SCHEDULING MULTICAST CELLS IN AN INPUT-QUEUED SWITCH *

NICK MCKEOWN
Departments of EE & CS
Stanford University.
Email: nickm@ee.stanford.edu.

BALAJI PRABHAKAR
BRIMS
Hewlett-Packard Labs, Bristol.
Email: balaji@hplb.hpl.hp.com.edu.

Abstract

In this paper we consider policies for scheduling cells in an 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 policies are assumed to be work-conserving, which means that cells may be copied to the outputs that they request over several cell times.

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. We prove that for a $2\times N$ switch, a policy that always concentrates the residue, subject to a natural fairness constraint, always outperforms all other policies.*

Simulation results indicate that this policy also performs well for more general $M\times N$ switches. We present a heuristic round-robin policy called mRRM *that is simple to implement in hardware, fair, and performs almost as well as the concentrating policy.*

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 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, 15].

In [16], Hayes et 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, subject to a natural fairness constraint. Finally, we present simulation results suggesting that this policy also performs well for more general M×N switches.

*This work was done when B. Prabhakar was a Visiting Scholar at the Department of EECS, University of California at Berkeley.

2 Our 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. Arriving multicast cells are assumed to contain a vector indicating which outputs the cell is to be sent to. For example, the 2 input and N output switch shown in Figure 1 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 constituent *output* cells. In the figure, the input cell at the head of queue Q_A will generate four output cells.

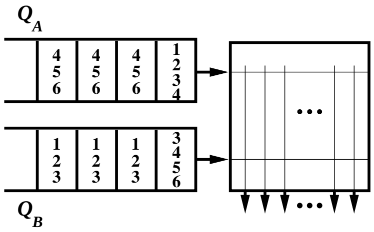

Figure 1: *2×N multicast crossbar switch with single FIFO queue at each input.*

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 1 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 now has to decide on 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 exclusively in Q_A or exclusively in Q_B (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 [16], 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, we consider only partial multicast policies here.

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.

Note that both the concentrating and distributing policies are defined in a constructive way (see also Section 6.1). Therefore, they are guaranteed to exist.

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, given a choice of inputs having the same number of output cells in common with the residue, it concentrates the residue on the input that has been at HOL for the shortest time.

Note that in the two input case this definition means that the residue alternates between the two inputs.

4 Main Result

We state and briefly discuss the main results of the paper. We believe that the following statement about general M×N multicast switch scheduling policies is true.

A scheduling policy that always "concentrates residue" at every possible instant subject to a natural fairness condition, leads to a higher output utilization than any other policy, with arbitrary arrival processes.

This belief is borne out by simulations presented in Section 6 and the sample path proofs presented for the 2×N case in the Appendix. Specifically, the following theorem is proved in the appendix.

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, with arbitrary arrival processes.*

Discussion: The theorem is proved in the appendix, where the fairness condition and a performance criterion are explicitly defined. Essentially, one policy is said to perform better than another if it leads to a higher output utilization. Although Theorem 1 covers the special case of 2×N switches, the methods developed are quite general and provide valuable insight into the general case. We believe that the arguments can be adapted with some suitable modifications to the case of M×N multicast switch scheduling policies.

5 Intuitive Explanation of Results

We now offer an intuitive explanation as to why a policy that always concentrates the residue outperforms one that does not.

Referring to Figure 1, 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 again in Figure 1. 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 rate as the distributing policy.

6 Simulation Results

In support of our argument that a "concentrating policy" outperforms all other fair policies, we present some simulation results.

6.1 Scheduling Policies

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 and find the input which has 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 fairness, although not in the sense of Definition 1 of the appendix.

2. Concentrate as much residue onto the input as possible and remove the input from further consideration.
3. Repeat steps (1) and (2) until no residue remains.

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

1. Determine the residue and 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.
2. Place one output cell of residue onto that input and remove the input from further consideration.
3. Repeat steps (1) and (2) until no inputs remain. If residue remains, consider all the inputs again and start at step (1).

Random: This policy is motivated by the work of Hayes et al. in [16], 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]. 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.

6.2 Traffic Types

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 (ignoring the null vector). As a result, for an M×N switch, the average fanout is $\mathrm{N}[2^{\mathrm{N}} - 1]/2^{\mathrm{N-1}}$, little more than N/2.

6.3 2×8 Switch

Figures 2 and 3 respectively compare 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.

6.4 8×8 Switch

Figures 4 and 5 respectively compare 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.

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. In particular,

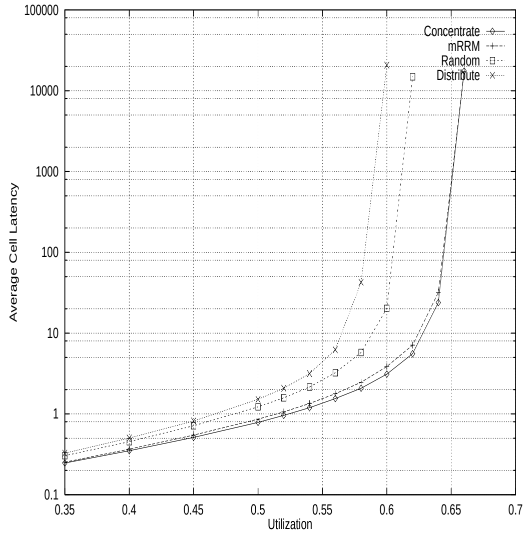

Figure 2: *Graph of average cell latency as a function of offered load for a 2×8 switch. Arrivals are uncorrelated.*

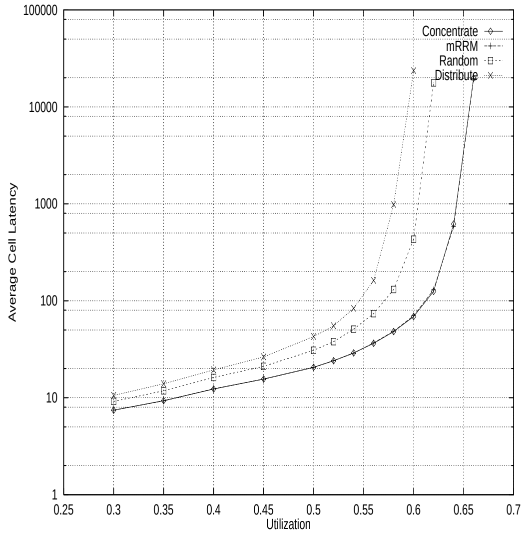

Figure 3: *Graph of average cell latency as a function of offered load for a 2×8 switch. Arrivals are correlated.*

we proved that, subject to a natural fairness constraint, 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 present the *mRRM* algorithm which performs favorably when compared to the concentrating algorithm, yet is simple to implement in hardware.

Acknowledgement

The authors thank Ritesh Ahuja of Stanford University for his prompt and timely help with the simulations and graphs.

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 Vs. 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 1993, pp.2174-5.

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

[12] 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.

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

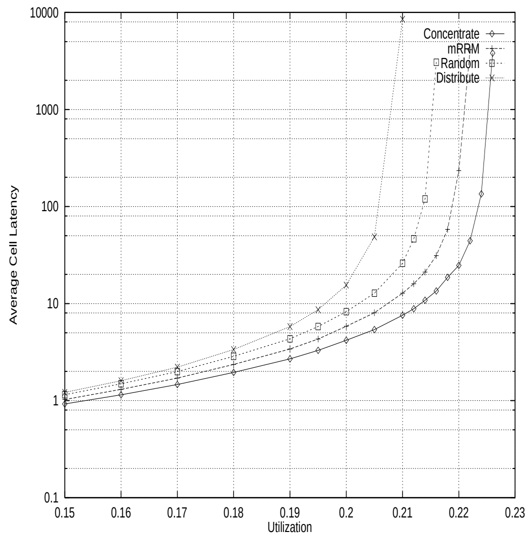

Figure 4: *Graph of average cell latency as a function of offered load for a 8×8 switch. Arrivals are uncorrelated.*

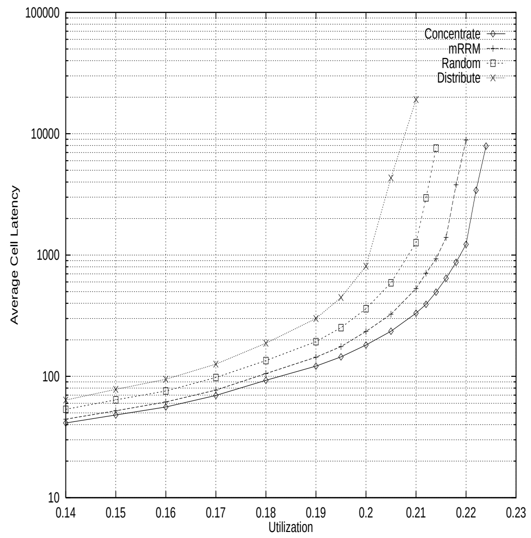

Figure 5: *Graph of average cell latency as a function of offered load for a 8×8 switch. Arrivals are correlated.*

[14] 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.

[15] Karol, M., Eng, K., Obara, H. "Improving the Performance of Input-Queued ATM Packet Switches," *INFOCOM '92*, pp.110-115.

[16] 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.

A Proof of Theorem 1

We first prove Theorem 1 under the static input assumption; that is, we assume that at time 0 both input queues have an infinite number of packets, placed according to some (possibly random) configuration. Fix one such configuration and label cells at inputs 1 and 2 as $\{a_i\}_{i=1,2,...}$ and $\{b_i\}_{i=1,2,...}$ respectively (see Figure 6). Once the theorem is proved under the static input assumption, a suitable modification of the performance criterion generalizes the same argument to dynamic inputs.

Definition 1 (fairness): A scheduling policy π is said to be *fair* if no cell from either input is held at HOL for more than one cell time.

Definition 2 (performance criterion): A fair scheduling policy π^1 for a 2×N multicast switch is said to *perform better* than another fair policy π^2 if every input cell, belonging to either input, departs no later under π^1 than under π^2.

As a consequence of Definition 1, a fair scheduling policy discharges the cell (or residue) at the HOL of each input buffer alternately, and the fairness constraint orders all input cells according to their departure times as follows: (1) $a_1 \leq_d b_1 \leq_d a_2 \leq_d b_2 \cdots$ if a_1 is the first cell to depart, and (2) $b_1 \leq_d a_1 \leq_d b_2 \leq_d a_2 \cdots$ if b_1 is the first cell to depart. Here $a \leq_d b$ is to be read as "a departs no later than b".

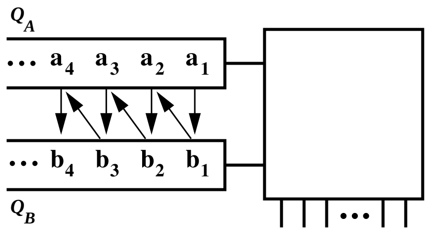

Figure 6:

Without loss of generality, we assume the first ordering for cells and $link$ them in a vertical or oblique fashion as shown in Figure 6. 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. The vertical link between a_i and b_i is labelled l_{2i-1} and the oblique link between b_i and

a_{i+1} is 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.*

Fact 3: *The residue concentrating fair policy, π^*, never takes two cell times to release a link. As a consequence, under π^*, the cell at the arrow end of any link is always a fresh cell.*

In light of Fact 2, Theorem 1 is proved if we show that the fair concentrating policy π^* releases each link i no later than any other policy π. To this end consider the plots in Figure 7. Each plot is a "time-link graph" showing the time a policy releases a certain link. Note that in Figure 7 the diagonal line (of slope 1) is the time-link graph of the worst policy - that is, this policy releases precisely one link per unit time. Similarly, the line of slope 1/2 is the time-link graph of the best policy - one that always releases 2 links per unit time (this is only possible if there is no contention at all). Clearly, the time-link graphs of all fair policies lie between these two extremes.

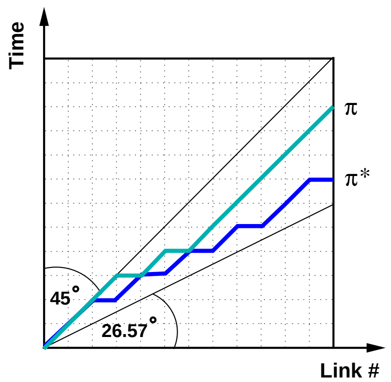

Figure 7: *Time-link graphs of a non-concentrating policy, π, and the concentrating policy, π^*.*

Thus, proving Theorem 1 is equivalent to showing that the time-link graph of the residue concentrating policy π^* *lies below* that of any other non-concentrating policy. In other words, it is sufficient to prove the following assertion.

Assertion 1: *The time-link graph of the optimal scheduling policy π^* is never above that of any other scheduling policy.*

The proof of the above assertion requires the following lemma.

Lemma 1: *Under the ordering of cells mentioned in Fact 1, consider any two scheduling policies π^1 and π^2. Suppose that the two policies are serving link l_i, which originates in cell c and terminates in cell d, say. Suppose also that cell d is a fresh cell for both π^1 and π^2. Denote by r_1^i (respectively, r_2^i) the cell c or some residue of it under π^1 (respectively, π^2). That is, r_k^i is the residue or cell connected to d by link l_i under policy π^k for $k = 1, 2$. If the policies are fair, then either $r_1^i \subset r_2^i$ or $r_2^i \subset r_1^i$.*

Proof: First of all, suppose that link l_{i+1}, originating in cell d and terminating in cell e, follows link l_i. Since $r_k^{i+1} = r_k^i \cap e$ (here r_k^{i+1} is the residue at link l_{i+1} under policy π^k), $r_k^i \supset r_k^{i+1} \supset \cdots \supset \Phi$. That is, the residue monotonically decreases to the empty set. Call this property the "residue-monotonicity property". Observe also that if $r_1^i \subset r_2^i$, then $r_1^{i+1} \subset r_2^{i+1}$. This is because $r_1^{i+1} = r_1^i \cap e$ and $r_2^{i+1} = r_2^i \cap e$. Call this property the "residue-domination property".

Now, if either $r_1^i = c$ or $r_2^i = c$, then Lemma 1 is trivially true. If neither equals c, then they are proper subsets of c which, using the "residue-monotonicity property" in the reverse direction, implies that $r_k^i \subset r_k^{i-1} \subset r_k^{i-2} \subset \cdots$ for $k = 1, 2$. Since the size of the residue is bounded by the number of outputs, either r_1^m or r_2^n will soon equal a fresh cell for some $m, n < i$. Without loss of generality suppose that $m < n$; that is r_2^n equals a fresh cell first while r_1^n is still a proper residue. Since r_2^n is a fresh cell, this means that $r_1^n \subset r_2^n$. Using the "residue-domination property" we are now able to conclude that $r_1^i \subset r_2^i$, thus proving Lemma 1. ∎

Proof of Assertion 1: Let π be any other policy. For ease of exposition, consider, as in Figure 7, a time-link graph showing the evolution of π and π^*. We will use induction over time to show that the time-link graph of π is always above that of π^*. Assume that π^* is optimal upto time n, we will show that it is optimal at time $n + 1$. Consider the following cases.

(1) At the beginning of time n, π is two or more links behind π^*: During time n, π can release at most two links while π^* must release at least one link. Therefore, at the beginning of time $n+1$, π is at least one link behind π^* which implies the optimality of π^* at the beginning of time $n+1$.

(2) At the beginning of time n, π is one link behind π^*: Again during time n, π can release at most two links while π^* must release at least one link. Therefore, at the beginning of time $n+1$, π has at best caught up with (but not overtaken) π^*.

(3) At the beginning of time n, π and π^* are both at the same link: Now, the time-link graph of π^* can go over that of π if, and only if, during time n, π^* releases only one link while π releases two links. By contradiction, we argue that this is impossible. First, suppose that the link at which π and π^* find themselves at time n is l_i, and suppose also that l_i originates in cell f and terminates in cell g. It is now necessary to determine the status of f and g (whether they are whole or fragmented), first under π^* and then under π.

(i) The status of l_i under π^*: By Fact 3, cell g must necessarily be a fresh cell while cell f can either be whole (fresh) or fragmented (residue).

(ii) The status of l_i under π: Now, under π, f and g cannot both be residues. Because this implies that π must have distributed residue at l_i at time $n-1$ and, therefore, must have first arrived at l_i at time $n-1$. However, since π^* is presently serving l_i, by Fact 3 it could not have been serving link l_i at time $n-1$ as well. Thus π^* must have arrived at link l_i at time n while π must have arrived at l_i at time $n-1$. This contradicts the optimality of π^* at time $n-1$. Therefore, under π, cell g must be fresh whereas cell f can either be whole or fragmented.

Observe from cases (i) and (ii) above that the whole cell g is common to both π^* and π at time n. Under π^* call the other cell at link l_i (this cell is f or a fragment of it) f_*, and under π call it f_π. Note that Lemma 1 now implies that either $f_* \subset f_\pi$ or $f_\pi \subset f_*$. Recall that, for the sake of contradiction, we have assumed that during time n π^* releases only one link whereas π releases two links. Given this, it follows that $f_\pi \subset f_*$ and the inclusion is proper.

Now if f_* is also a fresh cell then, since g is a fresh cell, we deduce that at time $n-1$ π^* must have released links l_{i-1} and l_{i-2}. In particular this means that π^* was at link l_{i-2} at time $n-1$. However, since f_π is strictly contatined in f_*, f_π is a proper residue which means that π was at link l_{i-1} at time $n-1$. Again this contradicts the optimality of π^* at time $n-1$.

If f_* is not a fresh cell, then using the "residue-domination property" backwards we arrive at a link, say l_j, where (at some time $m < n$) π^* was facing two fresh cells while π was facing a fresh cell and a proper residue. The above argument then leads us to conclude that π^* was not optimal at time $m-1$. This contradiction proves the assertion and Theorem 1. ■

Now, to extend the above proof to the dynamic-input case, the performance criterion is first modified as follows.

Definition 3 (perfomance criterion 2): A **possibly idling**, fair scheduling policy π_1 for a 2×N multicast switch subject to dynamic inputs is said to *perform better* than another **non-idling**, fair policy π_2 if every input cell (belonging to either input) departs no later under π_1 than under π_2.

Thus, given a *non-idling, non-concentrating* scheduling policy π for a 2×N multicast switch subject to dynamic inputs, it is easy to see that a *possibly idling, concentrating* policy π^* exists which performs no worse than π.

As before, consider two copies of the switch subject to the *same inputs*, one operating under π and the other operating under π^*. For convenience, we label the former switch S and label the latter switch S^*. Starting at time 0, the switches will experience identical periods when both inputs are active alternating with identical periods when at least one of the inputs is inactive. Since π^* always finishes earlier than π, we will allow S^* to idle whenever it is ahead of S in terms of releasing links. This idling time is then credited to π^* and can be used to measure how much better it is performing in comparison with π in processing the particular input that is applied.