank in each cycle.

Consider an arrival sequence of packets that leads to buffering of a total of R packets at the end of $T-1$ cycles. Let U_i be the map that maps packets that arrived at cycle $T-1-Z+i$ to the memories in which they were stored. Let $\mathbf{U} = (U_1, U_2 \cdots U_Z)$ and let V_m be the random variable denoting number of packets stored in memory m at the end of $(T-1)$-th cycle. Note that since all the packets in memory arrived within last Z cycles, $\mathbf{U}$ has sufficient information to compute V_m. Since there is no special bias for any of the memories, $\mathrm{E}(V_m \mid R) = R/M$. Define a sequence of random variables

$$W_i = \mathrm{E}(V_m | U_1, U_2 \ldots U_i), 0 \leq i \leq Z$$

where $W_0 = \mathrm{E}(V_m)$ and $W_Z = V_m$. Since $\mathrm{E}(W_i \mid W_{i-1}) = W_{i-1}$, the sequence of random variables W_i is a martingale. Furthermore if $\mathbf{U}$ and $\mathbf{U}'$ differ in only one of the U_i for $(T-Z-1+i)$-th cycle, at most s packet could be stored in memory m in that cycle, and at most one can leave. Therefore V_m satisfies $s-$Lipschitz condition, i.e., $|V_m(\mathbf{U}) - V_m(\mathbf{U}')| \leq s$. Thus using Azuma's inequality we obtain

$$Pr\left[|W_Z - W_0| > \sqrt{2cs Z \log M}\right] < e^{-2cs \log M/2s} = \frac{1}{M^c}$$

Thus,

$$\bigcup_{1 \leq m \leq M} Pr\left[V_m \geq R/M + \sqrt{2cs Z \log M}\right] \leq \frac{1}{M^{c-1}}.$$

Since $R \leq Q$ and $c > 1$, we have the desired result w.h.p. in M. ∎

Corollary 5.2 *Consider an SMS switch that emulates an output-queued switch with N ports and output buffer size L with M memory banks, each of size K, and each supporting s shared writes per cycle. If $K \geq LN/M + \sqrt{2csL \log M}$, where $c > 1$ is a constant, then with high probability, both of our schedulers will not drop a packet that will not be dropped by that output-queued switch.*

Proof: Use $L = Z$ and $Q = LN$ in the theorem. ∎

Note that in general, an output queued switch will use a conservative value for L to allow for occasional bursts of traffic for a single output. Thus the value of Q in the above theorem is typically much smaller than LN, and hence our scheduler would typically make much better use of the memory than a corresponding output-queued switch. Also, note that since typically $Q/M >> M >> \log M$, the value of K can be chosen to be only very slightly larger than Q/M, the minimum size needed, and the packet drop probability could be held very small even if Z is made very large. Note also that the value of Z in the above theorem is limited in only a weak way by the upper bound placed on the value on Q even if the value of K is to be held close to Q/M.

5.2 Number of Memory Banks

Even though the cumulative size of memories in an SMS architecture can be close to that of an output-queued router, having a large number of small memories is slightly more expensive than having a small number of large memories.

We have shown that that $(1 + \lceil 1/s \rceil + \epsilon)N$ memories are sufficient for an SMS router with speedup s to emulate an output-queued router. It is natural to investigate how many memories are actually necessary. First we examine what an off-line algorithm can achieve.

Lemma 5.3 *If an algorithm has knowledge of the complete arrival sequence then N memories are sufficient to store the packets while satisfying arrival and departure conflicts.*

Proof: Construct a bipartite multi-graph $G(V, W, E)$ in which the set of vertices V represent arrival times of packets, the set of vertices W represent the departure times of packets and one edge $(v, w) \in E$ is present for every packet that arrives at time v and departs at time w. Since at most N packets arrive at any cycle and at most N packets depart every cycle the maximum degree of any vertex in G is N. Thus by Birkhoff's theorem [29, Page 40] it can be edge-colored using N colors and packets belonging to every color-class can be stored in one memory. ∎

The requirement on N memories is also trivially a lower bound since there are potentially N new packets in a cycle.

Of course, in the context of a router, the algorithm has to operate on-line. Now we look at the absolute minimum number of memory banks that is required if an adversary is allowed to place packets in the memories.

Lemma 5.4 *If an adversary places packets in the memory then it is necessary to have $N + \lceil (N - 1)/s \rceil$ memories in order to satisfy arrival and departure constraints.*

Proof: Consider the case where at every cycle $T < N^2 - 1$, exactly 2 packets arrive for output $(T \bmod (n-1)) + 1$, one packet arrives for every output o such that $o \neq (T \bmod (n-1)) + 1$ and $o < N$, and no packet arrives for output N. At cycle $N^2 - 1$, the total number of arrivals at each output between 1 to $N-1$ would be $N^2 + N$ but the total number of packets that departed through each output would be $N^2 - 1$. Thus there would be $N + 1$ packets in the memory for each output from 1 to $N - 1$. Hence for each of the next $N + 1$ cycles we will have $N - 1$ packets scheduled to depart. An adversary could choose a set B of $N - 1$ memories and place all of these packets

into the memories in B such that each memory stores one packet of each time-stamp between N^2 and $N^2 + N$. Now if N packets arrive all destined for output N, then each packet will have a departure conflict with each memory in B. Thus all of these new packets must be stored in some memory that is not in B and no 2 packets can be stored in same memory. Therefore there must be additional N memories. Hence we need $2N - 1$ memories to store the packets. ■

Since our algorithm controls the placement of packets in the memory it is possible that such an algorithm can make do with a smaller number of memory banks than the bound in Lemma 5.4. We now show that it is impossible for an SMS router with less than $9N/8$ memories to behave identically to an output-queued router, regardless of how sophisticated its scheduling algorithm is. In particular this means that we cannot achieve the off-line optimal behavior in the on-line case with only N memory banks, or even with $N + \delta N$ memories, if $\delta < 1/8$.

Theorem 5.5 *There is no deterministic algorithm that can match any sequence of packet arrivals to memories while satisfying arrival and departure constraints if the number of memories is $M = N + \Delta$ and $\Delta < N/8$. Furthermore, for any randomized algorithm there exists an arrival sequence for which it will fail with probability at least $1/2$.*

In order to prove the theorem we will use a set of lemmas that show that if we have a sequence of subsets of size close to half of the original set such that any two consecutive sets are disjoint, then any pair sets with even sequence number have a significant intersection.

Lemma 5.6 *If $X, Y, Z \subseteq [N + \Delta]$ such that $|X| = |Y| = |Z| = N/2$ and $X \cap Y = Y \cap Z = \emptyset$ then $|X \cap Z| \geq N/2 - \Delta$.*

Proof: Since $X \cap Y = Y \cap Z = \emptyset$, both X and Z are subsets of Y^c. Since all sets are subsets of $[N + \Delta]$ and $|Y| = N/2$, we have $|Y^c| = N/2 + \Delta$. Hence the minimum size of $X \cap Z$ is $N/2 - \Delta$, i.e., $|X \cap Z| \geq N/2 - \Delta$. ■

Lemma 5.7 *For any three sets X,Y,Z of size $N/2$ if $|X \cap Y| \geq N/2 - \alpha$ and $|Y \cap Z| \geq N/2 - \beta$ then $X \cap Z \geq N/2 - \alpha - \beta$.*

Proof: The result follows from the observation that $|X \cap Y \cap Z| \geq N/2 - \beta - \alpha$. [] ■

Lemma 5.8 *For any series of sets $S_0, S_1 \cdots S_{2m} \in [N + \Delta]$ if $|S_i| = N/2$ and $S_i \cap S_{i+1} = \emptyset$ then, $|S_0 \cap S_{2m}| \geq N/2 - m\Delta$.*

Proof: The base case when $m = 1$ follows from Lemma 5.6. Let the lemma be true for some $m = p$. Thus $|S_1 \cap S_{2p}| \geq N/2 - p\Delta$ and $|S_{2p} \cap S_{2p+2}| \geq N/2 - \Delta$. Therefore from Lemma 5.7 we get $|S_1 \cap S_{2(p+1)}| = N/2 - (p + 1)\Delta$. ■

We can now prove Theorem 5.5. We will do so by defining two packet arrival sequences such that based on choices made by any algorithm, an adversary can always choose one of the arrival sequence for algorithm to fail if $\Delta < N/8$.

Assume the number of outputs is even. Let O_1 be a set of $N/2$ outputs and O_2 be remaining set of outputs. Define a_i (b_i) to be the set of packets that depart at time i and are destined for an output in O_1 (O_2). Our arrival process is such that $|a_i| = N/2$ or 0 and all the packets for any set a_i arrive in the same cycle. Similarly $|b_i| = N/2$ or 0 and all the packets in any set b_i arrive in the same cycle.

Now we will present two arrival sequences. The two arrival sequences are described in Table 1. Both sequences have a common prologue till time 9 as described in the first column of the table. The

second and third columns describe the packets that arrive in sequence 1 and sequence 2 respectively after prologue. A dash in the input column indicates that no packets arrived at those $N/2$ inputs. It is easy to verify that the time-stamp assignments are consistent with output queuing. We will use A_i^* (B_i^*) to represent the set of memories that the packet of a_i (b_i) will be stored in, where the superscript $*$ is either p, 1 or 2 based on whether the set of packets correspond to prologue, sequence 1 or, sequence 2 respectively. Since all the packets departing together must be stored in different memories, if $a_i \neq \emptyset$ then $|A_i^*| = |a_i| = N/2$. Similarly if $b_i \neq \emptyset$ then $|B_i^*| = N/2$.

For notational convenience, we introduce the infix binary relational operator $\not\Leftrightarrow$ denoting set disjointness, i.e., $U \not\Leftrightarrow V$ iff $U \cap V = \emptyset$. From arrival time constraints we get $A_{11}^p \not\Leftrightarrow A_{12}^p$, $B_{12}^1 \not\Leftrightarrow B_{13}^1$, $B_{12}^2 \not\Leftrightarrow B_{13}^2$, and $A_{14}^2 \not\Leftrightarrow B_{11}^2$, and from departure time constraints we get $A_{11}^p \not\Leftrightarrow B_{11}^2$, $A_{12}^p \not\Leftrightarrow B_{12}^1$, $A_{13}^p \not\Leftrightarrow B_{13}^1$, $A_{13}^p \not\Leftrightarrow B_{13}^2$, and $A_{14}^2 \not\Leftrightarrow B_{14}^2$.

Now since there are a total of $N + \Delta$ memories and A_{11}^p is connected to B_{11}^2 through a chain of 8 $\not\Leftrightarrow$ relations, from Lemma 5.8 we set $B_{11}^2 \cap A_{11}^p \geq N/2 - 4\Delta$. But we know that $B_{11}^2 \cap A_{11}^p = \emptyset$. Thus $N/2 - 4\Delta \leq 0$ or $\Delta \geq N/8$.

Therefore we conclude that if $\Delta < N/8$ any deterministic algorithm will fail. Furthermore, if any randomized algorithm, chooses A_{11}^p and A_{13}^p such that it works correctly for sequence 1 with probability θ then it must fail for sequence 2 with probability θ. Thus the worst case probability of failure for any randomized algorithm is at least $\max(\theta, 1-\theta) \geq 0.5$.

Prologue			Sequence 1			Sequence 2		
time	input		time	input		time	input	
1	a_1	a_2	10	b_{11}	–	10	b_{11}	a_{14}
2	a_3	a_4	11	b_{12}	b_{13}	11	b_{12}	–
3	a_5	a_6				12	b_{13}	b_{14}
4	a_7	a_8						
5	a_9	a_{10}						
6	a_{11}	a_{12}						
7	b_7	b_8						
8	b_9	b_{10}						
9	a_{13}	–						

Table 1: Adversarial arrival sequence.

A Detailed Analysis for Section 4.2

We now give a detailed analysis of the pipelined randomized scheduler based on the pipelined matching procedure in section 4.2, for the case when s is a positive integer, and $\epsilon > 0$ is an arbitrarily small constant. Let $\gamma = \epsilon/s$.

It is interesting to note that $log_b^* N$ is not defined for all values of N, if $b \leq e^{1/e}$. In fact, if $b \leq e^{1/e}$ then $(b \uparrow\uparrow i) \leq e$ for any value of i. Thus we cannot simply repeat the analysis in Section 4.2 with $b = e^{\frac{(1-\delta)\gamma}{2}}$. Here we present a more involved proof.

Recall that $z_i = \frac{N}{2^{i+1}(b \uparrow\uparrow i)}$. We will set $b = 2$ for this analysis. Let D be the smallest integer such that $z_D \leq \sqrt{N}$. Clearly $D = O(\log^* N)$. Let $Q_i(T,t)$ be the set of input ports that have i unmatched packets at the start of t-th iteration of the *pipelined matching procedure* in cycle T, and let $q_i(T,t) = |Q_i(T,t)|$. Let $s_i(T,t) = \sum_{k=i}^{D} q_i(T,t)$.

We define a series of predicates $\Lambda_0(T), \ldots, \Lambda_D(T)$. Predicate $\Lambda_j(T)$ is defined to be true iff for all $i \leq D$, $s_i(T,0) \leq z_{i-j}$, where $z_i = N$ if $i \leq 0$. Note that this is a refinement of the predicates

Λ_j defined in the extended abstract (as are s_i, q_i and Q_i).

Theorem A.1 *There exist a suitable constant ω such that if each stage executes ω iterations of* pipelined matching procedure *then $\Lambda_0(T)$ implies $\Lambda_0(T+1)$ w.h.p. in N.*

In order to prove the above theorem we will first need the following lemma.

Lemma A.2 *If $s_{i+1}(T,t) \leq a$ and $s_i(T,t) \leq a+b$ then w.h.p. in N we must have $s_i(T,t+1) \leq a + be^{-N\gamma/(2^i b)}/\alpha$.*

Proof: Let $q_i(T,t) = x$ and $s_{i+1}(T,t) = y$. If $x \leq \sqrt{N}$ then at the end of that iteration w.h.p. in N all the inputs in $Q_i(T,t)$ will get matched. Otherwise, we will have at most $xe^{-N\gamma/(x2^i)}/\alpha$ inputs with i unmatched inputs that were also in $Q_i(T,t)$ (similar to Lemma 4.2). Let δ be the number of inputs that got matched in $Q_{i+1}(T,t)$ thus $q_i(T,t+1) \leq xe^{-N\gamma/(2^i x)}/\alpha + \delta$ and $s_{i+1}(T,t+1) \leq y - \delta$. Therefore, $s_i(T,t+1) = s_{i+1}(T,t+1) + q_i(T,t+1) \leq y + xe^{-N\gamma/(2^i x)}/\alpha$. Thus,

$$s_i(T,t+1) \leq \max_{y \leq a,\ x+y \leq a+b} (y + \frac{xe^{-N\gamma/(2^i x)}}{\alpha}).$$

It is straightforward to show that the function on the R.H.S. achieves its maxima at $y = a$ and $x = b$. Substituting that we get the desired result. ∎

Substituting $a = z_{i+1}$, $b = z_i - z_{i+1}$ and $t = 0$ in the above lemma we get $s_i(T,1) \leq z_{i+1} + \frac{z_i - z_{i+1}}{\alpha}$. Since $z_{i+1} \leq z_i/2$ we get $s_i(T,1) \leq \beta z_i$, where $\beta = 1/2 + 1/2\alpha < 1$. Similarly $s_{i+1}(T,1) \leq \beta z_{i+1}$. Thus applying this argument repeatedly we get $s_i(T,f) \leq \beta^f z_i$. Let g be a constant such that $\beta^g \leq \min(1/2, \gamma/\ln 2)$. Thus $s_i(T,g) \leq z_i\beta^g$ and $s_{i+1}(T,g) \leq z_{i+1}\beta^g$.

Substituting $a = z_{i+1}\beta^g$ and $b = z_i\beta^g$ and $t = g$ in Lemma A.2 for the next iteration it is not difficult to show that

$$s_i(T,g+1) \leq \beta^g \left(z_{i+1} + z_i e^{-\frac{\gamma N}{\beta^g 2^i z_i}} \right) \leq z_{i+1}.$$

Thus if we set $\omega \geq g+1$, We have $s_{i-1}(T,\omega) \leq z_i$. Since at most one packet arrives in a cycle, $s_i(T+1,0) \leq s_{i-1}(T,\omega) \leq z_i$. Hence $\Lambda_0(T+1)$ holds with high probability in N.

Lemma A.3 *If $\Lambda_0(T)$ is true, w.h.p. in N, all packets that arrived in cycle $T-D$ have been matched at the end of cycle T.*

Proof: From the definition of $\Lambda_0(T)$ we get $q_D(T,0) = s_D(T,0) \leq \sqrt{N}$. Thus w.h.p. in N all the inputs in $Q_D(T,0)$ get matched in the first iteration of *pipelined matching procedure*. Thus $q_D(T,1) = 0$, i.e., no input has D unmatched packets. Thus all the packets that arrived $T-D$ cycles earlier are matched. ∎

Since $\Lambda_0(0)$ is trivially true, by Theorem A.1 we can argue inductively that $\Lambda_0(T)$ is true for $T = O(N)$. However as T grows large, the probability that $\Lambda_0(T)$ will continue to be true becomes small and then we can no longer guaranty that all the packets that arrived in cycle $T-D$ will be matched at the end of cycle T. However if we set $\omega \geq 2(g+1)$ our algorithm becomes "self-stabilizing" , i.e., if $\Lambda_0(T)$ becomes false for some T, then within D cycles the input queues get back to a state where the predicate Λ_0 is true.

Note that $\Lambda_j(T)$ implies $\Lambda_k(T)$ if $k \geq j$.

Theorem A.4 *If $j > 0$ and $\Lambda_j(T)$ is true then, w.h.p, $\Lambda_{j-1}(T+1)$ is true.*

Proof: Recall that in the proof of Theorem A.1 we proved that if $s_i(T,0) \leq z_i$ then $s_i(T,g+1) \leq z_{i+1}$. Using a similar argument if $s_i(T,0) \leq z_{i-j}$ then $s_i(T,(g+1)) \leq z_{i-j+1}$. If we apply another $g+1$ iterations we get $s_i(T,2(g+1)) \leq z_{i-j+2}$. Thus setting $\omega = 2(g+1)$, we get $s_i(T+1,0) \leq s_{i-1}(T,2(g+1)) \leq z_{i-j+1}$. Hence $\Lambda_{j-1}(T+1)$ holds. ■

Since $\Lambda_D(T)$ is trivially true, in D steps we get back to a state where $\Lambda_0(T)$ is true. This establishes the self-stabilizing feature of our pipelined algorithm.

References

[1] Noga Alon and Joel H. Spencer. *The Probabilistic Method.* Wiley, John & Sons, Incorporated, 2000.

[2] T. Anderson, S. Owicki, J. Saxe, and C. Thacker. High-speed switch scheduling for local area networks. *ACM Transactions on Computer Systems*, November 1993.

[3] A. Prakash, A. Aziz, and V. Ramachandran. Randomized Parallel Schedulers for Switch-Memory-Switch Routers: Analysis and Numerical Studies. manuscript, June 2003.

[4] R. Barker, P. Massiglia, and L. Krantz. *Storage Area Networking Essentials.* McGraw-Hill, 2001.

[5] C.-S. Chang, D.-S. Lee, and Y.-S. Jou. Load balanced Birkhoff-von Neumann switches, part I: one-stage buffering. *Computer Communications*, 2001.

[6] C.-S. Chang, D.-S. Lee, and C.-M. Lien. Load balanced Birkhoff-von Neumann switches, part II: multi-stage buffering. *Computer Communications*, 2001.

[7] S.-T. Chuang, A. Goel, N. McKeown, and B. Prabhakar. Matching output queueing with a combined input output queued switch. In *IEEE Infocom*, 1999.

[8] A. Czumaj, F. Meyer auf de Heide, and V. Stemann. Contention resolution in hashing based shared memory simulations. *SIAM Jour. Comput.*, 29(5), 2000.

[9] J. Duato. *Interconnection Networks.* Morgan-Kaufmann, 2002.

[10] A. E. Eckberg and T. C. Hou. Effects of output buffer sharing on buffer requirements in an atdm packet switch. In *IEEE Infocom*, 1988.

[11] M. Farley. *Building storage area networks.* McGraw-Hill, 2001.

[12] W. Futral. *InfiniBand Architecture: Development and Deployment–A Strategic Guide to Server I/O Solutions.* Intel Press, 2001.

[13] John Hennessy, David Patterson, and David Goldberg. *Computer Architecture: A Quantitative Approach.* Morgan-Kaufmann, third edition, 2002.

[14] M. Hluchyj and M. Karol. Queueing in high-performance packet switches. *IEEE Journal on Selected Areas in Communications*, 6(9), December 1988.

[15] S. Iyer, R. Zhang, and N. McKeown. High-speed policy-based packet forwarding using efficient multidimensional range matching. In *ACM SIGCOMM*, 2002.

[16] S. Keshav. *An Engineering Approach to Computer Networking.* Addison-Wesley, 1997.

[17] S. Keshav and R. Sharma. Issues and Trends in Router Design. *IEEE Communication Magazine*, 1998.

[18] G. Lev, N. Pippenger, and L. Valiant. A fast parallel algorithm for routing in permutation networks. *IEEE Transactions on Computers*, 30(2), February 1981.

[19] Y. Matias and U. Vishkin. Towards a theory of nearly constant time parallel algorithms. In *Proc. IEEE FOCS*, 1991.

[20] N. McKeown. iSLIP: A Scheduling Algorithm for Input-Queued Switches. *IEEE Transactions on Networking*, 7(2), April 1999.

[21] N. McKeown, V. Anantharam, and J. Walrand. Achieving 100% throughput in an input-queued switch. In *IEEE Infocom*, 1996.

[22] N. McKeown, M. Izzard, A. Mekkittikul, W. Ellersick, and M. Horowitz. The tiny tera: a packet switch core. *IEEE Micro*, 17(1):27–33, January 1997.

[23] Juniper Networks. High speed switching device. US Patent 5,905,726, 1999.

[24] A. Pattavina. *Switching Theory.* Wiley, John & Sons, Incorporated, 2000.

[25] L. Peterson and B. Davie. *Computer Networks.* Morgan-Kaufmann, 2000.

[26] A. Prakash, S. Sharif, and A. Aziz. An $O(\lg^2 n)$ algorithm for output queuing. In *IEEE Infocom*, 2002.

[27] R. Ramaswami and K. Sivarajan. *Optical Networks: A Practical Perspective.* Morgan-Kaufmann, 2001.

[28] T. Stern and K. Bala. *Multiwavelength optical networks: a layered approach.* Prentice-Hall, 1999.

[29] J. van Lint and R. Wilson. *A Course in Combinatorics.* Cambridge University Press, 1992.

[30] A. Wilson, J. Schade, and R. Thornburg. *Introduction to PCI Express.* Intel Press, 2002.