Randomized Parallel Schedulers for Switch-Memory-Switch Routers: Analysis and Numerical Studies

Amit Prakash Adnan Aziz Vijaya Ramachandran[1]
The University of Texas at Austin
{prakash, adnan}@ece.utexas.edu, vlr@cs.utexas.edu

***Abstract*— We present new results and numerical studies of very fast schedulers for SMS (Switch-Memory-Switch) routers, which emulate output-queuing by buffering packets in a partitioned shared-memory located between input and output ports. The architecture of Juniper's core routers and Brocade's storage switches is based on SMS.**

Our numerical results demonstrate that RiPSS, a randomized highly parallel SMS scheduler that we had developed recently, runs in just 3 rounds on switches with up to 4,096 inputs, and has a very low drop probability. We also show that RiPSS makes effective use of the shared-memory, with packets being uniformly distributed across the memory banks for both Bernoulli and bursty arrivals. We describe a new and improved randomized pipelined scheduler, PRiPSS, and analyze its performance. Both our analysis and our simulation results for PRiPSS show that it has better throughput than RiPSS with a slightly higher latency in terms of rounds of communication in the underlying hardware. Our analysis also shows that PRiPSS is self-stabilizing, i.e., if occasional lapses occur due to the probabilistic nature of the algorithm, it resumes normal behavior without the need for external intervention.

While the choice of RiPSS or PRiPSS would depend on whether throughput or latency is the primary concern, our results indicate that both schedulers are much faster than other schedulers for output-queuing, whether implemented directly or through emulation on SMS.

I. Introduction

Routers play a critical role in modern computing of all forms [12], [15], [7], [4], [6], [9], [19][10, Chapters 7.12, 8.12]. 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 [17], [18], the situation has reversed, and in many networks today routers are the bottleneck in moving data.

A router needs to be able to buffer packets because of contentions for output links. This buffering can be at the input port, at the output port, in the switching fabric, or in shared memories. The first and second cases are referred to as *input queuing* and *output queuing*, respectively [12].

Output-queued routers are appealing because they have better latency and throughput than input queued routers. However, a direct implementation of an output queued router needs to run the switching fabric and the buffer memory at N times the line speed for an N-input, N-output router (since at the start of a cycle, all packets at an input port may be destined to the same output port). Thus input queuing is preferred for implementation reasons, and considerable effort has been devoted to overcoming its limitations, e.g., the development of virtual output queuing to overcome head-of-line blocking [13].

It is natural to ask if it is possible to build a router whose external behavior is identical to an idealized output-queued router using slower components. Chuang *et al.* [5] define a router S to emulate output queuing, if, given identical input arrival patterns, the departure time of every packet from S is the same as that from the output queued router. They showed that a router with queues at both the inputs and the outputs that can support 2 reads and 2 writes per cycle can in principle be scheduled to emulate output queuing. However, computing the schedule itself involves solving an instance of the marriage problem. The standard "proposal algorithm" for finding a stable marriage takes $O(N^2)$ worst-case and $O(N \log N)$ expected time. The best parallel algorithm presently known for computing stable marriages [8] has complexity $O(\sqrt{N} \cdot \log^3 N)$ and uses N^4 processors. It is extremely complicated, based on interior point methods for linear programming. Thus it is neither theoretically efficient (i.e., does not have $O(\text{polylog}(N))$ complexity), nor of practical significance.

Prakash, Sharif, and Aziz [16] proposed the Switch-Memory-Switch (SMS) architecture as an abstraction of the M-series Internet core routers from Juniper Networks. The set of input ports is connected via an $N \times M$ interconnect to M packet memories; these M memories are connected to the set of output ports through another interconnect. In every cycle one packet can be read from and written to each memory. It is shown that when $M \geq 2N - 1$, the SMS architecture could emulate an output-queued switch. Specifically a scheduler based on computing perfect matchings in bipartite graphs is presented, whose time complexity is $O(\log^2 N)$ on a parallel random access machine. However, this algorithm requires $M = 3N$. The number of processing elements and memory cells used by the PRAM are a small multiple of the size of

[1]The work of the third author was supported in part by NSF Grant CCR-9988160.

the idealized router. The most significant technical contribution in [16] was the $O(\log^2 N)$ parallel algorithm for computing a schedule under which the SMS architecture emulated output-queuing.

Subsequent to the results in Prakash *et al.* [16], Iyer *et al.* [11] reproved that an SMS router[2] with $2N-1$ packet memories running at the line rate could emulate an N input, N output output-queued switch applying the pigeonhole principle (as in [16]). Iyer *et al.* did not consider the implementation of the scheduler, and the scheduler arising from their proof has time complexity $\Omega(N)$.

Although the results in Prakash *et al.* [16] represent the first implementation of output queuing that runs in polylog time using a polynomial number of processors, the scheduler requires building and manipulating complex data structures, and its practical utility remains unclear.

The SMS router architecture has benefits beyond just the ability to emulate output queuing. High performance packet switches use SRAM rather than DRAM for packet buffers because the access time of an SRAM is less than that of DRAM by a factor of up to 40. SRAM is considerably more expensive than DRAM, and the cost (measured in terms of power, footprint, as well as price) of memory is a significant factor in the design of routers. The SMS architecture is therefore appealing because of its ability to "pool" available memory, and thereby achieve better memory utilization. Indeed, Juniper's core routers and Brocade's storage switches are based on the SMS architecture precisely because it reduces memory cost.

With the above motivation, we began work on developing practical schedulers for routers based on the SMS architecture. We developed RiPSS [3]—a very simple randomized parallel scheduler for SMS routers—which we review in detail in Section II. In essence we proved that RiPSS computes a complete assignment of packets to memories in $O(\log^* N)$ basic matching rounds with high probability (w.h.p.), *independent of the input traffic pattern*. The intuitive idea is that when there are $(2+\epsilon)\cdot N$ memories, the ratio of unmatched memories to unmatched inputs increases by an amount exponential in N (rather than a constant, which would yield a $O(\log N)$ bound on the number of iterations).

The formal proof of $O(\log^* N)$ result for RiPSS uses the machinery of Martingale analysis and Azuma's inequality. Further, although the RiPSS intuitively is very simple and fast, the constants derived in the proof are quite large, and we suspected them to be loose.

In this paper we make two contributions toward the goal of developing practical schedulers for SMS routers:

1) We use a numerical bounding technique to compute much sharper bounds for the constants in the analysis of RiPSS' runtime for specific values of N. We also use simulation to demonstrate that RiPSS rapidly converges to a complete assignment on stochastic traffic, and that it balances packets across the memories more effectively than output queuing.
2) In [3] we presented a pipelined scheduling algorithm that uses an $O(\log^* N)$ deep pipeline of packets at the input to further reduce scheduler complexity compared to RiPSS. It requires only a constant number of rounds per cycle—independent of N and the input traffic pattern—in contrast to RiPSS, which requires $O(\log^* N)$ rounds per cycle. However this algorithm requires both memories and inputs to send and process multiple messages. In this paper we present PRiPSS, an improved version of the pipelined scheduler in [3] that requires less communication and hardware. PRiPSS also uses an $O(\log^* N)$ deep pipeline of packets at the input and requires only constant number of rounds per cycle. The advantage of PRiPSS over the earlier algorithm is that it requires each memory to send just one request message and process just one grant messages per round. We also show that PRiPSS is *self-stabilizing*, i.e., it resumes normal behavior if occasional lapses occur due to the probabilistic nature of the underlying algorithm. We provide simulation studies that demonstrate the effectiveness of pipelining over RiPSS in improving throughput.

[2]The architecture in [11] is isomorphic to SMS, although it is called Parallel Shared Memory Router in that paper.

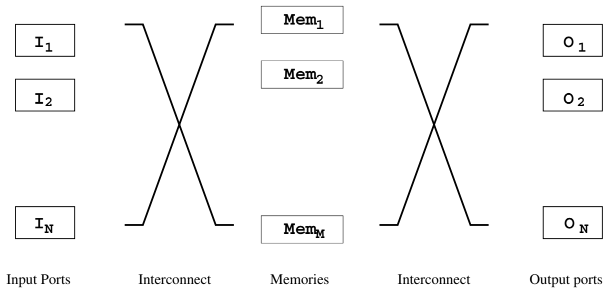


Fig. 1. The Switch-Memory-Switch (SMS) architecture.

II. Background

In this section we review germane results on SMS routers from our prior work [16], [3]. The SMS architecture is depicted in Figure 1. Input ports are connected via an $N \times M$ interconnect to M packet memories; these M memories are connected to the output ports through another interconnect. In every cycle one packet can be read from and written to each memory. (The generalization of our results to faster or slower memories is straightforward.)

A. Emulating output queuing with the SMS architecture

Since output-queuing is highly desirable, ideally, SMS scheduling should result in the SMS router emulating an output-queued router. 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*. 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.

1) The existence of a suitable schedule: In the SMS architecture packets cannot be arbitrarily placed in the memories due to two kinds of conflicts. No more than one packet that arrives at a given time can be written to a given 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 $(N-1)$. *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 $M \geq 2N-1$ by the pigeonhole principle 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) Computing a schedule efficiently: In order to construct a conflict-free schedule for transfer of packets there are three tasks to be performed in each 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. Task 1 can also be performed efficiently using a parallel prefix sum computation, as described in [16].

Task 2 is the most complex step and is the main focus of this paper. For routers that are relatively small and support slow links, 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. Prakash *et al.* [16] presented the first parallel algorithm for computing the assignment; however, the algorithm requires building and manipulating complex data structures. Subsequently, we developed RiPPS—a simple, highly-parallel randomized scheduler [3], which we describe in the next section.

B. A Randomized Parallel Scheduler for SMS

1) Computational Model for SMS Scheduler: We describe here 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 the basic algorithm of [3] I is 1; in the pipelined algorithm we present in Section IV of this paper, $I = O(\log^* N)$. 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; we assume $K \gg N$.
- 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], [14]). 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.

2) RiPSS: We describe here RiPSS—the $\underline{\text{r}}$andom$\underline{\text{i}}$zed $\underline{\text{p}}$arallel $\underline{\text{s}}$cheduler for $\underline{\text{S}}$MS introduced in [3]. RiPSS works in rounds of the Basic Matching Process given below.

At the start of each cycle, all memory banks are unmatched, each input port broadcasts the time-stamp of its packet to each memory, using which each memory bank constructs a list of the inputs that are compatible with it.

- **A Round of the Basic Matching Process**:

  In parallel do

  1) Each unmatched memory sends a grant message to a randomly selected compatible input port.
  2) Each input port i accepts a granting memory bank at random; it broadcasts a bit to all other memory banks to inform them that it is no longer available to be matched.
  3) Each memory bank that receives an accept declares itself matched.

We analyzed the above process in detail [3], and proved the following result:

Theorem 2.1: If $M = (2+\epsilon) \cdot N$, $O(\log^* N)$ rounds of the basic matching process suffice to match all inputs to memories, with high probability in N.

The key to establishing the above theorem is the following observation:

Lemma 2.2: Let $M = (2+\epsilon)N$, for any $\epsilon > 0$. If k of the N inputs remain unmatched at the start of a round then

- each input is compatible with at least $\epsilon N + k$ memories,
- the expected number of unmatched inputs at the end of the round is at most $ke^{-(1+\epsilon N/k)}$, and
- the probability that the number of unmatched inputs at the end of the round exceeds the expected number by more than $\sqrt{2M \log M}$ is at most $1/M$.

3) Pipelined Scheduler for SMS: In addition to RiPSS, a pipelined scheduler is presented in [3]. This 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. This scheduler requires both input ports and memories to communicate multiple messages in each round, but with this scheme, it is shown in [3] that $D = O(\log^* N)$ suffices to match all packets while using only a constant number of rounds per cycle. Since we will describe describe an improved pipelined scheduler that in the next section, we do not describe this algorithm further.

C. Memory Considerations

1) Load Balance: One useful feature of RiPSS is that it distributes packets evenly across the memory banks. This enables us to achieve the effect of a pure shared memory, *independent of the packet arrival process*. The following theorem and corollary are established in [3].

Theorem 2.3: Consider an SMS switch with N input and output ports, M memory banks, each of size K. Let Q be given as an upper bound on the total number of packets in the memories in any cycle. If $K \geq Q/M + \sqrt{2cZ\log M}$, with $c > 1$, then w.h.p. in M RiPSS will buffer packets for up to Z cycles without dropping any packets.

Corollary 2.4: Consider an SMS switch that emulates an output-queued switch with N input and output ports, and output buffer size L with M memory banks, each of size K. If $K \geq LN/M + \sqrt{2cL\log M}$, where $c > 1$ is a constant, then with high probability, RiPSS will not drop a packet that will not be dropped by that output-queued switch.

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. In this context the following results were established in [3].

Lemma 2.5: If an adversary places packets in the memory then at least $2N - 1$ memory banks are needed in order to satisfy arrival and departure constraints in SMS.

Since a well-designed algorithm can control 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 2.5. However, the following result is shown in [3].

Theorem 2.6: 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 $1/2$ if $\Delta < N/8$.

III. RiPSS—A Numerical Study

A. Worst case bounds on performance

The theorems presented in Section II tell us only the asymptotic behavior of RiPSS. In particular they do not tell us what the hidden constants are and for what value of N (the number of input ports) and M (the number of memory banks in the SMS architecture) we obtain acceptably small probability of failure. Since we do not have simple closed form expressions for the exact probability of failure and for the number of rounds, in this section we present concrete upper bounds on the probability of failure and number of rounds needed to limit probability of failure to a certain value.

In a given cycle, we know that each input can be incompatible with at most $N-1$ memories. Thus each input must have at least $M - N + 1$ compatible memories. We consider the *worst case* scenario, where each input has exactly $M - N + 1$ compatible memories and all inputs contend for the same set of $M-N+1$ memories. Thus the compatibility graph would be a complete $N \times (M - N + 1)$ bipartite graph. Let $P_m(i, j, k)$ be the probability of the event that if k balls are thrown uniformly at random into j bins then there are exactly i non-empty bins. Thus probability of i packets being matched in such a scenario would be $P_m(i, N, M - N + 1)$. For this to happen, if the first $k - 1$ balls fall into exactly i bins then the last ball must also fall in one of these i bins. Alternately if they fall in $i - 1$ bins then the last one must fall in a new bin. This gives us the following recurrence relation,

$$P_m(i, j, k) = P_m(i, j, k-1) \cdot \frac{i}{j} + P_m(i-1, j, k-1) \cdot \frac{j-i+1}{j}$$

Now let the probability $P_r(n, m, r)$ be the probability of matching n packets to a subset of m memories in r rounds when each input of the n inputs are compatible with each of the m memories. Thus,

$$P_r(n, m, r) = \sum_{i=1}^{n} P_m(i, n, m) \cdot P_r(n-i, m-i, r-1)$$

A similar approach can be used to computing the expected number of rounds. We emphasize that the solution to the recurrence relations provides only an upper bound, since the relations assume a very pessimistic scenario.

Table I shows the minimum number of rounds needed to ensure that the probability of all packets being matched is at least 0.999. The numbers in table show that, if $\epsilon \geq 0.5$ we never need more than 3 rounds.

Figure 2 depicts the expected number of rounds for various values of N and M. It can be seen from the figure that even for the case where $M = \lfloor 2.1N \rfloor$ (i.e., $\epsilon = .1$) we need less than 4.01 rounds for a 4096 port switch on average. With $N = 3N$ the expected number of rounds remains below 2.02. In Figure 3 we plot the probability of matching all inputs at the end of a fixed number of rounds. With $M = \lfloor 2.1N \rfloor$ and 4 rounds, the probability of matching all inputs is very close to one for a switch with up to 4096 ports. In Figure 4 we look at the effect of increasing the number of memory banks on the expected number of rounds. We use $M = \lfloor (2+\epsilon)N \rfloor$ and plot the expected number of rounds for different values of N and ϵ. Obviously, as ϵ increases the expected number of rounds decreases. However there does not seem to be much gain after $\epsilon = 0.5$.

B. Simulation studies on stochastic traffic

The failure probabilities and expected number of rounds computed in the previous section provide upper bounds on worst case traffic. However, we do not know of any explicit arrival sequence that would achieve those bounds and it may be the case that no such sequence exists. Similarly, we do not know whether the bound on M provided in Theorem 2.6 is

TABLE I

UPPER BOUND ON NUMBER OF ROUNDS NEEDED TO MATCH ALL THE PACKETS WITH PROBABILITY ≥ 0.999 IN A SWITCH WITH N INPUTS AND $(2+\epsilon)N$ MEMORIES (COMPUTED FROM THE RECURRENCE RELATION **for worst case traffic**).

ϵ \ N	16	32	64	128	256	512	1024	2048	4096
0.1	4	4	4	5	5	5	5	5	5
0.5	3	3	3	3	3	3	3	3	3
1.0	3	3	3	3	3	3	3	3	3

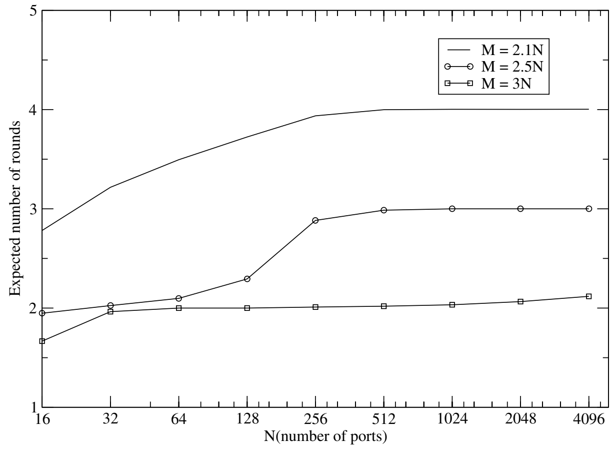


Fig. 2. Bound on the expected number of rounds needed to match all the packets as a function of N (obtained from the recurrence relation **for worst case traffics**)

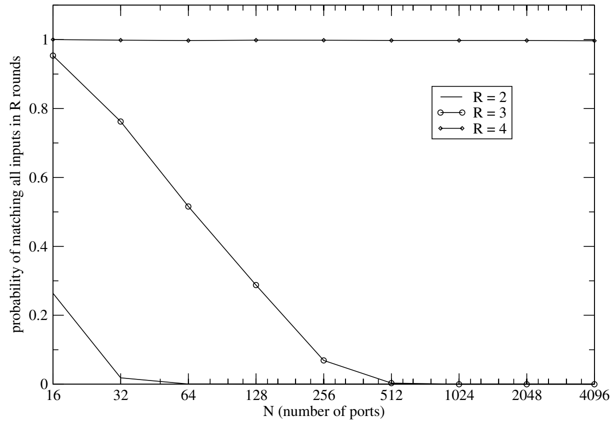


Fig. 3. Lower bound on probability of matching all inputs in R rounds (obtained from the recurrence relations **for worst case traffic**)

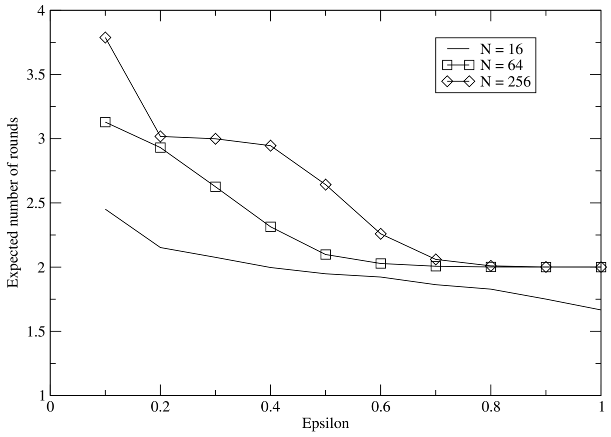


Fig. 4. Upper bound on average number of rounds needed to match all the packets as a function of ϵ, where $M = \lfloor (2+\epsilon)N \rfloor$ (obtained from the recurrence relations **for worst case traffic**)

tight or not. In this section we present results of simulation of SMS switch for different values of N and M and packet arrival patterns. Even though Lemma 2.5 guarantees the existence of a perfect match only when $M \geq 2N-1$, in our simulations we observed that RiPSS never failed to match all the inputs if $M \geq 1.6N$ (with $M = 1.5N$ we saw failures in 0.1% of the cycles).

Figure 5, Figure 6, and Table II refer to results obtained by simulating uniform Bernoulli arrival, in contrast to the worst case arrivals assumed in the previous section. Figure 5 plots the average number of rounds needed to match all inputs with uniform Bernoulli traffic; as might be expected, the average number of rounds needed here is less than the worst-case upper bound in the previous section. Even when only $1.6N$ memories are used, the packets are matched in less than 3.1 rounds on average for N up to 4096. Figure 6 shows the number of rounds needed for different values of M/N.

Table II shows the number of rounds needed to bound the probability of failing to match all inputs to at most 0.1%, for different values of N and M, as observed by simulating the system for 100,000 cycles. Even with $1.6N$ memories, all packets are matched 99.9% of the time if only 3 rounds are used. Further, in a cycle in which the scheduler does not construct a perfect match, only a small number of packets remain unmatched, hence only a small number of packets are dropped.

In [3] we claim that one of the advantages of SMS over output queuing is that SMS creates the effect of a shared memory by pooling all memory banks together (see also Theorem 2.3). Specifically, in output-queuing it may happen that one of the output buffers is full, resulting in packet drops while the other buffers are empty, but in SMS, with RiPSS and the analysis in [3], the packets are evenly distributed across all memory banks and hence packets are dropped only if all

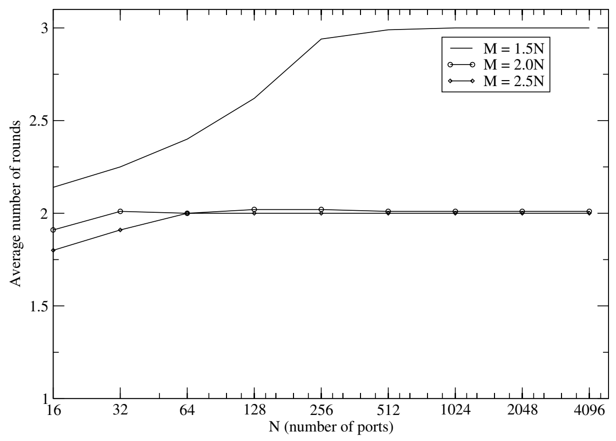


Fig. 5. The average number of rounds needed to match all packets as function of number of ports (as observed in simulations with **uniform Bernoulli traffic**, simulated for 100,000 cycles)

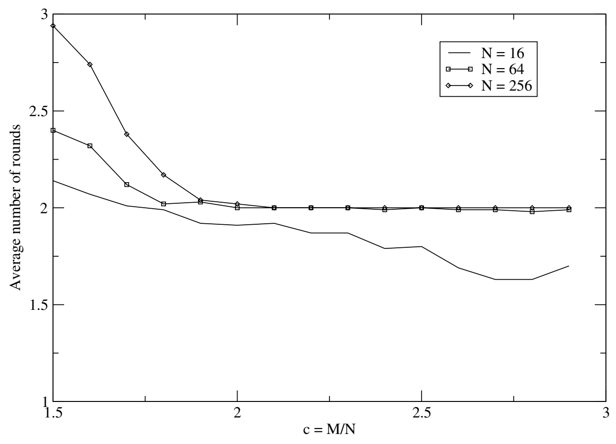


Fig. 6. The average number of rounds needed to match all the inputs, plotted against ratio of number of memories to that of input ports (as observed in the simulations with **uniform Bernoulli traffic**, simulated for 100,000 cycles)

buffers are almost full. Further, according to the analysis in [3], this is independent of the arrival pattern and works even with bursty traffic. To study this theoretical prediction, we simulated both SMS and output queued routers with 64 input and output ports and bursty traffic (geometrically distributed bursts for randomly chosen outputs). For each cycle we measured the ratio of number of packets in a buffer to that of average number of packets in each buffer. Ideally, if all buffers are equally full, all measurements should be close to one. A spread in these numbers indicates that buffers are not evenly occupied. Figure 7(a) shows distribution of this measurement for both SMS and OQ for bursty traffic. Here, the plot for SMS buffers is concentrated around one, indicating that packets are evenly balanced across all buffers, while for OQ, a buffer can have as much as 15 times the average number of packets. Figure 7(b) shows similar results for uniform Bernoulli arrivals. Since traffic arrives uniformly at all outputs, in this case the output queues also remain balanced and small, but even here, the distribution of packets across buffers is more balanced in the SMS router than in the OQ router.

TABLE II

NUMBER OF ROUNDS NEEDED TO MATCH ALL THE PACKETS MORE THAN 0.999 OF THE TIMES, IN A SWITCH SIMULATED FOR 100,000 CYCLES WITH N-INPUTS AND $c \cdot N$ MEMORIES FOR **uniform Bernoulli traffic**)

c \ N	16	32	64	128	256	512	1024	2048	4096
1.6	3	3	3	3	3	3	3	3	3
2.0	3	3	3	3	3	3	3	3	3
2.5	2	2	2	2	2	2	2	2	2

IV. PIPELINING

When multiple rounds of the basic matching procedure are used, the memories and inputs that are matched in earlier rounds will remain idle till the end of the cycle. In this section we present a simple pipelined scheduling algorithms, PRiPSS (Pipelined RiPSS) that makes fuller use of the interconnect in each round, thereby executing only a constant number of rounds of communication per cycle. PRiPSS is an improvement over an earlier pipelined scheduler in [3].

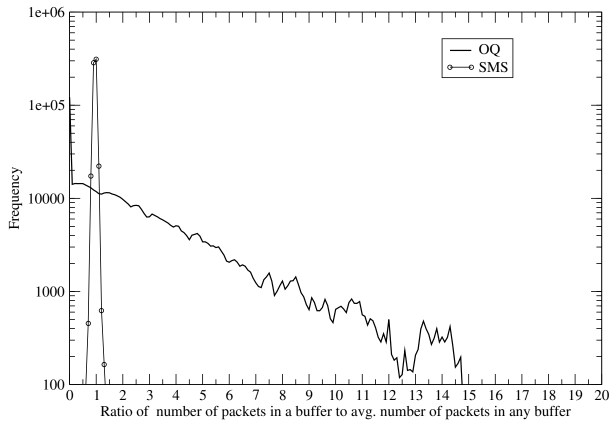
PRiPSS uses multiple cycles to construct a matching for each set of packets that arrive together. However matchings are constructed for multiple sets of packets simultaneously in a pipelined fashion. Consequently, the number of rounds per cycle reduces relative to RiPSS, but packets wait for $D = O(\log^* N)$ cycles at the inputs before they are transferred to the memories. The value D is the *latency* of the pipelined scheduler. 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

(a) Bursty traffic



(b) Uniform Bernoulli traffic

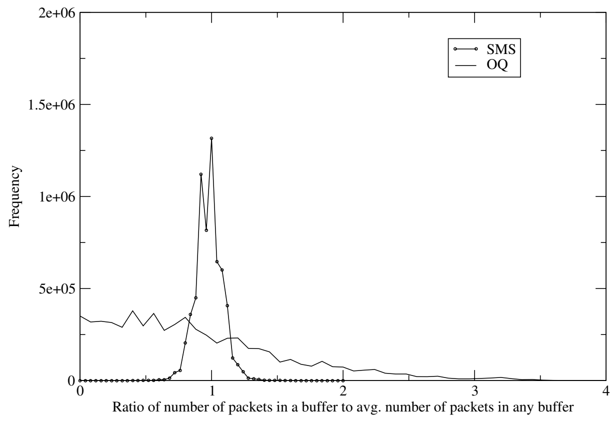


Fig. 7. Distribution of the ratio of number of packets in a buffer to the average number of packets across all the buffers in that cycle (from simulations of 64 port SMS and OQ routers with (a) bursty traffic and (b) uniform Bernoulli traffic). Both systems were simulated for 100,000 cycles. A spread in the curve around one indicates that the packets are not evenly distributed across different memory banks. In both simulations we used $N = 64$ and $M = 128$. The average number of packets in the buffer for bursty arrivals was 4218 while in uniform arrivals it was 610.

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 RiPSS) 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.

Analysis: Our analysis assumes that $M = (2 + \epsilon)N$, where ϵ is an arbitrarily small positive constant. For ease of explanation, here we present a simplified analysis for the case when $\epsilon = 1$. The complete analysis is given in the Appendix I.

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 $e^{1/e}$. The more detailed analysis is given in the Appendix I (which works for any $\epsilon > 0$) establishes the result using the traditional base 2.

Sampled Matching Process:

for $i = 0, 1, \cdots$

1) in parallel 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) 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.

Define $x \uparrow\uparrow y$ to be a y-high tower of x. 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 4.1: 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 [1] we can show that with 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 [1] 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 4.2: 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 4.1.

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 4.3: 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 4.2 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 4.4: 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 4.2 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 PRiPSS. We summarize the result in the following theorem.

Theorem 4.5: The pipelined scheduler PRiPSS uses a constant number of rounds per cycle and w.h.p. in N, matches all packets that arrived D cycles earlier, for $D = O(\log^* N)$. Further, when the low-probability event of failure in matching all packets occurs, PRiPSS resumes its normal behaviour of matching all packets within $\log_\phi N$ cycles, for a suitable constant $\phi > 1$.

A. Simulation of PRiPSS

We ran simulations of PRiPSS for 100,000 cycles with Bernoulli arrival sequences, for number of ports N varying between 16 and 4096, and number of memories M varying from $1.6 \cdot N$ to $2.5 \cdot N$.

PRiPSS ran without any packet being dropped in any of the runs even when the number of rounds in each cycle was held to $\omega = 2$. For $N \leq 2048$ we needed $D = 2$. For $N = 4096$, we needed $D = 3$.

We compared PRiPSS with two other pipelined scheduling strategies that we developed, PRiPSS-1 and PRiPSS-2. We describe both algorithms below and we prove that both work with $D = I = O(\log \log N)$. The time-stamp computation and notion of active packet remains the same in both PRiPSS-1 and PRiPSS-2. They differ from PRiPSS in the pipelined matching procedure. Below we describe the corresponding matching procedures.

PRiPSS-1: Procedure for Cycle T

(a) Execute the following three phases:
Phase 1:
(i) In parallel each new active input broadcasts to all memory banks the time-stamp of its active packet together with its arrival time mod D.
(ii) In parallel each memory sends a message to a random compatible input port.
(ii) In parallel each input port picks a memory bank that sent it a message and assigns its active packet to that memory bank. It then replaces its matched active packet by the first unmatched packet in its buffer.
Phase 2: Repeat phase 1 on the current active packets.
Phase 3: Repeat phase 1 *only on current active packets that arrived in cycle $T - D$.*

(b) Transfer all matched packets that arrived in cycle $T - D$ to the memory banks.

PRiPSS-2: Procedure for Cycle T

In the following, δ is a suitably small constant.

(a) **Phase 1:**
(i) In parallel each new active input broadcasts to all memory banks the time-stamp of its active packet together with its arrival time mod D.
(ii) In parallel each memory bank m sends a message to an input port chosen as follows: Let A_m be the set of inputs compatible with m that have an active packet that arrived in cycle $T - D$. If A_m is non-empty then with probability δ one of the inputs in A_m is chosen and with probability $1 - \delta$ some other compatible input is chosen; otherwise the input is chosen uniformly at random from all compatible inputs.
(ii) In parallel each input port picks a memory bank that sent it a message and assigns its active packet to that memory bank. It then replaces its matched active packet by the first unmatched packet in its buffer.
Phase 2: Repeat phase 1 on the current active packets.

(b) Transfer all matched packets that arrived in cycle $T - D$ to the memory banks.

A sketch of the proofs that both PRiPSS-1 and PRiPSS-2 are able to match all packets that arrived $T - D$ cycles, for $D = \log \log N$, w.h.p. in N is given in Appendix II and III.

TABLE III
NUMBER OF STAGES NEEDED FOR DIFFERENT VALUES OF M AND N IN PRiPSS, PRiPSS-1, AND PRiPSS-2, $\delta = 0.5$. THESE VALUES WERE OBTAINED BY SIMULATION FOR 100,000 CYCLES WITH UNIFORM BERNOULLI TRAFFIC.

$M/N =$	1.6	2.0	2.5	1.6	2.0	2.5	1.6	2.0	2.5
N	PRiPSS			PRiPSS-1			PRiPSS-2		
16	2	2	2	2	2	2	2	2	2
32	2	2	2	2	2	2	2	2	2
64	2	2	2	2	2	2	2	2	2
128	2	2	2	2	2	2	2	2	2
256	2	2	2	2	2	2	2	2	2
512	2	2	2	2	2	2	2	2	2
1024	2	2	2	3	3	2	3	3	2
2048	2	2	2	3	3	2	3	3	2
4096	3	3	2	3	3	2	3	3	2

We simulated PRiPSS, PRiPSS-1, and PRiPSS-2 for 100,000 cycles with uniform Bernoulli arrivals and varied D. For PRiPSS-2 we found $\delta = 0.5$ gave the best results. Table III shows the minimum value of D needed such that all the packets are matched in the simulations. Note that the algorithm for PRiPSS-1 requires 3 rounds of communication between memories and inputs while PRiPSS-2 and PRiPSS use only 2 rounds in our simulations. It appears that PRiPSS is an attractive alternative to the basic non-pipelined RiPSS and performs better than PRiPSS-1 and PRiPSS-2. It placed every packet in memory using just 2 rounds per cycle while keeping the latency to only two cycles for $N \leq 2048$, and using only $M = 1.6N$ memory banks. While PRiPSS-2 also requires only 2 rounds per cycle, for it needs 3 stages for $N \in [1024, 2048]$ and $1.6N \leq M \leq 2N$ in contrast to PRiPSS that requires only 2 stages.

V. DISCUSSION

In this paper we have presented several results on practical routers for output-queued switches based on the SMS architecture. We have presented extensive numerical results for RiPSS, a randomized, parallel scheduler for SMS described in our earlier work [3]. We have presented a new and improved pipelined randomized parallel scheduler, PRiPSS, and analyzed its performance, and we have presented numerical results evaluating the performance of PRiPSS and two other pipelined heuristics.

Our results for RiPSS and PRiPSS are very encouraging. For switches with up to N= 4,096 input ports, RiPSS placed all incoming packets in compatible memory banks using just 3 rounds in 99.9% of the cycles even when the number of memory banks M was only 1.6N. Earlier results in [3] have shown that under adversarial conditions, no placement is possible unless $M \geq 2N-1$, and there exist arrival sequences for which no randomized scheduler can place packets more than half the time unless $M \geq 9N/8$. The fact that RiPSS places all packets under Bernoulli arrivals in just 3 rounds in 99.9% of the cycles when $M \geq 1.6N$ is encouraging.

The effective use of available memory by RiPSS relative to the output-queued switch it simulated is impressive. For both Bernoulli arrival and bursty traffic, most of the memory banks in the SMS switch using RiPSS had load very close to the average load in most cycles. In practical terms, this means that if one uses RiPSS in an SMS architecture, the total amount of buffer space required needs to be only slightly larger than the total number of packets that need to be buffered. This had been proved analytically in [3] and our simulations support this result convincingly.

The pipelined scheduler PRiPSS that we presented and analyzed in this paper is superior to the pipelined scheduling algorithm in [3] since it uses less communication. In the pipelined algorithm in [3] each memory needs to send D request messages and can receive upto D grants, where $D = O(\log^* N)$ is the depth of the pipeline. This requires extra control communication bandwidth as well as processing. In contrast, in PRiPSS, each memory bank needs to send only one request per round. As with the pipelined scheduler in [3], our analysis shows that PRiPSS is self-stabiling.

In our simulations with the pipelined scheduler PRiPSS using Bernoulli arrivals, for switches with N up to 2,048 and $M = 1.6N$, a two-stage pipeline with two rounds of communication per stage sufficed to place all packets in memory in every cycle, without exception. When $N = 4,096$ we needed 3 stages in the pipeline, with each stage continuing to have 2 rounds. In view of the extremely small growth rate of the $\log^*$ function, we expect that PRiPSS will continue to need only 3 stages for much larger values of N.

We compared PRiPSS with two other pipelined strategies, PRiPSS-1 and PRiPSS-2, that appeared to be natural heuristics, but for which we could prove only an $O(\log\log N)$-stage bound on the delay, rather than the $O(\log^* N)$ bound that we proved for PRiPSS.

Our simulations reinforced our analytical results: Though all three pipelined strategies needed at most $D = 3$ stages for the range of values we considered, PRiPSS-1 and PRiPSS-2 needed to switch to 3 stages at smaller values of the parameters than PRiPSS.

PRiPSS is superior to RiPSS in terms of throughput. The cycle time of PRiPSS has to be only long enough to accommodate 2 rounds of communication, hence the cycle time can be short, thereby increasing the number of packets processed per second. However PRiPSS does have the extra overhead of buffering D packets at each input and of computing active packets. If latency is more important, and a small drop rate is tolerable, then RiPSS is better than PRiPPS since for uniform Bernoulli traffic it placed all packets in just 3 rounds in 99.9% of the cycles over the entire range of parameters we considered; in contrast PRiPSS needed 2 stages of 2 rounds each for N up to 2,048, and needed 3 stages of 2 rounds each for $N = 4,096$.

While the choice of RiPSS and PRiPSS may differ with the primary consideration for efficiency, our results indicate that both schedulers perform far better than any other scheduler proposed in the literature for output-queuing, whether implemented directly, or emulated through SMS. In our experiments we also saw that memory utilization in RiPSS is excellent. Even though we did not explicitly measure memory balance for PRiPSS, we expect it to be similar, since the placement method in both RiPSS and PRiPSS is the same.

References

[1] N. Alon and J. 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. Aziz, A. Prakash, and V. Ramachandran. A near optimal scheduler for switch-memory-switch routers. In *ACM Symposium on Parallelism in Algorithms and Architectures*, San Diego, CA, June 2003.

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

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

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

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

[8] T. Feder, N. Megiddo, and S. Plotkin. A sublinear parallel algorithm for stable matching. In *Symposium on Discrete Algorithms*, 1994.

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

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

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

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

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

[14] 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.

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

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

[17] R. Ramaswami and K. Sivarajan. *Optical networks: a practical perspective.* Morgan-Kaufmann, 2001.

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

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

Appendix I
Detailed Analysis of PRiPSS

We now give a detailed analysis of the pipelined randomized scheduler based on the pipelined matching procedure in Section IV, for the case when $\epsilon > 0$ is an arbitrarily small constant.

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 $b = e^{\frac{(1-\delta)\epsilon}{2}}$.

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 Section IV (as are s_i, q_i and Q_i).

Theorem 1.1: 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 1.2: The number of unmatched inputs in the set $Q_i(T,t)$ after a execution of single iteration of pipelined matching procedure is no more than, $q_i(T,t)e^{\epsilon N/2^{i+1}q_i(T,t))}/\alpha$, w.h.p. in N, where $\alpha > 0$ is a constant independent of N.

Proof: First we bound the expectation. Let j be an input in $Q_i(T,t)$. Let $\eta(j)$ be the set of unmatched memories that can be matched to input j. Clearly $|\eta(j)| \geq \epsilon N + q_i(T,t)$.

Let C_m be the index of the input to which memory m sends a request. Thus if $m \in |\eta(j)|$, then $\Pr[C_m = j] = 1/(2^{i+1} \cdot q_i(T,t))$. Let $\mathbf{C} = (C_1, C_2 \ldots, C_M)$ and define the random variable $X_j(\mathbf{C})$ to be 1 if $\forall m. (C_m \neq j)$ and 0 otherwise. Informally $X_j(\mathbf{C})$ indicates that input j 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, $X_j(\mathbf{C}) = 1$ implies input j did not get a match in that round. Let $X(\mathbf{C}) = \sum_{j \in Q_i(T,t)} X_i(\mathbf{C})$ be the total number of unmatched inputs in $Q_i(T,t)$ at the end of the round. Then,

$$\begin{aligned} \mathrm{E}(X(\mathbf{C})) &= q_i(T,t)(1 - 1/2^{i+1}q_i(T,t))^{(\epsilon N + q_i(T,t))} \\ &\leq q_i(T,t)e^{-(1+\epsilon N/2^{i+1}q_i(T,t))}. \end{aligned}$$

Thus defining a martingale and using Azuma's inequality we can say that w.h.p. in N, number of unmatched inputs in the set $q_i(T,t)$ would be no more than $q_i(T,t)e^{-(\epsilon N/2^{i+1}q_i(T,t))}/\alpha$, where α is a suitable constant. ■

Lemma 1.3: 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 $s_i(T,t+1) \leq a + be^{-N\epsilon/(2^{i+1}b)}/\alpha$.

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 be matched. Otherwise, we will have at most $xe^{-N\epsilon/(x2^{i+1})}/\alpha$ inputs with i unmatched inputs that were also in $Q_i(T,t)$ (from Lemma 1.2). Let δ be the number of inputs that got matched in $Q_{i+1}(T,t)$ thus $q_i(T,t+1) \leq xe^{-N\epsilon/(2^{i+1}x)}/\alpha + \delta$ and $s_{i+1}(T,t+1) \leq y - \delta$. Therefore, $s_i(T,t+1) = s_{i+1}(T,t+1) + q_i(T,t+1) \leq y + xe^{-N\epsilon/(2^{i+1}x)}/\alpha$. Thus,

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

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 $\beta^g \leq \min(1/2, \epsilon/\ln 2)$. 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 1.3 for the next iteration it is not difficult to show that

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

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 1.4: 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 $Q_D(T,0)$ are 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 1.1 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 1.5: 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 1.1 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 PRiPSS.

Appendix II
Analysis of PRiPSS-1

Here we show that w.h.p. in N, every packet that arrived in cycle $T-D$ will be matched by PRiPSS-1 at the end of this cycle, and that it has the 'self-stabilizing' property.

At any cycle T, let $Q_i^0(T)$ be the set of inputs that have exactly i unmatched packets at the beginning of the cycle after arrivals and $Q_i^j(T)$ be the set of inputs with exactly i unmatched packets after phase j. Let $\phi = 50/49$

Definition 1: We say predicate P_j holds for a cycle T iff $\sum_{k=i+1}^{D} |Q_k^0(T)| \leq \phi^j \cdot N/2^i$ for all $0 < i < D$.

We will now argue that P_0 holds for almost all cycles and if it does not hold for a particular cycle, within a small number of cycles it will continue to hold again.

Theorem 2.1: Let the number of memory banks $M = (2+\epsilon)N$, for some $\epsilon > 0$. If P_0 holds for cycle T then P_0 holds for cycle $T+1$ w.h.p. in N. Furthermore, if for $j > 0$ P_j holds for cycle T then, P_{j-1} must hold for the cycle $T+1$ w.h.p. in N.

This will be proved through a sequence of lemmas.

Lemma 2.2: In Phase 3, if the number of current active packets that arrived in cycle $T-D$ is $O(N/\log N)$ then with high probability in N all of these packets are matched after Phase 3 is executed.

Proof: The result follows by viewing the set-up as an instance of the 'coupon collector problem'. ■

Lemma 2.3: For any set S of inputs that are participating in the *basic matching procedure* if $|S| \geq N^{0.6}$ then at most $0.4|S|$ inputs remain unmatched from the set S, w.h.p. in N.

Proof: By using a similar argument to that in [3] to prove Lemma 2.2, we can show that w.h.p. in N the number of unmatched inputs from the set S is at most $\kappa|S|/e$, where $\kappa > 1$ is a constant. Choosing $\kappa = 0.4e$, we get the result. ■

Let P_j hold for the cycle T. Now the set of inputs that will be in $Q_i^1(T)$ will correspond the inputs in $Q_i^0(T)$ that were not matched and the inputs from $Q_{i+1}^0(T)$ that were matched. Thus if $|Q_i^0(T)| \geq N^{0.6}$ we get,

$$\begin{aligned}\sum_{k=i}^{D} |Q_k^1(T)| &\leq \sum_{k=i+1}^{D} |Q_k^1(T)| + 0.4|Q_k^1(T)| \\ &= 0.4\sum_{k=i}^{D} |Q_k^1(T)| + 0.6\sum_{k=i+1}^{D} |Q_k^1(T)| \\ &= \phi^j 0.7N/2^i\end{aligned}$$

If $|Q_i^0(T)| < N^{0.6}$ then also for sufficiently large N we can show, $\sum_{k=i}^{D} |Q_k^1(T)| \leq \phi^j 0.7N/2^i$. By using a similar argument, we get

$$\sum_{k=i}^{D} |Q_k^2(T)| = \phi^j 0.49N/2^i = \phi^{j-1}N/2^{i+1}.$$

In the third phase only elements from $Q_D^2(T)$ be matched. Thus $Q_D^3(T) \subseteq Q_D^2(T)$, $Q_D^3(T) \cup Q_{D-1}^3(T) \subseteq Q_D^2(T) \cup Q_{D-1}^2(T)$ and $Q_i^2(T) = Q_i^3(T)$ for all $i < D-1$. Hence for $i < D-1$ we get,

$$\begin{aligned}|Q_D^3(T)| &\leq |Q_D^2(T)| \leq \phi^{j-1}N/2^{D+1} \\ |Q_D^3(T)| + |Q_{D-1}^3(T)| &= |Q_D^2(T)| + |Q_{D-1}^2(T)| \\ &\leq \phi^{j-1}N/2^D\end{aligned}$$

Thus we get,

$$\sum_{k=i}^{D} |Q_k^3(T)| = \sum_{k=i}^{D} |Q_k^2(T)| \leq \phi^{j-1}N/2^{i+1}$$

Now at the cycle $T+1$ at most one packet arrives for each input the for each input the number of unmatched packets increases by 1. Therefore the $\cup_{k=i}^{D} Q_k^0(T+1) \subseteq \cup_{k=i-1}^{D} Q_k^0(T)$. Hence for $i \geq 1$ we get $\sum_{k=i}^{D} |Q_k^0(T+1)| \leq \sum_{k=i-1}^{D} |Q_k^3(T)| \leq \phi^{j-1}N/2^i$.

For $j > 0$ and $i = 0$ we trivially get $\sum_{k=i}^{D} |Q_k^0(T+1)| \geq \phi^{j-1}N/2^i$.

Thus if $j = 0$ then P_0 holds for cycle $T+1$ and if $j > 0$ then P_{j-1} holds for cycle $T+1$ with high probability.

Theorem 2.4: If P_0 holds for a particular cycle T then our algorithm does not drop any packets that arrived on cycle $T-D$ in the current cycle.

Proof: When P_0 holds, $|Q_D^2(T)| \leq N/2^D = N/\log N$. Thus by Lemma 2.2 with high probability all the inputs in $Q_D^2(T)$ will be matched in phase 3. Since each input other than $Q_D^2(T)$ had at least one packet matched now, all the packets that arrived at cycle $T-D$ must be matched. ■

Similar to the proof of PRiPSS, we now argue that our algorithm has a self-stabilizing property in the sense that if goes to a bad state, in a small number of cycles it will recover from the bad state to a state where P_0 is again true and no drops occur.

Since $\phi^j N/2^D = N$ if $j = \log_\phi \log N$, it follows that $P_{\log_\phi \log N}$ is trivially true for any cycle. Thus, by the second assertion in Theorem 2.1, any time P_0 is not true for a cycle, within $\log_\phi \log N$ cycles it must hold true with high probability. Thus P_0 will hold in almost all cycles during the execution of PRiPSS-1, and no packets will be dropped.

APPENDIX III
ANALYSIS OF PRiPSS-2

We now sketch a proof that PRiPSS-2 places all packets with high probability in N with $D = c \cdot \log\log N$ stages of pipeline, for a suitable constant c. For this we observe that the constant 0.4 used in the statement of Lemma 2.3 could be replaced by a somewhat smaller constant, call it c', while maintaining the validity of the claim. Thus if we use a sufficiently small constant δ in PRiPSS-2, Lemma 2.3 can be proved for the case when the basic matching procedure is executed by each memory only with probability $(1-\delta)$. Thus P_0 will hold w.h.p. in N for PRiPSS-2. Now, if P_0 is true, then the number of unmatched packets that arrived D cycles earlier must be less than $N/\log^c N$. Thus an input that has such a packet must receive a request from any compatible memory with probability at least $\frac{\delta \cdot \log^c N}{N}$. Since there are at least $\epsilon \cdot N$ compatible memories, the probability of such an input not receiving a grant would be at most $(1-(\delta \cdot \log^c N)/N)^{\epsilon \cdot N} < e^{-(\epsilon\delta \cdot \log^c N)}$. Thus with high probability in N all such inputs must be matched in a single round.