A Quantitative Comparison of Scheduling Algorithms for Input-Queued Switches*

Nick McKeown　　Thomas E. Anderson

Abstract

This paper quantitatively evaluates several alternative approaches to the scheduling of cells in a high-bandwidth input-queued ATM switch. In particular, we compare the performance of three algorithms described previously: FIFO queueing, parallel iterative matching (PIM), maximum matching and two new algorithms: *iterative round-robin matching with slip* (SLIP) and *least-recently used* (LRU). For the synthetic workloads we consider, including uniform and bursty traffic, SLIP performs almost identically to the other algorithms. Cases for which PIM and SLIP perform poorly are presented, indicating that care should be taken when using these algorithms. But, we show that the implementation complexity of SLIP is an order of magnitude less than for PIM, making it feasible to implement a 32x32 switch scheduler for SLIP on a single chip.

1 Introduction

The past few years has seen increasing interest in arbitrary topology cell-based local area networks, such as ATM [5]. In these networks, hosts are connected together by an arbitrary graph of communication links and switches, instead of via a bus, as in Ethernet[20], or a ring, as in FDDI[4]. One reason for the popularity of arbitrary topology networks is that they offer a number of potential advantages relative to other approaches[2]: (i) aggregate throughput that can be much larger than that of a single link, (ii) the ability to add throughput incrementally as the workload changes by simply adding extra links and switches, (iii) improved fault tolerance by allowing redundant paths between hosts, (iv) and reduced latency because control over the entire network is not needed to insert data.

But to realize the potential advantages of arbitrary topology networks, a high performance switch is needed to take data arriving on an input link and quickly deliver it to the appropriate output link. (In this paper, we will only consider switches that deal in fixed-length packets, or cells, as in ATM.) A switch requires three functions: a switch fabric, scheduling to arbitrate when cells arrive on different inputs destined for the same output, and (optionally) queueing at inputs or outputs to hold those cells that lose the arbitration. A large number of alternative switch designs have been proposed with varying approaches to these three functions [2, 6, 8, 12, 15, 21, 22], each with their own set of advantages and disadvantages.

The goal of this paper is to make a quantitative comparison of scheduling policies for input-queued switches, considering both hardware complexity and switch performance. We observe that while it can be tempting to throw hardware at improving switch performance, that strategy may be counter-productive in practice, since simpler solutions can be easier to implement at high speed.

The longstanding view has been that input-queued switches are impractical because of poor performance. If FIFO queues are used to queue cells at each input, only the first cell in each queue is eligible to be forwarded.

*McKeown is supported by Pacific Bell, Bellcore and California State MICRO Program Grant 93-152. Tom Anderson, is supported by the Advanced Research Projects Agency (N00600-93-C-2481), the National Science Foundation (CDA-8722788), California MICRO, Digital Equipment Corporation, the AT&T Foundation, Xerox Corporation, and Siemens Corporation. He was also supported by a National Science Foundation Young Investigator Award and a Sloan Research Fellowhsip. The authors are both with the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, Berkeley, CA 94720. By email nickm@eecs.berkeley.edu

As a result, FIFO input queues suffer from *head of line (HOL) blocking* [14] – if the cell at the front of the queue is blocked, other cells in the queue cannot be forwarded to other unused inputs. Because HOL blocking has very poor performance in the worst case [18], the standard approach has been to abandon input queueing and instead use output queueing – by increasing the bandwidth of the internal interconnect, multiple cells can be forwarded at the same time to the same output, and queued there for transmission on the output link.

Our work is motivated by a different approach. HOL blocking can also be eliminated by using a separate queue for each output [1, 2, 14, 23], where any cell queued at an input is eligible for forwarding across the switch. The central problem with random-access input queues is the need for fast *scheduling* – quickly finding a conflict-free set of cells to be forwarded across the switch, such that each input is connected to at most one output, and vice versa. As observed in [2], this is equivalent to *matching* inputs to outputs [24].

This paper examines a number of scheduling algorithms for scheduling cells across an input-queued switch, including several proposed in the literature and two novel algorithms we have devised. We simulate the behavior of these algorithms on a variety of synthetic workloads, comparing them on queueing delay, fairness and burst-reduction. We also compare the hardware complexity of two of the more promising approaches.

Our study has two main results. First, we describe a novel switch scheduling policy, called SLIP, which performs as well as the best previously proposed algorithm. We find that there are some special workloads for which SLIP will perform better and some for which it will perform worse. SLIP, however, has the advantage of being much simpler to implement. We demonstrate that a SLIP scheduler for a moderate-scale switch ($\leq$ 32 inputs/outputs) can fit onto a single chip; this appears not to be the case with other approaches. Single chip implementations will be crucial if we are to be able to keep up with increasing link speeds – a 32 way ATM switch with 10 Gbit/sec links would need to make more than a billion scheduling decisions per second.

Second, we demonstrate that seemingly trivial changes in switch scheduling policy can have a large impact on performance. SLIP and LRU are very similar, except for the ordering of the entries in the schedules. This small difference can lead to very different performance characteristics.

The remainder of this paper discusses these issues in more detail. Section 2 describes the various switch scheduling policies which we consider. Section 3 outlines our methodology for comparing the policies. Section 4 considers hardware implementation complexity; Section 5 presents performance results and Section 6 summarizes our comparative study.

2 Alternative Algorithms

In this section, we describe the scheduling algorithms that we examine in this paper, outlining the key design issues for each algorithm. We defer until afterwards the quantitative comparison of performance.

We assume that the switch synchronously forwards cells from all inputs at the same time. In an input-queued switch without any internal speedup, an input can only send one cell into the switch fabric in a cell time, or slot. Likewise, each output can only receive one cell in a slot. A conflict-free match is a set of connections through the switch that does not violate these two requirements. Each algorithm has the same goal: to try and find a conflict-free match of inputs to outputs in a single cell time.

2.1 FIFO Scheduling

The simplest approach to managing queues and scheduling cells is to maintain a single FIFO queue at each input; the queues hold cells which have not yet been transmitted through the switch. To schedule connections for the next cell time, the scheduling algorithm examines the cells at the head of each FIFO queue. The problem is that two or more FIFO queues may have cells destined for the same output. In this case, the scheduling algorithm must select just one cell from among the contending inputs. This is necessary to meet our requirement that each output can only accept one cell per slot for an input-queued switch with no internal speed-up.

On its own, the above description isn't enough to specify how FIFO works. If there are two inputs both with cells

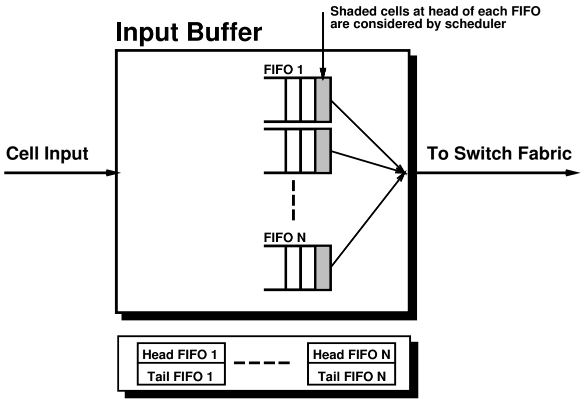


Figure 1: A single input queue containing N separate FIFOs managed within a single buffer memory.

for the same output, which input will be selected to send its cell first? One possibility would be to maintain a fixed priority and, for example, always select the lowest numbered input. This would be fast and simple to implement, but would be unfair among input-output pairs: in fact, it can be so unfair that one or more connections may be starved.

One option would be for the scheduling algorithm to *randomly* select among the contending inputs. This leads to a fair allocation among connections, but is hard to implement at high speed: the algorithm must perform a random number lookup uniformly over a varying set of inputs. Yet we could meet the same objective — starvation avoidance — with a rotating priority. This method has the advantage of not requiring a random number lookup, but we must vary the priority carefully to maintain fairness and avoid starvation. This choice between random and varying priorities is found in all of the approaches we consider.

Before considering other scheduling approaches, we need to examine a serious performance problem caused by maintaining FIFO queues at the inputs: *head of line (HOL) blocking* [14]. The problem is that when a cell is blocked at the head of an input queue (due to contention with a cell from a different input destined for the same output) it also blocks all the cells behind it in the queue *even though those cells may be destined for an output that is currently idle.* HOL blocking has been shown to limit the aggregate throughput of the switch to just 58% of its maximum for independent Bernoulli arrivals and destinations uniformly distributed over outputs [14]. With less benign traffic patterns, the situation can be much worse [18].

Several methods have been proposed for reducing HOL blocking [11, 14, 15, 21, 23], but a simple way described in [14, 15], to eliminate HOL blocking is to maintain a separate FIFO queue at each input for each output[1]. This is illustrated in figure 1. If cells for one output are blocked because the output is busy, the scheduling algorithm may select a cell for some other output. This method was implemented in [2], where they showed that the scheme is not only straightforward to build, but yields large performance gains.

Now that we have arranged the buffering to eliminate HOL blocking, we must change the connection scheduling algorithm: the algorithm must now consider the cells at the head of all N^2 input FIFOs and determine a pattern of conflict-free connections that will provide a high throughput. (The algorithm is an example of bipartite graph matching [24] and is also an example of the "rooks on a chessboard problem" [1]).

2.2 Maximum Matching

Ideally, we would like an algorithm that finds the maximum number of matches between inputs and outputs. This would provide the highest possible throughput in each slot for an input-queued switch.

[1]In practice, we would maintain a separate FIFO for each flow (in ATM terms, each virtual circuit). When a particular input-output connection is selected, we would then choose among those flows in a random, pre-determined or time-varying manner. But as we are only considering the scheduling of input-output connections in this paper, we shall assume that each input maintains only N output FIFOs.

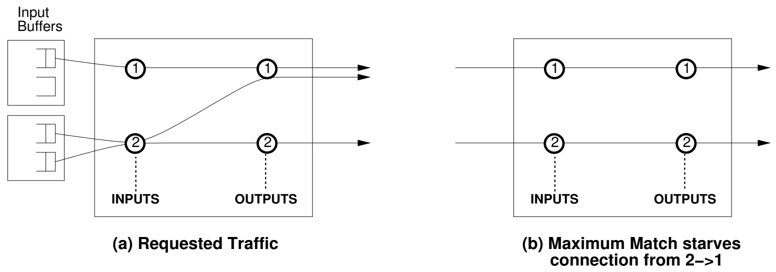


Figure 2: Starvation for a 2x2 switch using a Maximum Matching scheduling algorithm.

There are several algorithms for achieving a maximum match [7, 10, 16, 24], but these are not suitable for this application for two reasons. First, a maximum match can take a long time to converge and, second, they can lead to starvation of an input-output connection under certain traffic patterns. Referring to Figure 2 we see that even a simple traffic pattern can lead to starvation for a Maximum Matching algorithm. Input 2 will never be able to send cells to output 1 as this would lead to a match of only one connection.

We have a dilemma: the maximum throughput will always be achieved by a maximum match; but the maximum match can take too long to compute and will potentially starve some connections.

Each of the algorithms that follow attempt to achieve a maximal, not a maximum match, by iteratively adding connections to fill in the missing connections left by the previous iteration. Because the connections made in previous iterations may not be removed, this technique does not always lead to a maximum match. However, we shall see that it is possible to achieve a close approximation to the maximum for many traffic patterns. Furthermore, by introducing randomness or time-varying priorities, we shall see that starvation can be avoided.

2.3 Parallel Iterative Matching (PIM)

The first iterative algorithm that we shall describe is called Parallel Iterative Matching [2]. It uses *randomness* to (i) reduce the number of iterations needed, and (ii) avoid starvation.

PIM attempts to quickly converge on a conflict-free match in multiple iterations where each iteration consists of three steps. All inputs and outputs are initially unmatched and only thoses inputs and outputs not matched at the end of one iteration are eligible for matching in the next iteration. The three steps of each iteration operate in parallel on each output and input and are as follows:

1. Each unmatched input sends a request to *every* output for which it has a queued cell.
2. If an unmatched output receives any requests, it grants to one by *randomly* selecting a request uniformly over all requests.
3. If an input receives a grant, it accepts one by selecting an output among those that granted to this output.

By considering only unmatched inputs and outputs, each iteration only considers connections not made by earlier iterations.

Note that in step (2) above the independent output schedulers *randomly* select a request among contending requests. This has three effects: first, in [2], the authors show that each iteration will match or eliminate at least 3/4 of the remaining possible connections and thus the algorithm will converge to a maximal match in $O(\log N)$ iterations. Second, it ensures that all requests will eventually be granted. As a result, no input-output connection is starved of service. Third, it means that no memory or state is used to keep track of how recently a connection was made in the past. At the beginning of each cell time, the match begins over, independently of the matches that were made in previous cell times. Not only does this simplify our understanding of the algorithm, but it also makes analysis of the performance straightforward: there is no time-varying state to consider, except for the occupancy of the input queues.

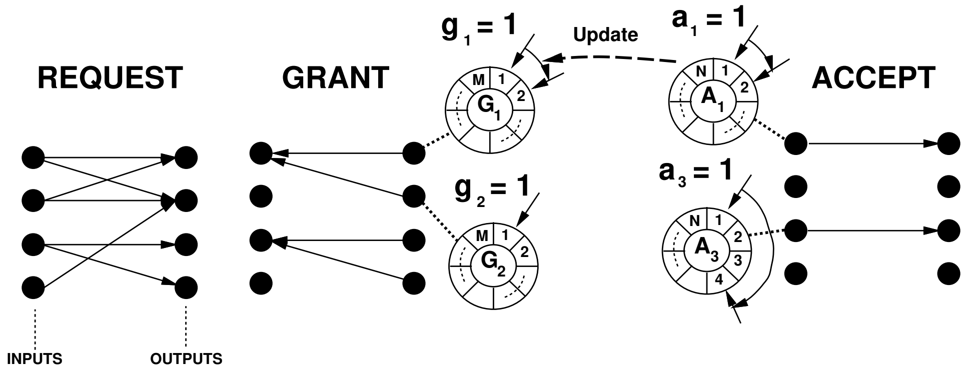


Figure 3: One iteration of the three phase SLIP scheduling algorithm.

But using randomness comes with its own problems. First, it is difficult and expensive to implement at high speed (we shall consider this more later) and second, it can lead to unfairness between connections. This will be discussed more when we present simulation results in section 5.

The two algorithms that we shall consider next (SLIP and LRU), avoid the use of randomness when making selections. Instead they use a *varying priority* based on state information that is maintained from one time slot to the next.

2.4 Iterative Round Robin Matching with Slip (SLIP)

SLIP, like PIM, is an iterative algorithm: the following three steps are iterated for an $N \times N$ switch:

1. Each unmatched input sends a request to *every* output for which it has a queued cell.
2. If an unmatched output receives any requests, it chooses the one that appears next in a fixed, round-robin schedule starting from the highest priority element. The output notifies each input whether or not its request was granted. The pointer g_i to the highest priority element of the round-robin schedule is incremented (modulo N) to one location beyond the granted input if and only if the grant is accepted in step 3 of the first iteration. The pointer is not incremented in subsequent iterations.
3. If an input receives a grant, it accepts the one that appears next in a fixed, round-robin schedule starting from the highest priority element. The pointer a_i to the highest priority element of the round-robin schedule is incremented (modulo N) to one location beyond the accepted output.

An example of one iteration of the three phases is illustrated in Figure 3. In the example, input 1 has one or more cells for outputs {1,2}, input 2 has one or more cells for outputs {1,2} and so on. The grant schedules are shown for outputs 1 and 2. The accept schedules are shown for inputs 1 and 3. At the end of the time slot, g_1 and a_1 are incremented to 2 and a_3 is incremented to 4; input 2 will match output 2 in the next iteration, but this will have no effect on the priorities.

So, each output scheduler decides among the set of ordered, competing requests using a rotating priority. When a requesting input is granted and the input accepts that grant, the input will have the lowest priority at that output in the next cell time. Also, whatever input has the highest priority at an output will continue to be granted during each successive time slot until it is serviced. This ensures that a connection will not be starved: the highest priority connection at an output will always be accepted by an input in no more than N cell times.

Moving the pointers not only prevents starvation, it tends to *desynchronize* the schedulers. Each of the outputs that matched in the previous time slot will have a different highest priority input. Thus they will each grant to different inputs. Consider an example in which two inputs are both requesting the same two outputs. Initially, both outputs may grant to the same input; in that case only one connection will be made in the first iteration.

Input Choice	Output Choice		
	Random	Rotating	LRU
Random	PIM [2]		
Rotating		SLIP	
LRU			LRU

Table 1: Alternative Selection methods for Input and Output Schedulers.

The successful output will increment its pointer and in the next cell time, the outputs will no longer contend: one will have moved on to grant to another input and the other will grant to the same input as before. This leads to a better match in the first iteration of the next cell time. This is because the output schedulers have become desynchronized (or "slipped") with respect to each other[1]. We shall see that this leads to high performance, even for a single iteration of SLIP.

Because of the round-robin movement of the pointers, we can also expect the algorithm to provide a fair allocation of bandwidth among competing connections and to be burst-reducing. The burst-reduction is simplest to understand under high load when all input queues are occupied: the algorithm will visit each competing connection in turn, so that even if a burst of cells for the same output arrives at the input, the burst will be spread out in time if there is competing traffic. We will examine fairness and burstiness further when we present our simulation results in Section 5.

2.5 Least Recently Used (LRU)

When a output scheduler in SLIP successfully selects an input, that input becomes the *lowest priority* in the next cell time. This is intuitively a good characteristic: the algorithm should least favor connections that have been served recently. But which input should now have the *highest* priority? In SLIP, it is the next input that happens to be in the schedule. But this is not necessarily the input that was served *least* recently. By contrast, LRU gives highest priority to the least recently used and lowest priority to the most recently used.

LRU is identical to SLIP except for the ordering of the elements in the scheduler list: they are no longer in ascending order of input number but rather are in an ordered list starting from the least recently to most recently selected. If a grant is successful, the input that is selected is moved to the end of the ordered list. Similarly, an LRU list can be kept at the inputs for choosing among competing grants.

Thus, for each of the iterative algorithms that we have described, there are several alternatives for choosing among contending inputs and outputs. Table 1 shows the alternative ways of making selections at the input (accept) and output (grant) schedulers; we only examine the symmetric form of SLIP and LRU in which the same scheduling scheme is used at the input and output, but the other combinations are possible.

3 Methodology

To evaluate the performance of the iterative PIM, SLIP and LRU scheduling algorithms, we have compared them to each other and to FIFO queueing, perfect output queueing and maximum matching. FIFO queueing is used as a lower bound: it is the simplest to implement, but suffers from HOL blocking. On the other extreme, perfect output queueing assumes not queueing due to contention for the switch fabric; only queueing caused by contention for the output links. This provides an upper bound on the performance that any switch can attain; unfortunately, output queueing is impractical at high speeds due to internal speed-up requirements. Maximum

[1] Desynchronization would in general be improved if we allowed the scheduler pointers to move in every iteration of the algorithm. Unfortunately, this leads to the pointers moving past a higher priority connection that was not made in the first iteration. This can lead to starvation.

matching indicates the best throughput that could be achieved in a given time slot for an input queued switch.

For each of the iterative algorithms we need to decide how many iterations to perform within a single cell time. For extremely high bandwidth switches, limited only by the link technology, there may only be time to perform a single iteration of the scheduling algorithm. For this reason, we compare the algorithms for a single iteration.

In other applications, there may be time for more than one iteration, particularly if the scheduler can be integrated on a single chip. In [2], the authors were able to achieve four iterations of the PIM algorithm within a single 53-byte ATM cell time (420ns) at link speeds of 1Gbits/sec. For comparison, we also consider the PIM, SLIP and LRU algorithms for four iterations.

3.1 Bases for Comparison

There are a number of factors that would lead a switch designer to select one scheduling algorithm over another. First, it must be simple to implement. A more complex implementation is not only more expensive in area and power, but is likely to provide less performance—particularly if each iteration requires off-chip communication. Second, the algorithm should provide high throughput and avoid starvation for any connection pattern. This is because real network traffic is rarely uniformly distributed over inputs and outputs. Fourth, the algorithm should provide high throughput for bursty traffic. Real network traffic is highly correlated from cell to cell [17] and the scheduling algorithm should not be unduly affected by the burstiness. However, we can expect all algorithms to perform worse under more bursty traffic. This is not necessarily because of the algorithm, but rather because of the temporary increase in queue length that occurs when the arrival rate momentarily exceeds the service rate. Last, it would be desirable for the algorithm to reduce the burstiness of the traffic as it passes through the switch. Burst reduction at one switch will help to reduce the queueing delay through subsequent switches.

In the next two sections, we will first consider the implementation complexity of the algorithms before considering their relative performance. Using simulation, we examine the performance for uniform and asymmetric work loads as well as for non-bursty and bursty traffic.

4 Complexity Comparison

For PIM, SLIP and LRU, the scheduler chip needs to know the status of all the input queues (empty/non-empty). This takes N^2 bits of state within the scheduler. To maintain this state for unicast traffic, each input must indicate $\log N$ bits to the scheduler per cell time: whether or not a queue has become empty or non-empty. When the scheduler completes its matching decision, it informs each input which output, if any, it can send a cell to.

A PIM scheduler is difficult to implement—random selection is an expensive operation, particularly for a variable number of input requests. In the implementation described in [2], the scheduler filled approximately 1.5 Xilinx 3190 [27] devices for each port [25]. For a 16x16 switch, this is approximately 64,000 *complex* gates. If we assume that each gate is approximately eight 2-input gate equivalents this totals 512,000 simple gates.

A SLIP scheduling chip contains $2N$ schedulers, each one is an N input priority encoder with a programmable priority level. A schematic is shown in figure 4. Using the "misII" tools from the Berkeley Octtools VLSI design package, we estimate (without any attempt to optimize the design) that the scheduler for a 16x16 switch will require approximately 52,000 2-input gate equivalents. This is an order of magnitude less complex than for PIM for the same sized switch. Further, since a custom VLSI chip should have room for at least 200,000 gates, this indicates that a single chip version of SLIP would be feasible— but not a single chip version of PIM.

Although we did not design a full implementation for LRU, it is much more complex than SLIP, requiring an ordered set of N registers each of width $\log N$, and N comparators to determine which is the highest priority that was selected.

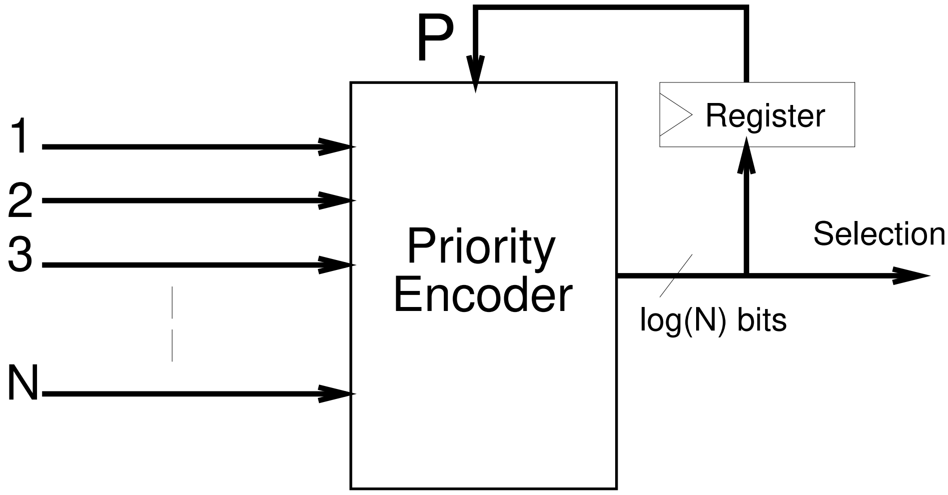


Figure 4: A SLIP scheduler is a programmable priority encoder. The priority level P indicates the lowest priority. (P+1) mod N is the highest priority.

5 Performance Results

In this section, we compare the performance results by simulation for PIM, SLIP and LRU against FIFO, maximum matching and output queueing for a 32x32 switch. We start in section 5.1 by considering the performance of one and four iterations of each algorithm with a Bernoulli arrival process (i.e. in each cell time, a random choice is made for each input whether a cell is to arrive in that time slot) and destinations uniformly distributed over outputs. This is followed by a discussion of some bad performance cases for PIM and SLIP under asymmetric work loads. Finally, we consider performance for bursty traffic and examine the burst reduction properties of the algorithms.

5.1 Performance Under Uniform Traffic

5.1.1 A Single Iteration

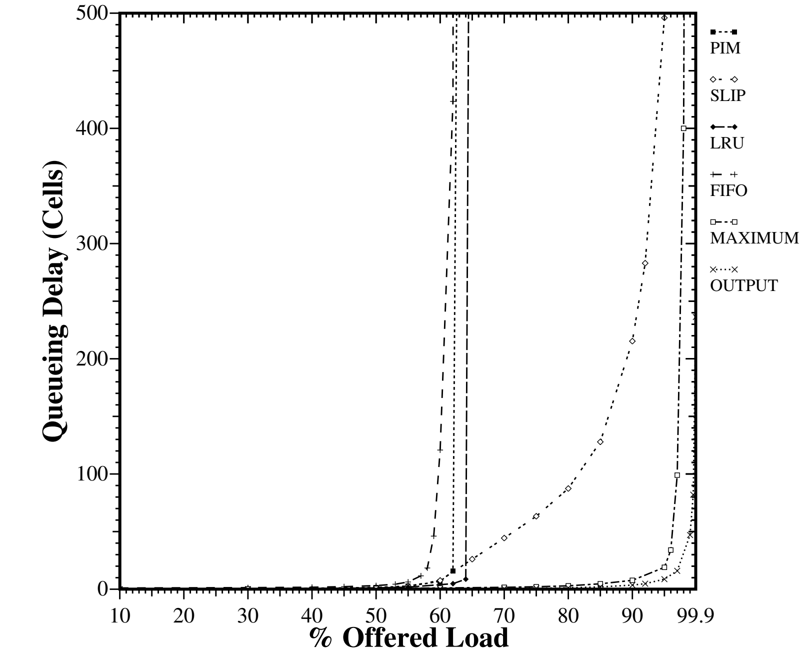
We consider first the case where cells arrive as a Bernoulli process with cell destinations distributed uniformly over all outputs. Figure 5(a) shows the average queueing delay (measured in cells) versus the average offered load for PIM, SLIP and LRU with one iteration contrasted with FIFO, maximum matching and perfect output queueing.

Because PIM, SLIP and LRU all avoid HOL blocking we expect them to achieve a higher maximum offered load than FIFO queueing. This is the case: PIM, SLIP and LRU achieve maximum offered load of 63%, 100% and 64% respectively, whereas FIFO queueing achieves slightly more than 58%[1]. PIM is limited to less than 63% because this is the probability that any given input will be able to connect to an output when all its input queues are occupied. The probability that no output will grant to a particular input is $((N-1)/N)^N$ which, for $N = 32$, limits the probability that a connection will be made to 63%[2].

SLIP can deliver throughput asymptotic to 100% of each link, but with a much higher average queueing delay than for either maximum matching or perfect output queueing. SLIP achieves the high offered load because of the *desynchronization* discussed in section 2.4. If all the input queues become non-empty, then the schedulers "slip" into a configuration that is 100% efficient until the backlog is cleared. This explains why SLIP has a higher

[1] FIFO is limited to 58% for a switch with an infinite number of inputs and outputs. For finite $N = 32$ the maximum offered load is slightly higher than 58%.

[2] As $N \to \infty$, maximum offered load for PIM with one iteration will tend to $1 - 1/e = 63.2\%$.



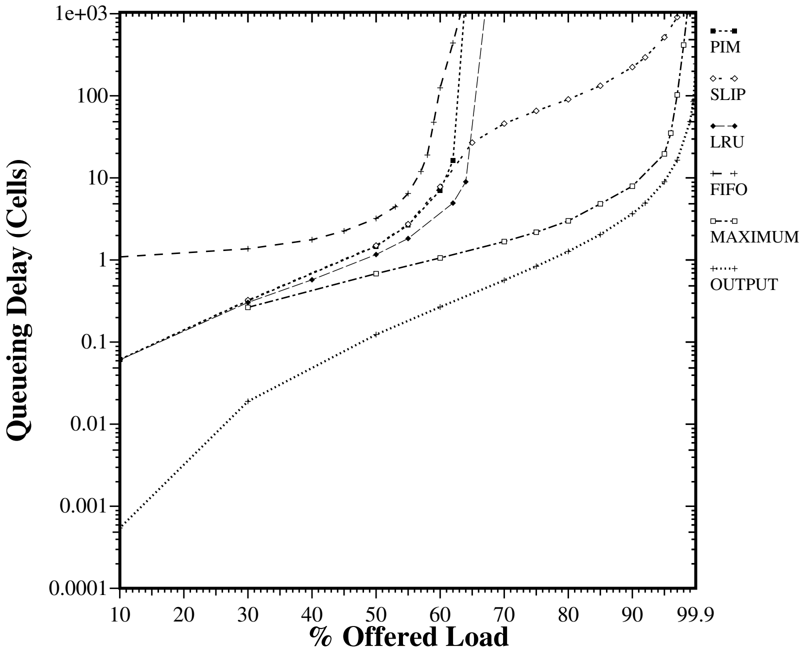


Figure 5: (a)Delay vs. utilization performance of PIM, SLIP, LRU, FIFO, for a single iteration with uniform workload and Bernoulli arrivals. (b)Log(Delay) vs. utilization.

queueing delay than maximum matching: it takes multiple cell times for the schedulers to slip into an efficient schedule.

We might have expected LRU to perform as well as SLIP. But as we can see, it performs significantly worse. This is because the *desynchronization* of the schedulers does not happen as effectively as it does for SLIP. Each schedule can become re-ordered at the end of each cell time which, over many cell times, tends to lead to a random ordering of the schedules. This in turn leads to a high probability that the pointers at two or more outputs will point to the same input: the same problem encountered by PIM. This explains why the performance for PIM and LRU are very similar for the single iteration case.

The highest throughput attainable per time slot for input queueing is illustrated by the maximum matching algorithm. This still leads to higher queueing delay than for perfect output queueing. With output queueing a cell is only competing for service with cells at the same output, whereas for input queueing a cell must compete with cells at the same input *and* output.

To clarify the difference between these algorithms over the whole utilization range, figure 5(b) plots the logarithm of the latency versus the utilization. This shows that PIM and SLIP are indistinguishable for a single iteration under low load, but separate as PIM becomes saturated. At approximately 70% utilization, the rate of increase of SLIP with utilization decreases. We do not have a good explanation for this effect, but it appears to be related to the onset of desynchronization at higher utilizations.

5.1.2 Four Iterations

Figure 6 shows the average queueing delay (measured in cells) versus the average utilization for PIM, SLIP and LRU with *four* iterations and the same traffic pattern as before. We chose four iterations as this was the number of iterations performed in a single slot in the implementation of PIM [2]. There is a large improvement for PIM, SLIP and LRU for multiple iterations and their performance is now almost indistinguishable. All three algorithms are now significantly better than FIFO due to the elimination of HOL blocking and all approach the performance to maximum matching. They still, of course, perform worse than output queueing: contention still exists at both inputs and outputs.

In summary, it appears that for a single iteration, the determinism of SLIP outperforms PIM and LRU due to the desynchronization of its output schedulers. However, after just four iterations for a 32x32 switch, all three algorithms achieve almost maximum matching and become indistinguishable.

5.2 Performance Under Asymmetric Workload

Real traffic is not usually uniformly distributed over all outputs, but is asymmetric. A common example of this would be a client-server work load in which a large number of clients communicate with a small number of servers. Although we cannot simulate all such work loads, we can highlight characteristics of each algorithm under certain patterns.

We have already seen in Figure 2 that maximum matching can be so unfair under an asymmetric work load that some connections may receive no bandwidth at all. Although PIM, SLIP and LRU will not starve connections, all can perform badly under asymmetric work loads. Figure 7(a) is a workload for which PIM will allocate bandwidth *unfairly*. Ideally, every connection would receive 1/2 of a link bandwidth. However, in PIM if the input queues are non-empty, input 2 will be able to send cells on connections to outputs 1 and 2 *less often* than inputs 1 and 3. The algorithm tends to discriminate against inputs that have contention. The figure shows the actual bandwidth allocation for PIM with 2 or more iterations. With a single iteration, the values will be slightly different, but PIM is still unfair. It is straightforward to construct more complex examples for which PIM will allocate bandwidth even more unfairly.

On the other hand, SLIP will allocate bandwidth fairly for this example: independent of the number of iterations. Whatever the starting conditions, so long as the input queues are occupied, SLIP will slip into a round-robin schedule providing each connection with exactly 1/2 of the link bandwidth. In fact, we have not been able to construct an example where SLIP allocates bandwidth unfairly between connections. But SLIP has a different

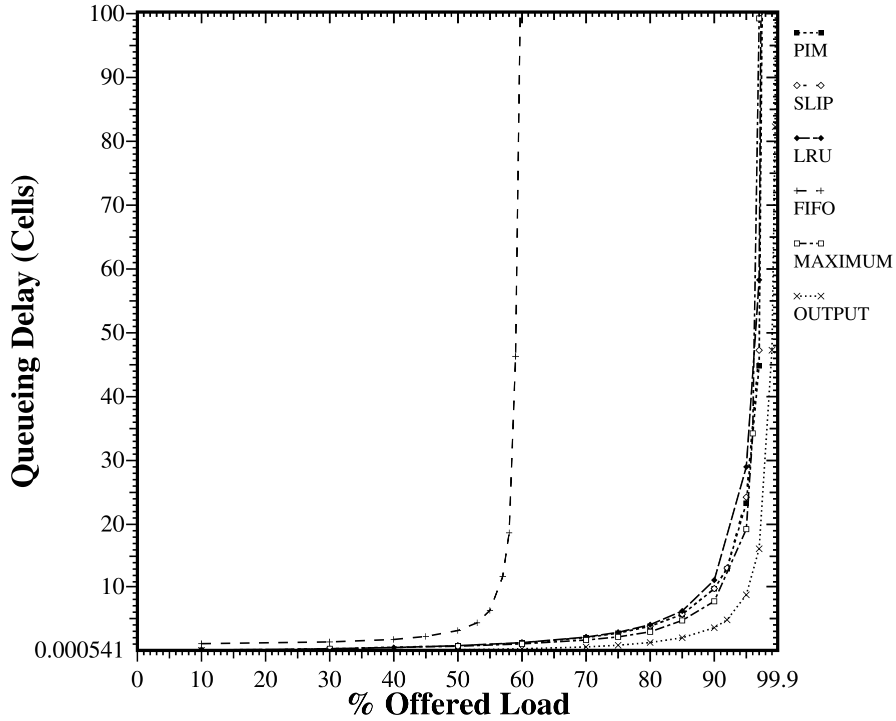


Figure 6: Delay vs. utilization performance for four iterations of PIM, SLIP, LRU, FIFO, maximum matching and output queueing with uniform workload and Bernoulli arrivals.

problem that is illustrated in Figure 7(b). In this example, if the schedulers start synchronized, they will visit each connection in turn allocating just two connections in each cell time (although up to $N-1$ connections could be made in more iterations). So, as a result of servicing each connection fairly and in turn, the algorithm does not slip into an efficient schedule with a high throughput. On the other hand, a single iteration of PIM will match about $(N-1)/2$ connections. Although this workload is poor for SLIP with a single iteration, further iterations will efficiently find other connections. We suspect that example connection patterns exist for which multiple iterations of SLIP will perform poorly, although we have as yet not been able to find any.

5.3 Performance Under Bursty Traffic

Real traffic is not only asymmetric—it is also bursty. Many ways of modeling bursts in network traffic have been described [9, 13, 3, 17]. Recently, Leland et al. have demonstrated that measured network traffic is bursty at every level and that, contrary to popular belief, burstiness in Ethernet networks typically *intensifies* as the number of active traffic sources increases [17]. So it is very important to understand the performance of switches in the presence of bursty traffic, whether it is a small private switch with a small number of active users, or a large public switch with many thousands of aggregated flows.

We illustrate the effect of burstiness on PIM, SLIP and LRU using a simple packet train model similar to the introduced in [13]. The source alternately produces a burst (train) of full cells (all with the same destination) followed by one or more empty cells. The bursts contain a geometrically distributed number of cells. The model contains two parameters: the average burst length and the overall average utilization of the input link.

Figure 8 illustrates the performance of the different algorithms for bursty traffic. In this example, we use an average burst length of 32 cells and then vary the utilization. Although the performance of all the algorithms will degrade as the average burst length is increased, we found that they all degrade in a similar manner. An average burst length of 32 cells is representative of our findings.

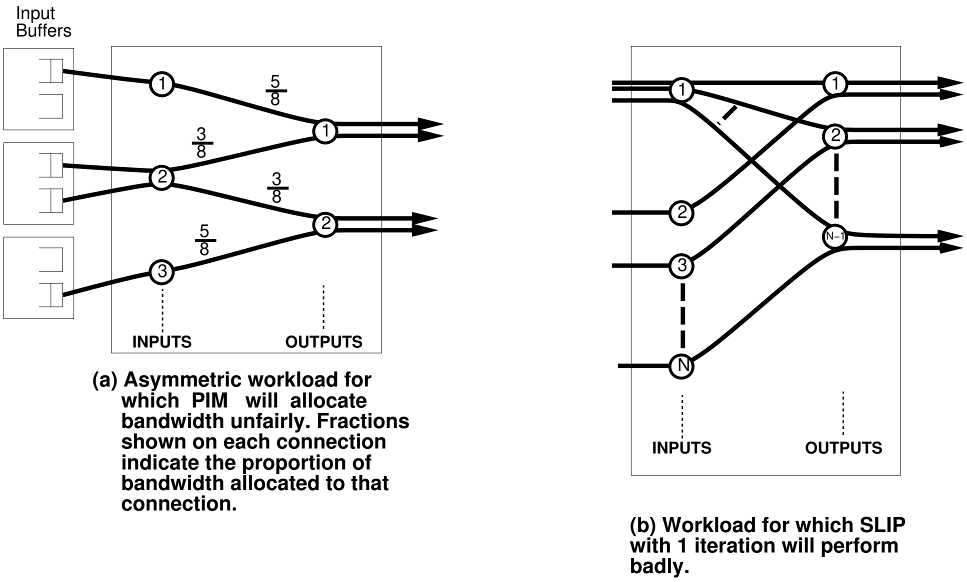


Figure 7: Examples of (a) unfairness for PIM and (b) poor performance for SLIP.

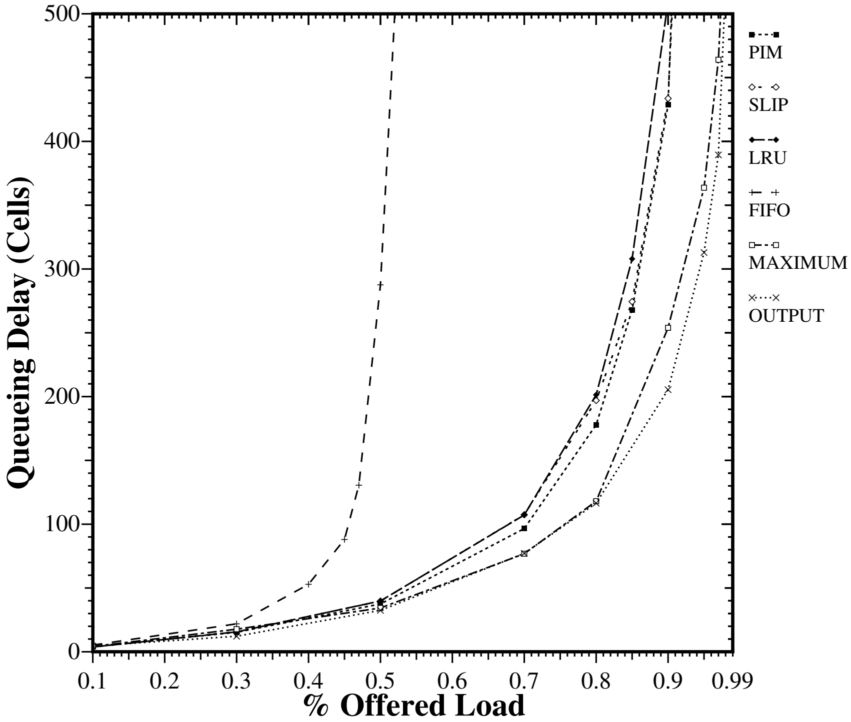


Figure 8: Delay vs. utilization performance for four iterations of PIM, SLIP, LRU, FIFO, maximum matching and output queueing with uniform workload and bursty traffic.

PIM, SLIP and LRU are shown for four iterations and are once again compared to FIFO, maximum matching and output queueing. The three most important characteristics of this graph are: (i) PIM, SLIP and LRU are again indistinguishable, as they were for non-bursty traffic, (ii) maximum matching performs similarly to output queueing, but (iii) all PIM, SLIP and LRU are all noticeably worse than maximum matching.

Once again, the three iterative algorithms achieve a very similar number of matches in four iterations. It appears that even for bursty traffic, connections not made in the first iteration of each algorithm are effectively and similarly filled in by subsequent iterations.

The similarity of maximum matching to output queueing indicates that the performance for bursty traffic is not heavily influenced by the queueing policy. Recall that for maximum matching, conflicts occur at both inputs *and* outputs, whereas for output queueing they occur only at the outputs. Burstiness tends to concentrate the conflicts on outputs rather than inputs: each burst contains cells destined for the same output and each input will be dominated by a single burst at a time. As a result, the performance is limited by output contention.

5.3.1 Burst Reduction

It should come as no surprise that burstiness increases queueing delay: this is generally the case for queueing systems [26]. Comparing Figure 8 with Figure 5(a) we indeed see the serious effect of burstiness on switch performance. In addition to the performance of a single switch for bursty traffic, it is important to consider the effect that the switch has on other switches downstream. If the switch *decreases* the burstiness of the traffic, then it will improve the performance of its downstream neighbor. We next examine the burst-reduction performance of each algorithm.

Figure 9 shows the burstiness at the switch outputs when the inputs receive bursty traffic with an average burst length of 32 cells. We define the burst length at an output as the number of contiguous full cells that came from the same input. We ignore all empty cells: the arrival of an empty cell at an output does *not* terminate the burst; we do this to make the metric less sensitive to offered load.

Our results indicate that PIM, SLIP and LRU will all reduce the average burst length, and will tend to be more burst-reducing as the utilization increases. This is because the probability of switching between multiple connections increases as the utilization increases. It is quite apparent, however, that the more deterministic algorithms, LRU and SLIP, will be more burst reducing than the random algorithms, PIM, FIFO and maximum matching. In fact, the SLIP and LRU algorithms with a single iteration will reduce the average output burst length to only one cell when the utilization is 70% or above.

It is also the case that for the iterative algorithms, output burst length *increases* with the number of iterations. Maximum matching leads to the longest average burst length. This is because for the duration of a burst, the pattern of traffic at the inputs does not change much between cell times. The maximum match will therefore not change much between cell times and so the same connection pattern may be selected several cell times in a row.

In order to understand the real effect of burst length reduction, we should consider the queueing delay and queue occupancy through multiple switches. This will be the subject of further work.

So, to summarize the effect of burstiness on the different algorithms, we find that PIM, SLIP and LRU perform similarly for bursty traffic, but much less effectively than maximum matching. Maximum matching performs similarly to output queueing because the conflicts are concentrated at the outputs. SLIP and LRU will tend to reduce output burst length more than PIM because of their similarity to round-robin scheduling. We plan to study the effect of burst length reduction in the future.

6 Conclusions

We have described and compared three realizable iterative algorithms, PIM, SLIP and LRU, for scheduling connections in an input-queued switch. SLIP and LRU differ from PIM by allocating connections according to priorities which vary according to the traffic pattern.

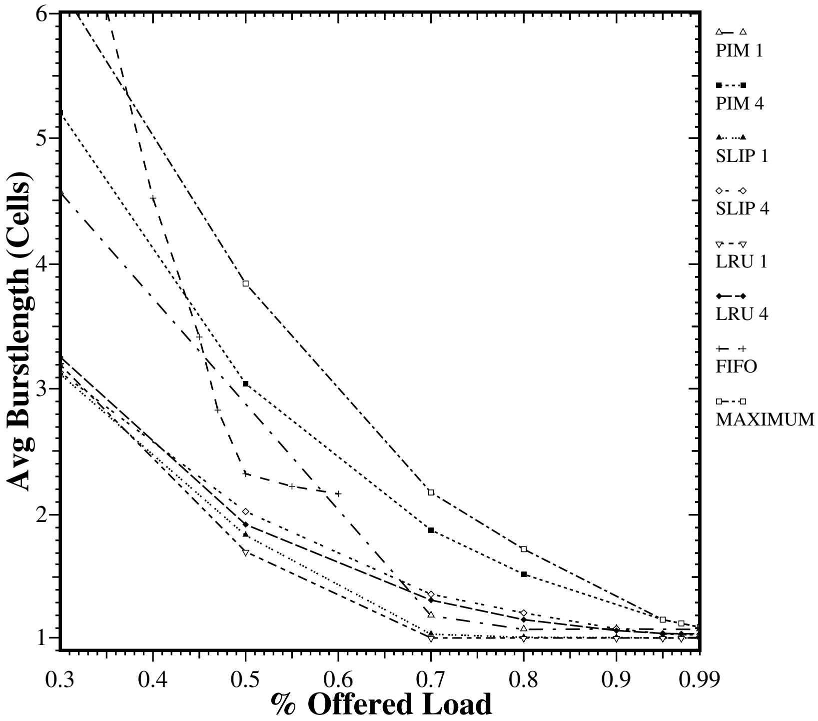


Figure 9: Average output burst length vs. utilization for PIM, SLIP, LRU, FIFO and maximum matching queueing with bursty traffic with a mean burst length of 32 cells.

Simulation results indicate that SLIP will significantly outperform both PIM and LRU for a single iteration with a uniform workload and Bernoulli arrivals. But as the number of iterations increases, the difference between the algorithms diminishes. With four iterations, for a 32x32 switch, all three algorithms do almost as well as the maximum for input-queued switches, and their performance becomes indistinguishable. Furthermore, we show that for multiple iterations, the algorithms perform almost identically for bursty traffic.

The algorithms differ considerably in the presence of asymmetric workloads. Workloads exist for which PIM will allocate bandwidth unfairly between connections, for any number of iterations. This appears not be the case for SLIP with a single iteration. On the other hand, we can construct workloads for which SLIP with a single iteration will only achieve a low throughput. Although we have not been able to find any, we suspect SLIP may perform badly on some workload, with multiple iterations. The designer of such a scheduler should be aware of this potential problem. It is our intention to study the performance of SLIP further.

We also observe that all the algorithms we considered tend to reduce burst length at the output of the switch. This is particularly the case for the SLIP and LRU algorithms. The impact that this will have on downstream switches will be the subject of further work.

Finally, and perhaps most importantly, we estimate that SLIP is an order of magnitude less complex to implement than PIM; LRU appears to be more complex to implement than SLIP with no performance gain. We believe that it is feasible to implement the scheduler for a 32x32 switch on a single chip using the SLIP algorithm. The consequence is that SLIP should be more scalable than PIM as link speeds increase in the future.

References

[1] Ali, M., Nguyen, H. "A neural network implementation of an input access scheme in a high-speed packet switch," *Proc. of GLOBECOM 1989*, pp.1192-1196.

[2] Anderson, T., Owicki, S., Saxe, J., and Thacker, C. "High speed switch scheduling for local area networks," *ACM Trans. on Computer Systems.* Nov 1993, pp. 319-352.

[3] Anick, D., Mitra, D., Sondhi, M.M., "Stochastic theory of a data-handling system with multiple sources," *Bell System Technical Journal 61,* 1982, pp.1871-1894.

[4] ANSI. "Fiber distributed data interface (FDDI). Token ring media access control(MAC)," *ANSI Standard X3.139*, American National Standards Institute, Inc., New York.

[5] The ATM Forum. "The ATM Forum user-network interface specification, version 3.0, " *Prentice Hall Intl.,* New Jersey, 1993.

[6] Eng, K., Hluchyj, M., and Yeh, Y. "Multicast and broadcast services in a knockout packet switch," *INFOCOM '88*, 35(12) pp.29-34.

[7] Even, S., Tarjan, R.E. "Network flow and testing graph connectivity" *SIAM J. Comput., 4* (1975), pp.507-518.

[8] Giacopelli, J., Hickey, J., Marcus, W., Sincoskie, D., Littlewood, M. "Sunshine: A high-performance self-routing broadband packet switch architecture," *IEEE J. Selected Areas Commun.*, 9, 8, Oct 1991, pp.1289-1298.

[9] Heffes, H., Lucantoni, D. M., "A Markov modulated characterization of packetized voice and data traffic and related statistical multiplexer performance," *IEEE J. Selected Areas in Commun.*, 4, 1986, pp.856-868.

[10] Hopcroft, J.E., Karp, R.M. "An $O(n^{5/2})$ algorithm for maximum matching in bipartite graphs," *Society for Industrial and Applied Mathematics J. Comput., 2* (1973), pp.225-231.

[11] Huang, A., Knauer, S. "Starlite: A wideband digital switch," *Proc. GLOBECOM '84* (1984), pp.121-125.

[12] Hui, J., Arthurs, E. "A broadband packet switch for integrated transport," *IEEE J. Selected Areas Commun.*, 5, 8, Oct 1987, pp 1264-1273.

[13] Jain, R., Routhier, S. A., "Packet Trains: measurements and a new model for computer network traffic," *IEEE J. Selected Areas in Commun.*, 4, 1986, pp.986-995.

[14] Karol, M., Hluchyj, M., and Morgan, S. "Input versus output queueing on a space division switch," *IEEE Trans. Comm*, 35(12) (1987) pp.1347-1356.

[15] Karol, M., Eng, K., Obara, H. "Improving the performance of input-queued ATM packet switches," *INFOCOM '92*, pp.110-115.

[16] Karp, R., Vazirani, U., and Vazirani, V. "An optimal algorithm for on-line bipartite matching," *Proc. 22nd ACM Symp. on Theory of Computing*, pp.352-358, Maryland, 1990.

[17] Leland, W.E., Willinger, W., Taqqu, M., Wilson, D., "On the self-similar nature of Ethernet traffic" *Proc. of Sigcomm*, San Francisco, 1993, pp.183-193.

[18] Li, S.-Y., "Theory of periodic contention and its application to packet switching" *Proc. of INFOCOM 1988*, pp.320-325.

[19] McKeown, N., Varaiya, P., Walrand, J., "Scheduling Cells in an Input-Queued Switch," *Electronics Letters*, Vol. 29, No.25 pp.2174-2175.

[20] Metcalfe, R., Boggs, D. "Ethernet: Distributed packet switching for local computer networks," *Communic. ACM*, 19, 7, July 1976, pp.395-404.

[21] Obara, H. "Optimum architecture for input queueing ATM switches," *Elect. Letters*, 28th March 1991, pp.555-557.

[22] Obara, H., Okamoto, S., and Hamazumi, Y. "Input and output queueing ATM switch architecture with spatial and temporal slot reservation control" *Elect. Letters*, 2nd Jan 1992, pp.22-24.

[23] Tamir, Y., Frazier, G. "High performance multi-queue buffers for VLSI communication switches," *Proc. of 15th Ann. Symp. on Comp. Arch.,* June 1988, pp.343-354.

[24] Tarjan, R.E. "Data structures and network algorithms," *Society for Industrial and Applied Mathematics,* Pennsylvania, Nov 1983.

[25] Thacker, C., *Private communication.*

[26] Wolff, R.W. "Stochastic modeling and the theory of queues," *Prentice Hall Intl.*, New Jersey, 1989.

[27] Xilinx, Inc. "Xilinx: The programmable gate array data book" *Xilinx, Inc.*, 1991.