Report Number: WUCS-93-30

1993-01-01

An Optimal Nonblocking Multicast Virtual Circuit Switch

Jonathan S. Turner

This paper describes an architecture for a multicast virtual circuit switch using cell recycling. This is the first nonblocking switch architecture that is optimal in both the switching network complexity and the amount of memory required for multicast address translation. Furthermore, it is optimal in the amount of effort required for multicast connection modification. This architecture makes it both technically and economically feasible to construct the large switching systems that will ultimately be needed for wide scale deployment of Broadband ISDN to residential users. In particular, we estimate that systems with tens of thousands of 620 Mb/s ports can... Read complete abstract on page 2.

An Optimal Nonblocking Multicast Virtual Circuit Switch

Jonathan S. Turner

Complete Abstract:

This paper describes an architecture for a multicast virtual circuit switch using cell recycling. This is the first nonblocking switch architecture that is optimal in both the switching network complexity and the amount of memory required for multicast address translation. Furthermore, it is optimal in the amount of effort required for multicast connection modification. This architecture makes it both technically and economically feasible to construct the large switching systems that will ultimately be needed for wide scale deployment of Broadband ISDN to residential users. In particular, we estimate that systems with tens of thousands of 620 Mb/s ports can be produced for a per port cost of under $500.

AN OPTIMAL NONBLOCKING MULTICAST VIRTUAL CIRCUIT SWITCH

Jonathan S. Turner

WUCS-93-30

June 11, 1993

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

Abstract

This paper describes an architecture for a multicast virtual circuit switch using cell recycling. This is the first nonblocking switch architecture that is optimal in both the switching network complexity and the amount of memory required for multicast address translation. Furthermore, it is optimal in the amount of effort required for multicast connection modification. This architecture makes it both technically and economically feasible to construct the large switching systems that will ultimately be needed for wide scale deployment of Broadband ISDN to residential users. In particular, we estimate that systems with tens of thousands of of 620 Mb/s ports can be produced for a per port cost of under $500.

This work was supported by the National Science Foundation (grant DCI 8600947), Ascom Timeplex, Bell Communications Research, Bell Northern Research, Goldstar Information and Communications, Italtel SIT, NEC, NTT and SynOptics.

AN OPTIMAL NONBLOCKING MULTICAST VIRTUAL CIRCUIT SWITCH

Jonathan S. Turner

1. Introduction

Multicast virtual circuit networks support communication paths from a sender to an arbitrary number of receivers, as illustrated in Figure 1. As shown, multicast virtual circuits induce a tree in a network connecting a sender to one or more receivers. Switching systems participating in the virtual circuit replicate received cells using *virtual circuit identifiers* in the cell headers to access control information stored in the switching system's internal control tables, then use this information to identify the outputs the cells are to be sent to and relabel the copies before forwarding them on to other switching systems.

Figure 2 illustrates in more detail, the function of a multicast virtual circuit switch. The switch includes control information, shown here as a table, which for each incoming virtual circuit provides a list of outputs and outgoing virtual circuit identifiers. For a cell received on input link i and virtual circuit z, the switch forwards copies to outputs $j_1, j_2, \ldots$ after relabeling them with new virtual circuit identifiers, $y_1, y_2, \ldots$ Notice that if the switch has n inputs and outputs and each output supports up to m virtual circuits, one can describe any collection of multicast virtual circuits with mn words of memory. One simply provides for each (output,VCI) pair, the identity of the (input,VCI) pair from which it is to receive cells. Unfortunately, this method of defining a set of multicast connections is not particularly helpful in switching, as it does not give one a way to go from an (input,VCI) pair to the desired list of (output,VCI) pairs. Existing virtual circuit switch architectures describe multicast virtual circuits in different ways, which while suitable for switching, use far more than mn words of memory. The broadcast packet switch [5, 6], for example, requires $mn^2/2$ words of memory under worst-case conditions. Moreover, the time required to update a multicast connection grows with the size of the connection. Other architectures require even greater amounts of memory. For example, Lee's multicast switching system [3] requires $mn^3/2$ words of memory under worst-case conditions.

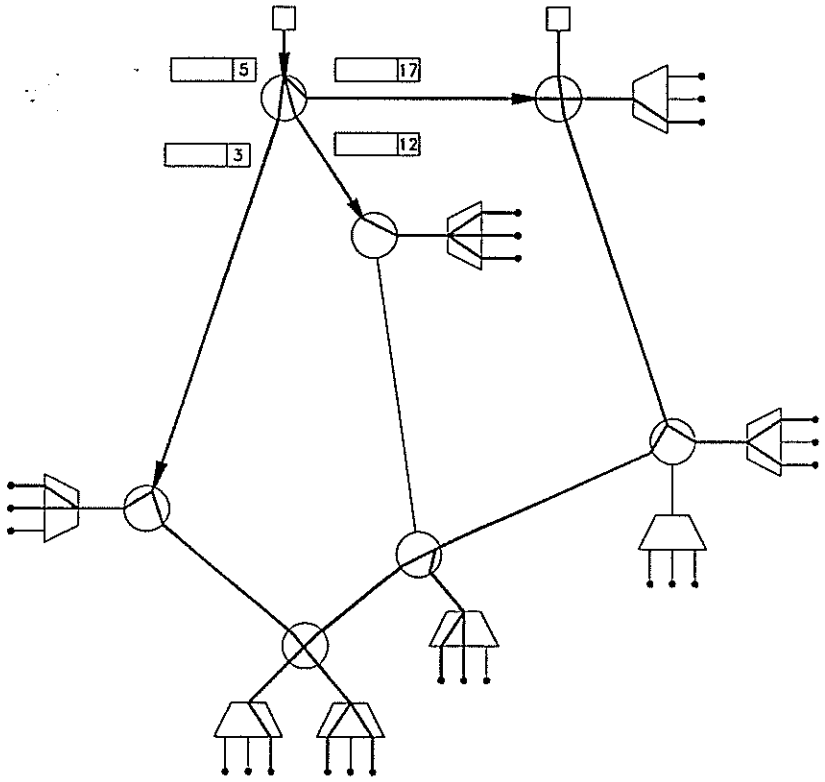

Figure 1: Multicast Virtual Circuit Switching

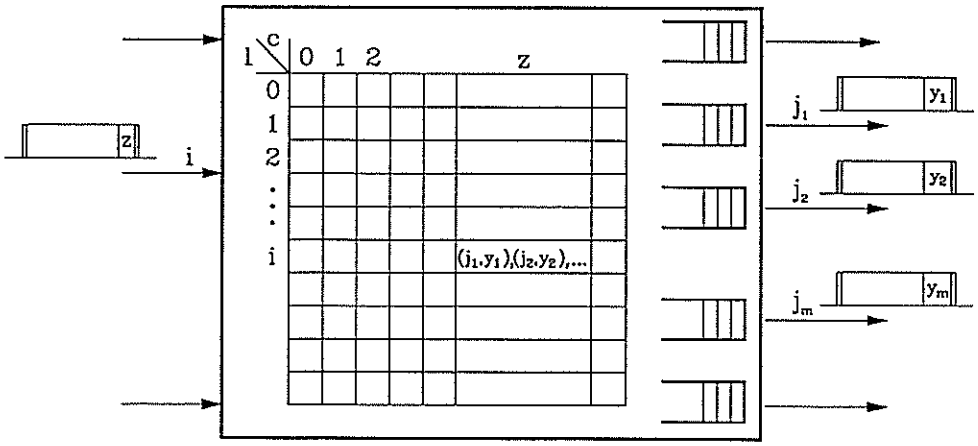

Figure 2: Multicast Switch Functionality

This paper describes a multicast switch architecture with $O(n \log n)$ hardware complexity that is nonblocking, in the sense that it is always possible to accommodate a new multicast connection or augment an existing one, so long as the required bandwidth is available at the external links, and which requires $< 2mn$ words of memory for multicast address translation. Moreover, the overhead for establishing or modifying a multicast connection is independent of the size of the connection or the switching network. Other architectures require even greater amounts of memory. These issues preclude many architectures in the large configurations that will ultimately be required for public network applications. (To put things in perspective, existing telephone switches can support as many as 100,000 individual telephone lines. ATM switches with tens of thousands of ports will be needed to extend advanced information services to residential users.)

2. Basic Operation

The basic principle behind the recycling architecture is illustrated in Figure 3. To implement a multicast connection, a binary tree is constructed with the source switch port at its root and the destination switch ports at its leaves. Internal nodes represent switch ports acting as relay points, which accept cells from the switch but then recycle them back into the switch after relabeling the cells with a destination pair identifying the next two switch ports they are to be sent to. There are many possibilities for constructing the switching network. A Beneš network in which the switches in the first half of the network distribute cells randomly in order to balance the load evenly, and in which local buffers are used to resolve contention, provides the lowest cost solution. This is illustrated in Figure 4 which shows a 16 port network of binary switches in which two cells with two destinations each are forwarded from inputs to outputs. Note that cells are copied at the latest possible point in the network and this point is easily determined by bit-wise consideration of the destination addresses. This scheme can easily be extended to networks constructed from larger switches. We show in a later section that given any collection of virtual circuits, the load placed on any of the switching network's internal links is at most equal to the load on the most heavily loaded external port. In other words, there is no collection of virtual circuits that can be handled by the external links that cannot also be handled by the network. That is, this network is nonblocking. Other switching networks, suitably extended to provide the copy-by-two function, can also be used in the recycling architecture.

The lower part of Figure 3 details the hardware associated with each port of the switching system. Because networks such as the one described in [5, 6] may deliver cells in a different order than that by which they enter, the ports are typically augmented with a resequencing buffer to restore the proper ordering on output [7]. In the recycling architecture, the resequencer also ensures proper ordering of cells during additions and deletions to multicast connections. The resequencing buffer is labeled RSQ in Figure 3. Given a virtual circuit identifier, obtained from a cell's header, the *Virtual Circuit Translation Table* (VXT) provides two (output,VCI) pairs that are added to the cell header

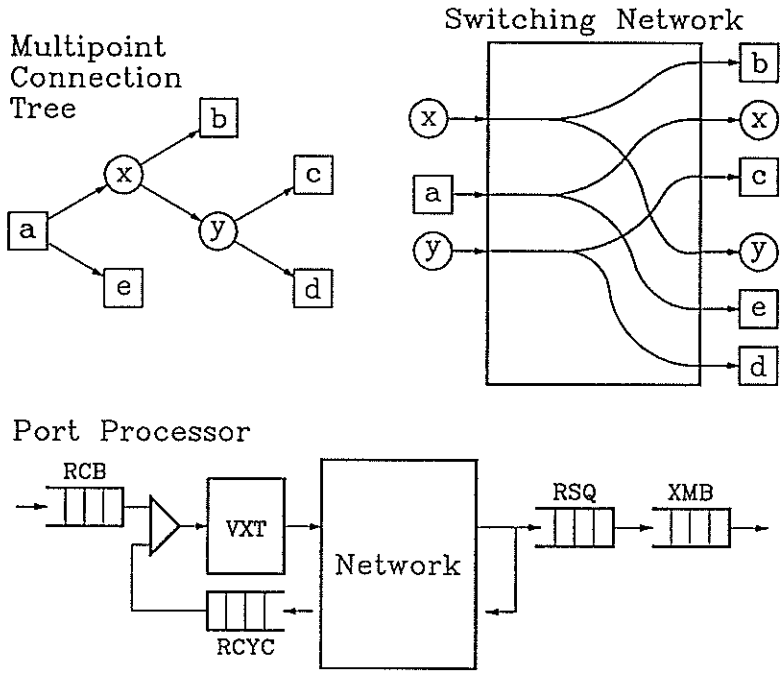

Figure 3: Multicasting by Recycling Cells

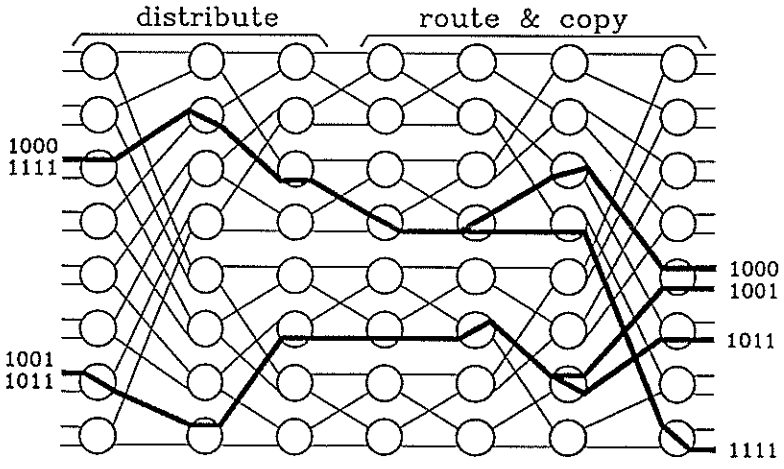

Figure 4: Beneš Network with Copy-Twice Routing

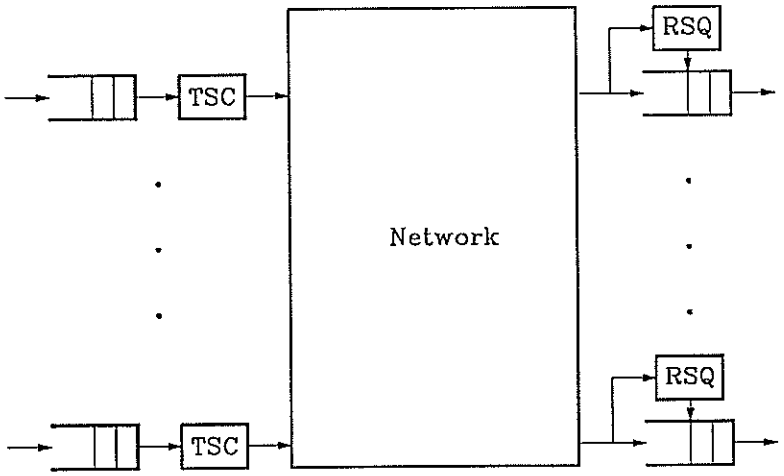

Figure 5: Resequencing Buffer

plus two additional bits that indicate, for each pair, whether it is to be recirculated another time, or not. The *Receive Buffer* (RCB) holds cells received from the input link that are waiting to enter the switching network, while the *Transmit Buffer* (XMB) holds cells waiting to be transmitted on the outgoing link.

Figure 5 illustrates the resequencing operation. Cells entering the network pass through a *Time Stamp Circuit* (TSC), which records the time the cell enters the network in its header. On output, the cell is placed in a buffer, while the RSQ notes its time of entry in order to calculate its age. The RSQ keeps track of the age of all cells stored in the buffer and allows cells to leave the buffer in oldest-first order. Cells are also delayed in order to allow cells that require an unusually long time to pass through the switching network to make it to the output. This is accomplished by comparing the age of the oldest cell to an *age threshold*, which is chosen to match the longest delay cells are expected to encounter when passing through the switching network. The size of the resequencing buffer is typically just slightly larger than the age threshold. This resequencing mechanism is described in more detail in [7].

Figure 6 illustrates the operation of the multicast switch in more detail. In this example, a multicast connection delivers cells from input a to outputs b, c, d and e, using ports x and y as relay points. In the lower part of the diagram, the implementation of the connection is shown in an 'unrolled' form, to clarify the flow of cells through the system. It should be understood however, that this is purely illustrative. There is in fact just one switching network, not three, and cells are simply sent through it multiple times in order to reach all the destinations. In the example, cells entering at input a with VCI i, are forwarded to output e, VCI k and output x, VCI j. At x, the cell is recycled, with VCI j used to select a new table entry from x's VXT. The resulting information causes the cell to be forwarded to output b, VCI n and output y, VCI m. At y, the cell is recycled again, with the resulting copies delivered to c and d.

We can also construct multicast connections to which multiple input ports can send

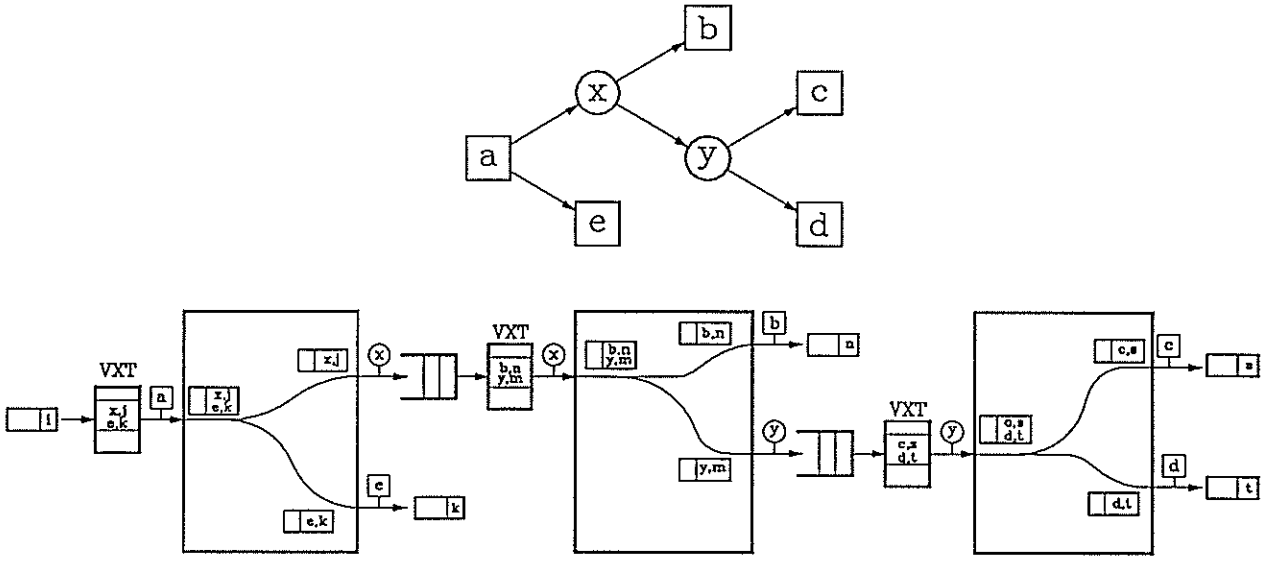

Figure 6: Example of Multicast Connection

cells. One simply sets up the virtual circuit tables of each of the source input ports so that they forward cells to the port at the root of the tree, which then recycles them along the tree. Of course, the total traffic from all the source ports must be limited to the total bandwidth allocated to the connection. In a connection where a port is both a source and a destination, we will usually not want to send a source a copy of a cell that it sent in the first place (although we do want the other participants to receive it). This is easily accomplished by including the identity of the original source in the cell and checking this at the destination in order to discard unwanted copies.

To add an endpoint to a multicast connection, some rearrangement of the connection is needed. This is illustrated in Figure 7. Let d be the output that is to be added to a connection, let c be an output closest to the root of the tree and let a be its parent. Select a switch port x with a minimum amount of recycling traffic. Enter c and d in an unused VXT entry at x and then replace c with x in a's VXT entry. These changes have the effect of inserting x into the tree, with children c and d, as illustrated in the figure.

Dropping an endpoint is similar, as illustrated in Figure 8. Let c be the output to be removed from a connection and let d be its sibling in the tree, x be its parent and a its grandparent. In a's VXT entry, replace x with d. If the output to be removed has no grandparent but its sibling has children, replace the parent's VXT entry with the sibling's children. For example in Figure 8, if b were the output to be deleted, we would copy x's VXT entry to a, effectively removing x from the connection. If the output to be removed has no grandparent and its sibling has no children, then we simply drop the output to be removed from its parent's VXT entry, and the connection reverts to a simple point-to-point connection. For example, in the bottom part of Figure 8, if b were to be dropped from this connection, we would be left with the point-to-point connection from a to d.

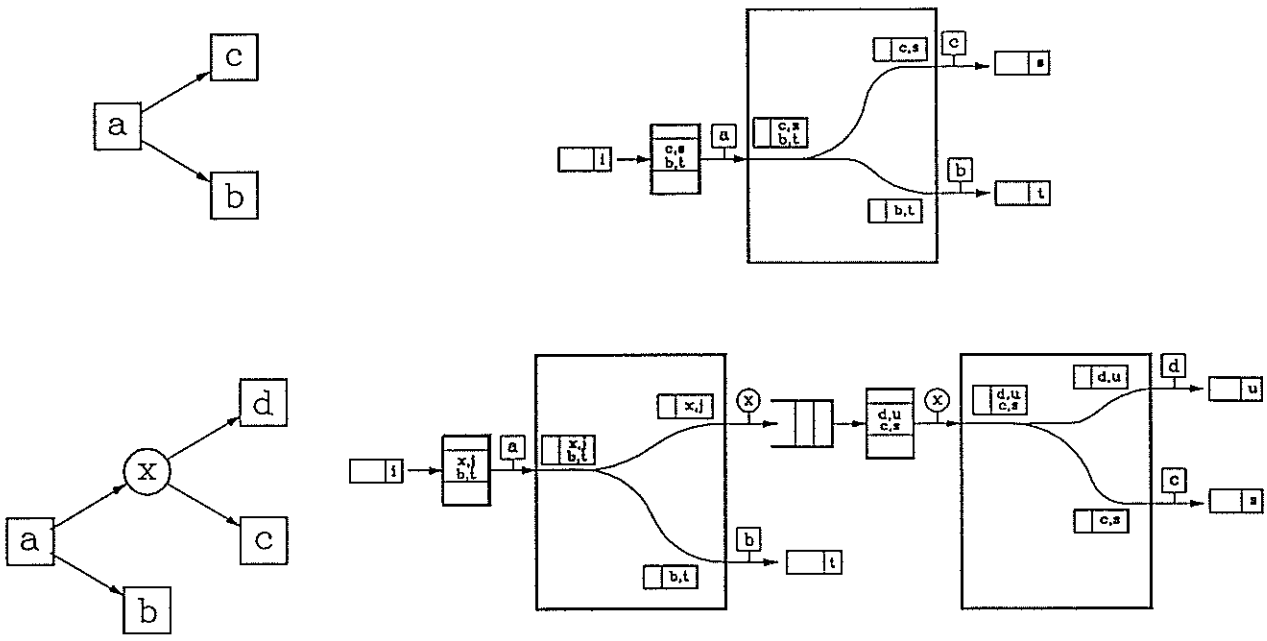

Figure 7: Adding an Endpoint to a Connection

3. Resequencing Options

Because cells are resequenced when they exit from the network, the resequencing buffer must be dimensioned to delay cells long enough so that slow cells have a chance to catch up with fast cells. That is, the resequencing buffer must be at least as large as the largest variation expected in the delay of cells through the system, when they recycle the maximum number of times. Since, both the total delay and the delay variation can change over time, the most practical approach appears to be to dimension the buffer to be equal to the maximum delay that would be expected under the heaviest loading conditions.

A naive analysis reveals how the delay grows with the number of inputs and outputs to the system (n). Let μ and σ be the mean and standard deviation of the delay in each stage of the switching network. Let μ_t and σ_t be the mean and standard deviation for cells passing through the network the maximum number of times. Let r be number of stages of switching that these cells pass through, altogether. Then $\mu_t = r\mu$ and if the delays in each stage are independent (often a reasonable approximation), then $\sigma_t = \sqrt{r}\sigma$. A reasonable engineering rule is to select the resequencer depth equal to the mean delay plus some number h of standard deviations past the mean. This gives a resequencer depth of $\mu_t + h\sigma_t = r\mu + h\sqrt{r}\sigma$. Consequently, the depth grows in proportion to r and for a Beneš network, $r = (2(\log_d n) - 1)(\log_2 F)$ where F is the maximum fanout. For $d = 2$ and $F = n$, this is too much if we are to obtain an overall system cost that grows in proportion to $n \log n$.

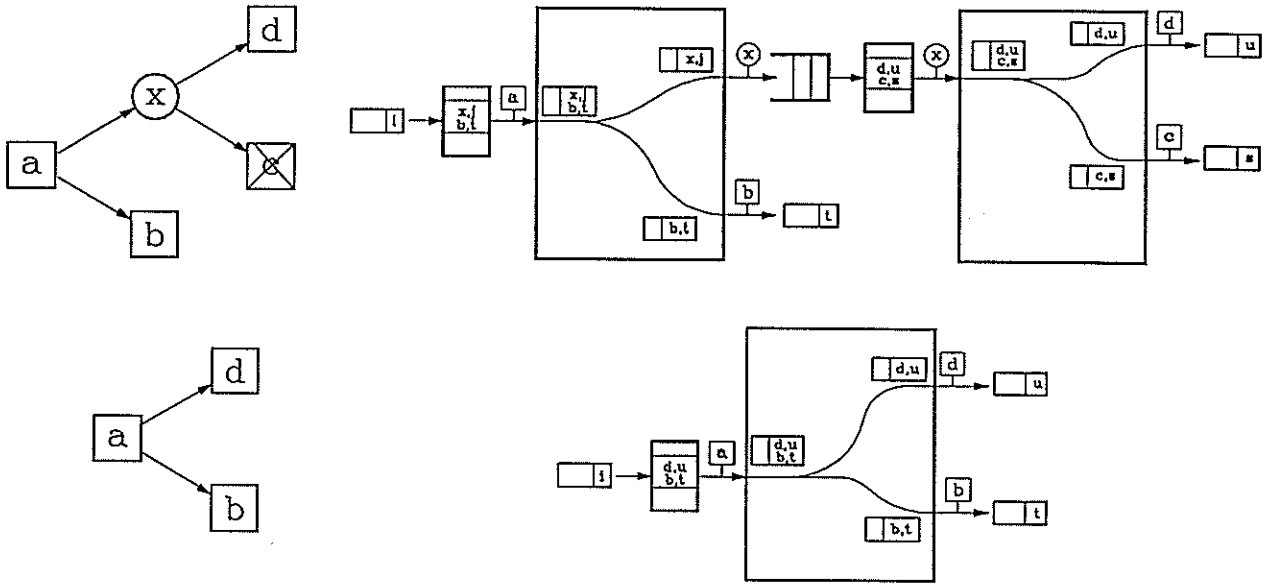

Figure 8: Dropping an Endpoint

We can obtain the desired complexity by resequencing cells after every pass, rather than waiting until the cells exit. This raises a new issue however, in that when we modify a connection, we potentially change the depth of the tree. This means that cells take a different number of passes through the network and introduces the possibility of cells getting out of sequence (even though they are correctly sequenced on each pass). When an endpoint is added to a connection its new sibling becomes repositioned in the tree and its cells experience a longer delay, because of the additional pass through the network. Consequently, there is a momentary gap in the flow of cells to the output, but the ordering of the cells is unaffected. However, when an endpoint is removed from a connection, outputs immediately following the *cut point*, are moved closer to the root of the tree and so the cells being sent to them experience a shorter delay and are at risk of being missequenced with cells that left the cut point just before the change.

To prevent cells from being delivered out of order, the resequencer must provide an extra delay for cells forwarded immediately after the cut occurs. Let T be the maximum delay we expect to see in one pass through the network (equal to $(2(\log_d n) - 1)\mu + \sqrt{2(\log_d n) - 1}h\sigma$ for the Beneš network). Let τ be the moment when the VXT at the cut point is changed and let R be a new register included in the time-stamping circuit of every input port processor. Assume the clock used for time stamping is incremented once for every operational cycle of the system (one cell time) and assume also that the time stamp field of the cell and the register R include an extra low order bit that can be used to represent a "half-step." Normally, cells are time stamped with the current time value. We modify this process for the affected virtual circuit in the time period immediately following the change in the following way. At time τ, the register R is set equal to $\tau + T$. After that time, cells in the affected virtual circuit are time stamped

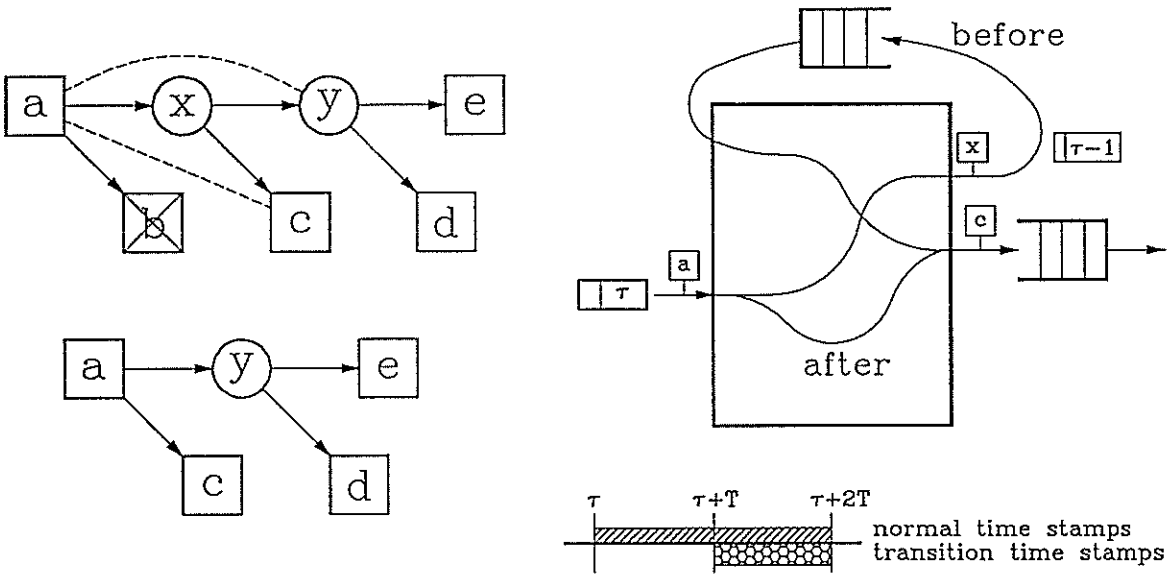

Figure 9: Maintaining Sequence During Transitions

			per pass reseq		multipass
n	r	$\sqrt{r}$	depth	max delay	$r\mu + h\sqrt{r}\sigma$
16	4	2	46	92	12+40=52
256	24	4.9	87	348	72+98=170
4K	60	7.7	119	714	180+155=335
65K	112	10.6	148	1184	336+212=548

Figure 10: Comparison of Per Pass Resequencing and Resequence on Exit

with either the current time or the value of R, whichever is larger. If R is chosen, we also add a $1/2$ to R. This process compresses the time stamps in the period of length $2T$ following the transition into the time period $[\tau + t, \tau + 2T]$ (see Figure 9). This ensures that cells immediately following the transition are delayed for an extra time period in the resequencer, giving cells that entered just before the transition, time to catch up and get placed in the proper sequence. The time stamping process returns to normal no later than $2T$ cycles following the transition.

These same ideas can be generalized to allow, resequencing after every p passes for some p. Letting $z = (2\log_d n) - 1$ and F be the maximum fanout, we obtain a resequencer depth of

$$\mu z(1+p) + h\sigma\sqrt{z}(1+\sqrt{p})$$

and a maximum delay of

$$\mu z \lg F + h\sigma\sqrt{z}(\lg F)/\sqrt{p}$$

The table in Figure 10 compares per pass resequencing ($p = 1$) to the case where we resequence only on exit ($p = \lg F$) when $\mu = 3$, $\sigma = 2$, $h = 10$ and $F = n$. In the table, r is the number of stages in a worst-case path. The expression given in the column labeled multipass gives the depth and delay (in cell times) for the resequence-on-exit case. For the largest system, the per pass resequencing delay is 1184 cell times, or under 700 μs for a system configured to support external link speeds of 620 Mb/s. To put things in perspective, this is less than the delay in many existing digital telephone switches, so even the largest value in the table is quite reasonable. The resequencer depth in the largest case is getting fairly large, although it's arguably still acceptable, since the transmit buffer of the output port is likely to be at least as large. We'll introduce mechanisms in the next section which can improve both of these cases, but the point to be made here is that even without further refinements, excellent performance is possible.

4. Configuring the Network to Avoid Blocking

In this section we show that the recycling architecture can be configured so that it never blocks a new connection request if the rate of the network's internal data paths

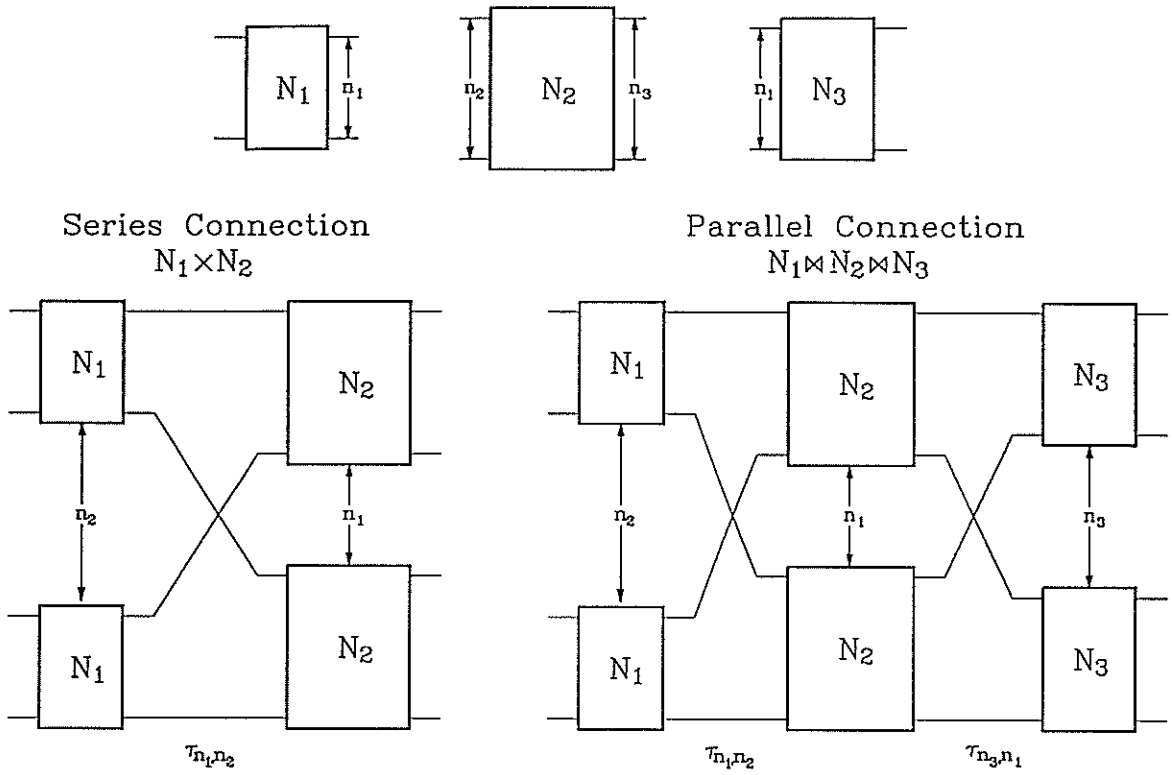

Figure 11: Series and Parallel Construction Operations

is sufficiently higher than that of the external links. We show that the necessary *speed advantage* is modest, making the recycling architecture practically useful. The analysis is in two parts. First we consider how the loading on the network's internal links depends on the loading of the network's ports (including the recycled traffic). Then we consider how the recycled traffic depends on the external traffic. By combining these two analyses, we obtain the speed advantage needed to make the system nonblocking.

To study how the internal link loading depends on the port loading, we apply an analytical technique developed in [4]. For the recycling architecture, we subdivide each multicast connection into "one-pass" segments, consisting of a single input and two outputs. With this understanding, we denote a connection by a triple (x, y, z, ω), where x is an input to the network, y and z are outputs and $0 < \omega < 1$ is a *weight*, which denotes the fraction of one of the network's internal data paths that would be consumed by the connection if it were to use that data path. (So, if the internal data paths operate at a speed of 400 Mb/s, a constant rate virtual circuit operating at a rate of 50 Mb/s would have $\omega = 1/8$.)

We say that a connection *induces a load* on the links that lie on paths joining the connection's input and output ports. In particular, we assume that the load is distributed evenly whenever there are multiple paths to the desired destination. We let $0 < \alpha \leq 1$ denote the maximum load allowed on any of the network's input or output ports. A

connection assignment C is a set of connections that satisfies this constraint and the load induced by C on link ℓ is denoted $\lambda_\ell(C)$. We say a network is *nonblocking* if for all connection assignments, $\lambda_\ell(C) \leq 1$ for *all* links ℓ.

The Beneš network is a special case of a class of networks known as *extended delta networks*. Extended delta networks can be defined using a combination of the *series* and *parallel* constructions of Cantor [2]. These are illustrated in Figure 11. Let N_1 be a network with n_1 outputs, N_2 be a network with n_2 inputs and n_3 outputs and N_3 be a network with n_1 inputs. The series connection of the two networks N_1 and N_2 is obtained by taking n_2 copies of N_1, n_1 copies of N_2, numbering the copies of N_1 sequentially from 0 $(N_1(0), \ldots, N_1(n_2 - 1))$, numbering the copies of N_2 similarly and then connecting output j of $N_1(i)$ to input i of $N_2(j)$, for all pairs i, j. The resulting network is denoted $N_1 \times N_2$. The parallel connection of N_1, N_2 and N_3 is obtained by taking the series network just constructed plus n_3 copies of N_3 and connecting output j of $N_2(i)$ to input i of $N_3(j)$. This network is denoted $N_1 \bowtie N_2 \bowtie N_3$. We also let X_{d_1,d_2} denote a switch element with d_1 inputs and d_2 outputs. Let $d \geq 2$, $k \geq 0$, $0 \leq h < k$ be integers and let $n = d^k$. The extended delta network $D^*_{n,d,h}$ is defined by

$$D^*_{n,d,h} = \begin{cases} X_{d,d} & \text{if } n = d \\ X_{d,d} \times D^*_{n/d,d,0} & \text{if } n > d \text{ and } h = 0 \\ X_{d,d} \bowtie D^*_{n/d,d,h-1} \bowtie X_{d,d} & \text{if } n > d \text{ and } h > 0 \end{cases}$$

When $h = 0$, the extended delta network is equivalent to the ordinary delta network and when $h = k - 1$, it is equivalent to the Beneš network. The number of stages in the extended delta network is $h + k$.

A network with the extended delta topology can be used for copy-twice routing by using the first h stages to distribute cells dynamically across the network and using the last k stages to route cells to the desired output. Hence, for $h \geq 1$, if a connection places a load ω on some input x of the network, the load on the links exiting from x's first stage switch is $1/d$, the load on the links exiting from the subsequent second stage switches is $1/d^2$ and so forth. This spreading of the load continues for the first h stages and then stops. In the last h stages, the load builds up again, increasing by a factor of d at every stage. Let ℓ be a link in stage i of an extended delta network and let $c = (x, y, z, \omega)$ be a connection. From the above discussion, if there is a path in the network from x to ℓ and a path from ℓ to either y or z then

$$\lambda_\ell(c) = \begin{cases} \omega_j d^{-i} & 0 \leq i \leq h \\ \omega_j d^{-h} & h \leq i \leq k \\ \omega_j d^{-(k+h-i)} & k \leq i \leq k + h \end{cases}$$

Let C be any set of connections on $D^*_{n,d,h}$ and let C_ℓ be the subset of connections $c_j = (x_j, y_j, z_j, \omega_j)$ for which there is a path from x to ℓ and a path from ℓ to either y or z. Note that there are paths to ℓ from at most d^i inputs and from ℓ to at most d^{h+k-i} outputs. Because the load on every input and output is limited to α,

$$\sum_{c_j \in C_\ell} \omega_j \leq \alpha d^i \quad \text{and} \quad \sum_{c_j \in C_\ell} \omega_j \leq \alpha d^{k+h-i}$$

Thus, for $0 \leq i \leq h$,

$$\lambda_\ell(C) = \sum_{c_j \in C_\ell} \lambda_\ell(c_j) \leq d^{-i} \sum_{c_j \in C_\ell} \omega_j \leq \alpha$$

For $k \leq i \leq k+h$,

$$\lambda_\ell(C) = \sum_{c_j \in C_\ell} \lambda_\ell(c_j) \leq d^{-(k+h-i)} \sum_{c_j \in C_\ell} \omega_j \leq \alpha$$

For $h \leq i \leq k$,

$$\begin{aligned} \lambda_\ell(C) &\leq d^{-h} \sum_{c_j \in C_\ell} \omega_j \leq \alpha d^{-h} \max_{h \leq r \leq k} \min\left\{d^r, d^{k+h-r}\right\} \\ &= \alpha d^{-h} d^{\lfloor (k+h)/2 \rfloor} = \alpha d^{\lfloor (k-h)/2 \rfloor} \end{aligned}$$

Hence we have proved the following theorem.

THEOREM 4.1. *For any assignment C on $D^*_{n,d,h}$, $\lambda_\ell(C) \leq \alpha d^{\lfloor (k-h)/2 \rfloor}$ for all links ℓ.*

For $h = k-1$, this is simply $\lambda_\ell(C) \leq \alpha$. That is, the load on the internal links of the network is bounded by the load on the inputs and outputs. Hence, the only speedup needed in the network is that required to accommodate queueing effects. From [1] it is apparent that if the network is constructed from shared buffer switches with $d \geq 8$ and $4d$ buffer slots, a speed of 20% can be sufficient.

We now consider the impact of the recycling strategy on the total traffic in the network. A binary tree with r leaves and in which every internal node has two children, has exactly $r-1$ internal nodes. Hence, a one-to-many connection with rate ω and r leaves, places a total load of ωr on the outgoing links and generates a recycling load of $(r-2)\omega$. A many-to-many connection (one with multiple transmitters as well as multiple receivers) creates an output load of $(r-1)\omega$ and a recycling load of $(r-1)\omega$. That is, the recycling load never exceeds the exiting load. Also, notice that recycling is used only for one-to-many connections with at least three destinations and for many-to-many connections with at least two. Call these the *recycling connections.*

Let β denote the ratio of the external link rate to the switch's internal data path speed and let δn denote the total traffic exiting the system from the recycling connections (where n is the number of inputs and outputs of the network). From the above discussion it is clear that the total recycling traffic is at most $(\beta+\delta)n$, so there is always some output port where the recycling traffic is $\leq \delta$. If $B \leq \beta$ is the maximum weight for a single connection then we can always accommodate a new connection if $\beta + \delta + B \leq 1$. The worst-case occurs when $\delta = B = \beta$; in this case a three-to-one speed advantage is needed to ensure that a new connection does not block. If however, $\delta = B = \beta/2$, a two-to-one speed advantage suffices. Note that since the required speed advantage is independent of n, the complexity of the switching network is $O(n \log n)$.

There are some variations on the recycling architecture that are worth considering. One variation is to add a distribution pass before copying begins, for those cells that

must be recycled. In effect, this replaces the single connection tree with a group of γ parallel trees, all with the same leaves, but different internal nodes. The input port (or ports) sends each cell to the root of one of the trees, distributing them in a round-robin fashion. This means that the impact of single connection on a recycling port is reduced by a factor of γ, making the system nonblocking if $\beta + \delta + B/\gamma \leq 1$. If $\delta = B = \beta/2$ and $\gamma = 8$, we require a speed advantage of about 1.56. Obviously, this increases the amount of memory needed for forwarding multicast cells by a factor of γ, so there is a trade-off of memory vs. bandwidth that can be resolved to meet specific application requirements.

Another variation is to do the cell replication at the recycling port instead of within the switching network. This would allow a network designed for point-to-point switching to be used for recycling. However, there is the obvious drawback of increased traffic at the ports and a larger speed advantage. In particular, the required speed advantage goes up to $\beta + 2(\delta + B/\gamma)$.

A third option is to design the system so that it can make more than two copies per pass. A larger branching factor reduces the number of passes required to produce a given number of copies and can reduce the total recycling traffic. In particular, if the branching factor is b, it suffices to have $\beta + \delta/(b-1) + B/\gamma \leq 1$ if copying is done in the network and $\beta + b(\delta/(b-1) + B/\gamma) \leq 1$ if copying is done at the recycling ports. When multiple copies are separately addressed, we require additional space in virtual circuit translation tables and in cell headers. Another option is to make more than two copies in a pass but simply send all the copies to a contiguous set of outputs, so that separate addresses are not needed for each copy. This is easy to accomplish by designing the switch hardware to (optionally) interpret the two addresses in the header as lower and upper bounds for a range of outputs, rather than using the usual copy-by-two algorithm. All the copies would use the same virtual circuit identifier to recycle through the network, meaning that some portion of the table entries (and bandwidth) must be managed across groups of ports. Using this approach, large branching factors become practical and it's possible to dramatically reduce the number of passes required to support a given connection. In particular, if we use a fixed branching factor of b copies per pass (in order to simplify management of recycling bandwidth and table entries) along with the original copy-by-two mechanism, it's possible to reduce the number of passes from $\log_2 F$ (where F is the maximum fanout), to $1 + \log_b F$. Even further reductions are possible with more sophisticated connection management algorithms.

5. Design of a Gigabit Switch

The recycling architecture can be used to implement very large switching systems with a modest cost per port. In this section, we detail the design of such a system. To minimize overall system cost, we employ a bit-sliced structure for the switch elements used to construct the buffered Beneš network. The bit-sliced structure allows us to construct a large switch with a minimum number of chips, dramatically reducing the number of

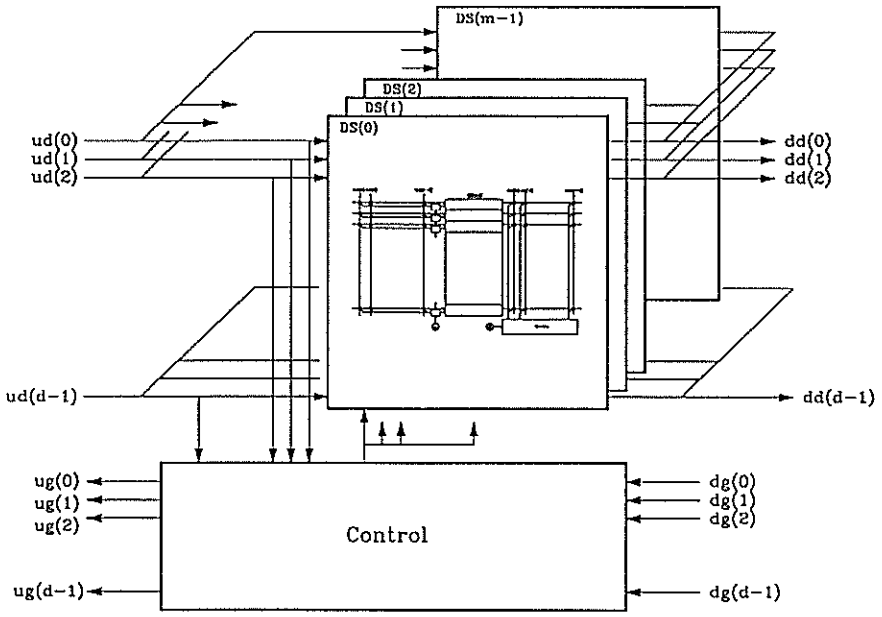

Figure 12: Bit-Sliced Switch

stages required to realize a large systems. Figure 12 shows the organization of a bit-sliced switch with d input and output ports and supporting m bit wide data paths. Cells enter on one of the d *upstream data* lines (ud(i)) at left, and the m bits of each cell are distributed across m separate *data slices* (DS). The cells exit from the switch element on the *downstream data* lines (dd(i)) at the right. The switch element contains sufficient internal buffering to store several cells for each port and implements a simple hardware flow control mechanism to prevent cells from overflowing these buffers.

The *control slice* shown at the bottom of the figure contains the circuitry used to control the operation of the switch. It receives a set of *downstream grant* signals (dg(i)) from the downstream neighbors and generates a corresponding set of *upstream grant* signals (ug(i)) which are sent to the upstream neighbors. In general, a switch element asserts an upstream grant signal ug_i if it is prepared to receive a cell on the upstream data lines ud(i). The cells flowing through the switch element are organized so that all the control information (in particular, the addressing information) passes through the first data slice DS(0). This allows the control circuit to easily monitor the control information for all cells entering the data slice. Using this information, together with the downstream grants and the internal status of the switch elements, it makes control decisions and broadcasts those decisions to the data slices. If the data paths are byte wide and the buffer size is three to four times the number of ports, a clock rate of 100 MHz is sufficient to handle external link speeds of 620 Mb/s.

The data slice is detailed in Figure 13. It includes an input crossbar on the left, a shared buffer in the center, and an output crossbar on the right. The shared buffer is a static memory array in which each row is used to store one bit slice of one cell.

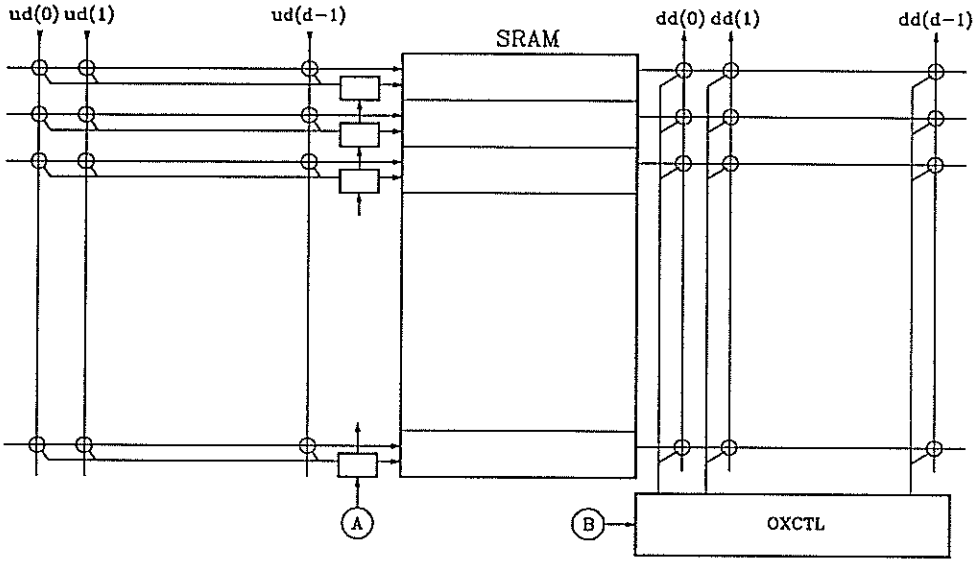

Figure 13: Data Slice

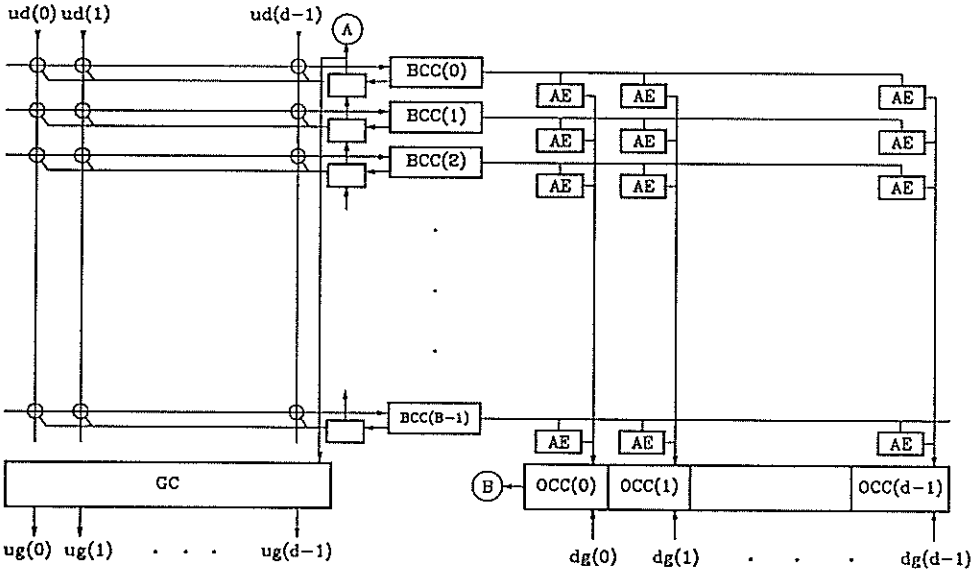

Figure 14: Control Circuit

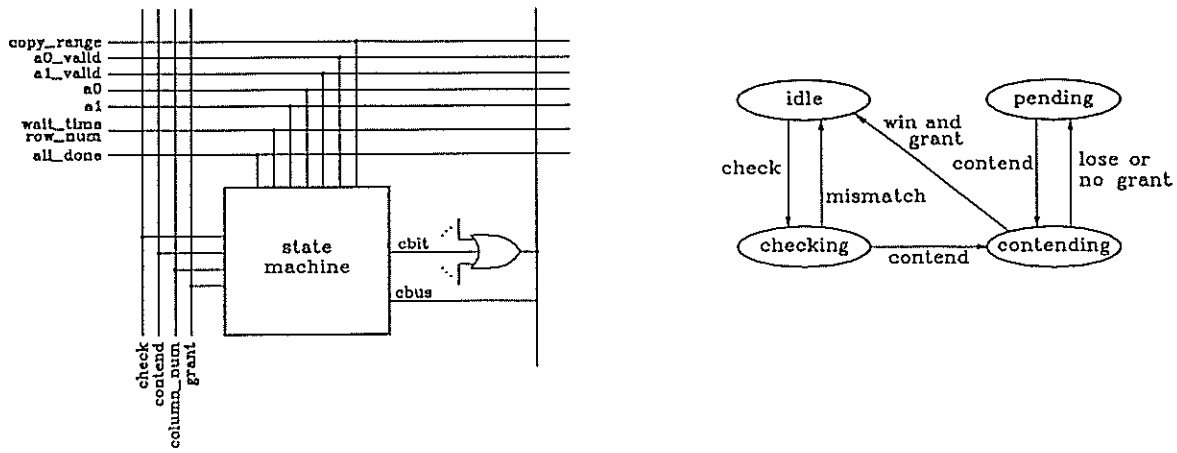

Figure 15: Arbitration Element

During an operational cycle, successive columns of the array are accessed during each bit time. Some rows are read and other rows written, corresponding to whether a cell is exiting or entering that row. A control bit, shown entering at the left of the array determines the mode of operation. The input crossbar delivers cells from the inputs to the buffer slots and is controlled by a column number supplied along each row, which selects the crosspoint in that row that is to be closed. The output crossbar delivers cells from the buffer slots to the outputs, with possibly more than one output receiving a copy at the same time. The output crossbar is controlled by a row number supplied along each column, that selects the crosspoint in that column that is to be closed. Different organizations are used for the input and output crossbar to facilitate the computation of the control information in each case. The complexity of the crossbar is roughly $2dBx_1 + (LB/m)x_2$ where x_1 is the complexity of a single crosspoint and x_2 is the complexity of one bit of memory. We estimate x_1 at five gate equivalents and x_2 at one gate equivalent, giving a complexity of under 50 thousand gates for ATM cells, $d = 32$, $B = 4d$ and byte wide data paths. Two such data slices can be placed on a single chip comfortably, and indeed a 64 port data slice is well within current technology constraints. While the crossbars are shown as being implemented in one-level, large implementations may require a two or three level structure to reduce the capacitive loads that must be driven by individual crosspoints. We estimate that this would increase the complexity of the crossbars by 10–15%.

The control circuit is detailed in Figure 14. It includes an input crossbar, a collection of *Buffer Control Circuits* (BCC), an array of *Arbitration Elements*, a set of *Output Control Circuits* (OCC) and a *Grant Circuit* (GC). Each BCC interprets arriving address information, forwards an appropriate request to the Arbitration Elements in its row and sends a control bit to the corresponding data slice buffer slot, to control whether a new cell is to be accepted or not. The AEs in a given column of the array compete for the corresponding outputs, if selected. Each OCC passes its downstream grant to the AEs in its column and passes the row number of the winning AE to the output crossbar control

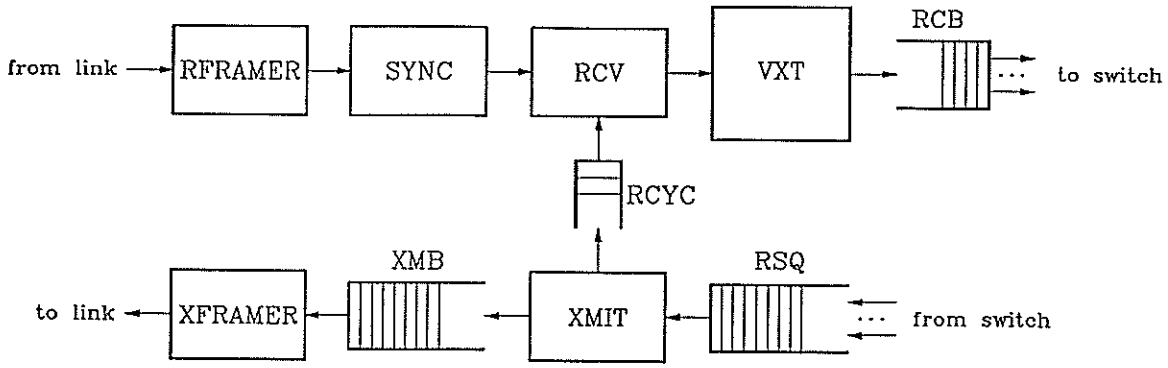

Figure 16: Port Processor

circuits in the data slices. The Grant Circuit determines which inputs are to receive grants, based on the number of available buffer slots. The complexity of the Control Circuit is about $dB(x_1+x_3)+Bx_4+dx_5$ where x_3 is the complexity of an AE, x_4 is the complexity of a BCC and x_5 is the complexity of an OCC plus the per port cost of the GC. We estimate $(x_1+x_3)=50$, $x_4=x_5=200$. For $d=32$, $B=4d$ this gives about 240K gates, which is feasible with current technology.

An Arbitration Element is detailed in Figure 15. It receives six signals along the row, from the BCC and generates a single reply bit. It also receives four signals along the column and contends on a contention bus that runs along the column. The three most important signals sent along the row are the address signals `a0`, `a1` and the `wait_time, row_num` signal. The address signals determine which AEs in a row are being selected and the `wait_time, row_num` signal is the serial concatenation of the number of cycles the requesting BCC has been waiting and the row number in the arbitration array (presented high order bit first). A selected AE passes this information bit serially to the OR gate at the right, where it is OR-ed with the corresponding signals from other AEs in the column. If at any point, the bit that the AE inputs to the OR gate differs from output of the OR gate, the AE ceases contending. When all selected AEs in a column do this in parallel, the effect is to select an AE with the largest waiting time. The signals `a0_valid, a1_valid` are asserted when the corresponding address is valid; otherwise the address is ignored. If the `copy_range` signal is asserted, the AEs interpret the two addresses as defining lower and upper bounds on a range of column numbers; otherwise they're treated as simply two column numbers. The `all_done` signal is a wire-NOR line that can be pulled low by any selected AE in a row that is not given access to its output. The `check` and `contend` signals are timing signals that indicate the start and end of different phases in the AE's operational cycle. The column number is passed serially on the `column_num` line and compared by the AEs to the values on the address lines to determine whether they are selected or not.

The right hand side of Figure 15 shows the major states of the AE state machine. The machine starts in the `idle` state. From there, it enters the `checking` state in which

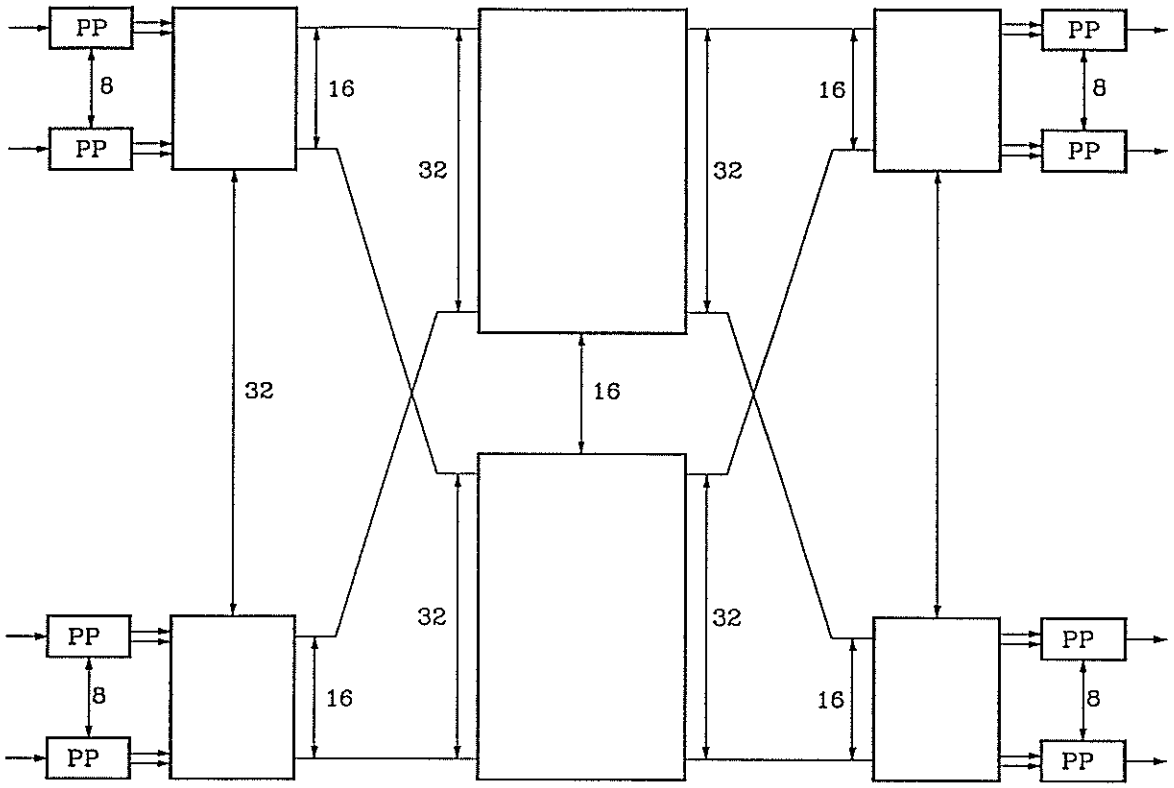

Figure 17: Gigabit Switch Organization

it compares the `a0, a1` signals to the `column_num` signal to determine if it is selected or not. If not selected, it returns to the `idle` state. If it is selected, it goes to the contending state during which it passes the `wait_time, row_num` signal to the contention bus, going to the pending state if it loses the contention or there is no grant, and back to the idle state if it wins and there is a grant. If the AE goes to the pending state, it will contend again on the next cycle and continue to do so until it succeeds.

Figure 16 shows the organization of a port processor for a gigabit switch. It consists of several components. The *Receive Framer* (RFRAMER) and *Transmit Framer* are responsible for cell delineation and checking of the ATM Header Error Check field. The *Synchronizer* (SYNC) synchronizes incoming cells to the switching systems timing reference. The *Virtual Circuit/Path Translation Table* (VXT) provides the address translation for cells received either from the external port or recirculated from the output side. The *Receive Buffer* (RCB) holds cells waiting to be sent to the switch and can send cells to multiple switch ports. The *Resequencer* (RSQ) does cell resequencing and can receive cells from multiple switch ports. The *Recycling Buffer* holds cells that are being recycled back to the switching network. Notice that cells are recycled *after* they are resequenced. The *Transmit Buffer* (XMB) holds cells waiting to be transmitted to the output link.

Two versions of this port processor are conceived. One version, designed for 620

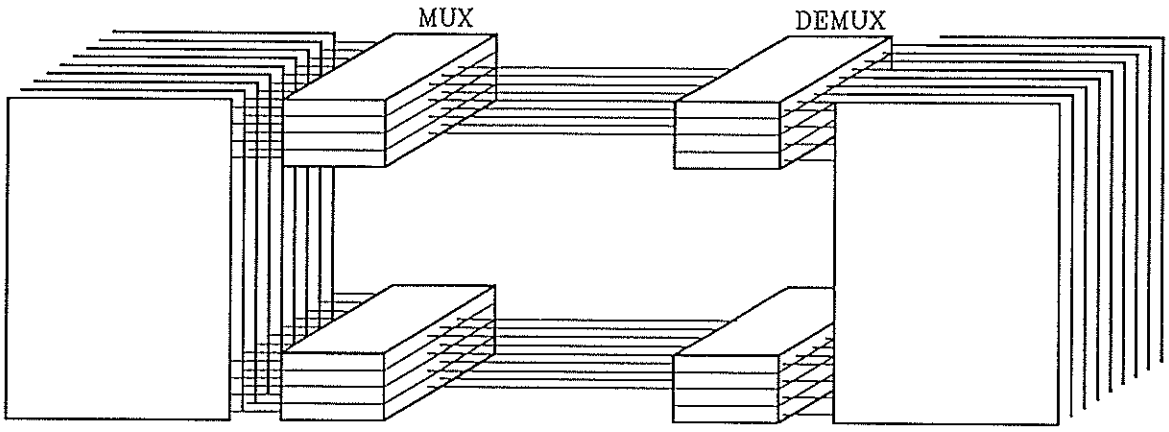

Figure 18: Board to Board Multiplexing Mechanism

Mb/s external links, would distribute cells from the RCB dynamically across two switch ports (and receive from two) in order to obtain a 2:1 speed advantage. The internal data paths would be two bytes wide in order to accommodate the necessary speedup in the recycling path. A second version, designed for 2.4 Gb/s external links would distribute cells across eight switch ports to obtain the necessary switch bandwidth. We estimate that a port processor with a 32 cell RCB, a 128 cell RSQ, a 256 cell XMB and 1024 entries in the VXT can be implemented with about 80K gates and 340K bits of memory, making a two chip implementation feasible. We estimate that for the 620 Mb/s version, the packages will require under 100 pins, and for the 2.4 Gb/s version, the packages will require under 250 pins.

Figure 17 is a block diagram of switching system that can support up to 256 ports at 620 Mb/s each or up to 64 ports at 2.4 Gb/s each (or any combination). We propose to package eight port processors together with a pair of switches (one for the input side, one for the output side). These boards would require 21 port processor and switch chips, plus additional chips for transmission formatting and conversion between optical and electrical signals. We propose to package four of the central 32 port switches per board. These boards will require 20 switch chips.

The interconnection between boards is one of the major implementation challenges. For the switch boards, the proposed packaging requires 2560 signals entering and leaving the board (32 inputs plus 32 outputs per switch element, 10 signals per input and output, 4 switch elements per board), well beyond the limits of current connector technology. The multiplexing scheme shown in Figure 18 shows how this can be reduced to a manageable level. On the left, are nine bit planes of one switch element. These nine signals are multiplexed to three signals for transmission between boards. The demultiplexor on the receiving board, demultiplexes the received signals and passes them on to the receiving data slices in synchronism with the local clock. Four multiplexor/demultiplexor pairs can be put on a single gallium arsenide gate array with about 150 pins. This adds 32 chips to each of the switch boards (making the total 52) and four chips to each of the port

processor boards. It reduces the number of signals per switch board to 768. Assuming production costs of $15 per chip for the data slice chips and $40 per chip for the others, the overall cost per port is $155 per 620 Mb/s port and $370 per 2.4 Gb/s port. This does not include the cost of the transmission interface chips and optical devices required to complete the system. Currently, these would raise the costs by over $1000 per port, but as the costs of these components come down, gigabit switches with per port production costs of under $500 appear feasible.

With the proposed packaging, the system described could be implemented with a total of 32 port processor boards and 4 switch boards and could be packaged in a volume of about four cubic feet. Using the same packaging approach, a five stage system, supporting 8192 ports at 620 Mb/s, would require a total of 1024 port processor boards plus 384 network boards and could be housed in eight equipment frames. The added network and multiplexor chips required for this larger system add about $32 per port, and a similar increment is sufficient for systems with up to 262 thousand ports.

6. Summary

The recycling architecture is the first nonblocking multicast virtual circuit switch architecture that is optimal in the switching network complexity, the amount of memory required for multicast address translation and the amount of effort required for multicast connection modification. The architecture makes it both technically and economically feasible to construct switching systems with tens of thousands of high speed ports. While recycling cells leads to suboptimal delay, this has a negligible impact in most practical applications, since the delays are well under one millisecond in the basic architecture and can be cut even further using refinements to the basic scheme.

References

[1] Bianchi, Giusseppe and Jonathan Turner. "Improved Queueing Analysis of Shared Buffer Switching Networks," *Proceedings of Infocom*, March 1993.

[2] Cantor, D. G. "On Non-Blocking Switching Networks," *Networks*, vol. 1, 1971, pp. 367–377.

[3] Lee, Tony T. "Non-Blocking Copy Networks for Multicast Packet Switching," *IEEE Journal on Selected Areas in Communications*, 1455–1467, 12/88.

[4] Turner, Jonathan S, "Fluid Flow Loading Analysis of Packet Switching Networks," *Proceedings of the International Teletraffic Congress*, June 1988.

[5] Turner, Jonathan S., "Broadcast Packet Switching Network," United States Patent #4,734,907, March, 1988.

[6] Turner, Jonathan S., "Design of a Broadcast Packet Network," *IEEE Transactions on Communications*, June, 1988.

[7] Turner, Jonathan S., "Resequencing Cells in an ATM Switch," Technical Report WUCS-91-21, Department of Computer Science, Washington University, St. Louis, Missouri.