DESIGNING A MULTICAST SWITCH SCHEDULER [*]

BALAJI PRABHAKAR NICK MCKEOWN [†]

ABSTRACT

This paper presents the design of the scheduler for an M$\times$N input-queued multicast switch. It is assumed that each input maintains a single queue for arriving multicast cells and that only the cell at the head of line (HOL) can be observed and scheduled at one time. The scheduler is required to be work-conserving, which means that no output port may be idle as long as there is an input cell destined to it. Furthermore, the scheduler is required to be fair, which means that no input cell may be held at HOL for more than M cell times (M is the number of input ports). The aim is to find a work-conserving, fair policy that delivers maximum throughput and minimizes input queue latency. When a scheduling policy decides which cells to schedule, contention may require that it leave a *residue* of cells to be scheduled in the next cell time. The selection of where to place the residue uniquely defines the scheduling policy. It is demonstrated that a policy which always concentrates the residue, subject to our fairness constraint, always outperforms all other policies. We present one such policy, called TATRA, and analyze it geometrically. We also present a heuristic round-robin policy called *mRRM* that is simple to implement in hardware, fair, and performs quite well when compared to a concentrating algorithm.

1. INTRODUCTION

A growing proportion of traffic on the Internet is multicast, with users distributing a wide variety of audio and video material. This dramatic change in the use of the Internet has been facilitated by the MBONE [1], [2], [3]. It seems inevitable that the volume of multicast traffic will continue to grow for sometime to come. So, if ATM switches are to find widespread use in the Internet, either as standalone switches, or as the core of high performance routers, it is important that they be able to handle multicast traffic efficiently. Although a number of different architectures and implementations have been proposed for multicast switches [6, 7, 8], we restrict our attention to input-queued switches. In particular, we consider how an input-queued switch may schedule multicast cells so as to achieve a high throughput and hence efficient utilization.

Most of the work on input-queued ATM switches has concentrated on unicast traffic in which cells are destined for only a single output. It is well known that when FIFO queues are used, the throughput of an input queued switch with unicast traffic can be limited to just 58% under relatively benign conditions [4]. When arrivals are correlated, the throughput can be even lower [5]. However, numerous papers have indicated that by using non-FIFO input queues and using good scheduling policies, much higher throughputs are possible [9, 10, 11, 12, 13, 14, 16, 17]. In [18], Hayes et

[*]Computer Systems Lab Technical Report, Stanford University.

[†]Balaji Prabhakar is with the Basic Research Institute in the Mathematical Sciences (BRIMS), Hewlett-Packard Labs, Bristol. Email: balaji@lion.statslab.cam.ac.uk. Nick McKeown is with the Departments of Electrical Engineering and Computer Science, Stanford University. Email: nickm@ee.stanford.edu. This work was done when B. Prabhakar was a Visiting Scholar at the Department of EECS, UC Berkeley.

al. give an excellent queueing analysis of the performance of input-queued multicast switches. To maintain tractability, they assume a random scheduling policy for determining which cells are copied to each output during each cell time. Specifically, each output randomly and independently selects one input from among those that request it.

In this paper we consider the performance of different multicast scheduling policies. As may be expected, we find that the random scheduling policy is not the optimum policy. Instead, we find that a better algorithm is one that concentrates the cells that it leaves behind (the *residue*) on as few inputs as possible. In the next section we describe our model and various scheduling policies in more detail. We then prove that for a 2×N switch, the concentrating policy is the optimum policy. Following this, we present simulation results that demonstrate the optimality of the residue-concentrating policy. Finally, we describe TATRA - an efficient M×N multicast switch scheduler.

2. The Switch Model

It is assumed that the switch has M input and N output ports and that each input maintains a single FIFO queue for arriving multicast cells. The input cells are assumed to contain a vector indicating which outputs the cell is to be sent to.

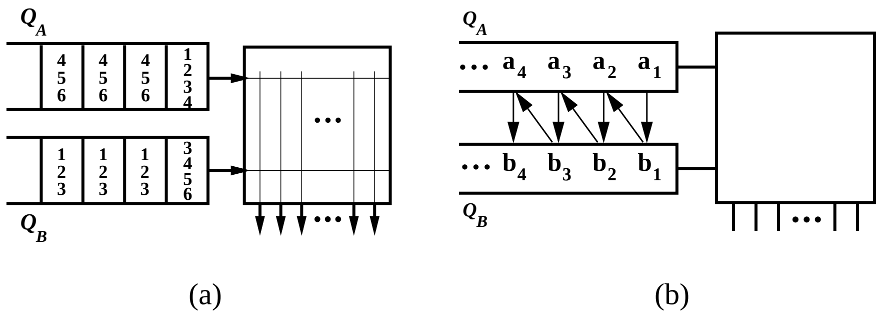


Figure 1:

For example, the 2 input and N output switch shown in Figure 1a has a cell at the head of each queue. Input queue Q_A has an input cell destined for outputs $\{1, 2, 3, 4\}$ whereas input queue Q_B has an input cell destined for outputs $\{3, 4, 5, 6\}$. We shall refer to the size of the vector as the *fanout*. In the figure, the input cell at the head of each queue has a fanout of four. For clarity, we distinguish an arriving *input* cell from its corresponding *output* cells. In the figure, the input cell at the head of queue Q_A will generate four output cells.

The input queues are necessary because cells at different inputs may wish to copy cells to the same output port. At the end of each cell time, a scheduling policy decides which input cells to copy to which output ports. The policy selects a conflict-free match between input and output ports such that each output receives at most one cell. Thus, at the end of every cell time, the scheduling policy discharges some output cells, possibly leaving behind some residual output cells at the head-of-line (HOL) of the input buffers. For example, in the situation depicted in Figure 1a the *discharge* will consist of output cells for outputs $\{1, 2, 3, 4, 5, 6\}$, and the *residue* will consist of output cells for outputs $\{3, 4\}$. Therefore, the scheduling policy has to decide where to place the residue. It may elect to place the residue on both inputs (i.e., it "distributes the residue"), or it may place the residue on either Q_A only or on Q_B only (i.e., it "concentrates the residue"). It is the purpose of this paper to argue, based on theoretical results and simulations, that a scheduling policy that always "concentrates the residue" performs better (improves output utilization, reduces input queue latency, etc.) than one that does not always concentrate residue.

To reduce the implementation complexity, we assume that an input cell must wait in line until all of the cells ahead of it have gained access to all of the outputs that they requested. Furthermore, it is assumed that the scheduling policy observes only the cell at the head of each input queue, without further knowledge of the contents of individual input buffers behind the HOL and traffic arrival patterns. Perhaps the simplest way to service the input queues is to replicate the input cell over multiple cell times, generating one output cell per cell time. However, this service discipline does not take advantage of the multicast properties of the crossbar switch. So instead, we assume that one input cell can be copied to any number of outputs in a single cell time for which there is no conflict.

Following the description in [18], we distinguish two different service disciplines. The first is *full multicast* in which all of the copies of a cell must be sent in the same cell time. If any of the output cells loses the contention for an output port, none of the output cells are transmitted and the cell must try again in the next cell time. The second discipline is *partial multicast* in which case output cells may be delivered to output ports over any number of cell times. Only those output cells that are unsuccessful in one cell time continue to contend for output ports in the next cell time. Because partial multicast is work conserving, it enables a higher switch throughput, for little increase in implementation complexity. Therefore in this paper we consider only partial multicast policies.

3. Some Definitions

Residue: The *residue* is the set of output cells that lose contention for output ports and remains at the HOL of the input queues at the end of each cell time. It is important to note that given a set of requests, every work-conserving policy will leave the same residue. However, it is up to the policy to determine how the residue is distributed over the inputs.

Concentrating Policy: A multicast scheduling policy is said to be *concentrating* if, at the end of every cell time, it leaves the residue on the smallest possible number of input ports.

Distributing Policy: A multicast scheduling policy is said to be *distributing* if, at the end of every cell time, it leaves the residue on the largest possible number of input ports.

A Non-concentrating Policy: A multicast scheduling policy is said to be *non-concentrating* if it does not always concentrate the residue.

Fairness Constraint: A multicast scheduling policy is said to be *fair* if each input cell is held at the HOL for no more than M cell times.

Note that in the two input case this definition of fairness means that the residue alternates between the two inputs. As will be seen, the fairness constraint leads to a round robin priority among the various inputs which is used to resolve contention between them. It can be seen that the above definition of fairness leads to a deterministic bound (of M cell times) on the latency of input cells at HOL.

4. Main Result

We state and briefly discuss the main results of the paper. The basic tenet of the paper is expressed below.

A work-conserving multicast switch scheduling policy that always "concentrates residue" at every possible instant subject to the fairness condition, leads to a higher output utilization than any other policy under arbitrary input processes.

Our goal is to demonstrate this observation and to use it to design an efficient multicast switch scheduler. Simulations are presented in Section 6 in support of this observation under various arrival process distributions; and a sample path proof is presented for the 2×N case. Specifically, we outline a proof of the following theorem.

Theorem 1: *A scheduling policy for a 2×N multicast switch that always "concentrates residue" at every possible instant subject to a natural fairness condition, performs better than any other policy*.

Outline of proof: An outline of the proof of Theorem 1 is given under the static input assumption. That is, it will be assumed that at time 0 both input queues have an infinite number of packets placed according to some (possibly random) packet configuration. Fix one such configuration and suppose that the cells in inputs 1 and 2 are labelled $\{a_i\}_{i=1,2,...}$ and $\{b_i\}_{i=1,2,...}$ respectively; where, for all i, packet a_i is ahead of packet a_{i+1} in Q_A and packet b_i is ahead of packet b_{i+1} in Q_B. See Figure 1b.

The fairness constraint now reduces to the following: the cell (or residue) at the HOL of each input buffer is discharged alternately. That is, no two cells belonging to the same input can depart in successive cell times without there being an input cell departure from the other queue. This leads to the following ordering of input cells: (1) $a_1 \leq_d b_1 \leq_d a_2 \leq_d b_2 \ldots$, if a_1 is the first cell to depart; and (2) $b_1 \leq_d a_1 \leq_d b_2 \leq_d a_2 \ldots$, if b_1 is the first cell to depart.

Without loss of generality, we will adopt the first contention resolution scheme and *link* input cells in a vertical or oblique fashion as shown in Figure 1b. The directions of the arrows on the links denote where the residue is to be concentrated, should a policy choose to concentrate residue at some time. We are only interested in fair scheduling policies as defined above, and so restrict our attention to such policies in this paper. Let the vertical link between a_i and b_i be labelled l_{2i-1} and the oblique link between b_i and a_{i+1} be labelled l_{2i}. The following facts now follow easily.

Fact 1: *All scheduling policies work their way through links $l_1, l_2, l_3, \ldots$ in that order releasing no links (when there is contention between cells at* HOL *and residue is distributed), one link (when there is contention between cells at* HOL *and residue is concentrated), or two links (when there is no contention between cells at* HOL*) in one cell time.*

Fact 2: *The time at which an input cell is completely served is exactly equal to the time at which the link emanating from it is released.*

To prove the theorem, it is sufficient to show that a fair concentrating policy π^* releases each link i before any other fair policy π. The details of how this is accomplished may be found in [15].

Discussion: We now offer an intuitive explanation as to why a policy that always concentrates the residue outperforms one that does not. Referring to Figure 1a, consider the options faced by a work-conserving scheduling algorithm at this time (t_1). Note that whatever decision the algorithm makes, the residue will be the same. The scheduling algorithm just determines where to place the residue. If at time t_1, the algorithm concentrates the residue on Q_B then all of a_1's output cells will be sent and cell a_2 will be brought forward at time t_2. At time t_2, the algorithm selects between a_2 *and* the residue leftover from t_1. If on the other hand, the algorithm distributes the residue over both input queues at t_1, then at t_2 the algorithm can only schedule the residue leftover from t_1. No new cells can be brought forward. So, *on average*, a concentrating policy will bring new work forward sooner, increasing the diversity of its choice. This enables it to schedule more output cells in the next cell time.

To demonstrate that the fairness constraint is necessary, consider the example of Figure 1a again. Assume that the concentrating policy is *not* fair and concentrates the residue at Q_A at both times t_1 and t_2. From then on, only one input cell can be completed per cell time. An algorithm that distributes residue at time t_1 would actually perform better. However, if the concentrating policy is fair and at time t_2 concentrates the residue at Q_B, it will, in this case, complete input cells at the same time as the distributing policy. Thus, in addition to being socially correct, the fairness constraint provides a means for comparing scheduling policies pathwise, *instantaneously*.

5. Simulation Results

The following simulations support the observation that a "concentrating policy" outperforms other fair policies.

5.1. Scheduling Policies

The four scheduling policies that we compare are as follows:

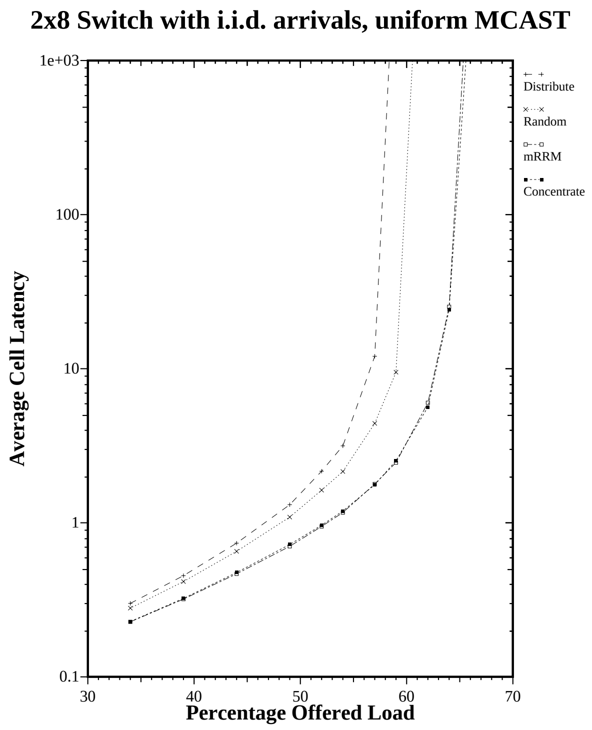
Concentrate: This policy always concentrates the residue onto as few inputs as possible. This is achieved by performing the following algorithm at the beginning of each cell time.

1. Determine the residue.
2. Find the input with the most in common with the residue. If there is a choice of inputs, select the one with the input cell that has been at the HOL for the shortest time. This ensures some fairness though not in the sense of the definition in Section 3 (see remark below).
3. Concentrate as much residue onto this input as possible.
4. Remove the input from further consideration.
5. Repeat steps (2)-(4) until no residue remains.

Remark: The allocation of residue uniquely defines which input queues are served. This algorithm does not guarantee that each input cell will remain at HOL for no more M cell times. In Section 6 we outline an algorithm, TATRA, which offers this guarantee in addition to concentrating residue in an efficient manner.

Distribute: This policy always distributes the residue onto as many inputs as possible. This is achieved by the following algorithm.

1. Determine the residue.



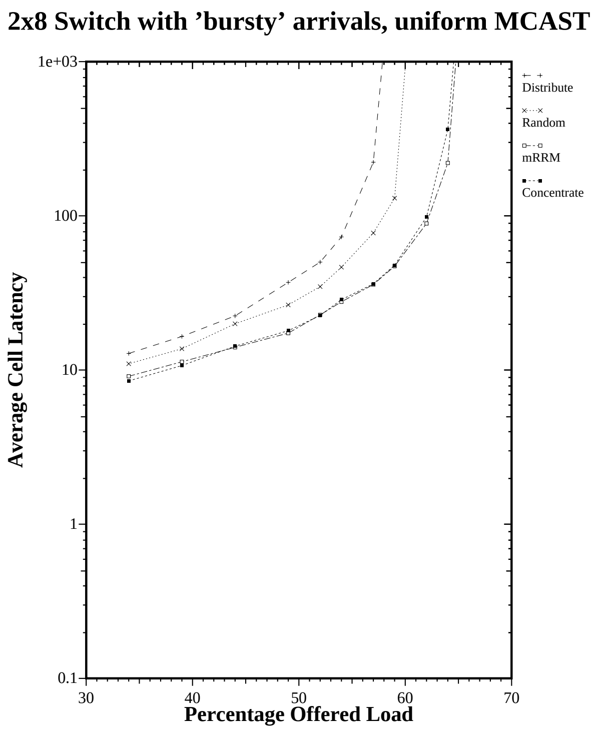


Figure 2: *Graph of average cell latency as a function of offered load for a 2×8 switch. Uncorrelated arrivals (left). Correlated arrivals (right).*

2. Find the input with at least one cell but otherwise the least in common with the residue. If there is a choice of inputs, select the one with the input cell that has been at the HOL for the shortest time.
3. Place one output cell of residue onto that input.
4. Remove the input from further consideration.
5. Repeat steps (2)-(4) until no inputs remain.
6. If residue remains, consider all the inputs again and start at step (2).

RANDOM: This policy is motivated by the work of Hayes et al. in [18], which is the multicast version of the algorithms described in [4] and [9]. Each output in turn, and independently of the other outputs, randomly selects one input from among those that request it.

MULTICAST ROUND ROBIN (*mRRM*): This policy is motivated by the algorithms described in [10, 11]. A single round-robin pointer is collectively maintained by all of the outputs. Each output selects the next input that requests it at, or after, the pointer. At the end of the cell time, the pointer is moved to one position beyond the first input that is served. Designed to be simple to implement in hardware, *mRRM* tends to concentrate the selection onto a small number of inputs, yet maintain fairness. Note that for a 2×N switch this algorithm performs almost identically to the *concentrate* algorithm.

5.2. TRAFFIC TYPES

We compare each scheduling policy for two different arrival processes:

UNCORRELATED ARRIVALS: At the beginning of each cell time, a cell arrives at each input with probability p independently of whether a cell arrived during the previous cell time.

CORRELATED ARRIVALS: Cells are generated using a 2-state Markov process which alternates between BUSY and IDLE states. The process remains in each period for a geometrically distributed number of cell times. The expected duration of the BUSY state is fixed at 32 cells (corresponding approximately to the maximum length of a segmented Ethernet packet). When in this state cells arrive at the beginning of every cell time and all with the same set of destinations. No cells arrive during the IDLE state.

For both types of traffic, each arriving multicast cell has a multicast vector that is uniformly distributed over all possible multicast vectors. As a result, for an M×N switch, the average fanout is N/2.

5.3. 2×8 SWITCH

Figure 2 compares the four scheduling policies for a 2×8 switch, with uncorrelated and correlated arrivals. As predicted by our theorem, the *concentrate* algorithm leads to an average cell latency that is much lower than for the *distribute* algorithm. In fact, as intuition suggests, the *distribute* algorithm is always the worst algorithm: it maximizes the HOL blocking.

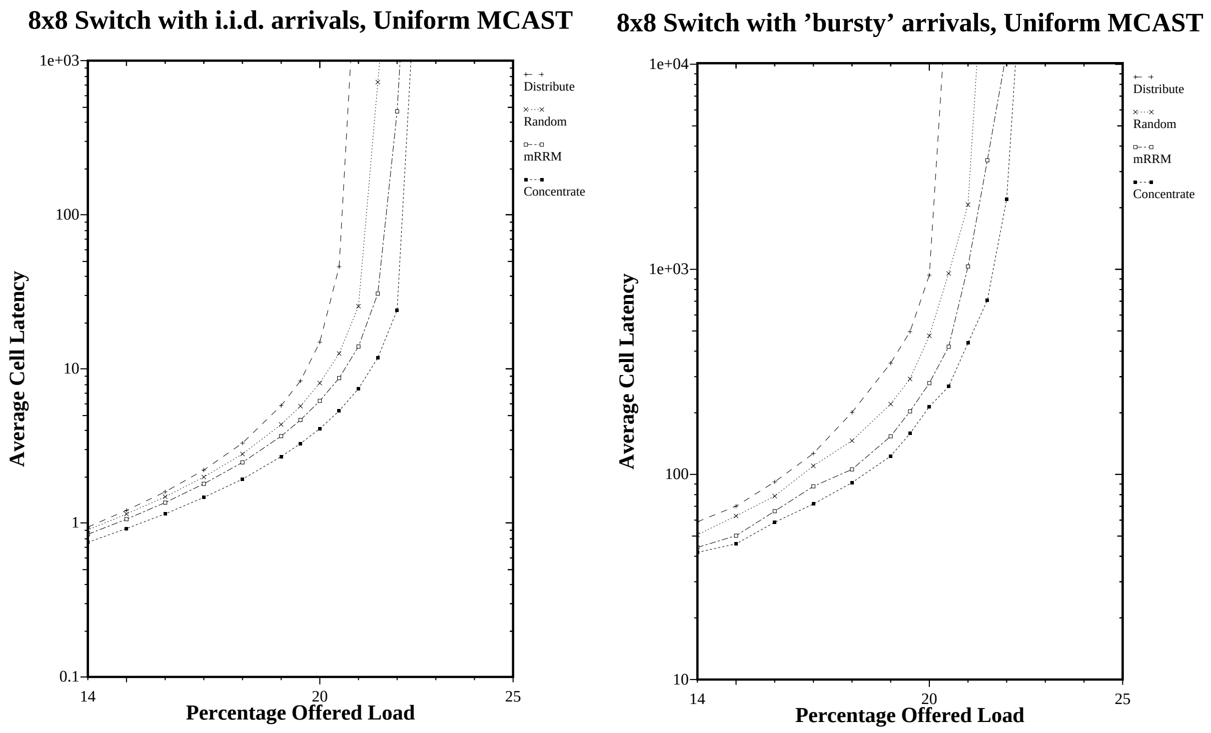


Figure 3: *Graph of average cell latency as a function of offered load for an 8×8 switch. Uncorrelated arrivals (left). Correlated arrivals (right).*

5.4. 8×8 SWITCH

Figure 3 compares the four scheduling policies for an 8×8 switch, with uncorrelated and correlated arrivals. Once again, the *concentrate* algorithm leads to an average cell latency that is much lower than for the *distribute* algorithm. This supports our argument, not proved in this paper, that the *concentrate* policy outperforms other algorithms.

Note that for an 8×8 switch *mRRM* performs worse than *concentrate*. This is because it does not necessarily concentrate the residue on as small a number of inputs.

6. TATRA: An M×N Multicast Switch Scheduling Algorithm

In this section we describe a multicast scheduling algorithm — TATRA. This algorithm achieves maximum residue concentration and is thus optimal among the class of fair, work-conserving algorithms. Due to limitations of space, we only provide a brief description of TATRA and show how it can be analyzed geometrically by mapping it onto a game similar to the popular block-packing game "Tetris". A more detailed analysis of its performance will be presented in forthcoming publications.

We begin by observing that the operation of a multicast switch can be mapped into a Tetris-like game in the following sense. Input cells will be mapped into Tetris blocks and since each input cell is composed of a set of output cells, this Tetris block will be an amalgamation of smaller blocks, one for each output cell. Suppose that an M×N switch begins processing input cells at time 1 having been idle before that time. At the beginning of time 1, input cells at HOL are dropped into an empty box which has N slots, one for each output. This is similar to Tetris where blocks are dropped into a bin and the aim is to achieve maximum packing. The main difference here is that whereas Tetris blocks are rigid and cannot be decomposed, work conservation will require that an input cell be broken down into its constituent output cells.

We now explain what work conservation and fairness mean in this context. If an input has a cell destined to output j, then this output cell is dropped into the slot corresponding to the j^{th} output. The output cell will drop to the lowest unoccupied position in that slot. This ensures *work conservation*, since gaps in the output slots lead to an idling of outputs. The order in which input cells are dropped is given by the priority rule arising from the *fairness constraint*. All scheduling policies must obey these rules. Figure 4a traces the evolution of a 5×5 switch operating under a sub-optimal policy π and Figure 4b traces its evolution under the optimal policy TATRA.

Evolution of the switch under π: At the beginning of time 1 input 1 drops cells for outputs 1, 3 and 5 into the corresponding output slots. After input 1, input 2 drops its output cells into slots 2, 3, 4 and 5. (Without loss of generality, we have assumed that input i has priority over input $i+1$). Proceeding thus, all 5 input cells are dropped into the box.

At the end of time 1, all output cells at the bottom-most layer of the box are discharged. That is, they are assumed to be served. Again, for the example in Figure 4a, this means that input 1 is completely served and can advance a new cell to HOL at time 2. Input 2 manages to discharge cells to outputs 2 and 4 and is left with a residue for outputs 3 and 5. Note that the *discharge* at any time is the set of output cells in the bottom-most layer and the *residue* is everything that's left behind.

At the beginning of time 2, all residue cells drop down one level. The new cell at input 1 drops into the box taking its place behind the residue in the manner described above. In Figure 4 the shaded cells are new input cells. Under π, at the end of time 2, input cell 2 is completely discharged along with a part of input cell 3. This goes on until *in four cell times π discharges all five original input cells and at least one fresh input cell (from input 1).*

Evolution of the switch under TATRA: During the first time instant, TATRA advances input cell 5 all the way to the bottom giving it priority over cells from inputs 2, 3 and 4; thus allowing it to depart at the end of time 1. Note that this does not conflict with the priority rule since it is impossible for inputs 2, 3 and 4 to leave at the end of time 1.

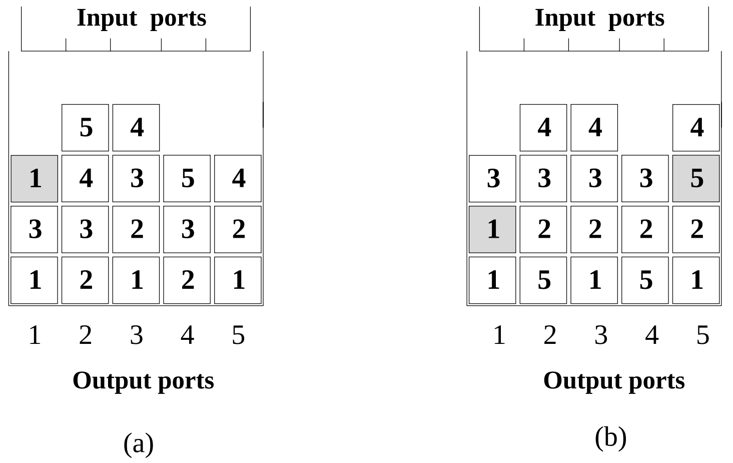


Figure 4:

At the beginning of time 2, under TATRA, two new cells (from inputs 1 and 5) are brought forth. Suppose that the order in which they drop down is again given by the priority rule (that is, 1 before 5)[1] Again, without delaying the departure of the original cells from inputs 2, 3 and 4, the priority of the new cells from inputs 1 and 5 can be bumped up as shown in Figure 4b.

Thus, *in four cell times* TATRA *discharges all five original input cells and at least two new cells (from inputs 1 and 5). Furthermore, individual input cells are served earlier under* TATRA *than under* π. The following observation is the key aspect of multicast switching (subject to fairness, work conservation and knowledge of HOL cells only) exploited by TATRA in scheduling output cells:

No input cell is completely served until every output cell that belongs to it has been discharged. This means that no input cell is completely served until the output cell belonging to it that is furthest from the bottom of the box has departed. Call this furthest output cell the *peak cell*. (In Figure 4 cells from inputs 1, 2, 3 and 4 destined to output 3 are peak cells.) Therefore, it pays to bump the priority of a fresh input cell j over that of an input cell i which is already in the box *so long as the peak cell of i is not raised in the process.*

7. CONCLUSION

Scheduling policies for input-queued multicast switches have been studied. We observed that when designing a multicast scheduling policy, it is important to determine the placement of the residue. Subject to a natural fairness constraint, it is shown that for a 2×N switch the optimum policy is one that always concentrates the residue. Our simulation results indicate that the concentrating policy also outperforms a distributing or random policy for M×N switches. In addition, we presented two fair algorithms: (1) *mRRM* which performs favorably when compared to

[1]It turns out that this ordering does not matter, since there are never more than M input cells in the box.

the concentrating algorithm, yet is simple to implement in hardware; and (2) TATRA which achieves maximum residue-concentration.

References

[1] Paxson, V; “Growth trends in wide-area TCP connections,” *IEEE Network*, vol.8, (no.4):8-17. July-Aug 1994.

[2] Eriksson, H.; “MBone: the Multicast Backbone,” *Communications of the ACM,*vol.37, (no.8):54-60. Aug 1994.

[3] Deering, S.E.; Cheriton, D.R.; “Multicast Routing in datagram internetworks and extended LANs,” *ACM Transactions on Computer Systems*, vol.8, (no.2):85-110. May 1990.

[4] Karol, M., Hluchyj, M., and Morgan, S. “Input Versus Output Queueing on a Space Division Switch,” *IEEE Trans. Comm*, 35(12) pp.1347-1356

[5] Li, S.-Q; “Performance of a nonblocking space-division packet switch with correlated input traffic,” *IEEE Trans. Comm*, vol.40, (no.1):97-108. Jan 1992.

[6] Lee, T.T.; “Nonblocking copy networks for multicast packet switching,” *IEEE J. Select. Areas Comm.*, vol.6, pp.1455-1467. Dec 1988.

[7] Turner, J.S.; “Design of a broadcast switching network,” *Proc. IEEE INFOCOM ’86*, pp.667-675.

[8] Huang, A.; “Starlite: A wideband digital switch,” *Proc. IEEE GLOBECOM ’84*, pp.121-125.

[9] Anderson, T., Owicki, S., Saxe, J., and Thacker, C. “High Speed Switch Scheduling for Local Area Networks,” *Proc. Fifth International Conference on Architectural Support for Programming Languages and Operating Systems* Oct 1992, pp. 98–110.

[10] McKeown, N.; Varaiya, P.; and Walrand, J.; “Scheduling Cells in an Input-Queued Switch,” *IEE Electronics Letters*, Dec 9th 1993, pp.2174-5.

[11] McKeown, N.; “Scheduling Algorithms for Input-Queued Cell Switches,” PhD Thesis, University of California at Berkeley, May 1995.

[12] Chen, M.; Georganas, N.D.; “A Fast Algorithm for multi-channel/port traffic scheduling,” *Proc. IEEE Supercomm/ICC ’94*, pp.96-100.

[13] Obara, H. “An Efficient Contention Resolution Algorithm for Input Queueing ATM Switches,”
Intl. Jour. of Digital & Analog Cabled Systems, vol. 2, no. 4, Oct-Dec 1989, pp. 261-267.

[14] Obara, H. “Optimum Architecture For Input Queueing ATM Switches,” *Elect. Letters*, 28th March 1991, pp.555-557.

[15] McKeown, N. and Prabhakar, B. “Scheduling Multicast Cells in an Input-Queued Switch,” *Technical Report: Computer Systems Lab, Stanford University.*

[16] Obara, H., Okamoto, S., and Hamazumi, Y. “Input and Output Queueing ATM Switch Architecture with Spatial and Temporal Slot Reservation Control” *Elect. Letters*, 2nd Jan 1992, pp.22-24.

[17] Karol, M., Eng, K., Obara, H. “Improving the Performance of Input-Queued ATM Packet Switches,” *INFOCOM ’92*, pp.110-115.

[18] Hayes, J.F; Breault, R.; and Mehmet-Ali, M; “Performance Analysis of a Multicast Switch," *IEEE Trans. Commun., vol.39, no.4*, pp. 581-587. April 1991.