Resequencing Cells in an ATM Switch

Jonathan S. Turner

WUCS-91-21

September 20, 1994

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

Abstract

ATM switching systems are required to maintain ordering for cells within a given virtual circuit. This is most commonly achieved using a switching system in which cells in a given virtual circuit are routed along a common path through the switching system. This solution has the drawback that it can lead to significant virtual circuit blocking unless the switching network is engineered with a large speed advantage, relative to the external links. Switching systems in which cells are routed independently, on the other hand, can balance the load dynamically and operate with a minimal speed advantage. This report describes a resequencer that can be used with such a switching system to put cells back in the proper order, gives an economical implementation of such a resequencer and reports on a simulation study of its effectiveness.

Resequencing Cells in an ATM Switch

Jonathan S. Turner

Figure 1 illustrates an ATM switching system, based on a three stage Beneš network topology with fixed path routing. Switching systems using this architecture are being developed by several manufacturers [**?**, **?**]. The left hand diagram shows several virtual circuits passing through the network, each consuming a specified fraction of the capacity of the network's internal data paths (for example, the virtual circuit that enters at the top left consumes 60% of the capacity of the data paths it is routed through). In these networks, virtual circuits may share a common data path so long as the sum of their bandwidth requirements is less than the system's internal data path capacity (normalized to 1 here). Notice that if one attempts to establish a connection with a traffic requirement of 0.7 between the input marked a and the output b, it will be blocked, as there is no path joining these two points with enough available bandwidth to support the connection. To ensure that blocking never occurs, the Beneš network can be engineered so that it's internal data paths are three times faster than the external links. For a $2k-1$ stage Beneš network, the speed advantage required to achieve nonblocking operation for point-to-point traffic is $(2k-1) - 2(k-2)/n^{1/k}$ where n is the number of inputs and outputs to the network (see [**?**, **?**] for details).

In the case of broadcast or multicast virtual circuits, an even larger speed advantage is required to achieve nonblocking operation. The right hand diagram in Figure 1 shows a multicast virtual circuit with a bandwidth requirement of 0.6. Note that any connection request from c that has a bandwidth requirement exceeding 0.4 will be blocked. Virtual circuits such as this one, in which branching occurs in the first stage, can lead to large amounts of blocking, but cannot be avoided in general. Nonblocking operation in a three stage Beneš network requires a speed advantage of $\sqrt{n}$ in this case.

ATM switching systems such as the one described in [**?**] allow cells passing through the system to follow different paths. This allows even load distribution and eliminates the need for any speed advantage beyond that required to absorb short-term contention. On the other hand, this approach does allow cells to exit the system in a different order from that in which they entered. This can cause problems for applications, which often require that cells be received in the same order as they were sent. In this paper we describe a mechanism for reordering cells after they exit an ATM switch that effectively eliminates this problem, detail an implementation of this mechanism and report on simulations performed to evaluate its effectiveness.

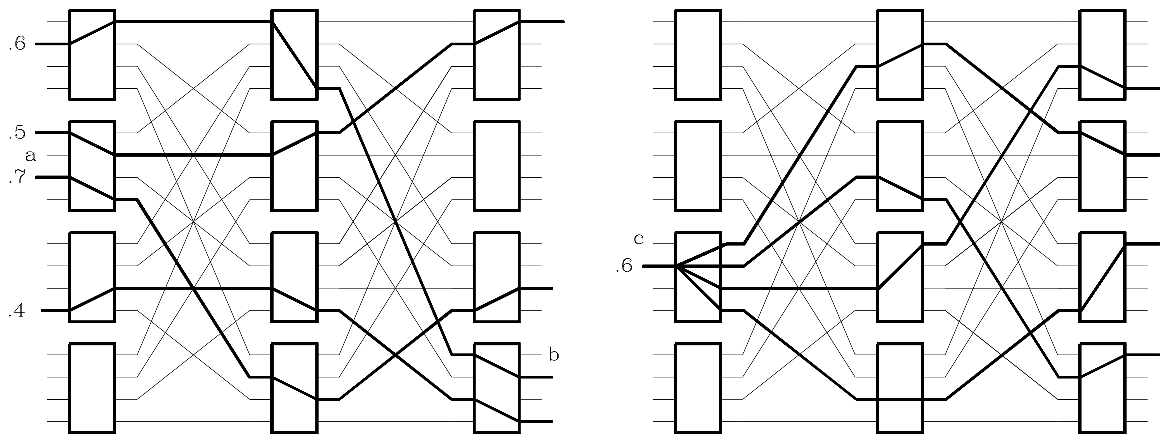

Figure 1: Fixed Path Routing in ATM Switches

1. Overview of Resequencing Mechanism

Figure 2 shows the overall organization of an ATM switching system that includes a resequencing mechanism. As cells are read from the input buffers on the left side of the switch, a time stamp is added by a *Time Stamp Circuit* (TSC), which records the time at which the cell enters the switch (the TSCs are all driven from a common clock). When cells leave the switch they enter a resequencing buffer which is managed by a *Resequencing Buffer Controller* (RBC). As the cell leaves the switch, the RBC computes its age from the time of entry and current time, then stores the age and the cell's location in the output buffer in an internal table.

When a cell is to be read from the buffer, the buffer controller selects the oldest cell in the buffer and provides its address to the output buffer which then sends it out. If the oldest cell is "not old enough," no cell is output. The purpose of this is to allow cells stored within the switching network to catch up with cells that have already reached the output buffer. This mechanism can ensure that all but a vanishingly small fraction of misordered cells are put back in order. Simulation results described below, indicate that a 64 port resequencing buffer provides ample protection against out-of-sequence cells for a network such as that described in [**?**].

The key element in this mechanism is the buffer controller. In our judgement, a reasonable buffer controller should have a complexity of no more than 25% of the buffer itself. The design we present below can, with careful implementation, meet that target. The buffer controller has two main functions. First, during a buffer output cycle, it must select the cell with the oldest time stamp, check that the cell is old enough and if so, supply the cell's address to the buffer. Second, during a buffer input cycle, it must select an empty slot that the arriving cell can be placed in and supply its address to the buffer controller.

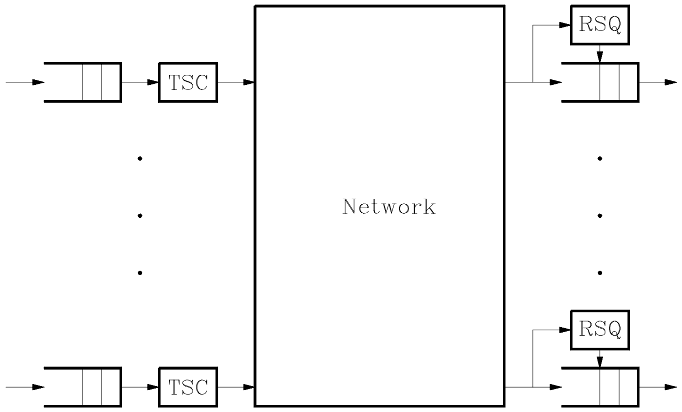

Figure 2: ATM Switch with Resequencing Mechanism

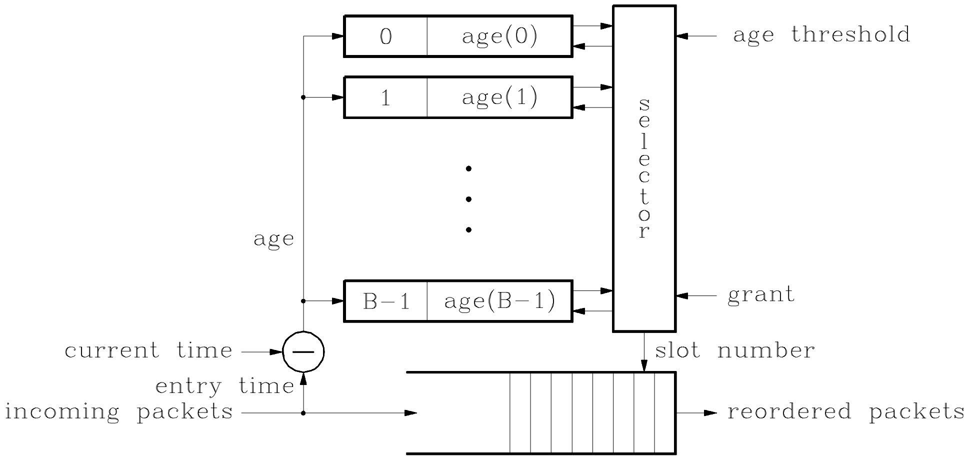

Figure 3: Resequencing Buffer Organization

2. Controller Organization and Operation

Figure 3 shows the organization of the resequencing buffer and buffer controller. The buffer is organized as a set of *slots* with each slot being large enough to hold a single ATM cell. The controller is organized similarly, with a *control slot* for each buffer slot. Each control slot contains two pieces of information, a *slot number* which specifies the buffer slot it is associated with and the age of the cell (if any) stored in that slot. The *selector*, at the right, selects the oldest cell during output operations, compares that cell's age to a given age threshold, and if appropriate, forwards the cell's slot number to the buffer which then forwards the cell to the downstream circuitry. The *grant* signal is asserted by the downstream circuit if it is prepared to receive a cell; this provides a simple form of flow control. During input operations, the selector selects any idle control slot inserts the age of the arriving cell into that slot, and passes the slot number to the buffer, which places the arriving cell in the specified slot.

The selector uses a distributed contention resolution mechanism illustrated in Figure 4.

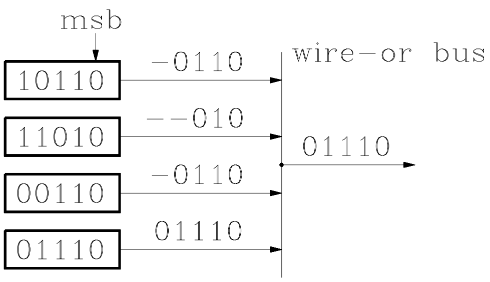

Figure 4: Distributed Contention Resolution

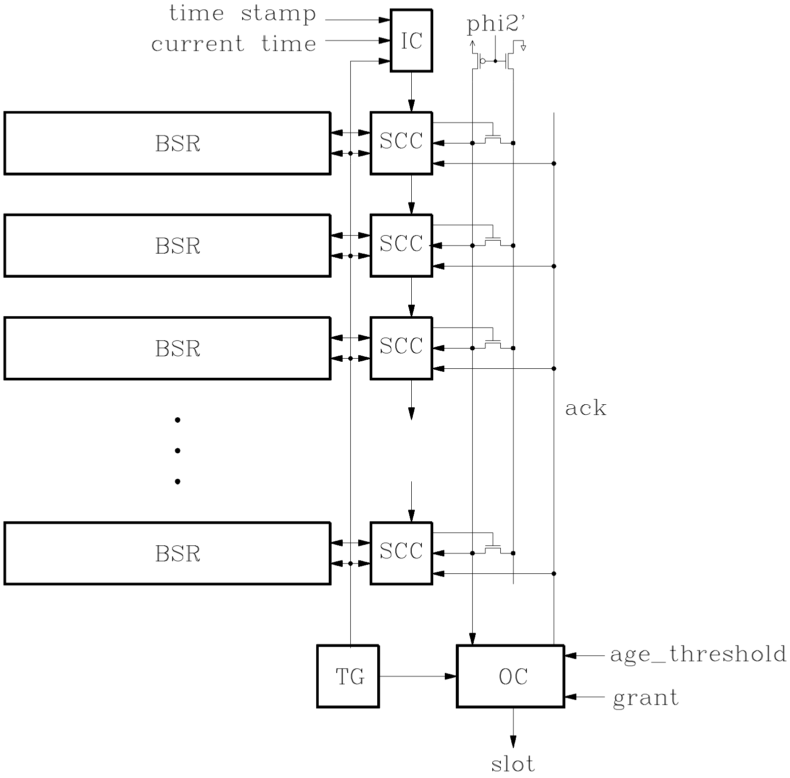

Figure 5: Buffer Control Circuit

The registers at left each contain a unique binary value which is placed sequentially on a wire-or bus, with the most significant bit (MSB) first. As the bits are placed on the bus, each register simultaneously examines the state of the bus and if the bus value differs from the value it put on the bus, it stops contending. Since the stored values are unique, only one register will remain in contention at the end of the cycle, and its value is the one placed on the bus.

Figure 5 gives a more detailed view of the buffer controller, as it might be implemented in a CMOS integrated circuit with a two phase, non-overlapping clock. Each control slot consists of two parts, a *Bidirectional Shift Register* (BSR), and a *Slot Control Circuit* (SCC). The BSR stores the slot number and age information and the SCC keeps track of whether the slot contains a cell or not and participates in the distributed contention resolution. The selector consists of a pre-charged bus that implements the wire-or contention mechanism (more precisely, it's a wire-nor) and an Output Circuit (OC), which on output compares the age of the oldest cell to the age threshold, checks the grant line, then provides an acknowledgment signal if the given cell can be output. Other components include an Input Circuit (IC) which computes the age of an arriving cell and the Timing Generator (TG) which generates the various control signals needed to coordinate the activities of the other components.

Figure 6 illustrates an output operation. The left side of the figure shows the initial state of the BSRs, with the slot number on the left and age on the right, with the most significant bit of each at the right hand end. The busy/idle status of each control slot is held in a flip flop indicated in the figure. During the write operation, the BSRs of the busy slots are rotated to the right, with successive bits placed on the bus. The contention is performed over both the age and the slot number, ensuring that there is a unique winner in the contention. Assuming the winner is acknowledged by the OC, its busy/idle flip flop is cleared, giving the final status shown at the right.

Figure 7 illustrates an input operation. In this case, contention is done using only the slot numbers of the idle slots. However, during the write operation, the age fields of the busy slots are also incremented. Incrementing is most efficiently implemented using a bit-serial incrementer, which requires that the bits be presented least-significant bit first. Consequently, the BSR is rotated to the left rather than to the right. This doesn't affect the selection of an idle slot significantly, since any idle slot will do as well as another. Notice in the figure, that the incoming cell is inserted into the idle slot that wins the contention and that the age fields of the busy slots are incremented appropriately.

3. Control Slot Implementation

Figure **??** shows details of the Slot Control Circuit. The lower flip flop (called the data present flip flop) is set if there is a cell stored in the buffer position associated with this slot control circuit. The upper flip flop (called the contention flip flop) is set when the the slot control circuit is contending for the bus. The *Incrementer* (INC) contains a serial incrementer plus some signal steering circuitry to pass signals between different pairs of its serial data ports `A, B, C` and `D`. For example, when the control signal `A->B` is asserted, data on the `A` port is passed to the `B` port with a one clock tick delay. When the control signal `B+1,D->A` is asserted and `selB+1` is asserted, the data on port `B` is incremented and passed to `A` with a one tick delay (the data is assumed to come in least significant bit first); if `selB+1` is not asserted, the data on the `D` port is passed to the `A` port with a one tick delay.

At the start of an operation, the `init` signal sets the contention flip flop to enable the contention resolution mechanism. During an output operation, the `right` and `A->B` signals are asserted causing the BSR's contents to be rotated to the right and the `read` signal is asserted allowing the slots whose data present flop flops are set to contend. Note that the contention flip flop is cleared when the bus state differs from the bit placed on the bus. At

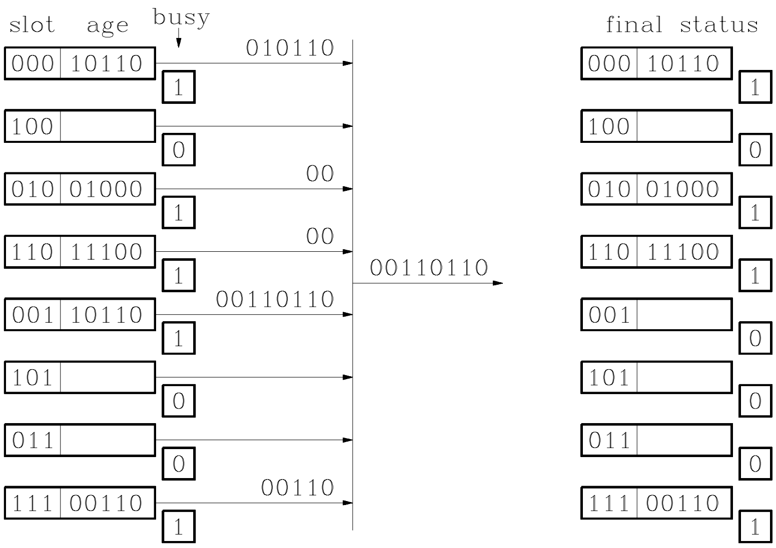

Figure 6: Slot Operation — Output

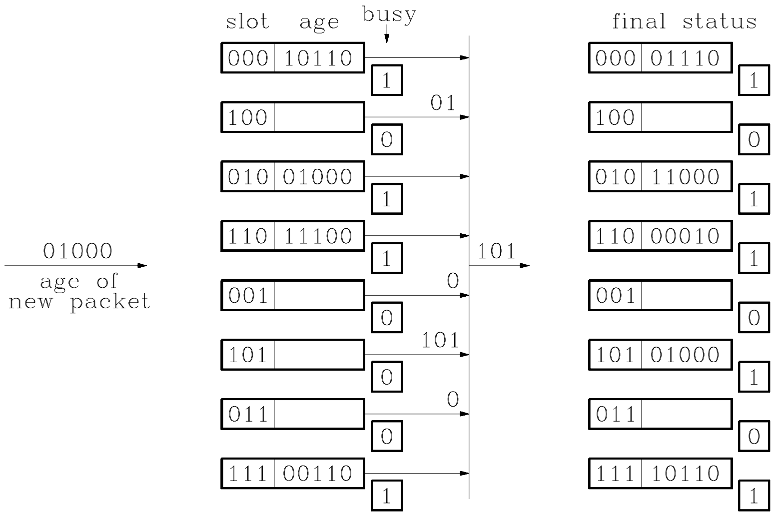

Figure 7: Slot Operation — Input