Tetris Models for Multicast Switches

Balaji Prabhakar
BRIMS
Hewlett-Packard Labs, Bristol.
Email: balaji@hplb.hpl.hp.com

Nick McKeown
Departments of EE & CS
Stanford University.
Email: nickm@ee.stanford.edu

Jean Mairesse
BRIMS
Hewlett-Packard Labs, Bristol.
Email: jem@hplb.hpl.hp.com

Abstract

This paper presents a unified approach to the analysis of schedulers for input-queued multicast ATM *switches. It is shown that the general multicast scheduling problem can be mapped onto a variation of the popular block-packing game Tetris. Within this common framework, one is able to describe and analyze any multicast scheduling policy in an intuitive and geometric fashion. One such policy -* TATRA*, is described and its salient features are discussed. This builds on earlier work presented in [4] and [5].*

1 Introduction

This paper presents a unified approach to the analysis of schedulers for multicast ATM switches. A number of architectures have been proposed for the design of multicast switches [1, 2, 3]; however, we restrict attention to input-queued crossbar switches. This allows us to exploit the inherent copying properties of the crossbar fabric; that is, the ability of the crossbar fabric to copy an input cell to a plurality of outputs in one cell time. By choosing an input-queued architecture, we are also able to design high-speed switches without being constrained by the (currently unsatisfiable) need for high memory bandwidth required for the implementation of high-speed output-queued switches.

Briefly, we now describe our model. The switches are assumed to have M input ports and N output ports. Each input maintains a single FIFO queue for arriving multicast cells. It is also assumed 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 idle as long as there is an input cell destined to it; and it 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 goal 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. Theoretical results and simulations were presented in in [4] and [5] in support of concentrating the residue on as few inputs as possible. In [5], a sample path proof established the optimality of the residue-concentrating policy for the case of 2×N switches, and simulation results presented in [4] and [5] supported the residue-concentrating heuristic for general M×N switches.

As a starting point we take the simple description of the Tetris analogy to multicast scheduling presented in [4]. We make this description more precise and further elaborate upon one such policy, TATRA. In particular

an algorithmic implementation of TATRA is presented and we conclude by exploring some its properties.

2 Tetris Models

Recall 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. The size of this vector is referred to as its *fanout*. It is imagined that each input cell is constituted of a number of identical copies of itself - one destined to each output. Each of these copies is then referred to as an *output cell*. (The terminology is clarified shortly with the aid of Figure 1.) An input cell is considered to be completely served when all of its constituent output cells have been discharged to their respective outputs.

The class of policies considered: In addition to requiring that policies be fair and work-conserving, we also require that they assign departure dates to input cells once the cells advance to HOL. This *departure date* (DD) is some number between 1 and M specifying how long, from the current cell time, the input cell will be held at HOL before being discharged. Once assigned, the DD of a cell cannot be changed (except for decremented it by 1 at the end of each cell time). Clearly, this class of policies is smaller than the class of fair and work-conserving policies, since fairness allows one to reassign departure dates to input cells at HOL (but not beyond M cell times).

Class of inputs: Following [5], we make the "static input assumption" for switches. That is, it is assumed that at time 0 an infinity of cells has been placed in each input buffer according to some (possibly random) configuration. The description of the policies holds equally well for "dynamic inputs" since scheduling is based only on cells at HOL and there is no look-ahead.

2.1 Tetris models: a sketch

We map the operation of an M×N multicast switch onto a Tetris-like game in the following fashion. Input cells are mapped onto 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. Upon assignment of DDs, the input cells at HOL will be dropped into a box of size M×N - each slot in the box capable of holding one output cell. Each of the N columns of the box holds cells destined to a specific output; i.e., column j holds cells destined to output j. Refer to the example of a 5×5 switch shown in Figure 1. In this figure and the others, the labels on the cells denote the input port from which the cell has arrived. The cells in the bottom-most row of the box in Figure 1 at columns 1, 3 and 5 are all identical copies of a cell from input 1 destined to outputs 1, 3 and 5 (note that this input cell has a fanout of 3). Similarly, the cell at the HOL of input 2 wishes to access outputs 2, 3, 4 and 5.

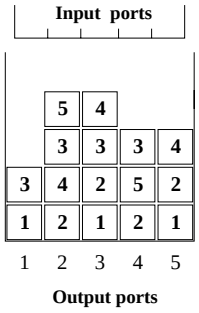

Figure 1: *An example. Cells from inputs 1, 2, 3, 4 are assigned DDs 1, 2, 3, 4 respectively, while the cell from input 5 is assigned a DD of 4.*

Suppose that, at time n, the switch is to schedule k input cells which have advanced to HOL; i.e., there have been precisely k departures in the previous cell time (recall the "static assumption" on inputs). After the

scheduler has assigned DDs to these input cells, they are dropped into the box which currently holds the cells or residues at the HOL of the other (M - k) inputs[1]. Each new output cell may occupy any position in its appropriate output slot as long as it (1) does not alter the DD of any other cell, and (2) does not leave any slots beneath it unoccupied. Again, referring to Figure 1, note that there are no unoccupied slots between cells in any output column. The reason for this will become clear momentarily.

At the end of time n, all output cells at the bottom-most layer of the box are discharged. That is, they are assumed to be served. For the example in Figure 1, 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. It should now be clear that we do not allow unoccupied slots in output columns because of the restriction to policies which are work-conserving. That is, these gaps may lead to an idling of the output in the future.

At the beginning of time $n+1$, all residue cells drop down one level and their DDs are decremented by one. Those inputs which have been completely served in the previous cell time advance a new cell to the HOL. These cells are assigned DDs, and the cycle continues.

This is reminiscent of 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 at times require that the various output cells constituting an input cell depart at different cell times. Note also that *there are never more than M input cells in the box.* Thus when an input cell is dropped into the box, it is *guaranteed to depart within M cell times*, since input cells arriving in the future do not alter its departure date. This automatically ensures fairness.

2.2 Tetris models: the details

We now make the description of Tetris models given in Section 2.1 mathematically precise. As noted earlier, if a plurality of cells advance to HOL at the beginning of a cell time, it is important to determine the order in which they are assigned DDs. In general, it is better to allow this ordering to depend on the constitution of the current residue and of the new cells. However, for simplicity, we choose the following fixed ordering: for $i < j$ the new cell at input i will be assigned its DD before the new cell at input j. Before proceeding to define a scheduling algorithm, we make the following definitions.

Tetris Box. The *Tetris box* is specified by a matrix $T_{i,j}, 1 \leq i \leq M, 1 \leq j \leq N$, where the rows are numbered from *bottom to top* and the columns are numbered from left to right. Thus $T_{1,1}$ is the bottom-left slot of the box and $T_{M,N}$ is the top-right slot.

Occupancy Set of an Input Cell. The *occupancy set* of the cell or residue at the HOL of input l at time n is given by $O_l(n) = \{T_{i,j} :$ an output cell of l resides at $T_{i,j}$ at time $n.\}$

Peak Cell and Departure Date. An output cell belonging to input l is said to be a

[1] The order in which the scheduler assigns DDs to the k new cells is important, because if the cells contend for the same outputs it may not be possible to assign them DDs in parallel. For example, suppose that two of the new cells have a fanout of 1 and are the only cells contending for a specific output. Then, deciding who goes first is important since no two cells in an output column can have the same DD. In general, the order of assignment of DDs can either be pre-fixed or made to depend upon some criterion (e.g., size of fanout). However, for ease of exposition, we will assume a pre-fixed ordering.

peak cell at time n if it occupies a slot in the row whose number is given by $\max\{i : T_{i,j} \in O_l(n)\}$. The corresponding row number is the *departure date (DD)* of the input cell at time n. That is, the peak cell of an input is one which is furthest from the bottom of the box and the distance from the bottom is its departure date. Note that there may be more than one peak cell for a given input.

Scheduling Policy. Given $k \leq$ M new cells $c_1, c_2, \cdots, c_k$ at the HOL of inputs $i_1 < i_2 < \cdots < i_k$ at time n, a *scheduling policy* Π is given by a sequence of decisions $\{\pi(n), n \in \mathcal{Z}^+\}$, where $\pi(n)$ associates to each of $c_1, c_2, \cdots, c_k$ (in that order) the corresponding occupancy sets $O_{c_1}(n), O_{c_2}(n), \cdots, O_{c_k}(n)$ subject to the following rules.

1) No cell should change the DD of a cell that is already scheduled. This means that no peak cells should be raised or lowered.

2) For every i and j, if $T_{i,j}$ and $T_{i+2,j}$ are occupied, then so must $T_{i+1,j}$ be. That is, there should be no gaps in the output columns.

Algorithm for Π. Given the above definitions, the algorithm for implementing a policy Π just requires a specification for transitioning from one cell time to the next. The following steps enumerate the procedure.

a) At the end of time n, all output cells occupying slots in the set $\{T_{1,j}, 1 \leq j \leq N\}$ are discharged. In particular, input cells (or residues thereof) with DDs $= 1$ are completely served.

b) Each output cell occupying slot $T_{i,j}$ for i and j in the set $\{2 \leq i \leq M, 1 \leq j \leq N\}$ is assigned to the slot $T_{i-1,j}$. The occupancy set, peak cell(s), and the departure dates of the residue are recomputed. For example, the occupancy set of the residue at input l is given by $O_l(n+1) = \{T_{i,j} : T_{i+1,j} \in O_l(n)\}$. From this peak cells and DDs are easily computed.

c) New cells advancing to HOL are then scheduled according to $\pi(n+1)$.

2.3 An Example

Consider the example of Figure 1 again. The input cells were scheduled in the order 1, 2, 3, 4 and 5. The occupancy sets are:
$O_1(1) = \{T_{1,1}, T_{1,3}, T_{1,5}\}$
$O_2(1) = \{T_{1,2}, T_{2,3}, T_{1,4}, T_{2,5}\}$
$O_3(1) = \{T_{2,1}, T_{3,2}, T_{3,3}, T_{3,4}\}$
$O_4(1) = \{T_{2,2}, T_{4,3}, T_{3,5}\}$
$O_5(1) = \{T_{4,2}, T_{2,4}\}$.

From this the peak cells and DDs are easily calculated:
$PC_1(1) = O_1(1); \quad DD_1(1) = 1$
$PC_2(1) = \{T_{2,3}, T_{2,5}\}; \quad DD_2(1) = 2$
$PC_3(1) = O_3(1) - \{T_{2,1}\}; \quad DD_3(1) = 3$
$PC_4(1) = \{T_{4,3}\}; \quad DD_4(1) = 4$
$PC_5(1) = \{T_{4,2}\}; \quad DD_5(1) = 4.$

At the end of cell time 1, input 1 is completely served and advances a new cell to HOL. Suppose that this new cell wishes to access outputs 1 and 5. Figure 2 shows two different ways of scheduling the new cell.

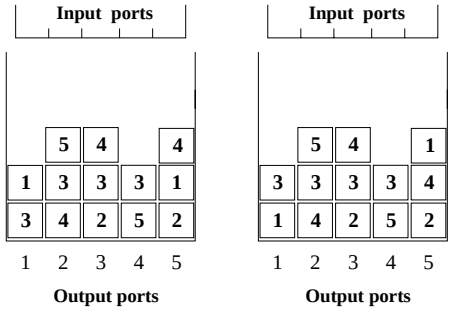

Figure 2:

3 TATRA: A multicast scheduling algorithm

In this section we formally describe a specific multicast scheduling algorithm, TATRA, first introduced in [4] and discuss its optimality properties.

Again we assume that the switch has been

idle prior to time 0 and that the "static input assumption" holds. We denote by $\Pi^* = \{\pi^*(n), n \in \mathcal{Z}^+\}$ the policy TATRA. Since TATRA schedules input cells solely based on their DDs, we assume that this number is stamped upon all the output cells belonging to a specific input cell (both peak and non-peak cells). In this section, the departure date of an input cell is not merely a number to be inferred *after* scheduling the cell. Rather, as will be shown below, it is used in the very process of scheduling.

For time $n \geq 1$, the algorithm is specified by the following steps.

(1) At the beginning of time n, $\pi^*(n)$ assigns a DD to each new cell at HOL according to the formula given in Equation 1 below. The order in which the DD is assigned when there is a plurality of new cells is in increasing order of their input port numbers.

(2) Each new output cell is dropped to the lowest possible level in the appropriate output slot, without getting ahead of another cell whose DD is less than or equal to its own.

Remark: It follows that a non-peak cell cannot be ahead of a peak cell unless it has the same DD as the peak cell. If such a non-peak cell exists, we call it a *pseudo-peak cell*. Corresponding to each output slot, there is thus a (possibly empty) column of peak/pseudo-peak cells. This column is called the *peak column*.

(3) Cells in the bottom-most row are discharged. New DDs are computed for the residue cells. Time is advanced to $n + 1$.

Using the terminology introduced in the remark above, and from the constitution of a new input cell its DD is computed as follows

$$\text{DD} = \max\{\text{height of peak columns across fanout}\} + 1 \quad (1)$$

3.1 An Example

By applying the above algorithm to the example of Figure 1, it is fairly easy to see that TATRA schedules the cells as shown in Figure 3 below.

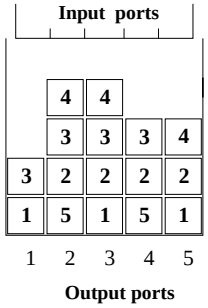

Figure 3:

3.2 Properties of TATRA

In this subsection we discuss some desirable properties possessed by TATRA. Due to a lack of space, the properties are stated and only briefly explored. There is no claim to completeness, indeed not all conditions under which properties hold are stated. However, we have strived to be as clear as possible.

Property 1: Under TATRA there is guaranteed to be a departure every cell time. This is equivalent to the statement that there is a peak cell in every row of the Tetris box. To see this, merely observe that (1) under every peak cell there is a column of peak (or pseudo-peak) cells, and (2) the cell furthest from the bottom of the box must be a peak cell.

Property 2: Suppose that we are given the occupancy sets, $O_l(n)$ and $O_m(n)$, of two input cells l and m. If $T_{i,j} \in O_l(n)$ and $T_{i+k,j} \in O_m(n)$ for some j and for some $k > 0$, then it is impossible that there exists an output $j' \neq j$ such that $T_{i',j'} \in O_l(n)$ and $T_{i'-k',j'} \in O_m(n)$, where $k' > 0$. That is occupancy sets cannot "criss-cross". This follows from the fact that output cells are arranged in output columns according to a monotonic increase of

DDs. The "no criss-crossing" property corresponds to residue concentration (otherwise, there could be a situation where some output cells are discharged from one input and some from another input leaving residues on both).

An interesting point is that as opposed to the $2 \times N$ case, in the general case of M×N switches, residue concentration may not be the "optimal" thing to do at all times. The following counter-example shows that by deliberately distributing residue at some times one can benefit at future times.

3.3 Counter-example

Consider a 4×4 switch. Assume that there is only a finite number of input cells to be scheduled (they all are in the input queues at time 0 and the switch has been idle prior to that time). There are 6 input cells in all, 3 in input 1, and 1 each in inputs 2, 3 and 4. Their fanout (FA) is given by: (I/p 1) $FA = \{1,2,3\}$ then $FA = \{3\}$ then $FA = \{4\}$, (I/p 2) $FA = \{1\}$, (I/p 3) $FA = \{1,2\}$, and (I/p 4) $FA = \{2,3\}$.

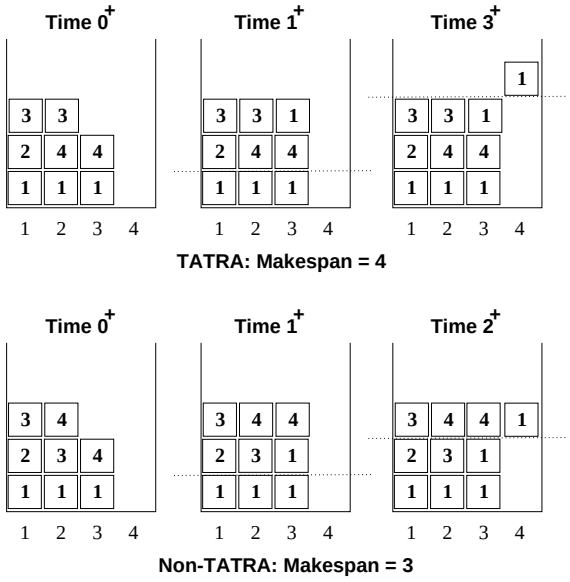

Figure 4:

From Figure 4, one gathers that the last input cell leaves earlier under non-TATRA than under TATRA. Note that non-TATRA does not concentrate residue (at time 2, there are residues from inputs 1, 3 and 4 as opposed to just 1 and 3 for TATRA). One might think that the policy non-TATRA has somehow anticipated future inputs and has doctored a situation that is favourable for the specific input pattern[2]. However, more elaborate counter-examples exist which are not input-specific.

Conclusions: TATRA operates by scheduling cells as early as possible, given knowledge of HOL cells and the current state of the residue. Each input cell is pushed as far down towards the bottom of the box as allowed, and is therefore able to depart early. Thus, despite the counter-examples, the heuristic of residue-concentration and the algorithm TATRA are simple and effective ways of scheduling multicast cells.

Acknowledgement: B. Prabhakar and N. McKeown thank R. Ahuja of Stanford University for several interesting discussions on multicasting switching.

References

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

[2] Turner, J.S.; "Design of a broadcast switching network," *Proc. IEEE INFOCOM '86*, pp.667-675.

[3] Huang, A.; "Starlite: A wideband digital switch," *Proc. IEEE GLOBECOM '84*, pp.121-125.

[4] Prabhakar, B. and McKeown, N.; "Designing a Multicast Switch Scheduler," *Proc. of the 33rd Annual Allerton Conference*, 1995.

[5] McKeown, N. and Prabhakar, B.; "Scheduling Multicast Cells in an Input-queued Switch," *Proc. IEEE INFOCOM '96.*

[2] Input 1 holds 3 cells and thus will take 3 cell times to be served out completely. Non-TATRA has ensured that input 1 is able to send out cells in each successive cell time.