# Extending ATM Networks for Efficient Reliable Multicast

Jonathan S. Turner, jst@cs.wustl.edu
Washington University, St. Louis

*Abstract*

One of the important features of ATM networks is their ability to support multicast communication. This facilitates the efficient distribution of multimedia information streams (such as audio and video) to large groups of receivers. Because ATM networks do not provide reliable delivery mechanisms, it is up to end systems to provide end-to-end reliability where it is needed. While this is straightforward for point-to-point virtual circuits, it is more difficult for multicast virtual circuits. This paper proposes extensions to the hardware of ATM switches that enables end systems to implement reliable multicast in a more efficient and scalable manner than is otherwise possible. We essentially provide a *network assist* to enable end systems to implement reliable multicast more effectively. With our approach, the amount of work that must be done by any receiver in a reliable multicast is independent of the number of participants in the multicast and the amount of work that must be done per packet transmission by any sender, is independent of the number of participants in the multicast. Of course, in large multicasts, a given packet may have to be transmitted multiple times and the number of such transmissions does depend on the number of participants. However, one can show that even under very conservative assumptions this effect does not limit scalability to any significant extent. The proposed reliable multicast mechanisms are being implemented in a scalable ATM switch support 2.4 Gb/s transmission links.

## 1 Introduction

Efficient multicast communication is a key feature of ATM network technology. Originally designed to support distribution of multimedia information streams (audio and video), it can also be useful for more general distributed computing applications. However, because such applications generally require reliable data delivery, end systems must provide mechanisms for ensuring reliable transport over the unreliable communication channels provided by ATM networks. While providing reliable transport is straightforward for point-to-point communication channels, it is more complex in the context of multicast channels which may have hundreds, thousands or even millions of participants.

This paper develops an approach to reliable multicast communication in ATM networks. Our objective is to provide *scalable* mechanisms by which we

[0] This work was supported by the Advanced Research Projects Agency.

mean mechanisms in which the amount of work done by senders and receivers in a multicast connection is independent of the total number of participants. Reliable multicast can be implemented entirely by mechanisms in end systems or by some combination of mechanisms in end systems and the network. Our approach is to provide a *network assist* to enable end systems to implement reliable multicast more effectively. There are actually four distinct aspects of the proposed reliable multicast support.

- *Redundant acknowledgement suppression* elimates redundant acknowledgements in a multicast connection, so that a sender receives just a single positive acknowledgement for each packet sent and at most one negative acknowledgement.
- *Targeted retransmission* ensures that packet retransmissions are forwarded only to the receivers that did not get a packet when it was first sent.
- *Dynamic channel sharing* allows multiple senders in a many-to-many multicast to dynamically share a single virtual circuit without the occurrence of packet loss due to cell interleaving of the different packets. This is accomplished using a dynamic virtual circuit subchannel assignment technique.
- *Path trace/retrace* is used in applications with multiple senders and allows control cells used in the reliable multicast protocol to accumulate path information that can be later used by cells going in the reverse direction to retrace the path back to the original sender.

These mechanisms can be used separately or in combination to provide reliable multicast for one-to-many and many-to-many applications. In addition, the dynamic channel sharing mechanism addresses a general issue for many-to-many virtual circuits and is of value even where fully reliable transmission is not required.

There is a substantial literature on providing reliable multicast mechanisms in networks. One of the earliest contributions was [3]. Some selected recent contributions include [1, 5, 7, 8, 9, 10, 11]. Much of the more recent work is directed toward multicast in the context of the internet protocol suite and shifts much of the responsibility for recovering lost data onto the receivers. The recent paper by Floyd, Jacobson, et. al. [6] is a good example of receiver-based recovery. They also advocates the use of application-specific reliable multicast mechanisms, rather than more general mechanisms supported at lower levels. While some of the prior work is applicable in the ATM context, there seems little prior work directly related to extending ATM switching mechanisms to assist in the implementation of reliable multicast.

In section 2, we describe the acknowledgement suppression mechanisms that enable scalable reliable one-to-many multicast. Section 3 describes the mechanisms used to support many-to-many multicast with consistent delivery order using one end system as a relay. Section 4 describes how the relay can be eliminated (at the cost of sacrificing consistent delivery order). Section 5 outlines how the mechanisms are being implemented in the Washington University Gigabit Switch. Section 6 gives a brief performance analysis.

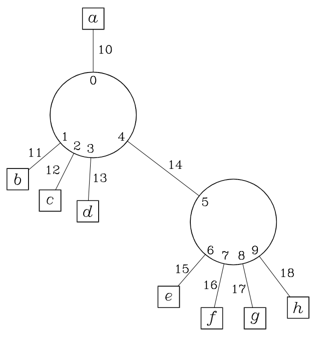


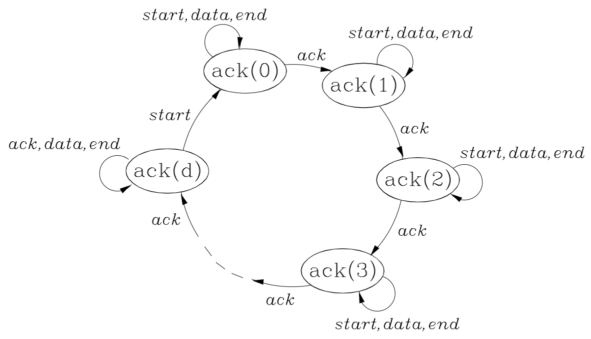
**Fig. 1.** Reliable Multicast Message Transmission

## 2 Reliable One-to-Many Multicast

Our basic approach to achieving reliable one-to-many multicast is illustrated in Figure 1. To send a packet to end systems $b$ through $h$, end system $a$ sends the packet on a reliable multicast virtual circuit, through the two ATM switching systems, shown in the figure. The packet is delineated by start and end of packet cells (these, and the other control cells required, can be implemented using a variant of the ATM Resource Management cell type). The ATM switches note the passage of the start and end cells, and when acknowledgement cells are received from the downstream neighbors, they propagate only the last acknowledgement cell expected. The acknowledgement cells are originated by the receiving end systems; the switches merely propagate them selectively, so that the sender receives only one acknowledgement indicating that all destinations have received the message. If the acknowledgement is not received until after a timeout has expired, source $a$ can send the packet again, preceded by a new start cell. The switches propagate the retransmitted packet, only to those downstream neighbors that did not acknowledge the original transmission, so that destinations that received the first copy will not have to process a duplicate.

To allow end systems to pipeline multicast packet transmissions through the network, all control cells processed by the switches (start, end, ack) contain *transmission slot numbers* which are used to access stored state information relevant to a particular packet. Individual data cells do not contain slot numbers, but the data cells are assumed to be sent using AAL 5 (or something equivalent) and include the slot number or an equivalent transport protocol sequence number somewhere in the end-to-end protocol header. A receiver acknowledges a packet only if it is correctly received (as indicated by an end-to-end error check).

The switches maintain a state machine for each transmission slot, to keep track of which downstream neighbors have acknowledged a given packet and which have not. One version of this state machine is shown in Figure 2. In the state diagram, the state $ack(i)$ denotes all states in which $i$ acknowledgements (out of a total expected number of $d$) have been received. Assuming the state machine starts in state $ack(d)$, the arrival of a start cell, places it in state $ack(0)$. In



```
upstream_neighbor = input link and VCI on which packets arrive
downstream_neighbor = link and VCI on which a received ack arrived
output_set = set of output (link, VCI) pairs to which packets
             are to be forwarded
type = type of the cell being processed (data, start, end, ack)
i = slot number in the control cell being processed
    (i is undefined if processing data cell)
currslot = slot number in last start cell processed
status[j] = status of slot j (options are ack(0) ... ack(d))
ackset[j] = subset of downstream_neighbors that have ack'ed slot j

if type = start and status[i] = ack(d) then
    status[i] = ack(0)
    ackset[i] = {}
    currslot = i
    forward cell to output_set
if type = start and status[i] != ack(d) then
    currslot = i
    forward cell to (output_set - ackset[i])
if type = data then
    forward cell to (output_set - ackset[currslot])
if type = end then
    forward cell to (output_set - ackset[i])
if type = ack then
    ackset[i] = ackset[i] + {downstream_neighbor}
    update status[i]
    if status[i] = ack(d) then
        forward ack to upstream_neighbor
    else
        discard cell
```

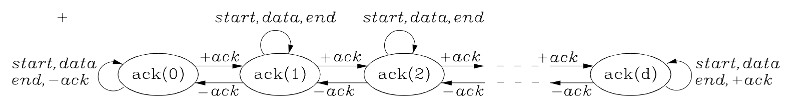
**Fig. 2.** State Machine for Monochromatic Algorithm

this state it forwards data cells and the end cell. When the first acknowledgement is received, it notes which downstream neighbor acknowledged the cell and proceeds to state $ack(1)$. Additional acknowledgements trigger further transitions. The transition to state $ack(d)$ also triggers the forwarding of an acknowledgement to the upstream neighbor. When in state $i$ for $i < d$, new start, data or end cells are simply forwarded to those downstream neighbors that have not yet acknowledged the packet. The program fragment in the figure shows the processing that would be done at a typical switch in a reliable multicast connection in response to the various types of cells. Note that `status[j]` need not really be a separate variable, since its value is implied by the value of `ackset[j]`. We've chosen to show it separately only for clarity of exposition.

Unfortunately, the state machine in Figure 2 is deficient in two respects. First, consider what happens if a given switch completes a packet and goes to state $ack(d)$, sends its acknowledgement to its upstream neighbor, and then the acknowledgement is lost before it reaches the upstream neighbor. Eventually, the sender will retransmit the packet beginning with a new start cell. The switch whose acknowledgement was lost should recognize this situation and simply convert the start cell to an acknowledgement cell and return it, discarding the subsequent data and end cells. However, the state machine as written, will treat this as a new transmission and forward it on to the downstream neighbors. Sequence numbers in the transport level packet can prevent the receiving end systems from being confused by this redundant transmission, but they will be forced to do some unnecessary work. The more important problem with this approach however, is that its correctness requires that the sender never initiate a retransmission if there is any possibility of the receiver still sending an acknowledgement. There are cases where the combination of a slow receiver, lost control cells and unlucky timing can result in the sender thinking a packet has been correctly received, when in fact, it has been lost.

To correct this problem, we add a *color bit* to the slot numbers carried in the various control cells. We require that senders alternate the color of consecutive packets sent with the same slot number. This leads to the state machine shown in Figures 3 and 4. Here, `cell_color` represents the color bit of the control cell being processed and `currcolor` represents the color bit of the most recent start cell. In this algorithm, the state machine is in either the leftmost or rightmost state when propagating a packet for the first time. As acknowledgements of the proper color are received, the state machine moves to the other end. Thus, when processing red packets, `ack(i)` designates states in which `i` acknowledgements have been received, while when processing black packets, it designates states in which `i` acknowledgements are still expected. To distinguish this algorithm from the original, we refer to it as the *bichromatic* algorithm and the original as the *monochromatic* algorithm. Note that the bichromatic algorithm correctly handles lost acknowledgements and its correctness is independent of timing considerations [1]

[1] The author thanks Andy Fingerhut for convincing him that the deficiencies of the monochromatic algorithm should be corrected and showing how they could be cor-



```
if status[i] = ack(0) then
    if type = start and cell_color = red then
        ackset = {}
        currslot = i
        currcolor = red
        forward cell to output_set
    else if type = start and cell_color = black then
        convert cell to ack and return to upstream_neighbor
    else if type = data or type = end then
        forward cell to output_set
    else if type = ack and cell_color = red then
        ackset[i] = ackset[i] + {downstream_neighbor}
        discard cell
    else if type = ack and cell_color = black then
        discard cell
else if status[i] = ack(d) then
    if type = start and cell_color = black then
        ackset = output_set
        currslot = i
        currcolor = black
        forward cell to output_set
    else if type = start and cell_color = red then
        convert cell to ack and return to upstream_neighbor
    else if type = data or type = end then
        forward cell to output_set
    else if type = ack and cell_color = black then
        ackset[i] = ackset[i] - {downstream_neighbor}
        discard cell
    else if type = ack and cell_color = red then
        discard cell
```

**Fig. 3.** State Machine for Bichromatic Algorithm

The use of timeouts to detect lost packets can delay retransmission unnecessarily, limiting overall performance. Frequently, it's possible for a receiver to detect that a packet has been lost and immediately send a negative acknowledgement to the sender, requesting a retransmission. (In a typical sliding window protocol operating over an "order-preserving" network, the arrival of a packet with a sequence number different from the "next" one in sequence indicates one or more lost packets.) When a packet is lost in a reliable multicast connection, all receivers downstream of the point where the loss occurs may detect the loss

rected withot adding any state.

```
else if status[i] = ack(1) or . . . or status[i] = ack(d-1) then
    if type = start then
        currslot = i
        currcolor = cell_color
        if currcolor = ews then
            forward cell to output_set - ackset[i]
        else
            forward cell to ackset[i]
    else if type = data or type = end then
        if currcolor = plus then
            forward cell to output_set - ackset[i]
        else
            forward cell to ackset[i]
    else if type = ack and cell_color = red and currcolor = red then
        ackset[i] = ackset[i] + {downstream_neighbor}
        if ackset[i] = output_set then
            forward ack to upstream_neighbor
        else
            discard cell
    else if type = ack and cell_sign = minus and currcolor = black then
        ackset[i] = ackset[i] - {downstream_neighbor}
        if ackset[i] = {} then
            forward ack to upstream_neighbor
        else
            discard cell
    else
        discard cell
```

**Fig. 4.** Bichromatic Algorithm (cont.)

and send a negative acknowledgement. To ensure scalability of reliable multicast protocols in the presence of negative acknowledgements, the network should return only the first negative acknowledgement for a particular lost packet. This requires that the switches keep track of whether a nack has been sent for a given packet, and if so to suppress forwarding of further nacks. This requires adding some additional state to the switch state machine.

In particular, for each transmission slot, the switch must keep an additional bit that is cleared when a new packet is sent using that slot and set when the first nack for that slot number is received. Start cells associated with retransmissions do not re-enable nack forwarding. Referring to the picture at the top of Figure 3, the nack bit would be cleared on a transition from `ack(d-1)` to `ack(d)` or on a transition from `ack(1)` to `ack(0)`. If a negative acknowledgement is received when the nack bit is 0, the nack is forwarded upstream and the nack bit is set, but the state machine remains in the same state. Note that the correctness of the protocol still relies on the positive acknowledgements; nacks merely improve performance when packets are lost.

In [13], it is shown how these mechanisms can be used in combination with a conventional sliding window protocol to provide reliable one-to-many multicast. This requires that the number of transmission slots be set equal to the maximum window size (in packets) of the protocol.

## 3 Many-to-Many Multicast with Fully Ordered Delivery

In distributed computing applications, it is often desirable to have not just reliable one-to-many communication, but also reliable many-to-many communication, in which any member of a group can reliably send a packet to the other members. Moreover, in some cases it can be important for the packets to be ordered so as to provide consistent delivery order to all receivers. The most straightforward (and efficient) way to implement reliable many-to-many multicast is to have one end system act as a *relay node* receiving packets from all senders, ordering them and forwarding them on a reliable one-to-many connection to all receivers. In this section we describe mechanisms for implementing reliable multicast in this way. However, we also show, in the next section, how these mechanisms can be extended to provide reliable many-to-many multicast without use of a relay node.

The use of a relay node breaks the reliable multicast problem into a many-to-one problem and a one-to-many problem. Since the one-to-many problem has already been addressed, we focus here on the many-to-one problem. There are two issues that must be addressed in the many-to-one problem. First, we need a way to enable the receiver to identify cells coming from different senders that are sharing a single virtual circuit. Second, we need a mechanism to allow the receiver to acknowledge a packet sent by a particular sender, while using a single virtual circuit. Of course, one could simply address this problem using point-to-point virtual circuits between the senders and the receiver, but this leads scaling limits that we seek to avoid.

There are a variety of approaches one can take to sharing a single many-to-ome virtual circuit among multiple senders. The different options are discussed in some detail in [13]. We have concluded that the most attractive general approach to this problem is a contention based scheme in which switches observe the flow of packets from different senders and perform collision resolution at every point where packets from different senders come together. This allows senders to transmit packets without coordinating their transmissions. In the simplest variant of this scheme, the switches only allow one packet at a time to propagate through a "merge point." Unfortunately, this leads to unacceptably high collision probabilities unless the connection is very lightly loaded.

To get better performance using local collision resolution, we allow multiple packets to propagate through a merge point at the same time by implementing dynamically assigned *subchannels* within each virtual circuit. When the start cell of a packet arrives at a merge point, the packet is assigned an outgoing subchannel, a local mapping is created and subsequent cells in that packet are forwarded on the assigned subchannel. The outgoing subchannel is released when the end cell of the packet is received, or on expiration of a timeout. If all outgoing subchannels are in use when the start cell of a packet arrives, then the packet is discarded.

To implement subchannels, we need to add a subchannel identifier to every cell. Because the subchannel field is required in data cells as well as start and end cells, it needs to be kept small so as not to take away too many bits from other parts of the ATM cell header. Fortunately, it does not take a large number of subchannels to give good performance. If we have $n$ sources in a many-to-one virtual circuit, $h$ subchannels and an average of $m$ busy sources, the probability of a burst being blocked due to the unavailability of any subchannels is approximately

$$\sum_{i=h}^{n-1} \binom{n-1}{i} p^i (1-p)^{(n-1)-i}$$

When $n = 1000$, $h = 15$ and $m = 1$ the probability of a burst being discarded is less than $10^{-12}$. If we increase $m$ to 4, this becomes .00002. For $m = 8$, it is .02. To accommodate applications in which we expect more simultaneously active sources, we can use multiple virtual circuits with sources randomly selecting a virtual circuit on which to transmit. While the subchannel remapping takes place only within each virtual circuit, the number of virtual circuits needed to give low collision probabilities is less than the average number of active sources.

With 15 subchannels, a subchannel number can be encoded in four bits, making it possible to use the GFC field of the ATM cell header to carry the subchannel number (we leave one value unused, as an idle subchannel indication). At each merge point, a switch must store the status of all the outgoing subchannels. This status information includes the incoming subchannel number of the burst using the given output subchannel (if no incoming burst is currently using the output subchannel, the stored value is the idle subchannel value), the incoming branch that the burst is coming in on and timing information used to determine when a given entry has been idle for more than the subchannel timeout period.

To make the reliable multicast mechanism useful in a general setting, it's important that there be some strategy for interoperability between switches that implement reliable multicast and those that don't. For the one-to-many mechanisms described in the previous section, interoperability is easy, since all that is required of switches that do not support the reliable multicast is that they be capable of forwarding the control cells and data on point-to-point virtual circuit segments. For the many-to-one case, the subchannel remapping mechanism can interfere with interoperability. For switches that propagate the GFC field without change along point-to-point virtual circuit segments, there is direct interoperability. However for switches that overwrite the GFC field (the usual case), some alternative approach is needed. The simplest way to address this is to allocate a block of consecutive virtual circuit identifiers on the link from a reliable multicast switch to a "standard" switch, and remap the virtual circuit subchannels to distinct virtual circuits. This is done by simply adding the subchannel number of forwarded cells to the first virtual circuit identifier in the allocated range. Similarly, when coming from a standard switch to a reliable multicast switch, distinct virtual circuits can be remapped to a single virtual circuit with subchannels. This mechanism for converting virtual circuit subchannels to consecutive virtual circuits is particularly useful in the context of delivery to end-systems, where the network interface circuitry will typically not be able to directly interpret the subchannel information. The use of distinct VCs at this point allows conventional network interface circuitry to properly demultiplex the arriving packets into separate buffers.

The collision resolution mechanism allows senders to transmit packets to the relay efficiently, but we still need a way for the relay to send acknowledgements back to the sender of a packet, without requiring a point-to-point virtual circuit for each sender. The most straightforward way we have found to do this is to implement a simple *trace-back* mechanism, that allows a cell sent in reply to a cell previously received on the many-to-one path to be returned to the same place. The trace-back mechanism requires a one-to-many virtual circuit that has the same branching structure as the many-to-one virtual circuit that it is used with. Control cells sent on the many-to-one path (i.e. start of packet cells) include a *trace field* that switches on the path can insert trace information into. The relay includes the same trace information in cells it sends on the upstream path back to the sender and each switch along the path uses this information to forward the cell to the proper upstream branch.

## 4 Many-to-Many Multicast Without a Relay

If one is not concerned with the provision of consistent delivery order to all receivers in a reliable multicast connection, it is possible to directly combine a many-to-one connection with a one-to-many connection, eliminating the intervening relay node. This requires extending the acknowledgement suppression mechanism to recognize different subchannels. The main cost of this extension is that for each of the 15 subchannels, the switch entry must keep track of which transmission slot the last start cell on that subchannel specified (so that the data cells for the subchannel can be forwarded to the proper downstream branches).

Now, because this allows multiple sources to send packets directly into the one-to-many part of the connection, sources that are sending information concurrently must use distinct transmission slot numbers. The end systems can handle this in a variety of different ways. One is to simply assign slot numbers to sources statically. This is a reasonable approach when the number of senders is small. The other option is for the senders to allocate transmission slot numbers dynamically, using any general distributed resource allocation algorithm.

The elimination of the relay node makes it possible to structure a multicast connection that does not funnel through some common point and then funnel out again to all the receivers. Instead we can have a more distributed situation in which data flows from senders through the multicast connection tree, branching at various points to reach the receivers.

## 5 Implementation in WU Gigabit Switch

The mechanisms described above are being implemented in the Washington University Gigabit Switch (WUGS). This section provides some of the implementation details. Readers are referred to [4] for background on the WUGS architecture and to [13] for a more detailed account of the reliable multicast mechanisms.

The WUGS architecture breaks large multicast connections into binary copy steps. For example, suppose we have a multicast connection in which each cell arriving at a given switch is to be forwarded to four outputs. In the WUGS architecture, we would select two ports of the switch to act as *recycling ports* for the connection, and send arriving cells, not directly to the output ports, but to the recycling ports, instead. Now, every output port of the switch has a direct data path back to its corresponding input port (that is, there is a data path from output $i$ back to input $i$), allowing cells to be recycled. So, when the two copies of the cell in the example arrive at the recycling ports, they are sent back to the input side of the interconnection network, and sent through the network again with the four copies produced in this pass forwarded to the four required outputs. To accomplish this, a virtual circuit table lookup is performed at the input port where the cell first arrives and at each of the two recycling ports, before it is sent back through the interconnection network. The table entries, in each case, specify the pair of switch output ports that the cell is to be sent to next and the virtual circuit identifier that is to be used to access the next table entry.

The use of binary copying and recycling, together with the particular interconnection network design used in the WUGS architecture yields a system that has optimal scaling properties. Moreover, it makes the implementation of reliable multicast particularly straightforward. With binary copying, the one-to-many mechanism described in section 2 has $d = 2$, which means that for each transmission slot defined for a given virtual circuit, we need just two bits of state information for positive acknowledgements and a third bit to suppress negative acks as well. This allows one to implement one-to-many multicast connections supporting large end-to-end transmission windows without using much memory in the switch's virtual circuit tables. If one considers the overall reliable multicast connection, the memory required is 3 bits per transmission slot for every receiver in the reliable multicast. Since each end-system maintains buffers that consume far more memory than this, the overhead in the switches is acceptably modest.

Collision resolution is also handled most easily when the merging of data flows from different senders is done on a binary basis. The ability to recycle cells from outputs back to inputs, makes it possible to do this as well. With binary merging the subchannel mapping information can be kept quite small. In particular, for each of 15 outgoing subchannels we require 1 bit to specify the upstream branch using that outgoing subchannel, 4 bits to specify the input subchannel number and two bits of timer information, used to ensure that subchannels are eventually released, in the event of a lost end-of-packet cell.

There are several cell types that make up the reliable multicast protocol:

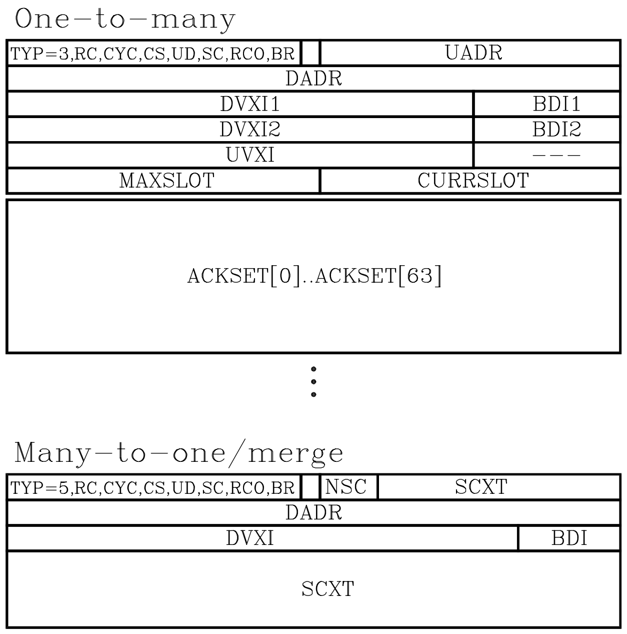


**Fig. 5.** Virtual Path/Circuit Table Formats for Reliable Multicast

start cells, end cells, acknowledgement cells, negative acknowledgement cells, reject cells and data cells. The start, end, ack, nack and reject cells are all encoded using the ATM Resource Management cell type (PTI=110), with the first byte of the payload equal to the Reliable Multicast Protocol ID (123). The second byte of the payload contains a cell type field (0 for NOP, 1 for start, 2 for end, 3 for ack, 4 for nack and 5 for reject). The next two bytes contain the transmission slot number used for the one-to-many part of the protocol. The first bit of this two byte field is taken as the current color bit. The next eight bytes contain trace information, the subsequent eight bytes are reserved for future use, and the remainder of the payload is available for end-to-end information and is passed through by the switches without modification. In both data cells and control cells the GFC field of the standard ATM cell header is replaced by a subchannel field in which values 0–14 represent valid subchannels and 15 is reserved as an idle subchannel indicator.

There are several different virtual circuit table entry formats. Figure 5 shows the two key formats for reliable multicast. The first of these is used in a one-to-many connection, as described in section 2. The second is used in a many-to-one connection as described in section 3. The one-to-many format implements one node in a binary tree. `UADR` and `UVXI` identify its parent in the tree and `DADR` and `DVXI1` and `2` identify its children in the tree. `MAXSLOT` is the maximum number of transmission slots that the connection can support and `CURRSLOT` is the current transmission slot number. Since different connections may require different numbers of transmission slots, we allow a given reliable multicast virtual circuit to use multiple consecutive table entries. The control software that sets up the virtual circuit is responsible for allocating these table entries. As shown in the figure, each entry contains six words of four bytes each, so each added table entry supports 64 transmission slots.

The many-to-one format implements one binary merge point in the many-to-one part of a connection. It stores a complete subchannel mapping table (tt SCXT) as shown in Figure 5. When a start cell is received on a connection, the VXT entry is read and an unused outgoing subchannel is selected from the SCXT stored in the VXT entry. The identity of the incoming branch and subchannel is stored in the entry. When data cells are received, their input branch and subchannel number are used to select the right entry from the SCXT. Their subchannel field is then modified appropriately and they are forwarded to the downstream node in the tree.

As discussed in section 3, start cells in a many-to-one connection acquire trace information to allow acknowledgement cells to be efficiently returned to the proper sender. This is done by adding one bit of trace information to the trace field of the start cell as it passes through a binary merge point. Acknowledgement cells flow back on a separate one-to-many connection with the same branching structure as the one-to-many connection and are routed using the trace information.

The two virtual circuit table entry types described here are also used in general many-to-many connections. The complete implementation requires several other entry types which cannot be described here due to space limitations. A more complete description of the implementation can be found in [13].

## 6 Performance Analysis

If a packet is delivered to all receivers in a one-to-many connection the first time it is sent, the amount of work done by the sender is about the same as for sending a point-to-point packet. Similarly, for each of the receivers. Furthermore, no matter how many times a given packet is lost, the receivers who got it the first time need do no extra work on behalf of other receivers. In fact, the amount of work that a receiver must do per packet received is essentially the same as for point-to-point transmission, since the probability of any single receiver not receiving the packet is essentially the same as in a point-to-point connection (there can be some difference, since the multicast connection path length may be longer than would be used in a point-to-point connection). However, in large multicast connections, the sender may have to send packets multiple times before they are received at all receivers. This is true of any protcol in which the network elements do not buffer and retransmit lost packets. The remainder of this section assesses the magnitude of this effect on the scalability of the reliable multicast mechanism.

Consider a multicast connection with $n$ receivers, a maximum sender-to-receiver path length of $d$ links and a probability of packet loss of at most $\epsilon$ on any single link. Let $x_i$ be the number of receivers that have not received a given packet following the $i$-th transmission (we assume that packet losses on different links and in different 'rounds' are independent). For any particular receiver, the probability that it fails to receive a packet when it is first transmitted is $\leq \epsilon d$, hence the expected value of $x_1$ is $\leq \epsilon dn$. Similarly, the expected value of $x_2$ is $\leq \epsilon d x_1 \leq (\epsilon d)^2 n$. Similarly, the expected value of $x_i \leq (\epsilon d)^i n$. Since $x_i$ is a non-negative random variable, the probability that $x_i$ is $\geq 1$ is less than or equal to its expected value. Using this, one can show that for

$$i \geq \frac{\ln n + \ln 1/\epsilon}{\ln 1/\epsilon d}$$

the probability that $x_i \geq 1$ is at most $\epsilon$. For $n = 1000$, $\epsilon = 10^{-5}$ and $d = 6$, this implies that with probability $1 - \epsilon$, all endpoints receive the packet after two transmissions. For $n = 10^6$, $\epsilon = 10^{-4}$ and $d = 10$, four transmissions are enough. For still larger connection sizes, it might be necessary to have some end systems act as repeaters, buffering packets for retransmission to smaller subsets. However, it's hard to imagine a real application that would require this.

## 7 Closing Remarks

We have shown that the mechanisms proposed for reliable one-to-many multicast are highly scalable. Indeed, it does not appear possible to do better unless the switches store packets and retransmit them when lost or unless the end systems play a more active role. For the many-to-one case, a single virtual circuit can support an arbitrary number of senders with uncoordinated bursty transmissions, so long as the average number of packets arriving concurrently at the receiver is small (ideally, an average of one or two). Since many-to-many multicast connections are naturally constrained by the ability of the recipients to sink data, we believe that even very large many-to-one connections will rarely have more than a few simultaneous senders.

This paper does not address the issues of flow control and congestion control, in the reliable multicast context. For a one-to-many reliable multicast, one can use adaptive windowing techniques like those used in point-to-point protocols to adapt the sender's rate to accommodate slow receivers and/or congested links. Similar techniques are applicable to a many-to-one connection. The use of flow/congestion control on a one-to-many multicast does have the effect of slowing the connection data rate to that of the slowest receiver or most congested link. For distributed computing applications, this may well be the right thing to do. For information distribution applications, it's not clear that this is an appropriate approach. Indeed, there may be no single approach that is really suitable for all applications, as argued in [6].

We have neglected the problem of dynamically updating a reliable multicast connection in progress. Adding a new endpoint requires proper initialization of the acknowledgement state information for the new branch. The most straightforward way to accomplish this is as follows. When requested to add a new endpoint, the network signaling system can allocate the appropriate resources for the new branch, clearing all the acknowledgement information, but not linking the new branch into the connection. It then asks the connection "owner" to inform it when it is safe to update the connection. When all outstanding packets have been acknowledged, the owner can signal to the network to perform the update. While the update is in progress, all senders must refrain from sending any new packets on the connection. If pausing a connection during updating is unacceptable, an alternative is to maintain two parallel connections, one for normal use and one for transitional purposes. When a new endpoint is to be added, it's first added to the second connection. After the new endpoint has been added to the second connection, the senders begins shifting transmission to the second connection. When there are no outstanding packets on the first connection, the owner signals to the network to add the new endpoint to that connection as well.

## References

1. Armstrong, S. A. Freier and K. Marzullo. "Multicast Transport Protocol," RFC 1301, 2/92.
2. Braudes, R and S. Zabele, "Requirements for Multicast Protocols," RFC 1458, 5/93.
3. Chang, J. and N. Maxemchuk. "Reliable Broadcast Protocols," *ACM Transactions on Computer Systems*, 8/84.
4. Chaney, Tom, J. Andrew Fingerhut, Margaret Flucke and Jonathan S. Turner. "Design of a Gigabit ATM Switch," Washington University Computer Science Department, WUCS-96-07, 2/96.
5. Crowcroft, J. and K. Paliwoda. "A Multicast Transport Protocol," *Proceedings ACM SIGCOMM*, 1988.
6. Floyd, S., V. Jacobson, S. McCanne, L. Zhang, C. Liu. "A Reliable Multicast Framework for Light-weight Sessions and Application Level Framing," *Proceedings of ACM SIGCOMM*, 9/95.
7. Holbrook, H., S. Singhal and D. Cheriton. "Log-Based Receiver-Reliable Multicast for Distributed Interactive Simulation," *Proceedings of ACM SIGCOMM*, 9/95.
8. Papadopoulos, Christos and Guru Parulkar. "Error Control for Continuous Media and Multipoint Applications," Washington University Computer Science Department, WUCS-95-35, 12/95.
9. Pingali, S., D. Towsley and J. Kurose. "A Comparison of Sender-initiated and Receiver-initiated Reliable Multicast Protocols," *Proceedings of SIGMETRICS*, 1994.
10. Shacham, N. "The Design of a Heterogeneous Multicast System and its Implementation Over ATM," *Proceedings of the IEEE Workshop on Computer Communications*, 9/95.
11. Whetten, B., S. Kaplan and T. Montgomery. "A High Performance Totally Ordered Multicast Protocol," *Proceedings of Infocom*, 1995.
12. Turner, Jonathan S. "An Optimal Nonblocking Multicast Virtual Circuit Switch," *Proceedings of Infocom*, 6/94.
13. Turner, Jonathan S. "Extending ATM Networks for Efficient Reliable Multicast," Washington University Computer Science Department, WUCS-96-16, 11/96.