Stress Resistant Scheduling Algorithms for CIOQ Switches

Prashanth Pappu, Jonathan Turner

Department of Computer Science

Washington University in St. Louis, MO, USA

{prashant,jst}@arl.wustl.edu

Abstract

Practical crossbar scheduling algorithms for CIOQ switches such as PIM and i-SLIP, can perform poorly under extreme traffic conditions, frequently failing to be work-conserving. The common practice of evaluating crossbar scheduling algorithms according to the packet delay under random admissible traffic tends to obscure significant differences that affect the robustness of different algorithms when exposed to extreme conditions. On the other hand, algorithms such as LOOFA with provably good worst-case performance, don't lend themselves readily to high performance implementation. We advocate evaluating crossbar scheduling algorithms using targeted stress tests *which seek to probe the performance boundaries of competing alternatives. Appropriately designed stress tests can reveal key differences among algorithms and can provide the insight needed to spur the development of better solutions.*

In this paper, we introduce the use of stress testing for crossbar scheduling and use it to evaluate the performance of PIM, i-SLIP and LOOFA. Our results show that PIM and i-SLIP need large speedups in order to perform well on stress tests, while LOOFA can deliver excellent performance, even for speedups less than 1.5. We then develop improved versions of PIM and i-SLIP, which take output queue lengths into account, making them much more robust. We also develop an algorithm which closely approximates the behavior (and performance) of LOOFA, but which admits a straightforward, high performance hardware implementation.

1 Introduction

The conceptual simplicity of an output queued router makes it an ideal candidate for router architecture design. An output queued router with N input and output ports, buffers packets only at the outputs. When two or more packets destined to the same output, arrive simultaneously at different input ports, they are transferred immediately to the output queue to avoid packet loss. Hence, the switching fabric of an output queued router needs to be scaled with the number of ports N. The increase in the link rates of individual ports and also the total number of ports supported on a single switch makes the task of scaling the switch fabric with the number of ports infeasible. Hence, most scalable switches employ some form of both input and output queuing. This Combined Input and Output Queuing (CIOQ) lets us use lower speed switching fabrics without any packet loss. When two or more packets contend to go to the same output, some of them are temporarily held in the input queues before being transmitted to the outputs. In typical CIOQ switches, the *speedup* of the switching fabrics over individual link rates is a small constant.

The inputs in CIOQ switches maintain separate queues for each output (called *Virtual Output Queues (VOQ)*) and use a scheduling algorithm to determine which cells from various inputs are transferred to outputs in a given cycle. The use of VOQs helps alleviate the problem of performance degradation due to head-of-line blocking [1]. The objective of the scheduling algorithms used in these swtiches is to approximate the work conserving properties of a pure output queued switch. A number of *stability results* have been proved regarding the performance of scheduling algorithms under admissible traffic. A traffic pattern is said to be admissible if no input or output is over-subscribed. A maximum size matching has been shown to be stable for i.i.d arrivals up to an offered load of 100% when the traffic is uniform and admissible [2]. It has also been demonstrated that a maximum weight matching algorithm can lead to a maximum throughput of 100% for independent and either uniform or non-uniform traffic [3]. The weights used in these algorithms can be the lengths of various VOQS or cell arrival times [4]. Unfortunately, the best known algorithms for performing a maximum size match or maximum weight match are too complex ($O(N^{\frac{5}{2}})$ [5] and $O(N^3 \log N)$ [6] , respectively) for high speed implementations. Hence, a number of heuristic algorithms have been proposed to approximate the behaviour of these complex algorithms. PIM [7] and iSLIP [8] are examples of algorithms

which attempt to converge on a maximal matching and iLPF [9] is an algorithm which approximates maximum weight matching. Though, these algorithms are comparatively simpler to implement, they do not match the performance of an output queued switch under extreme traffic conditions.

Efforts have been made to design scheduling algorithms which can retain the properties of output queued swtiches under all traffic patterns. These *worst case results* usually need increased speedup in the switch to maintain their throughput under all traffic conditions. Reference [10] proposes a scheduling algorithm called Critical Cells First (CCF) which (with speedup of 2) can exactly emulate an output queued swtich, i.e, it is both work conserving and preserves the cell ordering of an ideal output queued switch. Reference [11] presents a simpler ($O(n)$) algorithm called Lowest Occupancy Output First Algorithm (LOOFA) which keeps the switch work conserving under all traffic conditions. This algorithm can be augmented with timestamps to preserve the cell ordering in a switch with a speedup of 3. However, these significant algorithms are also not practical for high speed implementations.

For example, a switch with links operating at a rate of 10 Gb/s has less than 40 ns to make a scheduling decision! This implies that often, it is the implementation simplicity which is the primary factor in determining which scheduling algorithms are used in most high speed routers. Hence, these *worst case results* are only of theoretical interest and limited use. On the other hand, it is non-tivial to arrive at any definite conclusions about the performance of implementable scheduling algorithms, especially, under extreme traffic conditions. This is particularly of concern because of the unregulated nature of IP networks which can cause sustained overloads at output ports of routers. There are a number of factors which can lead to overload problems in IP networks

- limited route diversity which makes congested links common.

- use of route selection mechanisms which are not guided by session bandwidth needs.

- sudden route changes which can cause rapid traffic shifts.

- use of slow congestion control mechanisms.

- presence of malicious users.

These overload conditions in IP networks are essentially inadmissible traffic patterns that can potentially cause scheduling algorithms to underperform leading to a loss in throughput. Hence, it is not clear how most practical scheduling algorithms used in switches would perform under more realistic, inadmissible traffic conditions in IP networks.

To study the performance of scheduling algorithms under these extreme traffic conditions, we have designed a *stress test*. This stress test is a traffic pattern which simulates the unregulated nature of IP networks by overloading the various outputs of a switch with the objective of bringing about the worst case performance of the scheduling algorithms. The test while not providing any conclusive evidence, helps us in making meaningful distinctions among algorithms operating under extreme conditions. We use this stress test as a tool to gain insight into

- performance of practical scheduling algorithms under extreme conditions

- performance of work conserving scheduling algorithms under speedups < 2.

- in design of *stress resistant* scheduling algorithms which maintain their throughput both under uniform traffic and stress tests and still are simple enough to be used in high speed implementations.

In this paper, we first study the performance of crossbar scheduling algorithms PIM, *i*SLIP and LOOFA under uniform traffic and the stress test. We demonstrate that though PIM and *i*SLIP perform well under uniform traffic, they have low throughputs under the stress test. While LOOFA has good throughput even under the stress test, it is too complex for implementation. From the performance studies we observe that scheduling algorithms which favour outputs with smaller queue lengths over those which have greater queue lengths can maintain their throughput even under extreme conditions. Using this insight, we present a simple heuristic called the Lowest Layer Selection (LLS) that we then use to create stress resistant but still implementable versions of PIM and *i*SLIP. These algorithms called LLS-Random (LLS-R) and LLS-Slip (LLS-S) have good performance under both stress test and uniform traffic. We also present an approximate but implementable version of LOOFA called approximate LOOFA (A-LOOFA) which only maintains an approximate ordering of the outputs based on their queue lengths but is still stress resistant.

2 Stress Test

Most prior work compares the performance of scheduling algorithms for CIOQ switches by measuring the average queuing delay of packets for various traffic workloads. Average queuing delay as a metric can be used to study the relative performance of two algorithms for a given workload. But, it is difficult to quantify the absolute performance (*throughput*) of an algorithm using the measured average queuing delays, especially, when the traffic workload is inadmissible and/or non-uniform, where the inherent queuing
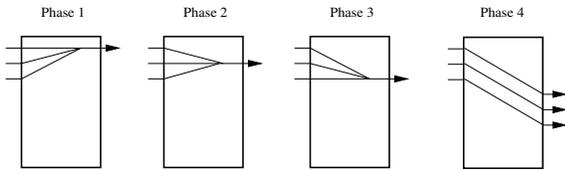
Phase 1   Phase 2   Phase 3   Phase 4

Figure 1. Example of stress test with 3 partic-ipating inputs and 4 phases

delays of the packets due to the traffic sources tends to dom-inate over the actual delays induced by the scheduling algo-rithm. Even when the traffic is uniform, while the maximum throughput achieved can be inferred from the measured av-erage queuing delays, it is non-trivial to quantify the exact throughput achieved at various traffic loads.

In this section, we use a metric called the *miss fraction* to quantify the throughput achieved by a scheduling algorithm used in a CIOQ switch. In a given measurement period, let N_A be the number of cells forwarded by a switch using scheduling algorithm A and N_I be the number of cells for-warded by the ideal output queued switch when subjected to the same workload as algorithm A. Then *miss fraction* is defined as

$$miss\ fraction = 1 - \frac{N_A}{N_I}$$

Thus, the metric essentially determines the relative loss in throughput of a switch using the given scheduling algorithm as compared to the ideal output queued switch under the same traffic conditions. The miss fraction proves to be a particularly useful metric in inadmissible traffic conditions where the average queuing delays are usually unbounded.

To test the performance of scheduling algorithms for CIOQ switches under extreme and inadmissible traffic con-ditions, we have designed a *stress test*. The stress test sim-ulates unregulated traffic by causing sustained overloads at various outputs of the router. Also, while stressing individ-ual outputs, the test attempts to bring about the worst case performance in the work-conserving nature of the schedul-ing algorithm. To achieve this, the test takes an adversarial approach to stressing various outputs with the goal of in-creasing the miss fraction of the scheduling algorithm. The adversarial approach of the stress test tries to create condi-tions where,

1. A single output which has an empty queue has cells queued for it at various inputs.

2. Inputs which have cells queued for an output with an empty queue, also have cells queued for other outputs.

A traffic pattern which can create such conditions can po-tentially cause a scheduling algorithm to incur greater miss

fractions. In particular, the stress test we have designed, consists of a series of phases, as illustrated in Fig. 1. In the first phase, the arriving traffic at each of several inputs is sent to a single output. This causes each of the inputs to build a backlog for the target output. The arriving traf-fic at all inputs is then switched to a second output, causing the accumulation of a backlog for the second output at each of the inputs. Successive phases proceed similarly, creating backlogs at each input for each of several outputs. During the final phase, the arriving traffic at each of the inputs is switched to distinct new outputs. Since, these inputs are the only source of traffic for the new target outputs, they must send packets to them as quickly as they come in, while si-multaneously clearing the backlog for other outputs, in time to prevent underflow at those outputs. This creates an ex-treme condition that can cause underflow and increase the miss fraction. The timing of the transitions is chosen to en-sure that all the VOQs at each of the participating inputs still have some backlog at the final transition. More specifically, to create the worst case traffic conditions for a given algo-rithm, the traffic is switched to a new target output when the input backlog for the current target rises to the same level as the input backlog for the previous target. However, when comparing the performance of different schedulers, the transition times and measurement periods are fixed and the same test is applied to all algorithms. The stress test can be varied by changing the number of participating inputs and the number of phases.

Fig. 2 better illustrates the progress of a stress test. Fig. 2(a) plots the queue lengths of various VOQs of the first input (0), of a switch under a stress test with 3 partici-pating inputs and 4 phases. (This test is illustrated in Fig. 1.) The algorithm used in the example is PIM and the speedup of the switch is 1.5. The plot shows how the input directs its traffic to a new output when the input backlog to the cur-rent output equals that of the backlog to a previous output. In the last phase input 0 is the only input sending traffic to output 3 but it still accumulates a backlog to that output in-dicating misses incurred by output 3. Fig. 2(b) shows the average miss fraction incurred by the algorithm in this test. We use the interval from the beginning of the last phase to the end of the simulation as a measurement period for the average miss fraction due to the stress test. This explains the spike in the miss fraction curve in Fig. 2(b). Algorithms which have smaller miss fractions under these stress tests, evidently maintain their throughput even in overload situa-tions.

We note that the stress test only exemplifies a general ap-proach to evaluating CIOQ algorithms under extreme con-ditions. There may well be other stress test scenarios that are more *stressful* at least for some algorithms and that al-gorithms that are designed to perform well on the stress test might perform poorly under other tests. However, the intu-
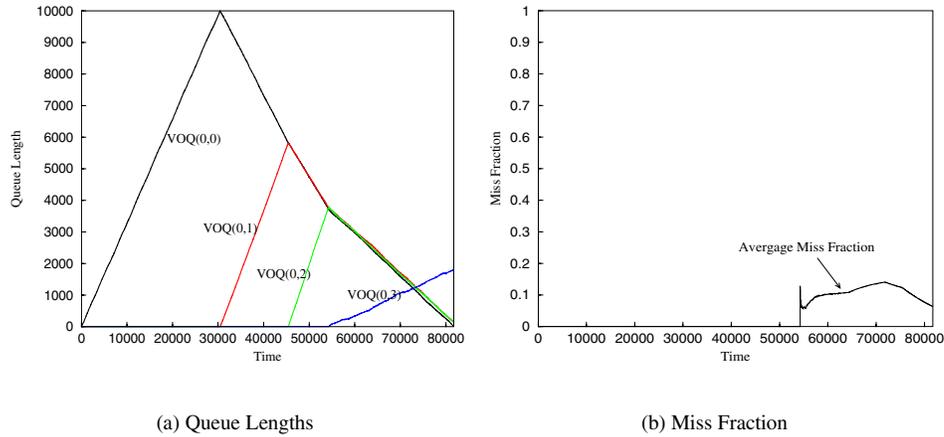
(a) Queue Lengths   (b) Miss Fraction

Figure 2. Queue lengths of various VOQs and miss fraction for PIM under a stress test with 3 participating inputs and 4 phases. (N=16, speedup=1.5)

itive basis of the stress test provides good evidence for distinguishing among algorithms which perform well in overload situations and those that do not.

We study the performance of various centralized crossbar scheduling algorithms under the stress test and present simple improvements which retain the desirable properties of these algorithms and make them *stress resistant*.

3 Crossbar Schedulers

In this section, we first study the performance of the crossbar scheduling algorithms, PIM, *i*SLIP and LOOFA by measuring their miss fractions under uniform traffic and the stress test. We show that the simple algorithms, PIM and *i*SLIP perform poorly under the stress test though they have good performance under uniform traffic. On the other hand, LOOFA has good performance under the stress test at reasonable speedups but is too complex for a high speed implementation. We note that the better performance of LOOFA under the stress test is primarily due to its output ordering and present heuristics to use this insight in designing stress resistant algorithms in the next section.

3.1 Parallel Iterative Matching (PIM)

PIM is an iterative matching algorithm which attempts to converge on a maximal match in multiple iterations. Each iteration consists of three steps where

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 one randomly to grant.

3. If an input receives any grants, it choose one to accept and notifies that output.

In [7], the authors show that the algorithm converges to a maximal match in $O(\log N)$ iterations by showing that each iteration eliminates 3/4 of the remaining requests. It is interesting to note that this property is independent of the way the inputs select a grant in the third step of the algorithm.

Also, since the outputs send their grants randomly, when all the input queues are occupied, the probability that no output will grant to a particular input in one iteration is $((N-1)/N)^N)$. Hence, in a single iteration, the throughput of PIM is limited to $(1 - \frac{1}{e})$ for large N, which is approximately 63% for $N = 16$.

3.2 Iterative Round Robin Matching with Slip (*i*SLIP)

*i*SLIP, like PIM, is an iterative algorithm but is designed to give higher throughputs even for a single iteration. The algorithm iterates the following three steps

1. Each unmatched input sends a request to every unmatched output for which it has a cell queued.

2. If an unmatched output receives any requests, it chooses one that appears next in a fixed, round-robin schedule starting from its input pointer. The output notifies each input whether or not its request was granted. The input pointer 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 its own output pointer. The output pointer of the round-robin schedule is incremented (modulo N) to one location beyond the accepted output.

The *iSLIP* algorithm maintains good performance even with a single iteration under heavy loads due to its *desynchronization* effect. Step 2 of the algorithm causes different outputs to send grants to different inputs, particularly, under heavy loads, causing larger matches in single iteration.

3.3 Lowest Occupancy Output First Algorithm (LOOFA)

LOOFA is also an iterative algorithm which iterates the following steps till no more matches can be made

1. Each unmatched input sends a request to an output with the lowest occupancy among those for which it has at least one queued cell.

2. Each output, upon receiving requests from multiple inputs, selects one and sends a grant to that input.

It has been proven that a switch using LOOFA with a speedup of 2 is work conserving under all traffic conditions [11]. But the algorithm requires $O(N)$ iterations to *perform correctly*. This means that, unlike PIM and *iSLIP*, the algorithm can have biased and unpredictable behaviour when used with fewer iterations. For example, under uniform traffic and heavy loads when all inputs have cells queued for all outputs, all the inputs send requests to the same output in a single iteration! Such behaviour makes LOOFA unsuitable for use in high speed implementations where there is only time to perform a few iterations.

3.4 Performance Evaluation

We first measure the miss fractions of these algorithms under uniform traffic for varying number of iterations, where the cells arrive as a Bernoulli process and are uniformly distributed over all outputs. The speedup of the switch is 1.0 implying that all queuing is done at the inputs. Fig. 3 and Fig. 4 show the difference between using miss fraction and average queuing delay as a metric versus offered load for PIM and *iSLIP*, respectively.

While the maximum load carried by a scheduler can be determined from the delay plots, the miss fractions curves also indicate the throughput achieved by the algorithms at all loads. It can be inferred from Fig. 3(a) that the queuing delays are unbounded for PIM for load > 0.63 and that the miss fraction for PIM in Fig. 3(b) increases to 0.36 for an offered load of 1. This is in agreement with the fact that the throughput of PIM is limited to 63% (for $N = 16$) for

a single iteration. *iSLIP* on the other hand performs much better even with a single iteration. The miss fraction curve of *iSLIP* in Fig. 4(b) shows that for load > 0.63 the rate of increase of miss fraction actually shows a sharp decrease. This is due to the *desynchronization effect* of *iSLIP* which comes into play when almost all VOQs have non-zero backlogs which happens for load > 0.63. The noise like variation seen in the curves where miss fraction is in the range $[0.001, 0.0001]$ is due to the very fine granularity of observation.

To compare the performance of the algorithms under the stress test, recall that the transitions times and the measurement periods of the test have to be fixed for all algorithms. To determine these basic parameters, we subject one of the algorithms to a stress test where the transitions between various phases takes place when the backlog of the VOQ at an input to a target output equals that of the backlog of a VOQ to a previous output. We then compare the performance of the rest of the algorithms under the same test with these basic parameters.

Fig. 5(a) and Fig. 5(b) compares the performance of PIM, *iSLIP* and LOOFA under a stress test with 5 participating inputs and 12 phases at various speedups. In Fig. 5(a) the basic parameters were determined to create worst case traffic scenario for PIM(4) under a speedup of 2.0. We denote the test with these parameters as Test *A*. As can be noted, PIM and *iSLIP* have poor performance under the stress test. *iSLIP* has almost the same performance for iterations 1 and 4 and has miss fraction of 30% even at a speedup of 2. PIM shows improvement with increase in iterations from 1 to 4 but still has a higher miss fraction at various speedups when compared to LOOFA. LOOFA has been proven to be work conserving for unlimited iterations under all traffic conditions for a speedup of 2. For this particular test, LOOFA needs a speedup of just 1.3 to eliminate all underflow.

Fig. 5(b) compares the performance of the algorithms under a stress test where the basic parameters were determined to create worst case traffic for LOOFA at a speedup of 2.0 (Test *B*). Again, LOOFA has zero miss fraction for speedups > 1.5 showing that the output ordering based on queue lengths in LOOFA effectively reduces all underflow even in extreme conditions. Though the same test doesn't cause PIM and *iSLIP* to perform as badly as in Fig. 5(a), the performance of these algorithms does not match that of LOOFA.

These tests demonstrate the underperformance of widely used algorithms like PIM and *iSLIP* under overload traffic conditions.
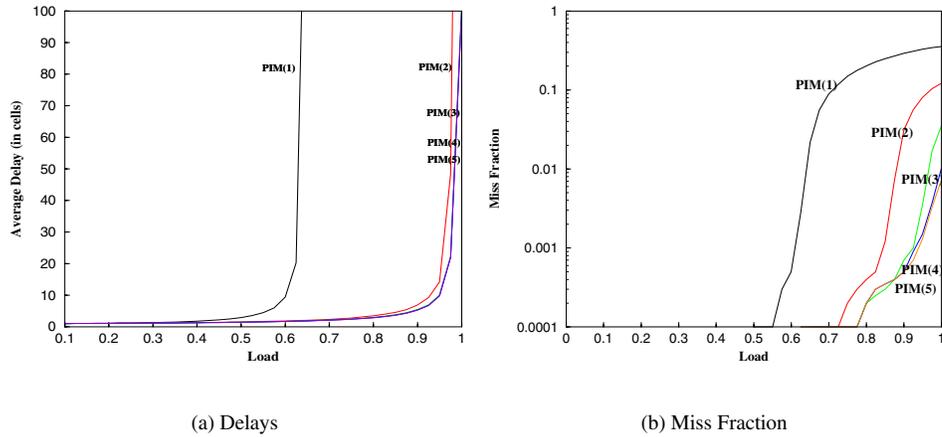
(a) Delays   (b) Miss Fraction

Figure 3. Average delays and miss fractions for various iterations of PIM, N=16, speedup=1.0

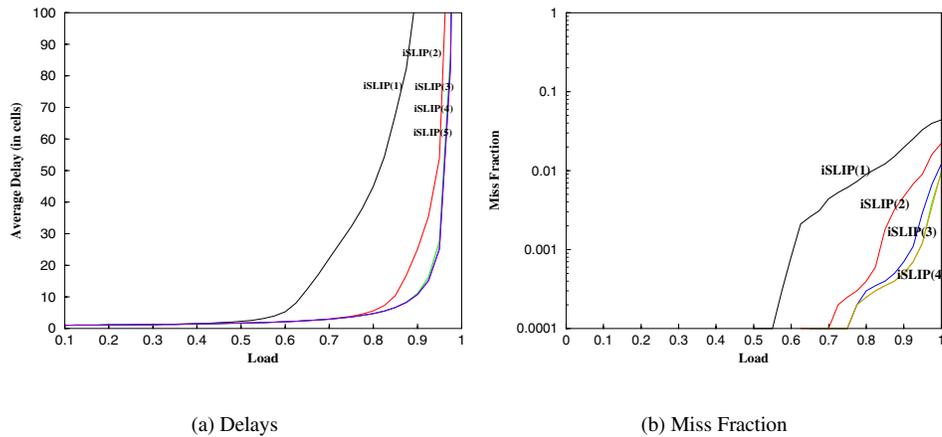(a) Delays   (b) Miss Fraction

Figure 4. Average delays and miss fractions for various iterations of *i*SLIP, N=16, speedup=1.0

4 Stress Resistant Algorithms

The better performance of LOOFA under the stress test suggests that biasing outputs to favour those with smaller queue lengths is the key to maintaining throughput even under extreme traffic conditions. Unfortunately, complete ordering of outputs and the large number of sequential iterations needed to use this ordering can themselves be obstacles to implementing these algorithms at high speeds. But, we note that the traffic conditions that are in consideration here are essentially persistent traffic conditions and that algorithms which achieve and use even approximate or partial ordering of outputs can perform significantly better than those that do not take output backlogs into consideration at all. In this section we introduce two simple heursitics

1. Lowest Layer Selection (LLS) heursitic which

achieves a *coarser* ordering of outputs based on their queue lengths.

2. Odd-even sorting which achieves only an approximation of the ideal ordering of outputs but converges to the ideal ordering under persistent traffic conditions.

We use LLS to design stress resistant variants of PIM and *i*SLIP calles LLS-Random (LLS-R) and LLS-Slip (LLS-S). We use the odd-even sorting technique to design an approximate version of LOOFA called approximate LOOFA (A-LOOFA). All these stress resistant algorithms have been designed for high speed implementations.

4.1 Lowest Layer Selection

PIM and *i*SLIP perform poorly compared to LOOFA under the stress test because they ignore output occupancies.

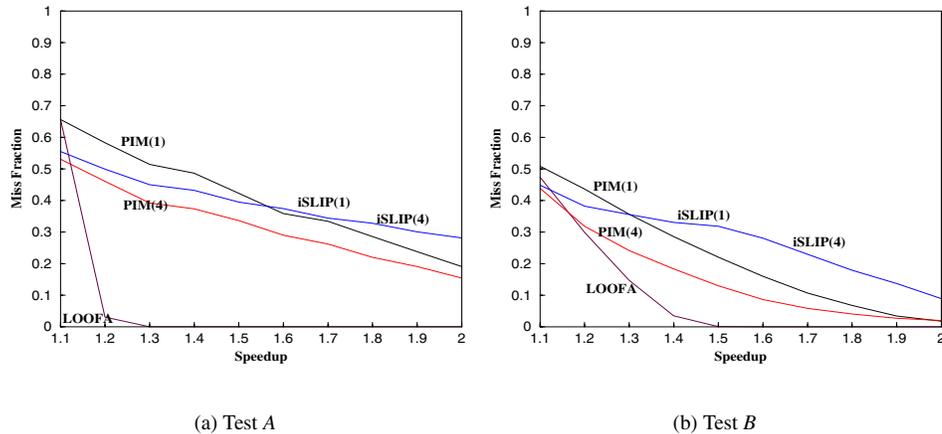(a) Test *A*　　　　　　　　　　　　(b) Test *B*

Figure 5. Miss fractions for PIM, *i*SLIP and LOOFA under stress test with 5 participating inputs and 12 phases.

On the other hand, they perform well under uniform traffic and also require fewer iterations to converge making them more suitable for high speed implementations.

In this section, we describe a simple low-cost mechanism that can be used to make PIM and *i*SLIP *stress resistant*. The improved algorithms have the same performance under uniform traffic and have greatly improved performance under the stress test. The idea is to prioritize the outputs based on their queue lengths since, underflow occurs only when an output queue is empty. The various outputs of the switch are divided into *layers* based on their queue length using an exponentially graded scale. Fig. 8 shows an instance of such a scale with 16 layers. In this scale, queues with length ≤ 8 are put in layer 0, queues with length > 8 and ≤ 16 are put in layer 1 and so on. The queue length corresponding to a layer i is given by 8×2^i. The last layer of the scale (15) holds all queues with lengths $> 8 \times 2^{15}$, indicating that the scale doesn't differentiate between outputs with the largest queue lengths. Thus the layering of queue lengths

- achieves a coarser ordering of outputs based on queue lengths.

- bigger layers are used for larger queue lengths, indicating that there is less chance of underflow at outputs with large queue lengths.

- beyond a queue length limit (indicated by the final layer), all outputs are treated equal as it is not necessary to order outputs with large queue lengths to avoid underflow.

Hence, the number of layers itself is independent of the number of ports of the switch (*N*) making it possible to use
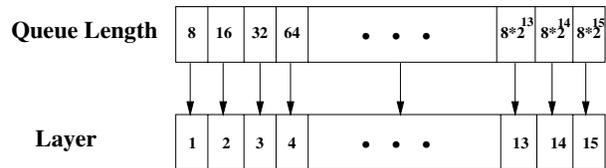

Figure 8. Exponentially graded scale used in assigning outputs to layers based on their queue length.

a single scale with 8 to 16 layers for high speed switches with large numbers of ports. Also, the layers to which various queues belong can be trivially updated whenever cells are added or removed from the various queues.

Algorithms use the layers to which the various outputs belong by employing a Lowest Layer Selection (LLS) heuristic. The algorithms (LLS-R and LLS-S) in their accept phase give priority to outputs in the lowest layer. Thus, the use of the Lowest Layer Selection heuristic in these algorithms introduces a bias towards outputs with smaller queue lengths. The exponential scale used in defining the layers determines the extent of this bias, since the algorithms still show their default behaviour to outputs which belong in the same layer. A scale which has *thin* (and hence, more) layers, forces the inputs to always accept outputs with smaller queue lengths and a scale with *thick* layers causes the inputs to pick outputs randomly (in case of PIM) or in a round-robin order (in case of *i*SLIP) irrespective of the queue lengths of the outputs since, outputs more often than not will belong to the same layer.

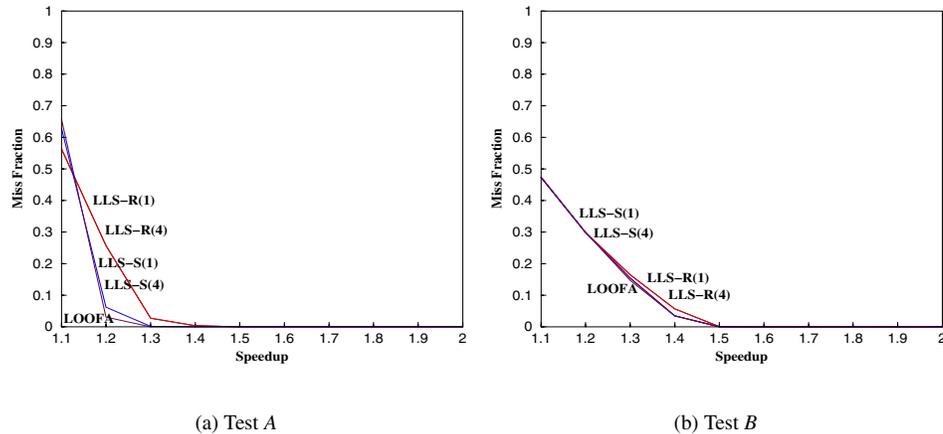The heuristic itself can be implemented at negligible cost

(a) Test *A*   (b) Test *B*

Figure 6. Miss fractions for LLS-R, LLS-S (using 16 layers) and Loofa under stress test with 5 partic-ipating inputs and 12 phases.

by maintaining per input *grant vectors*. These vectors have 1 bit corresponding to each layer. When an output sends a grant to an input in a scheduling algorithm, it also sets the bit corresponding to the layer to which its queue length belongs, in the input's grant vector. The input can then easily find the lowest layer of all granting outputs by using a priority encoder to find the first bit set to 1 in the grant vector. Also, for crossbars of moderate size (32 ports), we can quickly determine the output with the smallest layer index using an N-way minimum finding circuit.

4.1.1 Lowest Layer Selection - Random (LLS-R)

LLS-R is an interative matching algorithm which uses the LLS heuristic and per input grant vectors to improve the performance of PIM under the stress test. The LLS-R algorithm iterates the following three steps.

1. Each unmatched input makes a request to every unmatched output for which it has a queued cell.

2. If an unmatched output receives any requests, it chooses one randomly to grant.

3. Inputs use their grant vectors to determine the lowest layer among all the granting outputs and accept an output belonging to this layer and notify that output.

It is evident that the algorithm is actually similar to Parallel Iterative Matching in the first two steps and differs only in the *accept* phase where the inputs pick an output from the lowest layer. Hence, the algorithm will still converge on a maximal match in $O(\log N)$ iterations. The proof of this claim is similar to the argument made in [7] for PIM.

4.1.2 Lowest Layer Selection - Slip (LLS-S)

LLS-S is a variant of the *i*SLIP algorithm obtained by using the LLS heuristic and per input grant vectors. The algorithm iterates the following three steps

1. Each unmatched input sends a request to every unmatched output for which it has a cell queued.

2. If an unmatched output receives any requests, it chooses one that appears next in a fixed, round-robin schedule starting from its input pointer. The output notifies each input whether or not its request was granted. The input pointer 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. Inputs use their grant vectors to determine the lowest layer among all granting outputs, and accept an output from this layer, that appears next in a fixed, round-robin schedule starting from their output pointer. The output pointer is then incremented (modulo N) to one location beyond the accepted output.

Again, we note that LLS-S differs from *i*SLIP only in the last step where the inputs select one of multiple outputs belonging to the lowest layer.

4.1.3 Performance Evaluation

We first note that in purely input queued switches (speedup = 1.0), the algorithms, LLS-R and LLS-S behave exactly like PIM and *i*SLIP, since all output queue lengths are 0.
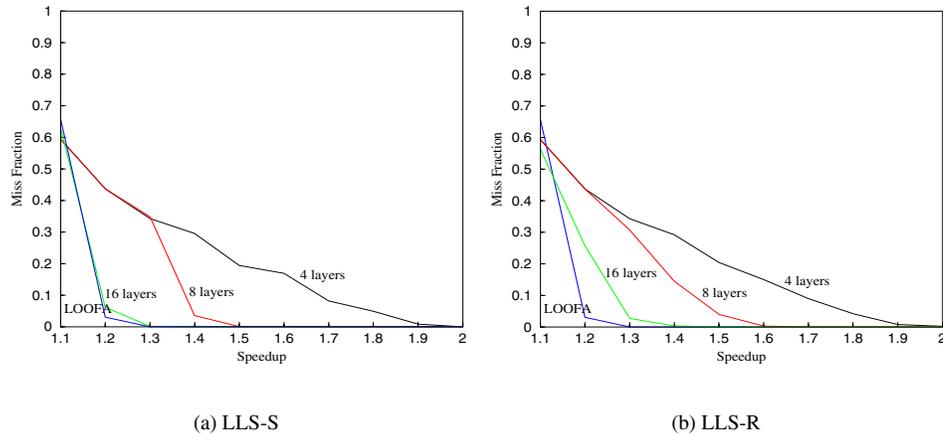
(a) LLS-S   (b) LLS-R

Figure 7. Miss fractions for LLS-R and LLS-S (single iteration) with varying layers under stress test with 5 participating inputs and 12 phases (Test A)

This causes all the outputs to belong to the same layer (layer 0) and the algorithms follow their default behaviour. This observation is also true for uniform random traffic under any speedup, since, all the outputs have approximately the same queue lengths and hence belong to the same layer. Thus the performance of the LLS-R and LLS-S algorithms under uniform random traffic at any load and speedup is identical.

Fig. 6 compares the performance of LLS-R, LLS-S and LOOFA under a stress test with 5 participating inputs and 12 phases. Under both tests A and B, the algorithms show greatly improved performance over PIM and iSLIP shown in Fig. 5. With just a single iteration, LLS-R has zero miss fraction for speedup ≥ 1.4 (Fig. 6(a)) and has similar performance even with 4 iterations. LLS-S also shows greatly improved performance very similar to that of LOOFA even for a single iteration. Under Test B (Fig. 6(b)), the algorithms have almost identical performance.

Fig. 7 compares the performance of LLS-S and LLS-R algorithms with varying number of layers. As can be seen from the figures, the algorithms show improvement with increasing number of layers and have performance comparable to that of LOOFA with 16 layers. This comparison of the performance of these algorithms with that of LOOFA shows that these simple algorithms can potentially provide high throughputs even in overload situations even by using only approximate output ordering schemes.

4.2 Approximate LOOFA (A-LOOFA)

Although LOOFA itself is too complex for a high speed implementation, it can be used as the basis for an algorithm that is practical and which in practice, can provide very similar performance. This algorithm, which we call
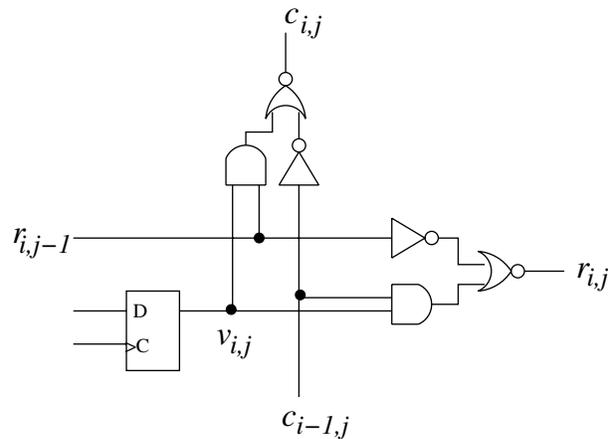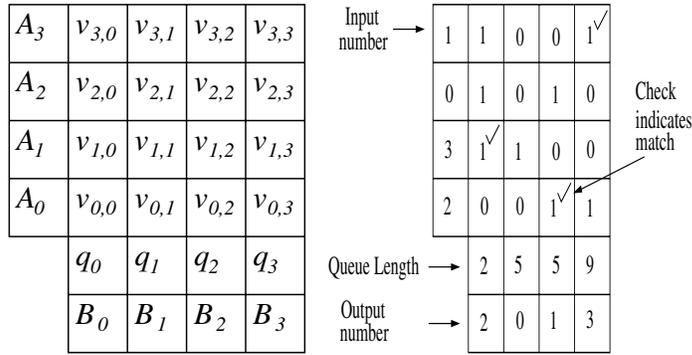

Figure 9. Match Logic

Approximate LOOFA (A-LOOFA), can be implemented in hardware in a way that makes it suitable for routers with 10 Gb/s links. Fig. 10 illustrates the basic concept behind A-LOOFA and its implementation. At the left, we have a set of *row registers*, A_i ($0 \leq i \leq N-1$), each containing the number of some input. At the bottom, we have a set of *column register pairs*, (B_j, q_j) ($0 \leq j \leq N-1$) each containing the number of an output (B_j) and its associated output queue length (in cells). The central area contains an $N \times N$ array of *VOQ occupancy bits* $v_{i,j}$ where $v_{i,j} = 1$ if and only if input A_i has one or more cells to send to output B_j. A-LOOFA attempts to maintains the set of column register pairs in sorted order, so that $q_0 \leq q_1 \leq \cdots \leq q_{N-1}$. As will be explained shortly, it only approximates the sorted order, in order to avoid a time-consuming sorting step.

					Input number					
A_3	$v_{3,0}$	$v_{3,1}$	$v_{3,2}$	$v_{3,3}$		1	1	0	0	1 ✓
A_2	$v_{2,0}$	$v_{2,1}$	$v_{2,2}$	$v_{2,3}$		0	1	0	1	0
A_1	$v_{1,0}$	$v_{1,1}$	$v_{1,2}$	$v_{1,3}$		3	1 ✓	1	0	0
A_0	$v_{0,0}$	$v_{0,1}$	$v_{0,2}$	$v_{0,3}$		2	0	0	1 ✓	1
	q_0	q_1	q_2	q_3	Queue Length →	2	5	5	9	
	B_0	B_1	B_2	B_3	Output number →	2	0	1	3	

Check indicates match

(a) Hardware Components   (b) Example Operation

Figure 10. Principal hardware components and example operation of A-LOOFA

(a) Test A   (b) Test B
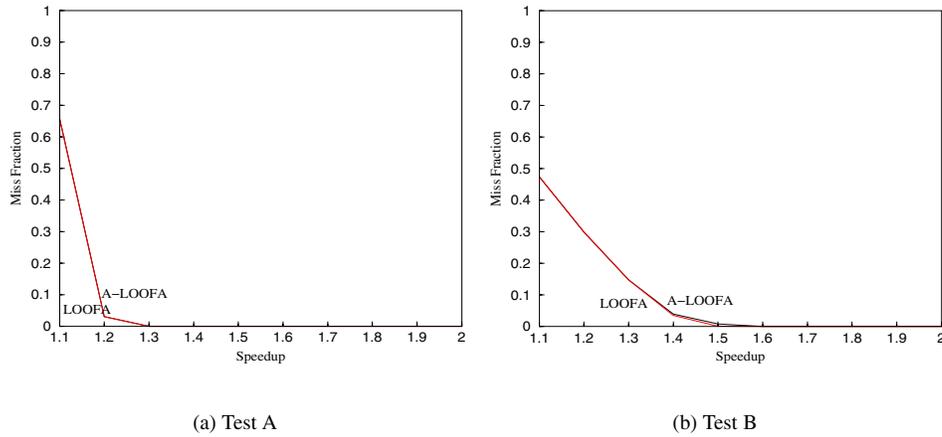
Figure 11. Miss fractions for A-LOOFA and LOOFA under stress test with 5 participating inputs and 12 phases.

Matching in A-LOOFA is accomplished using a simple combinational circuit. This circuit effectively implements the N step iterative matching process required by LOOFA. While it requires $O(N)$ time to complete, the constant factor is determined by gate delays, making it small enough to allow for high speed implementation. Fig. 9 shows the match logic that is associated with the VOQ occupancy bit $v_{i,j}$. The input signals $r_{i,j-1}$ and $c_{i-1,j}$ are high if output B_j is available for selection by input A_i. If both are high and $v_{i,j} = 1$ then A_i is matched with B_j and $r_{i,j}$ and $c_{i,j}$ are both pulled low. So,

$$r_{i,j} = r_{i,j-1}(\overline{v}_{i,j} + \overline{c}_{i-1,j}) = \overline{(\overline{r}_{i,j-1} + v_{i,j}c_{i-1,j})}$$
$$c_{i,j} = c_{i-1,j}(\overline{v}_{i,j} + \overline{r}_{i,j-1}) = \overline{(\overline{c}_{i-1,j} + v_{i,j}r_{i,j-1})}$$

To complete a matching operation, these signals must prop-

agate throughout the $N \times N$ array, but note that signals propagate upward and to the right, so the delay is $2N$ times the delay in each block, with each block contributing two gate delays. For a modern .13 μm ASIC process, the gate delays are 25-50 ps, allowing a match to be completed in 3.2-6.4 ns. A router with 10 Gb/s links and a speedup of 2 will need to complete a crossbar scheduling operation every 20 ns, making the matching delay small enough to allow for high speed implementation.

In order for the approach described to exactly implement LOOFA, it's necessary to maintain the column register pairs in sorted order. This is not practical in a high speed implementation. Fortunately, we can still get good (although not provably work-conserving) performance without sorting. Because the queue lengths change slowly, we can

maintain an approximate sorted ordering by doing a pair of nearest neighbor swaps (odd-even sorting) after each crossbar scheduling operation. Specifically, for all even $j < N$, we exchange the values of B_j and q_j with B_{j+1} and q_{j+1} if $q_j > q_{j+1}$. Then for all odd $j < N - 1$, we exchange the values of B_j and q_j with B_{j+1} and q_{j+1} if $q_j > q_{j+1}$. Whenever we perform such an exchange, we also exchange the values of the VOQ occupancy bits in the corresponding columns.

The combinational matching circuit favors inputs that occupy "lower" rows in the array of VOQ occupancy bits. To ensure fairness among the different inputs, we randomly permute the rows of the array at the end of each crossbar scheduling operation (both the row registers and the VOQ occupancy bits). Specifically, for all even values of $i < N$, we generate a pseudo random bit x_i. This is easy to do in hardware. If $x_i = 0$, then all the values in row i are moved to row $i/2$ and the values in row $i + 1$ are moved to row $(N+i)/2$. If $x_i = 1$, then all the values in row i are moved to row $(N+i)/2$ and all values in row $i+1$ are moved to row $i/2$. This permutation scheme is based on the well-known perfect shuffle, is easy to implement and ensures long-term fairness.

There are a few other issues that need to be addressed to complete the description of the implementation. First, when we get a match, we need a way to pass the identity of the matching input to the circuitry that controls the output, so that the appropriate crossbar control signals can be asserted. This requires a $\lg N$ bit wide data path for each row and column of the array and a switch that forwards the value on row i to column j if there is a match at location (i, j). Second, we need a way to load new values in the VOQ occupancy bit. To do this, the circuitry controlling an input sends an output number along its row, which is compared at each location (i, j) to $B(j)$. At the location where these values match, the VOQ bit is selected to receive a new value.

Finally, we need to maintain a connection between the IO pins of the device and the registers associated with each input and output. Since the pins of the device have a fixed association with specific inputs and outputs, we need to maintain connections between these fixed pin locations and the associated registers, which are constantly exchanging values, as the algorithm proceeds. This requires two special purpose crossbars, one on the input side and one on the output side. The crosspoint settings in these crossbars change with each row and column swap to maintain the required connections to the fixed IO pins. The output side crossbar carries a two bit signal from the output pins, indicating whether a given output queue length is to increase by one, decrease by one or stay the same. The input side crossbar carries a $2 + \lg N$ bit signal indicating whether the VOQ occupancy bit for a specified ouput is to be set, reset or stay

the same (note that during one operation cycle, only two VOQs at any input can change their status).

The gate complexity of the A-LOOFA control circuit has the form $C_1 N^2 \lg N + C_2 N^2 + C_3 N \lg N + C_4 N$, for constants $C_1, \ldots, C_4$. We estimate $C_1 \approx 10$, $C_2 \approx 30$, $C_3 \approx 50$ and $C_4 \approx 20$, yielding an overall estimate of less than 90,000 gates. While not a trivial circuit, to be sure, it is well within the capabilities of modern ASICs.

To compare the performance of LOOFA and A-LOOFA we subjected both the algorithms to the stress tests, Test *A* and Test *B*. As can be seen from Fig. 11, they have almost identical performance under both tests indicating that even partial ordering techniques like odd-even sorting used in A-LOOFA can perform well due to the slowly changing nature of the output queue lengths.

5 Conclusions

The problem of overload conditions in IP networks makes it important to study the performance of practical scheduling algorithms under extreme traffic conditions. The stress test that we have presented in this paper, helps us to determine which algorithms perform best under these conditions. Using the stress test, we have studied the performance of crossbar scheduling algorithms PIM, *i*SLIP and LOOFA under overload conditions and have designed improved and implementable *stress resistant* variants of these algorithms, LLS-R, LLS-S and A-LOOFA which can maintain their throughput under both uniform traffic and stress test.

References

[1] M. Karol, M. Hluchyj, and S. Morgan, "Input versus output queuing in a space division switch," *IEEE Trans. Comm*, vol. 35, no. 12, pp. 1347–1356, 1987.

[2] Sundar Iyer and Nick Mckeown, "Maximum size matching and input queued switches," in *Proc. of the 40thth Allerton Conference on Communication, Control, and Computing*, 2002.

[3] N. Mckeown, V. Anantharam, and J. Walrand, "Achieving 100% throughput in an input queued switch," in *Proc. of IEEE INFOCOM 96*, San Francisco, CA, Mar. 1996.

[4] Nick Mckeown, Adisak Mekkittikul, Venkat Anantharam, and Jean Walrand, "Achieving 100% throughput in input queued switches," *IEEE Transations on Communications*, vol. 47, no. 8, aug 1999.

[5] J. E. Hopcroft and R. M. Karp, "An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs," *SIAM*

journal of computing, vol. 2, no. 4, pp. 225–231, dec 1973.

[6] R. E. Tarjan, *Data Structures and Network Algorithms*, Bell Labs, 1983.

[7] T. E. Anderson, S. S. Owicki, J. B. Saxe, and C. P. Thacker, "High speed switch scheduling for local area networks," *ACM Transactions on Computer Systems*, vol. 11, pp. 319–352, Nov. 1993.

[8] N. McKweon, "islip: A scheduling algorithm for input queued switches," *IEEE Transactions on Networking*, vol. 7, no. 2, Apr. 1999.

[9] Adisak Mekkittikul and Nick Mckeown, "A practical scheduling algorithm to achieve 100% throughput in input queued switches," in *Proc. of IEEE INFOCOM 98*, San Francisco, CA, Apr. 1998.

[10] N. McKweon S-T. Chuang, A. Goel and B. Prabhakar, "Matching output queueing with a combined input output queued switch," *IEEE Journal of Selected Areas in Communication*, vol. 17, no. 6, pp. 1030–1039, June 1999.

[11] Pattabhiraman Krishna, Naimish S. Patel, Anna Charny, and Robert j. Simcoe, "On the speedup required for work-conserving crossbar switches," *IEEE Journal on Selected Areas of Communications*, vol. 17, no. 6, pp. 1057–1065, June 1999.