

Scheduling Algorithms for CIOQ Switches

Prashanth Pappu

WUCSE-yy-n

October 29, 2003

Department of Computer Science and Engineering
Campus Box 1045
Washington University
One Brookings Drive
St. Louis, MO 63130-4899

Abstract

This proposal deals with the design of scheduling algorithms for Combined Input and Output Queued (CIOQ) switches.

For crossbar based switches, we demonstrate the poor performance of commonly used scheduling algorithms under overload traffic conditions using targeted *stress tests* and present ideas to develop robust, *stress resistant* versions of these algorithms which are still simple enough to be implemented in high speed switches.

For buffered crossbar based multi-stage switches, we introduce a novel mechanism called distributed scheduling. Distributed scheduling is similar to crossbar scheduling used in switches with small port counts, but is both *distributed* and *coarse-grained* to enable high-speed implementations of scheduling algorithms in high capacity, high performance switches. In this proposal, we outline methods and ideas to comprehensively study and evaluate distributed scheduling.

1. Introduction

1.1. The Scheduling Problem

1.1.1. Anatomy of a Router¹. The main function of a router is to forward packets from its input ports to its output ports. Fig. 1 shows the various components of a router. A switching fabric connects the input side Port Processors (PPs) to the output side PPs. The port processors queue packets and perform all packet processing functions like packet classification, route lookup and packet scheduling. The input and output transmission interfaces (ITI and OTI) terminate the physical links and provide the requisite conversion and encoding functions for transmitting/receiving packets on the target physical layer. The control processor implements the routing and other network management protocols.

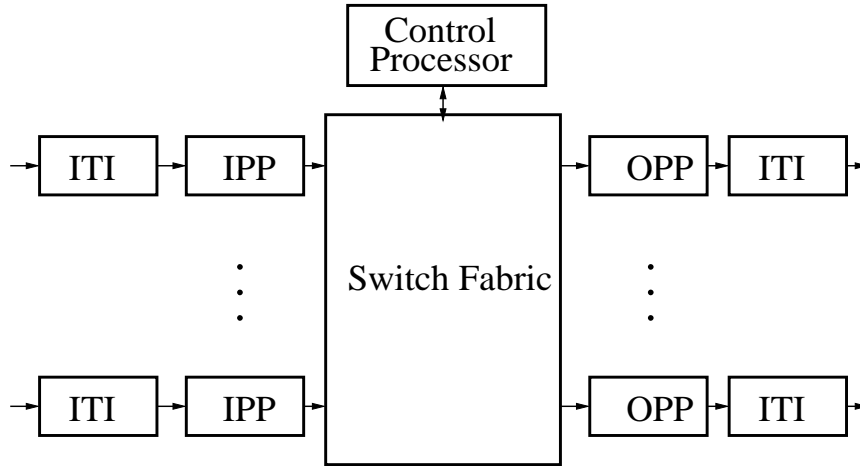


Figure 1: Router Architecture

1.1.2. Output Queuing (OQ). Ideally, we would like all packets in the router to be buffered only at the output ports. In such a router (called Output Queued (OQ) router), when two or more packets destined to the same output, arrive simultaneously at different input ports, they are immediately transferred to the output queue to avoid any packet loss. This architecture not only simplifies the design of the router but also

1. Maximizes the throughput of the router.
2. Enables the ready use of packet scheduling algorithms [22] for providing Quality of Service (QoS) guarantees to individual flows.

Since there is no queuing at the inputs, the output queues of an OQ router with N line cards each connected to a line operating at a rate R should have a bandwidth of $(N + 1) \times R$ (to support

¹We use the terms *router and switch* and *packet and cell* interchangeably, unless explicitly specified.

N writes for each read). Unfortunately, although router capacities ($N \times R$) have increased by about 2.2 times every 18 months (slightly faster than Moore's law), router buffer (DRAM) speeds have only increased by about 1.1 times every 18 months (slower than Moore's law) [20]. This mismatch makes the use of output queuing infeasible in scalable routers.

1.1.3. Combined Input and Output Queuing (CIOQ). To overcome the mismatch between the capacities of routers and the speed of memories, most switches queue packets at both input and output ports of the switch. This Combined Input and Output Queuing (CIOQ) lets us use lower speed memories for buffering packets. When two or more packets at different inputs, contend to go to the same output, some of them are temporarily held in the input queues before being transferred to the outputs.

The switching fabric of a CIOQ switch is itself operated at a speed S (called *speedup*) times the link rate R . Though, the speedup of a CIOQ switch can be any value between 1 and N , in practice, it is usually a small constant (typically, ≤ 2). Hence, a CIOQ switch with a speedup of S , needs memories with a bandwidth of just $(1 + S) \times R$. A switch with a speedup of 1 evidently queues all packets only at inputs and is called an Input Queued (IQ) switch.

While a CIOQ switch requires lower speed switching fabrics and memories, it also introduces a scheduling problem. A decision needs to be made every time slot to determine which inputs are allowed to transfer cells to which outputs. The design of scheduling algorithms to perform this function is the focus of this proposal. The objective in the design of these schedulers is to approximate the throughput and delay properties of a pure output queued switch.

1.2. Related Work

1.2.1. Crossbar based CIOQ switches. Commercial CIOQ switches with small port counts often use a non-blocking crossbar as the switching fabric. An $N \times N$ crossbar is organized as an $N \times N$ matrix to connect input ports to output ports as shown in Fig. 2. A crossbar allows multiple cells to pass in parallel to distinct outputs. Though a crossbar has quadratic complexity ($O(N^2)$), it concentrates this complexity within a single chip or a chipset for moderate scale switches, reducing impact on the system cost. The scheduling problem in crossbar based CIOQ switches is reduced to configuring the non-blocking crossbar using a centralized controller every time slot to determine which inputs are allowed to transmit to which outputs.

Though the throughput and delay properties of a scheduler determine its performance, it is the implementation simplicity which is the primary factor in determining which scheduling algorithms are used in high speed switches. For instance, in a switch with external link rates of 10 Gb/s, the scheduler has less than 40 ns to make a scheduling decision.²

It is known that if the input queues are maintained as FIFOs, then the maximum throughput achieved by any scheduling algorithm is limited to $2 - \sqrt{2} \approx 58\%$ of the link bandwidth for uniform random traffic. Because, with FIFO queueing, if a cell at the head of an input queue is blocked, all cells behind it in the queue are prevented from being transmitted, even when the output link they need is idle. This is called head-of-line (HOL) blocking. The problem of HOL blocking can be

²Assuming a minimum packet size of 50 bytes

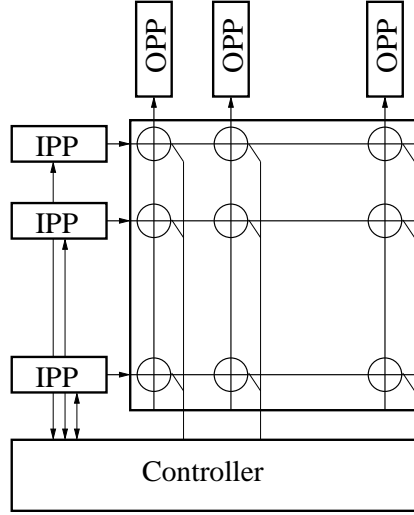


Figure 2: Crossbar based switch.

alleviated by maintaining separate queues at each input for every output (called *Virtual Output Queues* $VOQ(i, j)$) [8]. With $VOQs$, the task of the scheduler is to find a matching on a bipartite graph whose vertices are the inputs and outputs and an edge between an input and an output denotes that the input has a queued cell for that output. A number of results have been proved regarding the performance of crossbar scheduling algorithms under various traffic conditions.

The arrival traffic to a switching system can be viewed as a set of arrival processes $A_{i,j}(t)$, where $A_{i,j}(t)$ is the discrete-time arrival process of cells at input i to output j . The set of all arrival processes at various inputs is together simply referred to as the *arrival process*.

DEFINITION 1: An arrival process is said to be **admissible** if no input or output is oversubscribed, i.e., $\sum_{i=1}^N \lambda_{i,j} < 1$, $\sum_{j=1}^N \lambda_{i,j} < 1$, $\lambda_{i,j} > 0$, where, $\lambda_{i,j}$ is the average rate of arrival of traffic from input i to output j .

DEFINITION 2: An arrival process is **uniform**, if all arrival rates $\lambda_{i,j}$ are equal and destinations are uniformly distributed over all outputs, otherwise the traffic is **non-uniform**.

DEFINITION 3: An arrival process is called **independent and identically distributed (i.i.d)** if all arrivals (both at the same input and across all inputs) are independent of each other and are identically distributed.

DEFINITION 4: A scheduling algorithm is said to lead to **weak stability** if for every $\epsilon > 0 \exists D > 0$ such that, $\lim_{t \rightarrow \infty} P\{X_t(i, j) > D\} < \epsilon$, where, $X_t(i, j)$ denotes the queue length of $VOQ(i, j)$.

DEFINITION 5: A scheduling algorithm is said to lead to **strong stability** if $\lim_{t \rightarrow \infty} E(X_t(i, j))$ is finite.

DEFINITION 6: A **maximal** match on a bipartite graph is one for which atleast one endpoint of every edge in the graph is matched. A **maximum** match is one that matches the maximum number of inputs and outputs. A maximum match is maximal; the converse is not true.

A number of **stability results** have been proved regarding the performance of crossbar scheduling algorithms under admissible traffic conditions. Any scheduling algorithm that produces a maximum size matching has been shown to be strongly stable for i.i.d arrivals up to an offered load of 100% when the traffic is uniform and admissible [7]. It has also been demonstrated that a maximum weight matching algorithm can lead to a maximum throughput of 100% (strongly stable) for independent and either uniform or non-uniform traffic [13]. The weights used in these algorithms can be the lengths of various VOQs or cell arrival times [14]. Unfortunately, the best known algorithms for computing a maximum size matching or maximum weight matching are too complex ($O(N^{\frac{5}{2}})$ [6] and $O(N^3 \log N)$ [21], respectively) for high speed implementations. Hence, a number of heuristic algorithms have been proposed to approximate the behaviour of these complex algorithms. PIM [2] and *i*SLIP [15] are examples of algorithms which attempt to produce a maximal matching and *i*LPF [16] is an algorithm which approximates maximum weight matching. APSARA [4], DRD-SRR [11] and the algorithms introduced in [5] are examples of algorithms that aim to obtain an approximate MWM. Both PIM and *i*SLIP have a runtime complexity (convergence time) of just $O(\log_2 N)$. APSARA uses limited parallelism to find a MWM in just a single iteration. Though, these algorithms are comparatively simple 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 switches under all (both admissible and inadmissible) traffic patterns. These **worst case results** usually need increased speedup in the switch to maintain their throughput under all traffic conditions. Reference [3] proposes a scheduling algorithm called Critical Cells First (CCF) which (with speedup of 2) can exactly emulate an output queued switch, i.e, it is both work conserving and preserves the cell ordering of an ideal output queued switch. CCF needs to compute a stable matching which can take as many as N^2 iterations and also the algorithm has high information complexity and needs information (for each cell at an input) that depends on the state of all queues in the system. Though there are techniques to reduce the matching complexity to $O(N)$ and the number of cells that need to be considered at each input can be upper bounded by N , they cannot be applied together and as the authors note in [3] CCF remains a complex algorithm to implement using current technology. Reference [9] 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 [19]. However, these significant algorithms are only of theoretical interest and are not practical for high speed implementations.

Given this current situation, where algorithms with proven performance under a variety of traffic conditions are often not implementable and implementable algorithms are evaluated only according to their packet delays under admissible and/or random uniform traffic, it is not clear, how most commercially used algorithms perform under extreme traffic conditions frequently encountered in unregulated IP networks

1.2.2. Buffered multi-stage CIOQ switches. In crossbar based switches, all the line cards and the crossbars in the switching fabric must be synchronized to the centralized scheduler. The frequency (*once in less than 40 ns for a switch with 10 Gb/s links*) and the complexity ($O(\log_2 N)$ *even for simple algorithms*) of the decisions made by the scheduler, makes it infeasible to use crossbars for large switching systems.

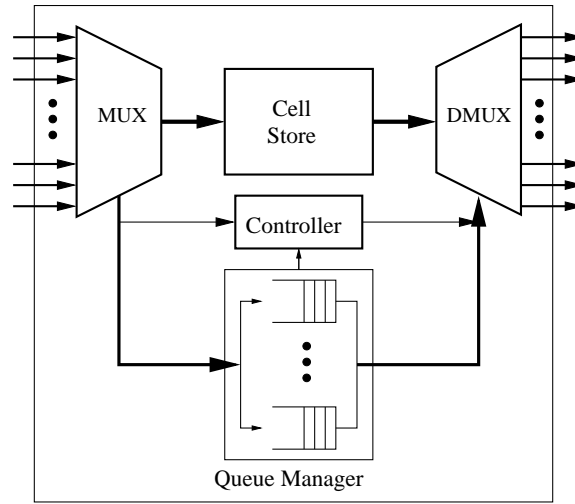


Figure 3: Buffered crossbar switch element.

One approach to alleviate the centralized scheduling problem is the introduction of *buffers* at the crosspoints of a crossbar. Such a switching element is called a *buffered crossbar*. With these buffers, cells are sent from inputs into the crossbar only if the corresponding crosspoint has empty buffers. Thus, the ingress scheduling problem is reduced to a simple flow control mechanism. A backpressure signal is used to indicate if the input can forward cells to the crosspoint. On the egress side, the outputs schedule cells from one of the N cross-points destined to them. Unfortunately, the discrete nature of buffering makes this architecture memory intensive. A 16×16 crossbar needs 256 buffers, each with a space for at least a few cells.

The buffered crossbar architecture can be further simplified by using shared memory within the crossbar as shown in Fig. 3. The switching element has an internal shared buffer (*cell store*) for upto a few thousand cells with current technology. The cells are multiplexed from the inputs of the switching element to free slots in the cell store. These cells are then forwarded to the outputs as and when they become idle. A controller uses per output queue information to configure the input side multiplexer and the output side demultiplexer. Such switching elements can be used in multi-stage switching fabrics with inter stage flow control to build large switching systems. The simple flow control mechanism along with a modest speedup can alleviate the need for a centralized scheduler and maintain throughput even under temporary overloads.

The performance of such switching systems can degrade drastically in the presence of sustained overloads. In extreme traffic conditions, when a single output port of a switch is under sustained overload, the shared memory buffers in the switching fabric can be congested with cells attempting to reach the overloaded output, interfering with other traffic directed to non-overloaded outputs. The unregulated nature of IP traffic makes such overloads a normal fact of life, which router designers must address if their systems are to be robust enough to perform under the most demanding of traffic conditions.

1.3. Proposal Overview

1.3.1. Stress Resistant Crossbar Scheduling Algorithms. 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. Commercially used 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. On the other hand, algorithms such as LOOFA with provably good worst-case performance, don't lend themselves readily to high performance implementation. In this proposal, 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 proposal, we introduce the use of stress testing for crossbar scheduling which can be used to evaluate the performance of PIM, *i*SLIP and LOOFA. We have presented results in [18] that 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 present ideas to develop improved versions of PIM and *i*SLIP, which take output queue lengths into account, making them much more robust (stress resistant). We also present an algorithm which closely approximates the behavior (and performance) of LOOFA, but which admits a straightforward, high performance hardware implementation.

1.3.2. Distributed Scheduling. In this proposal, we introduce *distributed scheduling* as a means of regulating the flow of traffic through large, high performance multi-stage routers which use buffered crossbar switching elements. The mechanism is both **distributed** and **coarse-grained** to enable high speed implementations of the algorithms. Distributed scheduling, unlike crossbar scheduling, does not seek to schedule the transmission of individual packets. Instead, it regulates the *rates* at which traffic is forwarded through the switching fabric from inputs to outputs. These rates are themselves determined and readjusted at pre-determined time periods using distributed algorithms to let the mechanism scale to switches with large capacities. This also implies that distributed scheduling can only approximate the performance of a pure output queued switch.

In this proposal we present ideas to comprehensively study and evaluate distributed scheduling. In particular, the proposal outlines

1. Work conserving distributed scheduling algorithms.
2. Basic non-iterative distributed scheduling algorithms.
3. Time-Sliced distributed scheduling algorithms for output queuing emulation.
4. Fair distributed scheduling algorithms for providing Quality of Service guarantees.

2. Stress Resistant Crossbar Scheduling Algorithms

2.1. Introduction

The unregulated nature of traffic in IP networks can cause sustained overloads at output ports of routers. There are a number of factors which can lead to such extreme traffic conditions in IP networks

1. limited route diversity which makes congested links common.
2. use of route selection mechanisms which are not guided by session bandwidth needs.
3. sudden route changes which can cause rapid traffic shifts.
4. use of slow congestion control mechanisms.
5. 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. The common practice of evaluating widely used algorithms based on packet delays under random admissible traffic tends to obscure significant shortcomings in the robustness of these algorithms when exposed to such extreme traffic conditions.

To study the performance of scheduling algorithms under extreme traffic conditions, we propose the use of targeted *stress tests*. A 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 intend to use stress tests as a tool to address several issues

- *How do practical scheduling algorithms perform under extreme conditions?*
- *How do work conserving scheduling algorithms perform under smaller (< 2) speedups?*
- *How do we design scheduling algorithms which can maintain their throughput both under admissible traffic and overload traffic conditions and still are simple enough to be used in high speed implementations?*

In [18], we have studied the performance of crossbar scheduling algorithms PIM, *i*SLIP and LOOFA under uniform traffic and a particular stress test. We've shown 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 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.

2.2. Simulating Overload Traffic Conditions

2.2.1. Miss Fraction. 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 delays of the packets due to the traffic sources tends to dominate over the actual delays induced by the scheduling algorithm. Even when the traffic is uniform, while the maximum throughput achieved can be inferred from the measured average 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 forwarded by the ideal output queued switch when subjected to the same workload as algorithm A . Then *miss fraction* is defined as

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

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.

2.2.2. Stress Test. To test the performance of scheduling algorithms for CIOQ switches under extreme and inadmissible traffic conditions, we have designed a *stress test*. The stress test simulates unregulated traffic by causing sustained overloads at various outputs of the router. Also, while stressing individual outputs, the test attempts to bring about the worst case performance in the work-conserving nature of the scheduling algorithm. To achieve this, the test takes an adversarial approach to stressing various outputs with the goal of increasing the miss fraction of the scheduling algorithm. The adversarial approach of the stress test tries to create conditions 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 potentially cause a scheduling algorithm to incur higher miss fractions. In particular, the stress test we have designed, consists of a series of

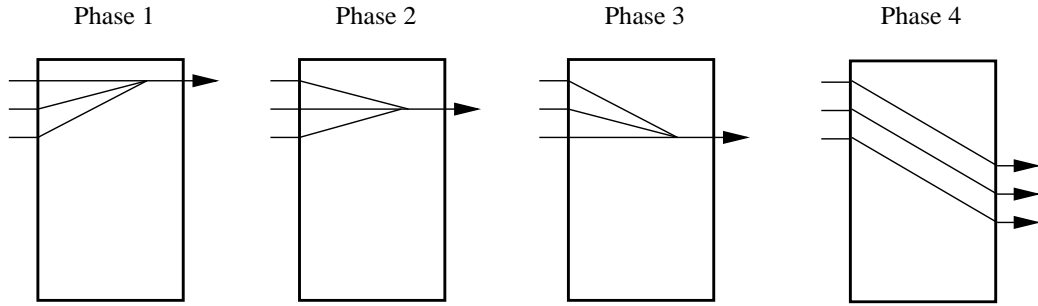


Figure 4: Example of stress test with 3 participating inputs and 4 phases

phases, as illustrated in Fig. 4. 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 traffic 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 simultaneously clearing the backlog for other outputs, in time to prevent underflow at those outputs. This creates an extreme condition that can cause underflow and increase the miss fraction. The timing of the transitions is chosen to ensure 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 algorithm, 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. 5 better illustrates the progress of a stress test. Fig. 5(a) plots the queue lengths of various VOQs of the first input (0), of a switch under a stress test with 3 participating inputs and 4 phases. (This test is illustrated in Fig. 4.) 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 current 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 indicating misses incurred by output 3. Fig. 5(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. 5(b). Algorithms which have smaller miss fractions under these stress tests, evidently maintain their throughput even in overload situations.

We note that the stress test only exemplifies a general approach to evaluating CIOQ algorithms under extreme conditions. There may well be other stress test scenarios that are more *stressful* at least for some algorithms and that algorithms that are designed to perform well on the stress test might perform poorly under other tests. However, the intuitive basis of the stress test provides good evidence for distinguishing among algorithms which perform well in overload situations and those that do not.

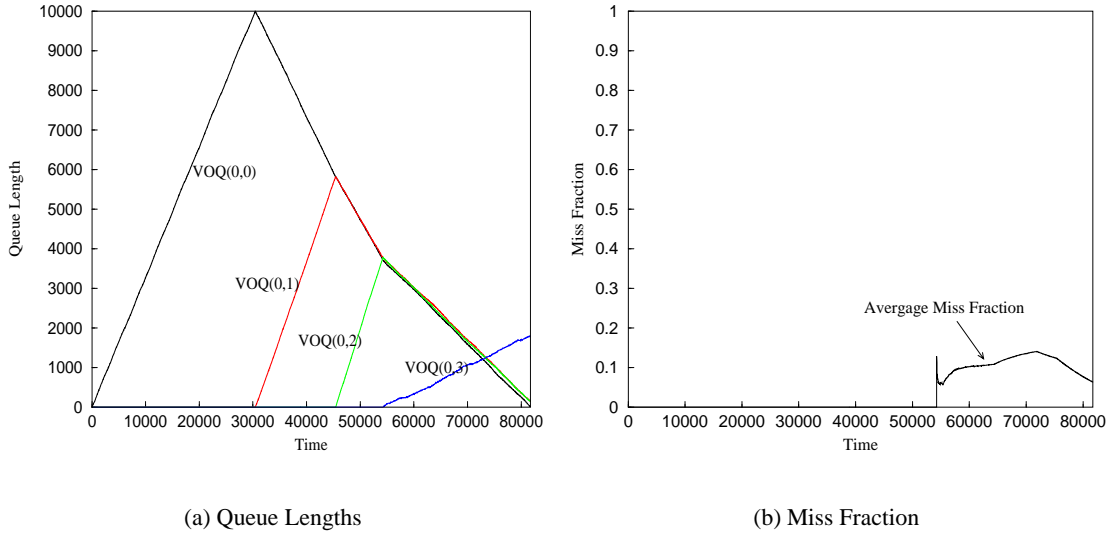


Figure 5: Queue lengths of various VOQs and miss fraction for PIM under a stress test with 3 participating inputs and 4 phases. ($N=16$, speedup=1.5)

2.3. Crossbar Schedulers

In [18], we study in detail 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.

2.4. 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 heuristics

1. Lowest Layer Selection (LLS) heuristic 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 called 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.

2.4.1. Lowest Layer Selection. PIM and *i*SLIP perform poorly compared to LOOFA under the stress test because they ignore output occupancies. 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. 6 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, reflecting the fact 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 as 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 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.

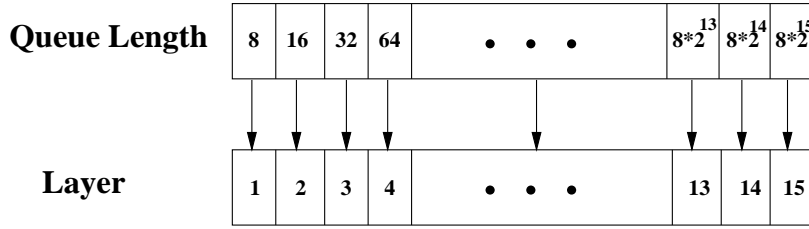


Figure 6: Exponentially graded scale used in assigning outputs to layers based on their queue length.

The heuristic itself can be implemented at negligible cost 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. 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.

The algorithms using the LLS heuristic and their performance are studied in detail in [18].

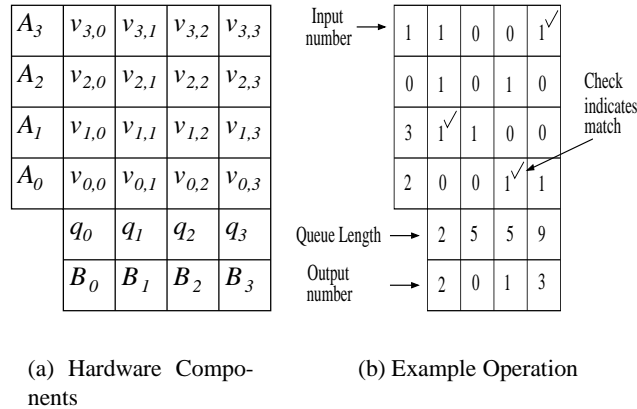


Figure 7: Principal hardware components and example operation of A-LOOFA

2.4.2. 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 *Approximate LOOFA* (A-LOOFA), can be implemented in hardware in a way that makes it suitable for routers with 10 Gb/s links. Fig. 7 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 $q_0 \leq q_1 \leq \dots \leq q_{N-1}$. As will be explained shortly, it only approximates the sorted order, in order to avoid a time-consuming sorting step.

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. 8 shows the match logic that is associated with the VOQ

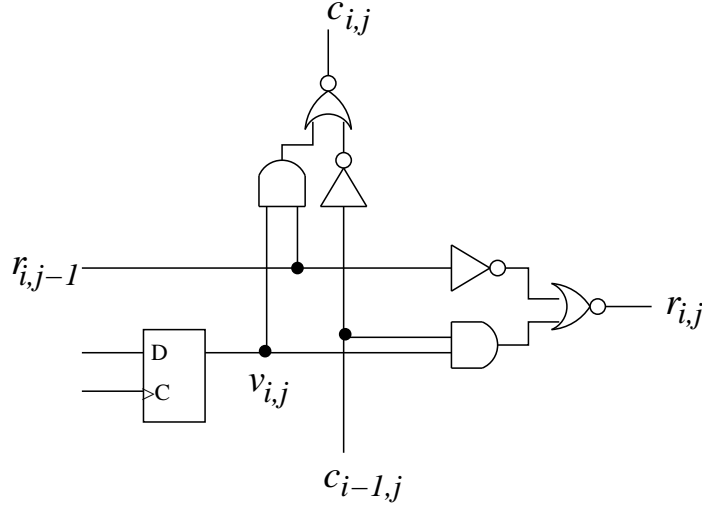


Figure 8: Match Logic

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,

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

To complete a matching operation, these signals must propagate 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 $.13 \mu\text{m}$ 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.

We note that attempts have been made to use similar combinational circuits to perform matching in other contexts. For example, [12] describes an implementation of the iMCRA algorithm. The bits in the rows in this problem represent the various outputs to which a single multicast packet is destined. However, this algorithm solves a maximal, non-redundant vertex cover problem unlike A-LOOFA which solves a maximal matching problem and has only $O(N)$ complexity.

Further implementation details and performance results for A-LOOFA can be found in [18].

2.5. 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 proposal, 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, iSLIP 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 traffic.

As noted earlier, the stress test only exemplifies a general approach to creating overload traffic conditions. Also, at a speedup of 1.5, LOOFA has zero miss fraction under the stress tests, while it has been proven that LOOFA requires a speedup of 2 to maintain 100% throughput under all traffic conditions. This suggests that work needs to be done to create more *stressful* tests.

Also, we need to prototype the improved algorithms (A-LOOFA in particular) in VHDL, to make more concrete conclusions on their implementation complexity.

3. Distributed Scheduling for Terabit Routers

3.1. Introduction

It is common knowledge that the increasing traffic demands in IP networks can quickly exhaust the number of free slots in low density routers. When a number of low density routers (with 16-32 ports) are configured to together behave like a single large router, a majority of the ports are used in interconnecting the routers themselves. Hence, most ISPs prefer large capacity switching systems for their core networks. Most high capacity switching systems currently under development are multi-rack systems [1] (to reduce power density) and employ distributed, multi-stage switching fabrics.

Unfortunately, multi-stage switching fabrics have unpredictable performance unlike single-stage crossbar switches. Network operators need switching systems that can operate at throughputs of 100% to use the full capacity of expensive long haul links. In this chapter, we introduce Distributed Scheduling as a scalable mechanism to provide performance guarantees in such large multi-stage switching systems.

3.2. Distributed Scheduling

Distributed Scheduling (DS) is a method for regulating the flow of traffic through large routers employing multi-stage switching fabrics which use buffered switching elements. DS, unlike crossbar scheduling, does not seek to schedule the transmission of individual cells every time slot. Instead, it regulates the *rates* at which traffic is forwarded through the switching fabric from various inputs to outputs using coarse-grained scheduling. This means that the *rates* themselves are determined and readjusted, only at a pre-determined update period (T). While this approach keeps the mechanism scalable, this also implies that DS can only approximate the throughput and delay properties of a pure output queued switch.

3.2.1. Mechanism. Fig. 9(a) shows a simplified block diagram of a router that implements distributed scheduling. Each output port contains a FIFO queue and each input port contains a set of N virtual output queues. The *VOQs* are rate controlled by a Distributed Scheduling Controller (DSC). The DSCs at various input ports execute the following pseudocode every update period (T).

```
DQ()
{
    Send_VOQ_status();
    Recv_VOQ_status();
    Allocate_rates();
    for(int i=1 to N){
        SetPace(VOQ(i),new_rate);
    }
}
```

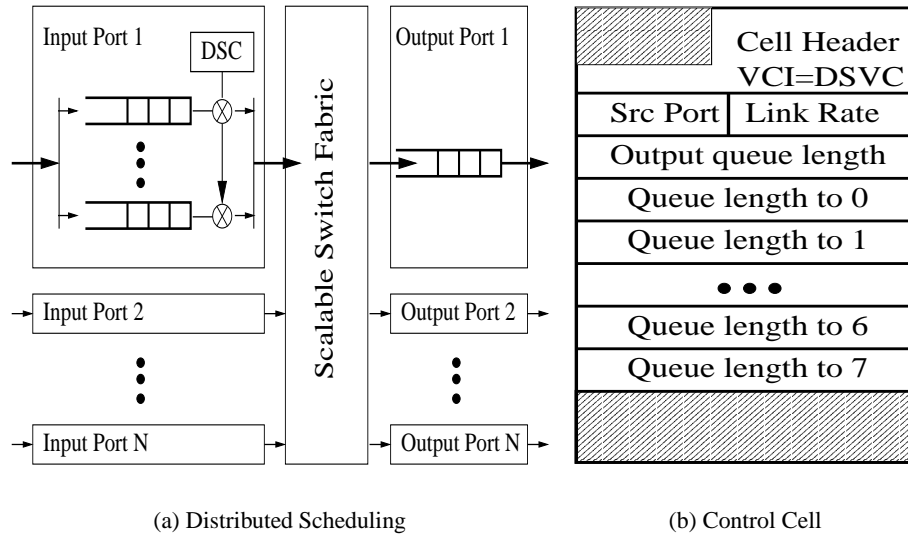


Figure 9: Router with distributed scheduling.

The DSCs periodically exchange information about the VOQs and the output queues and use this information to determine new rates at which their VOQs are to be paced. The exchange of information by inputs can be achieved by using control cells. Fig. 9(b) shows the control cells used for distributed scheduling in the Dynamically Extensible Router (DER) [10] developed at Washington University. The information exchanged in these particular cells is simply the queue length of various VOQs and outputs, though, in general, any information that incurs an acceptably small overhead can be exchanged. For example, in a system with 1000 links, each operating at 10 Gb/s and using an update period of 100 μ s, the overhead due to exchange of queue length status is just 5% of the system bandwidth. From the pseudocode, it is clear that the most important component of DS is the rate allocation algorithm.

3.2.2. Constraints. The rate allocation algorithms used in DS are limited by the following constraints

$$\sum_j r_{i,j} \leq S \times R \quad (1)$$

$$\sum_i r_{i,j} \leq S \times R \quad (2)$$

where $r_{i,j}$ is the rate at which DSC at input i forwards traffic to output j , S is the speedup of the switch and R is the external link rate. Equation 1 (*referred to as the output constraint*) restricts the net rate at which traffic is sent to each output to the speed of the switching fabric itself. Equation 2 (*referred to as the input constraint*) denotes that the total rate at which traffic is transferred from each input is limited by the speed of the switching fabric itself. A good rate allocation algorithm should satisfy these constraints and emulate the behaviour of an output queued switch as closely as possible.

3.3. Work Conserving DS algorithms

3.3.1. Problem Definition. We note that the DS mechanism is scalable because it is distributed and that it performs the scheduling decision only every update period (T). The primary issue we wish to investigate in this section is the effect of making a scheduling decision only at fixed time periods on the throughput of the switch. To address this problem we use the following model of a switching system.

A crossbar switch can be viewed as operating in three phases. An arrival phase, a transfer phase and a departure phase. In the arrival phase and departure phase a maximum of one cell can arrive at an input or depart from an output respectively and in the transfer phase, up to S (speedup) cells can be transferred from an input or to an output. In a switching system in which the scheduling decision is being made only every T time units, we extend the model so that in each arrival phase an input can receive up to T cells and during the departure phase, each output sends up to T cells on its output link. At the beginning of each transfer phase (every T time units) a scheduling decision is made to determine which cells are transferred from inputs to outputs with a limit of $S \times T$ cells on each input and output.

We ask the question, *is there a scheduling algorithm that can keep such a system work conserving implying that every output for which cells are queued at inputs has at least T cells in its output queue at the beginning of the departure phase.*

3.3.2. Work Conserving Scheduling Algorithms. Before we can find a distributed scheduling algorithm, we first identify a couple of properties, in effect, constructing a centralized but still *coarse-grained* algorithm which can keep the system work conserving even by making scheduling decisions only at time periods (T). The following definitions will be used in defining the algorithm.

At each input, we maintain an ordering of the VOQs. This ordering is changed (if need be) only at the beginning of any of the three phases. The VOQs are ordered in increasing order of the queue lengths of the corresponding outputs. This ordering can also be extended to an ordering of the cells at an input.

DEFINITION 1: We say a cell b precedes a cell c in the cell ordering at an input, if b is in a VOQ that comes before the VOQ that c is in, or if b is in the same VOQ as c , but appears earlier in the VOQ.

The scheduling algorithm determines which cells move from input to output during the transfer phase.

*DEFINITION 2: A scheduling algorithm is said to be **maximal** if for any cell c that is not transferred in the transfer phase, either $S \times T$ cells at c 's input are transferred or $S \times T$ cells are transferred to c 's output.*

*DEFINITION 3: A scheduling algorithm is said to be **ordered** if for any cell c that is not transferred, no cell preceded by c at the same input gets transferred unless c 's output gets $S \times T$ cells.*

3.3.3. Claim. *With a speedup of 2 ($S = 2$), a scheduling algorithm that is both maximal and ordered is work conserving.*

To prove this claim, we use the notation shown in Table 1. Let c be any cell at an input and $o(c)$ be the output to which c is destined. For a given phase (arrival, transfer or departure) let $q(c)$ be the queue length of $o(c)$ at the *beginning* of the phase and let $q'(c)$ be the queue length of $o(c)$ at the *end* of the phase. Similarly, let $p(c)$ be the number of cells preceding c at the beginning of the phase and $p'(c)$ be the number of cells preceding c at the end of the phase. Also, define *slack* of cell c at the beginning of the phase as $slack(c) = o(c) - p(c)$ and slack of c at the end of the phase, $slack'(c)$ as $slack'(c) = q'(c) - p'(c)$. Finally, let $r(c)$ be the number of cells received by $o(c)$ during the transfer phase.

Notation	Definition
$o(c)$	Output to which a cell c is destined.
$q(c)$	Queue length of $o(c)$ at the beginning of a phase.
$p(c)$	Number of cells preceding c at the beginning of a phase.
$slack(c)$	Slack of cell c at the beginning of a phase = $o(c) - p(c)$
$q'(c)$	Queue length of $o(c)$ at the end of a phase.
$p'(c)$	Number of cells preceding c at the end of a phase.
$slack'(c)$	Slack of cell c at the end of a phase = $o'(c) - p'(c)$.
$r(c)$	Number of cells received by $o(c)$ during transfer phase.
T	Maximum number of arrivals at an input in arrival phase.

Table 1: Notation.

LEMMA 3.1. *If all cells at an input have non-negative slack before the arrival phase then minimum slack of cells at the input after the arrival phase is $\geq -T$.*

Proof Consider a cell c at an input which didn't arrive in this phase. Clearly, at the beginning of the phase

$$slack(c) = q(c) - p(c) \geq 0 \quad (3)$$

The precedence of c can increase by a maximum of T during the arrival phase. Hence, the slack of c after arrival phase is

$$slack'(c) = q'(c) - p'(c) \geq q(c) - (p(c) + T) \geq -T \quad (4)$$

Now, consider a cell b which arrives in this arrival phase. Let c be the latest cell among those that didn't arrive in this phase and precede b . (If there is no such cell, clearly, $slack'(b) \geq -T$.) Since c precedes b after arrival phase, and at most T cells can arrive in one phase

$$q'(c) = q(c) \leq q'(b) \quad (5)$$

Also, since c is the latest among the cells that didn't arrive in this phase and precede b ,

$$(p'(c) - p(c)) + (p'(b) - p'(c)) \leq T \quad (6)$$

which simply denotes that the total number of cells which arrived in this phase and precede b is $\leq T$. Hence,

$$p'(b) \leq p(c) + T \quad (7)$$

Using equations 3, 5 and 7

$$q'(b) - p'(b) \geq q(c) - (p(c) + T) \quad (8)$$

$$\geq q(c) - p(c) - T \quad (9)$$

$$\geq -T \quad (10)$$

■

LEMMA 3.2. *If all cells at an input have slack $\geq -T$ before a transfer phase, then, slack of all cells at that input is $\geq T$ after transfer phase.*

Proof Let c be a cell at input i that is not transferred in the transfer phase. Also, let the slack of c before the transfer phase be

$$slack(c) = -T + x. \quad (11)$$

where $x \geq 0$ since, all cells have slack $\geq -T$ before the transfer phase.

Because of the change in output queue lengths during the transfer phase, a set of cells B which didn't precede c before the transfer phase might now precede it. If B is empty, i.e, $B = \{\}$, then either $2T$ cells preceding c at its input were transferred or $2T$ cells were received by the output to which c is destined. This is because of the maximality of the scheduling algorithm. Hence, the slack of cell c after the transfer phase ($slack'(c)$) is

$$slack'(c) \geq -T + x + 2T \geq x + T \geq T \quad (12)$$

Let B not be empty. Let B have exactly c_m cells, i.e, $|B| = c_m$. Let b be a cell such that $b \in B$ and $\forall b' \in B, p(b) \geq p(b')$. I.e, b is the cell latest in the list of cells which succeeded c before the transfer phase but now precede it.

Since, b precedes c after the transfer phase (but, succeeded it before the transfer phase)

$$q(c) \leq q(b) \quad (13)$$

$$q'(c) = q(c) + r(c) \geq q'(b) = q(b) + r(b) \quad (14)$$

And hence,

$$r(b) \leq r(c) \leq 2T \quad (15)$$

Equation 15 and the maximality of the scheduling algorithm imply that, $2T$ cells preceding b at input i were transferred during the phase. Let c_p of these $2T$ cells precede c and c_s of them succeed c , hence, $c_p + c_s = 2T$.

Firstly, as shown in Fig. 10,

$$p(b) \geq p(c) + c_s + c_m \quad (16)$$

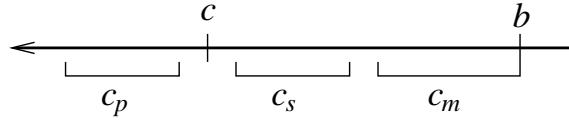


Figure 10: Precedence list.

Also, using equations 11 and 16 and the fact that $slack(b) \geq -T$, we have

$$q(c) - p(c) \leq q(b) - p(b) + x \quad (17)$$

$$\leq q(b) - (p(c) + c_s + c_m) + x \quad (18)$$

Or simply,

$$q(c) \leq q(b) + x - c_s - c_m \quad (19)$$

Using equation 19 in 14, we have

$$q(b) \leq q(c) + r(c) - r(b) \quad (20)$$

$$\leq q(b) + x - c_s - c_m + r(c) - r(b) \quad (21)$$

implying that,

$$r(b) - x \leq r(c) - c_s - c_m \quad (22)$$

Calculating the new slack of cell c after the transfer phase ($slack'(c)$), we have

$$slack'(c) = q'(c) - p'(c) \quad (23)$$

$$= (q(c) + r(c)) - (p(c) - c_p + c_m) \quad (24)$$

$$= q(c) + r(c) - (p(c) - (2T - c_s) + c_m) \quad (25)$$

$$= slack(c) + (2T + r(c) - c_s - c_m) \quad (26)$$

Using equation 22, we have

$$slack'(c) \geq slack(c) + (2T + r(b) - x) \quad (27)$$

$$\geq (-T + x) + 2T - x + r(b) \quad (28)$$

$$\geq T + r(b) \quad (29)$$

$$\geq T \quad (30)$$

■

LEMMA 3.3. *If the slack of all cells is $\geq T$ at the beginning of the departure phase, then, slack of the cells ≥ 0 after departure phase.*

Proof The queue length of an output can decrease by a maximum of T during the departure phase. Hence, if no output reordering occurs (which changes the precedence of cells at inputs), the slack of any cell c at an input $slack'(c) \geq 0$ after the departure phase (since, $slack(c) \geq T$ before departure phase).

Now, we prove that there can be no change in the precedence of cells at inputs due to output reordering during the departure phase. Consider two cells b and c at an input before the departure phase. Let,

$$q(b) \leq q(c) \quad (31)$$

Then, after departure phase, one of the following holds true.

$$q'(b) = q'(c) = 0 \quad (32)$$

$$q'(b) = 0 \leq q'(c) = q(c) - T \quad (33)$$

$$q'(b) = q(b) - T \leq q'(c) = q(c) - T \quad (34)$$

Hence, the relative order of output queue lengths does not change during the departure phase.

THEOREM 3.4. *For any cell c at an input, $slack(c) \geq -T$ after the arrival phase (before the transfer phase), $slack(c) \geq T$ after the transfer phase (before the departure phase) and $slack(c) \geq 0$ after the departure phase (before the arrival phase).*

Proof In the beginning when the system does not have any cells, trivially, the inequalities hold true.

From Lemmas 3.1, 3.2 and 3.3, we have shown that if the slack of all cells is ≥ 0 at the beginning of an arrival phase then all three inequalities hold true and the slack of all cells is ≥ 0 at the next arrival phase too. Hence, by induction, all three inequalities hold true. ■

From Theorem 3.4 it is clear that, no output with cells queued at inputs can have less than T cells in its output queue at the beginning of the departure phase. Hence, a scheduling algorithm that is both maximal and ordered is work conserving with a speedup of 2.

3.4. Backlog based DS algorithms

In practice, it is non-trivial to design distributed scheduling algorithms that are both maximal and ordered, because such algorithms are essentially iterative and require multiple exchanges of information between the various ports. Nevertheless, the above result aids us in evaluating and comparing the performance of more practical DS algorithms which seek to approximate the same behaviour without multiple exchanges.

In designing practical DS algorithms we first note that the rate allocation algorithms can only use the information the various DSCs exchange periodically. In the simplest case, the DSCs at various inputs exchange the size of the backlogs of their VOQs. The backlog from input i to output j is denoted by $B(i, j)$. Each input also receives the size of the output queue length from each of the outputs. The backlog at output j is called $B(j)$. Rate adjustment algorithms which use only these simple queue length measures in adjusting the rates $r_{i,j}$ are referred to as *backlog based DS algorithms*.

In this proposal, we present simple backlog based DS algorithms based on two heuristics to determine the rates $r_{i,j}$ according to the input and output constraints.

1. Output Constraint

To meet the output constraint, we require that for all i and j

$$r_{i,j} \leq out(i, j) \times S \times R \quad (35)$$

where

$$out(i, j) = \frac{B(i, j)}{\sum_j B(i, j)} \quad (36)$$

It is easy to see that allocation of rates using Equation 35 meets the congestion avoidance objective since for each output j , the sum (over all inputs i) of $r_{i,j}$ equals $S \times R$. Also, note that when the rates are allocated in proportion to the magnitude of the input backlogs, all input backlogs to a given output are cleared at the same time (assuming no new traffic arrives). This is the motivation for the *backlog proportional allocation* heuristic.

2. Input Constraint

The input constraint is not satisfied by the above defined values of $r_{i,j}$ only when $\sum_j out(i, j) > 1$. This can be easily overcome by again proportionally scaling the various $out(i, j)$ values at input i by defining

$$in(i, j) = \frac{out(i, j)}{\sum_j out(i, j)} \quad (37)$$

and by allocating rates $r_{i,j}$ using

$$r_{i,j} = S \times R \times \min(in(i, j), out(i, j)) \quad (38)$$

Equation 38 ensures that the scaled down values of $out(i, j)$ ($in(i, j)$) are used only when $\sum_j out(i, j) > 1$.

The performance of the algorithm under non-uniform traffic conditions can be improved by taking output queue lengths into account. While, equation 38 satisfies both the input and output constraints, it does not take backlogs of output queues into account. The output queue backlogs can be greater than 0 for switches with speedup > 1 . To take these output queue backlogs into account, we use an *urgency proportional allocation* heuristic which biases the rate allocation at an input towards outputs with smaller queue lengths, hence, potentially avoiding underflow conditions at outputs. The output queue lengths can be included by using

$$in(i, j) = in_1(i, j) = \frac{out(i, j) \times \frac{1}{B(j)}}{\sum_j out(i, j) \times \frac{1}{B(j)}} \quad (39)$$

$$in(i, j) = in_2(i, j) = \frac{out(i, j) \times (K - B(j))}{\sum_j out(i, j) \times (K - B(j))} \quad (40)$$

Equations 39 and 40 both give preference to outputs with smaller queue lengths in determining $in(i, j)$. And the final rates are determined again by using Equation 38. The performance of backlog based distributed scheduling algorithms has been studied in detail in [17].

3.5. Time Sliced DS algorithms

As mentioned in previous sections, the goal of scheduling algorithms is to approximate the delay and throughput properties of a pure output queued switch. Backlog based DS algorithms can result in good throughput but can also badly misorder packets when the arrival rates of incoming traffic change abruptly. This is primarily because the information exchanged (backlog) between the various DSCs doesn't have any timing information.

DS algorithms can better approximate the properties of an output queued switch by exchanging *queue length slices* instead of total backlog in the control cells. A queue length slice refers to the amount of data an input port receives for an outgoing link during an update interval. Let $A(i, j, t)$ be the amount of data received at input i for output j during update period t . These values are stored in a data structure at output j , for all inputs, and used to determine the target rates at which inputs should send traffic to it. The target rates are chosen to keep the inputs synchronized with each other, with respect to output j . As with earlier algorithms, we first determine $out(i, j)$ to satisfy the output constraint. We start by determining for each output j , the smallest time period t_1 such that

$$\sum_{\tau \leq t_1} A(+, j, \tau) > S \times R \times T \quad (41)$$

We then let,

$$R(j) = S \times R \times T - \sum_{\tau < t_1} A(+, j, \tau) \quad (42)$$

and determine $out(i, j)$ using

$$out(i, j) = \left(\frac{1}{S \times R \times T} \right) \left(\sum_{\tau < t_1} A(i, j, \tau) + \frac{A(i, j, t_1)}{A(+, j, t_1)} \times R(j) \right) \quad (43)$$

It is easy to see that the sum of all $out(i, j)$ for a given j is 1, indicating there will be no congestion in the switch. This allocation allocates the switch bandwidth among all inputs so as to transfer traffic in the order in which it arrived. The input constraint is again satisfied by using one of equations 37, 39, 40 and the final bandwidth allocated is determined using equation 38.

We have omitted the discussion about a data structure that can be used to efficiently determine t_1 for all outputs j . A data structure that combines ideas from binary search trees and heaps can be used for this purpose.

3.6. Fair Distributed Scheduling Algorithms

In routers that support fair queuing [22], packets belonging to different user data flows are placed in different queues and the packet scheduler for each link attempts to give each flow its share of the link bandwidth. The addition of a fair queuing packet scheduler at each output of a router implementing one of the distributed scheduling algorithms discussed in the proposal, can give each flow a share of the output bandwidth, only so long as the links are not overloaded. To provide fair distributed scheduling, it's necessary to augment the scalable

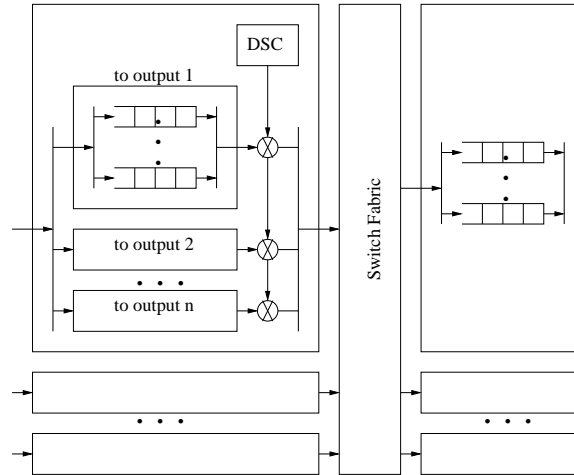


Figure 11: Fair Distributed Scheduling

router architecture as shown in Fig. 11. Note that each output has separate queues for each flow and each input has per flow queues as well. The queues at each input are grouped according to the output they forward packets to. The distributed scheduling controller (DSC) regulates the rates at which packets are forwarded from each group of queues.

To ensure that each flow gets its share of the output link bandwidth, the switch fabric bandwidth should be allocated among the inputs in proportion to the sum of the weights of the backlogged queues they have. While, this ensures that all flows get the right share when they are backlogged, any excess bandwidth can be allocated by an input only among its backlogged flows. In particular, an input with a large number of flows of which only a few are backlogged will always receive excess bandwidth and will render the output link non-work conserving. Suitably designed reallocation mechanisms which take the net backlog of the flows at an input also into consideration can alleviate this problem.

3.7. Summary

In this section, we have introduced Distributed Scheduling (DS) as a scalable mechanism for regulating the flow of traffic through multi-stage switching fabrics using buffered switching elements.

A fair part of the work related to the study and design of work conserving and basic non-iterative DS algorithms has already been completed and can be found in [17]. We intend to use the ideas presented in this proposal to further study time-sliced DS algorithms for output queuing emulation and fair DS algorithms for providing QoS guarantees.

References

- [1] The essential core: Juniper networks t640 internet routing node with matrix technology. <http://www.juniper.net/solutions/literature/solutionbriefs/351006.pdf>, April 2002.

- [2] 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*, 11:319–352, Nov. 1993.
- [3] S.-T. Chuang, A. Goel, N. McKweon, and B. Prabhakar. Matching output queueing with a combined input output queued switch. *IEEE Journal of Selected Areas in Communication*, 17(6):1030–1039, June 1999.
- [4] P. Giaccone, D. Shah, and B. Prabhakar. An implementable parallel scheduler for input-queued switches. *IEEE Micro*, 22(1):19–25, Jan. 2002.
- [5] P. Giaccone, B. Prabhakar, and D. Shah. Towards simple, high-performance schedulers for high-aggregate bandwidth schedulers. In *Proc. of the Twenty-First IEEE Conference on Computer Communications (INFOCOM)*, New York, NY, June 2002.
- [6] J. E. Hopcroft and R. M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM journal of computing*, 2(4):225–231, dec 1973.
- [7] S. Iyer and N. Mckeown. Maximum size matching and input queued switches. In *Proc. of the 40th Allerton Conference on Communication, Control, and Computing*, 2002.
- [8] M. Karol, M. Hluchyj, and S. Morgan. Input versus output queueing in a space division switch. *IEEE Trans. Comm*, 35(12):1347–1356, 1987.
- [9] P. Krishna, N. S. Patel, A. Charny, and R. j. Simcoe. On the speedup required for work-conserving crossbar switches. *IEEE Journal on Selected Areas of Communications*, 17(6):1057–1065, June 1999.
- [10] F. Kuhns, J. Dehart, A. Kantawala, R. Keller, J. Lockwood, P. Pappu, D. Richard, D. Taylor, J. Parwatikar, E. Spitznagel, J. Turner, and K. Wong. Design and evaluation of a high performance dynamically extensible router. *Proceedings of the DARPA Active Networks Conference and Exposition*, 2002.
- [11] J. Liu, H. C. Kit, M. Hamdi, and C. Y. Tsui. Stable round-robin scheduling algorithms for high-performance input queued switches. In *Proc. of the 10th Symposium on high performance interconnects HOT Interconnects (HotI'02)*, Stanford, CA, Aug. 2002.
- [12] J. W. Lockwood. *Design and implementation of a multicast, input-buffered ATM switch for the iPOINT testbed*. PhD thesis, Department of Electrical and Computer Engineering, UIUC, 1995.
- [13] 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.
- [14] N. Mckeown, A. Mekkittikul, V. Anantharam, and J. Walrand. Achieving 100% throughput in input queued switches. *IEEE Transactions on Communications*, 47(8), aug 1999.
- [15] N. McKweon. islip: A scheduling algorithm for input queued switches. *IEEE Transactions on Networking*, 7(2), Apr. 1999.
- [16] A. Mekkittikul and N. Mckeown. A practical scheduling algorithm to achieve 100% throughput in input queued switches. In *Proc. of IEEE INFOCOM 98*, San Francisco, CA, Apr. 1998.
- [17] P. Pappu, J. Parwatikar, J. Turner, and K. Wong. Distributed queueing in scalable high performance routers. In *Proc. of IEEE INFOCOM 2003*, San Francisco, CA, Mar. 2003.

-
- [18] P. Pappu and J. Turner. Stress resistant algorithms for cioq switches. In *International Conference on Network Protocols (ICNP)*, Atlanta, GA, Nov. 2003.
 - [19] T. L. Rodeheffer and J. B. Saxe. An efficient matching algorithm for a high-throughput, low-latency data switch. *Compaq Computer Communications Research Center, Research Report 162.*, 1998.
 - [20] R.R.Schaller. Moore's law: Past, present and future. *IEEE Spectrum*, 34(6):52–59, June 1997.
 - [21] R. E. Tarjan. *Data Structures and Network Algorithms*. Bell Labs, 1983.
 - [22] H. Zhang. Service disciplines for guaranteed performance service in packet switching networks. *Proc. of the IEEE*, 83(10):1374–96, Oct. 1995.