Adaptive Group Multicast with Time-Driven Priority

Mario Baldi [*] Yoram Ofek [†] Bülent Yener [‡]

Abstract

This paper shows how to provide an *adaptive real-time group multicast* (many-to-many) communication service. Adaptive means that the number of nodes that transmit to the multicast group is continuously changing. In order to meet deterministic quality of service (QoS) requirements of a real-time group multicast, some communication resources are reserved. In this work we show (i) how bandwidth is reserved for each multicast group, and (ii) how an active source can *dynamically* share the bandwidth allocated to this multicast group with other active group members.

Quality of service support for a real-time multicast group is based on *time driven priority* [7]. In this scheme the time is divided into *time frames* of fixed duration and all the time frames are aligned by using a global time reference which can be obtained from GPS (global positioning system). Bandwidth is allocated to a multicast group as a whole, rather than individually to each user. The allocation is done by reserving time intervals within time frames in some periodic fashion.

This sort of allocation raises two problems that are studied in this paper: (1) *Scheduling:* how time intervals are reserved to each multicast group, and (2) *Adaptive sharing:* how the active (transmitting) participants can dynamically share the time intervals that have been reserved for their multicast group. The proposed approach is based on embedding multiple *virtual rings*, one for each multicast group. By using the virtual rings it is simple to route messages to all the participants, while minimizing the bound on the buffer sizes and queuing delays. The last part of the paper introduces a *scalable growth* of the multicast group by adding multiple subtrees to the virtual ring.

Early version was published in IEEE INFOCOM 1997.

[*]Dip. di Automatica e Informatica, Politecnico di Torino, Torino, Italy 10129, baldi@leonardo.polito.it. This work was done in part while the author was visiting IBM T. J. Watson Research Center.

[†]IBM T.J.Watson Research Center, P.O.Box 218, Yorktown Heights, NY 10598, (914) 945-3171, ofek@watson.ibm.com.

[‡]Networking and Communication Lab. Department of Computer and Information Sciences, New Jersey Institute of Technology, Newark, NJ 07102-1982, (201) 596-2666, yener@cis.njit.edu. This work was supported in part by the Office of Naval Research under Grant N00014-96-1-0825.

1 Introduction

Group or many-to-many real-time multicast is a demanding service. It is usually exploited by applications which require multiparty communications with guaranteed (sometimes large) bandwidth, small loss, and bounded end-to-end delay and jitter[1].

As the number and the identity of transmitting nodes in a multicast group change over time, real-time multicast protocols should be *adaptive* and *efficient* to provide near full utilization of network resources. This has two implications: (i) the resources should be allocated to the group and not individually to each participant, and (ii) the active nodes (i.e., the source nodes) should *adapt* their transmission rate for dynamic sharing of the bandwidth allocated to their multicast group.

The solution presented in this paper exploits a combination of *space* (virtual ring embedding) and *time* (packet forwarding driven by time). Using virtual rings for many-to-many communications is simple and "natural", since a message sent from one member of the group travels around the ring until it comes back to the sender. We simply exploit this property by embedding multiple virtual rings, one for each multicast group. In a previous work it has been shown that the virtual ring can be instrumental in designing a *reliable multicast protocol* from *bursty sources* [10]. In this work the ring circular property is used to adapt, in real-time, the share of capacity each *active* (transmitting) *node* obtains. It is done in a fair manner according to the number of nodes that are trying to transmit to the group at any given time.

An external timing information which comes, for example, from the global positioning system (GPS) [2], is used to control the forwarding of packets inside the network. This time-driven forwarding is used for:
(i) Minimizing the *queuing delay* bound in the virtual ring in order to provide a guaranteed quality of service (QoS), as we discuss in Section 2.
(ii) Sharing the capacity of a link among multiple virtual rings traversing the link in the same direction, as described in Section 3.
(iii) Adaptive sharing of bandwidth to ensure that each active node in a virtual ring gets a fair share of the capacity, as discussed in Section 4.

Section 5 introduces a method for *scalable growth* of the multicast group by adding multiple subtrees to the virtual ring. Adding listeners (receiving only) nodes on subtrees is simple. However, adding active nodes that transmit from the subtree to the multicast group is more complex and may not be adaptive. This complexity helps understanding some of the virtues of using virtual rings for adaptive real-time multicast.

Real-time group multicast with allocation of resources to the group was addressed in depth by the Tenet Group in the framework of the Tenet Project [3]. Multicast communications are implemented through a set of independent trees, each one rooted at a sender [4]. QoS is guaranteed by giving priority to real-time traffic over best-effort traffic, by performing rate control and scheduling in network nodes, and

[1] The jitter is defined as the difference between maximum and minimum delay.

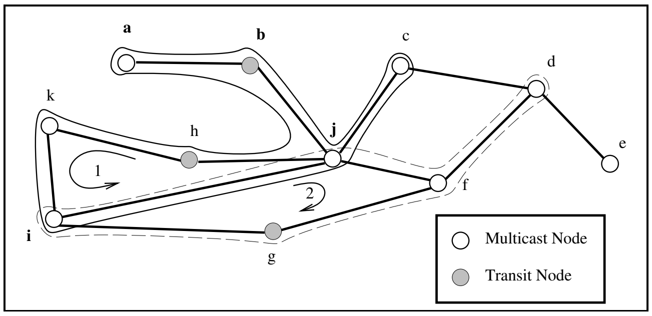

Figure 1: Virtual Rings (VRs) Embedded in the Network.

by restricting access to network resources through call admission control. When resources are allocated on a link which belongs to multiple trees used by a multicast group, a shared allocation is performed [5]. Network nodes, based on the traffic characterization provided by each source, allocate enough shared resource to guarantee the required QoS to each flow. When resource sharing is used, rate control and scheduling can be performed per group (instead of per single flow), which is an important scalable property. Some other real-time transport approaches are based on asynchronous packet forwarding. Such schemes, such as Packet Generalized Processor Sharing (PGPS) [11, 12], bound the packet delay by using scheduling algorithms that approximate the fluid flow service model. Such schemes guarantee a bound on the queuing delay which is inversely proportional to the connection (or session) rate, proportional to the number of nodes traversed by that connection, and proportional to the packet size.

2 Principles of Operation

2.1 Network Model

In this work we consider a general topology network with bidirectional links. A virtual ring (VR) is associated with each group multicast by constructing cycles that includes each participant at least once [10]. For example, Figure 1 shows two VRs (continuous and dashed lines) embedded onto an arbitrary topology network (bold lines). In this example, the two VRs share link (i,j) that will carry packets for both the VRs.

A virtual ring has three types of nodes: (i) active participants (ii) passive participants (e.g., listeners in a video conference), and (iii) the transit nodes that are not the members of the multicast group but they are needed to construct the VR (shaded circles in Figure 1).

Although, some nodes are traversed more than once by the packets pertain to the same multicast group (e.g., f and j in Figure 1), these packets will be received only the first time they reach these nodes.

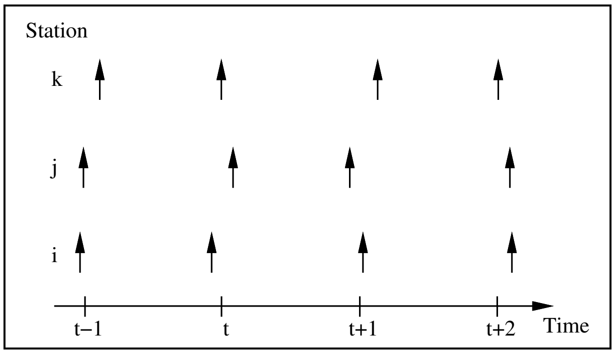

Figure 2: Clock Ticks.

2.2 Network Timing

All the nodes are provided with a global timing structure. The timing information can be obtained from internal sources, as proposed in [9], or external sources like GPS. If, for example, GPS is used for this purpose, each node in the network receives clock ticks from a GPS receiver and uses them to keep its internal clock synchronized. As shown in Figure 2, there is a period of time in which the clocks of all the nodes have the same time value [8].

The time between two successive ticks is called a *Time Frame* (TF) and has a typical duration of 125 μs or 8KHz. Thanks to this global timing structure all nodes have the same instant knowledge of the *temporal value* of the current TF.

2.3 Packet Forwarding

Nodes forward packets according to the *pseudo-isochronous* or *RISC-like* forwarding principle proposed in [8, 6, 7]. The basic idea is that packet hopping in the network is a function of TFs. Precisely, a packet received by a node in TF t, is forwarded by the node so that it reaches the next node along the path in TF t+i (for $i \geq 1$). Note that in wide area internetworking environments i may be large due to the processing and propagation delays. Furthermore, this scheme assumes that the delay between the output port of a node and the output port of the next node along the path is a constant integer number of TFs. If this does not fit actual delays, receiving nodes can introduce extra delay by buffering packets and moving them to the output port at the predefined instant.

Consider as an example the simple topology depicted in Figure 3; links are labeled with the delay (in TF units) between the output port of the node transmitting on the link, and the output port of the receiving node. Node A has a packet to send on its output port (buffer) at TF 2. This packet will arrive to node B and will be ready to be sent from B's output port at TF 6; the destination node C receives the packet at TF 9. The propagation delay from the source A to the destination C is fixed; the jitter is bound by two TFs and is due to the uncertainty on when node A's

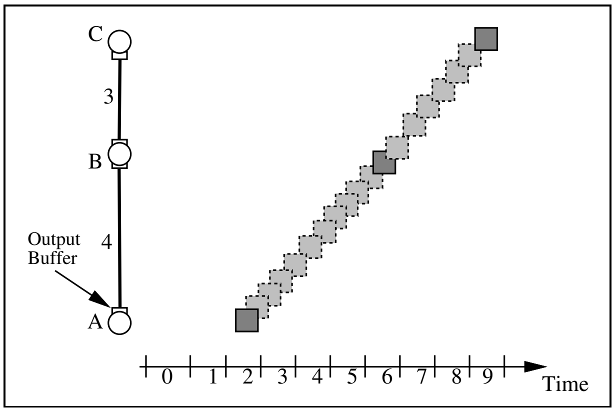

Figure 3: Packet Forwarding.

transmission begins, during TF 2, and when node C's reception ends, during TF 9.

The packet forwarding on a VR follows the same principle such that a packet is forwarded along the VR and after a fixed number of TFs (say n_r) it is received back by its source node. Note that the integer n_r gives the *length* of the VR in terms of time frames.

2.4 Bandwidth Allocation

Some bandwidth is allocated to each multicast group and is shared by the members of the group that are ready to transmit. Section 3 shows how bandwidth allocation is performed while Section 4 shows how bandwidth is dynamically shared among the active nodes of the group.

We assume that a node in the network can forward from its output port up to $125(\mu s) \cdot B(Mb/s)$ bits during each TF. For example, if the data rate (B) of a link is 100 Mb/s then 12.5 Kb can be transmitted in every TF. The source of the bits can be from this node, from its multicast group, or any other multicast group sharing this link. Note that $125(\mu s) \cdot B(Mb/s)$ is also the output buffer size requirement on every port. It is easy to see that this establishes a *small bound* on the queuing delay of every node along the virtual ring. If the many-to-many group multicast would be done over a tree, the propagation delay would be shorter but the queuing delay bound would be much larger. This is significant, since the delay bound is a critical parameter for real time applications.

The bandwidth is allocated on a link for a VR (i.e., multicast group) by reserving time intervals in some TFs in a periodic manner. (Unused reserved bandwidth can be used for best-effort traffic). For example, allocating a 64 Kb/s for a multicast group means the transmission of 1 byte[2] (i.e., $0.064 \cdot 125 = 8$ bits) during each TF

[2]In the following examples when dealing with bandwidth we consider the gross bandwidth allocated to a multicast group, not the net bandwidth offered to applications. I.e., the overheads due to protocol encapsulation are not taken into account. This is not a limitation because these overheads

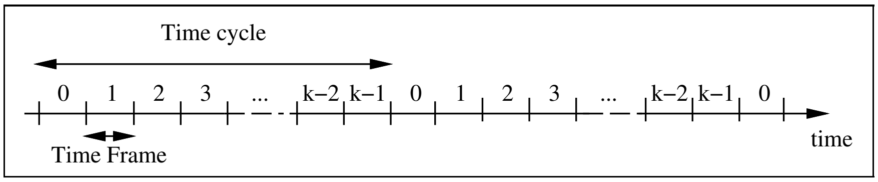

Figure 4: Timing Structure and Time Cycle.

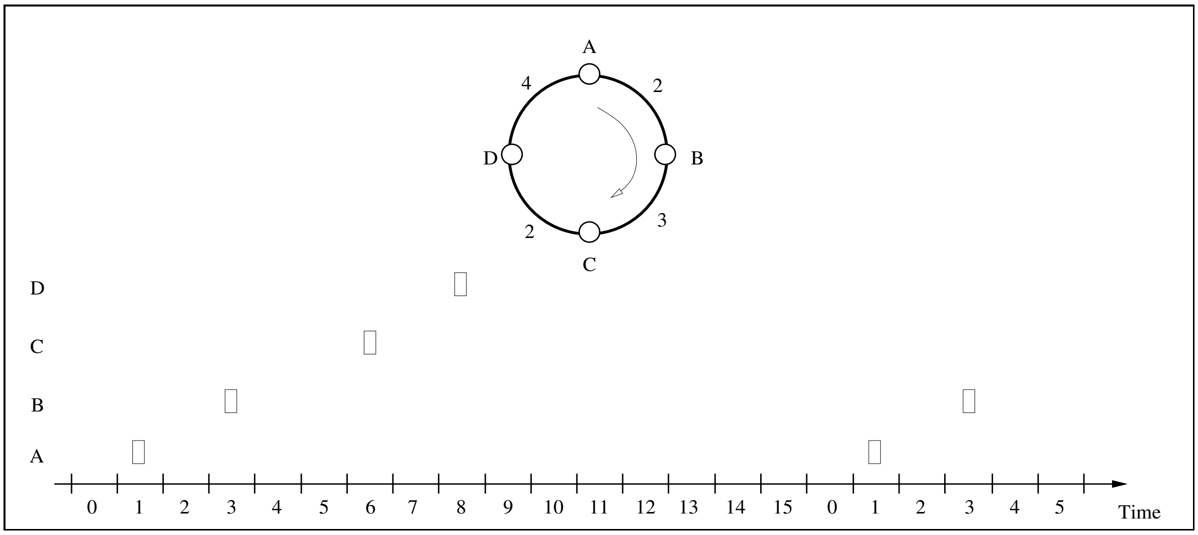

Figure 5: Sample Bandwidth Allocation for a Multicast Group.

(in each virtual node). A packet size of one byte is obviously not practical. Thus, we have to use a packet size that keeps low the percentage overhead of the packet headers. For example for video-conferencing over IP networks, a 500 bytes payload is a reasonable size in order to keep the overhead due to IP, UDP, and RTP headers under 10%. Therefore, in our example, the 64 Kb/s can be allocated to a multicast group by reserving 500 bytes every 500 TFs.

TFs are grouped into *Time Cycles* (TCs) of fixed length k and are enumerated from 0 to $k-1$, as shown in Figure 4. Allocation of bandwidth to a multicast group is done by reserving a fraction of the bits that a node in the VR can send during some of the k TFs in the TC.

Figure 5 shows a sample reservation of one packet per TC for a VR. In the figure a rectangle represents the bits reserved during the TF on the virtual ring nodes (i.e., A, B, C and D). Each node on the ring sends or forwards packets of its multicast group during the TFs in which transmission time for packets have been reserved, i.e., it uses always the same TFs in each TC for transmitting packets pertaining to its multicast group.

Assume for simplicity that packets are fixed size of s bytes. If we reserve a packet

---

can be taken into account by simply having the gross allocated bandwidth larger.

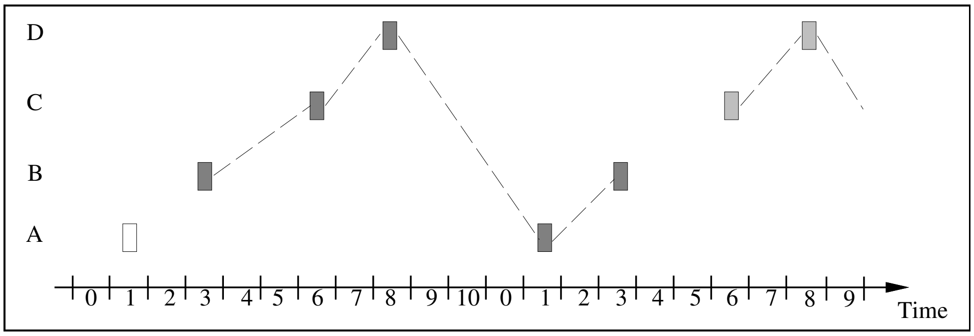

Figure 6: Forwarding of a Packet on a Virtual Ring.

worth of bytes in p_r TFs for a multicast group r, then the bandwidth allocated to the multicast group is given by

$$MB_r = \frac{p_r \cdot s \cdot 8}{k \cdot 125} \text{ Mb/s.} \tag{1}$$

Bandwidth is allocated in multiples of a base quantity which is given by the reservation of 1 packet per TC. In this work we assume $s = 500$ bytes and, as we want the base bandwidth allocation unit to be 64 Kb/s, we fix the number of TFs per TC (length of the TC) to

$$k = \frac{500 \cdot 8}{64 \cdot 10^3 \cdot 125 \cdot 10^{-6}} = 500.$$

Without loss of generality, we assume that only one packet is sent in each TF. As a consequence, bandwidth is allocated by giving to each virtual node on the ring exclusive usage of outgoing links during the whole reserved TFs. Such TFs are said to have been reserved on the node for a specific multicast group or VR.

2.5 Forwarding on a Virtual Ring

As an example, consider now the packet forwarding in the sample VR topology with the bandwidth allocation shown in Figure 5. If node B has a packet to transmit to the multicast group, it waits until TF 3 and then sends the packet which reaches the output port of C in TF 6, as shown in Figure 6. The packet travels on the VR and gets back to node B by TF 3 of the following TC. If node B at this point does not transmit any packet, C will be able to transmit a packet in TF 6 if it is not used for forwarding a packet from some other node.

Figure 7 shows the forwarding of packets on the same VR when $k = 16$. Virtual node A receives B's packet by TF 12, but it has no TF reserved until TF 1 of the following TC. Thus, node A buffers B's packet until the next TF reserved for its VR. Node A is named the *buffering node* of this VR. Each VR must have a buffering node in order to *adapt the length of the VR (virtual ring) to the length of the TC (time cycle)*. (Note that due to the presence of a buffering node a node does not necessarily

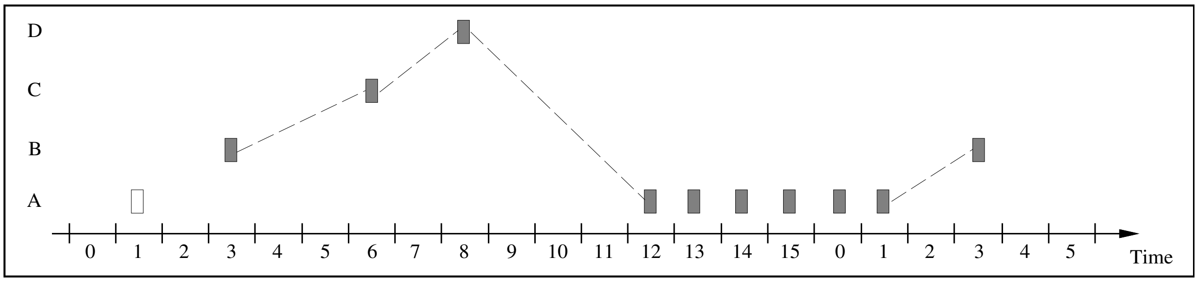

Figure 7: Forwarding of a Packet Originated from B on a Virtual Ring with Buffering Node A.

receive its own packet at the same TF, in the next TC, in which it has transmitted it.)

2.6 Relative Time

When the timing structure is applied to a ring, the concept of *Relative IDentifier* (RID) of TFs is useful for two purposes:

1. To handle the reservation of TFs in the nodes (Section 3) and
2. To dynamically share the allocated TFs among the participants to the multicast group (Section 4).

Each node j keeps its *relative time* by assigning a RID to TF i through the expression:

$$r_i(j) = (i - d_{b,j} + k) \bmod k, \tag{2}$$

where $r_i(j)$ is the RID given to TF i by node j and $d_{b,j}$ is the delay, expressed in number of TFs, experienced by a packet traveling from node b chosen as the *base* for the relative time, to node j. The base node has the property that

$$r_i(b) = i, \ \forall i \ 0 \le i \le k-1$$

i.e., the relative time in the base node equals the absolute time.

Figure 8 shows the relative time for the nodes of the VR shown in Figure 5 when node A is the base node. By comparing the two figures, it is easy to note that the TF reserved to the VR has RID 1 in each node. In general, the relative time has the useful property that each node of a VR identifies the TFs reserved to the multicast group through the same RID.

3 Scheduling

In this section we consider how to allocate resources to a new multicast group M_r with bandwidth request MB_r. As explained in section 2.4, the bandwidth request is

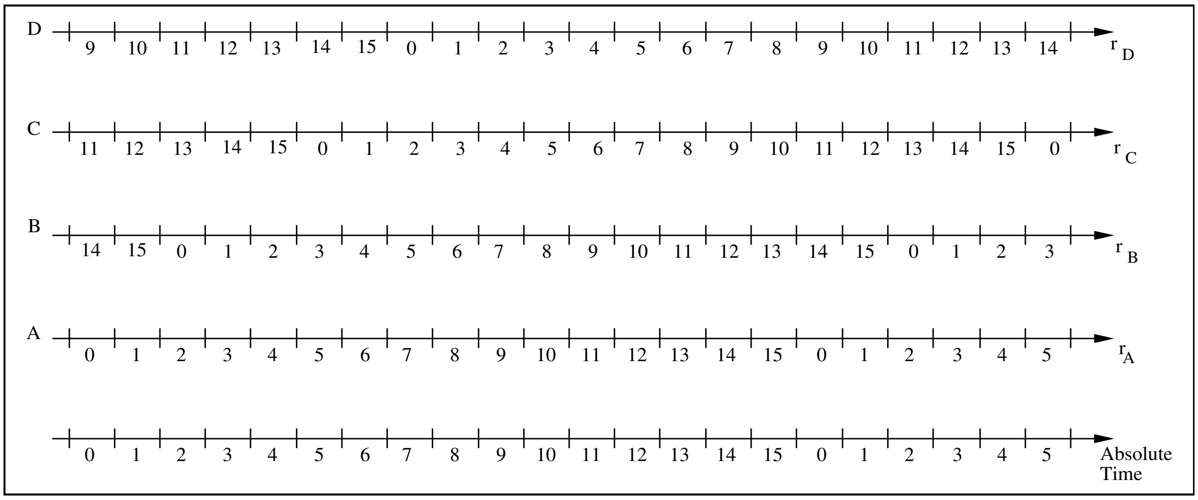

Figure 8: Relative Time.

mapped to a number (p_r) of TFs needed for the new multicast group. Let VR_r be the VR associated with this multicast group and let E be the set of links included in this VR. Since the links of a synchronized network are shared by various types of traffic, some of the TFs (say f) on a link e in E may already be reserved.

It is necessary (*not sufficient*) to reserve p_r unused (available) TFs on each link in E, in order to guarantee MB_r bandwidth for this multicast group. Having p_r available TFs on each link is not sufficient because a TF (on the same link) may need to be allocated to two or more different VRs.

In other words, it is possible that there is available bandwidth (i.e., time frames) for a new multicast group (or to increase the allocation of an existing group), but there is no available *schedule* to assign time frames (of fraction of time frames) such that a time frame is *not* allocated to more than one virtual ring, see an example in Figure 9.

This problem is like the Pigeon Hole Problem where there are enough available TFs on each link, but there is no schedule that can fit them in the right linear (sequential) order, on each of the VRs. The impact of such scheduling conflict is that some of the link bandwidth cannot be allocated to the multicast groups (however, note that this bandwidth can still be used by non-reserved traffic).

The scheduling problem is examined in a hierarchical way. First, we consider a single link used by the multicast group and define the necessary condition for *feasibility* of a link. Next, we consider the whole VR and define *feasible ring scheduling*.

A link of a VR is *feasible* if it has at least p_r available TFs such that p_r satisfies the equation (1). Note that if there is even one link in E with less available TFs than p_r (i.e., the link is highly loaded), then this multicast group cannot be allocated the required bandwidth. Next, we introduce the following definitions to be used in this section:

Definition 1 –Feasible Schedule: *Given a virtual ring VR_r, a schedule σ is called*

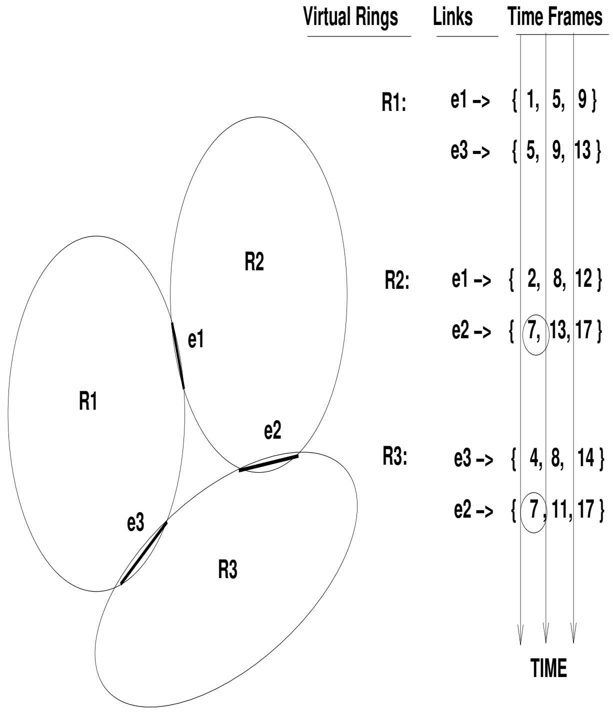

Figure 9: Inability to reserve bandwidth to virtual ring due to scheduling conflicts of time frames

feasible schedule for VR_r if σ chooses exactly p_r (available) TFs on each link such that each TF has the same RID at each node.

Definition 2 –Feasible Ring: *A virtual ring VR_r is called feasible if there exists a* feasible schedule σ *for VR_r.*

Definition 3 –Feasible Network: *Given a network and a set of multicast groups on it, the network is called feasible, if there exists a feasible scheduling for each one of the VRs.*

Proposition 1 *In a feasible network, no TF on a link is used by two different rings.*

3.1 Feasible Ring-Scheduling Algorithm

In this section we present a distributed algorithm to find a feasible assignment of TFs to a VR. The algorithm is initiated by a predetermined origin node (e.g., the base node) and executed by each node in a VR.

In the algorithm (see Figure 11) a node first assigns a RID to each TF by using equation (2). Next, a bit-vector BV_i of size k is constructed at node i. In BV_i a bit is set to 1 (see the example in Figure 10.a), if the corresponding relative TF is not available (i.e., it has been previously reserved for another VR or a point-to-point

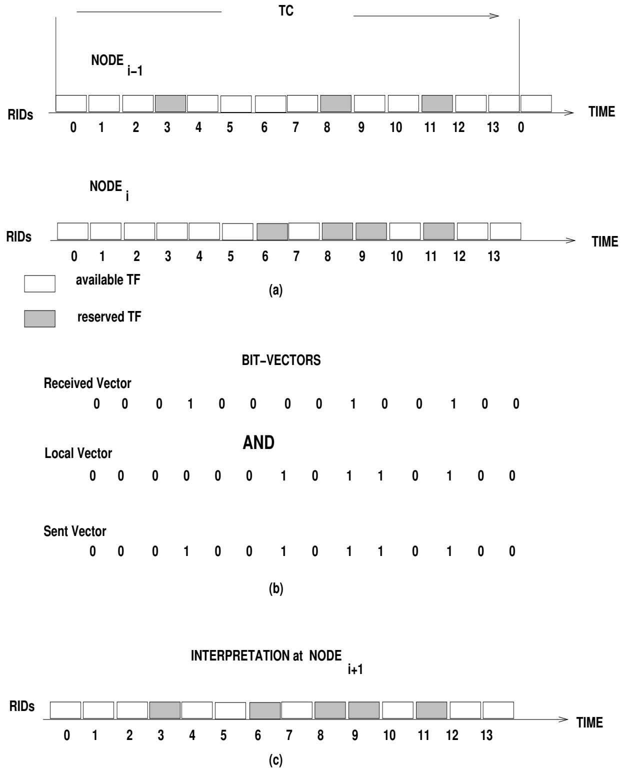

Figure 10: Example of Processing and Interpretation of Bit-Vectors to Determine a Feasible Schedule.

connection).

The origin node on the VR *sends* its bit-vector to its downstream neighbor. Upon receiving a bit-vector, a node performs a logical AND operation between the received and local vector (see Figure 10.b). The result is stored as the updated local bit-vector, and sent to the downstream node (see Figure 10.c).

When the origin receives a bit-vector from its neighbor (step 1), the first phase of the algorithm is completed. Note that at this point each node knows what all the upstream nodes (from the origin to its upstream neighbor) know about available TFs. Thus, when the origin receives the bit-vector from its neighbor, it learns the ring-wise availability of TFs (global information).

If the number of zeros in the received bit vector BV is less than p_r, the origin concludes (step 1.2.1) that the VR is *not feasible*. If BV contains at least p_r zeroes, then any choice of p_r TFs will result in a *feasible scheduling*. The origin calls the procedure $MinMax\_Link\_Schedule(VR, BV, p_r)$ (see algorithm in Figure 13), to choose p_r TFs for optimal scheduling (step 1.2.2). The bit-vector returned by this procedure indicates the TFs reserved for this multicast group. This bit-vector becomes the bit-vector of the reservation for this ring (RBV). The origin sends RBV to its downstream node and it is propagated on the ring to inform all the nodes. When the

Algorithm: *Feasible_Ring_Schedule* performed at node i
Input: VR_r; origin node o_r; p_r
Preprocessing: for each TF on this link
  compute its RID (for example, o_r can be used as base)
  construct BV_i to indicate the status of TFs
Begin
  RECEIVE a bit-vector BV_{i-1}
1 **if** node $i = o_r$ then
  1.1 **if** $BV_{i-1} == RBV$ then STOP
  1.2 **else**
    1.2.1 **if** number of 0's in RBV $< p_r$ then
        print INFEASIBLE
    1.2.2 **else**
        RBV = $MinMax\_Link\_Schedule(VR, RBV, p_r)$
        SEND RBV to downstream node $i + 1$
2 **else**
  2.1 **if** $BV_{i-1} == RBV$ then $BV_i = RBV$
  2.2 **else** $BV_i = BV_{i-1}$ AND BV_i
  2.3   SEND BV_i to downstream node $i + 1$
End

Figure 11: Ring Scheduling Algorithm to Ensure Ring Feasibility.

origin receives RBV (step 1.1), the algorithm terminates and each node on the ring has the same RID for the newly reserved TFs.

3.2 Optimal Selection of TFs

In this section we consider the problem of optimally choosing p_r TFs for a VR. First we introduce an objective function then an algorithm to construct a solution.

3.2.1 Objective Function to Minimize Jitter

Delay jitter determines buffer size at nodes and performance of real-time protocols. The larger the jitter the larger the buffers needed to compensate for it. A uniform distribution of TFs in the TC is beneficial in reducing the end-to-end delay, its variation (jitter), and the size of buffers in the buffering node. Ideally, if reserved TFs are uniformly distributed, the buffering node receives the packets at a (almost) regular pace and forwards them at almost the same regular pace. As a consequence

1. packets are buffered shortly in the buffering node. Thus, the buffer must not be very large and the delay experienced by packets is small;
2. subsequent packets experience similar delays thus yielding small jitter.

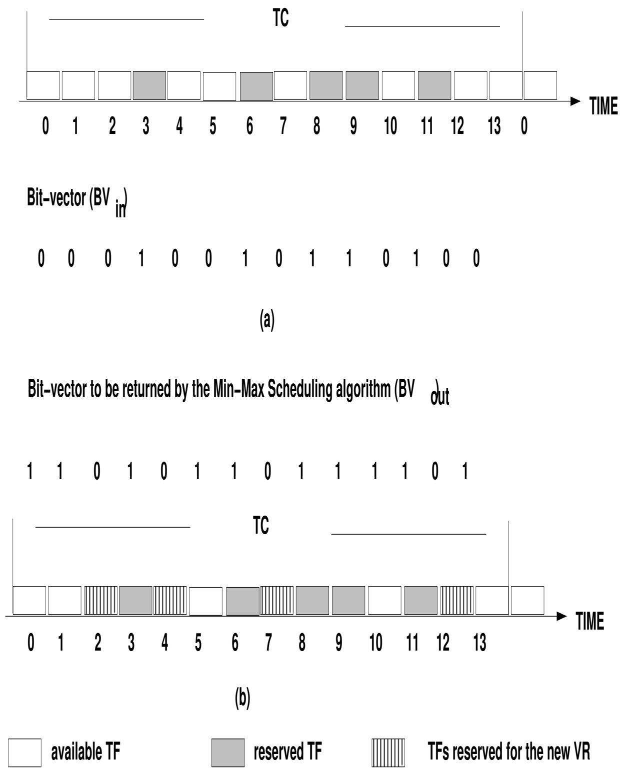

Figure 12: Example of a Time Cycle with 9 Available TFs.

Thus, one objective function for optimal schedule is to ensure equal *time distance* among TFs allocated in a TC. Let FS denote the set of all feasible schedules; we consider as optimal schedule σ the one which satisfies

$$\min_{\sigma \in FS} \left\{ \max_{1 \leq i \leq p_r} \left| [(t(\sigma)_{i \bmod p_r + 1} - t(\sigma)_i + k) \bmod k] - \frac{k}{f} \right| \right\} \tag{3}$$

where $t(\sigma)_i$ denotes the i^{th} TF reserved for the multicast group according to the schedule σ.

Note that if all the TFs were available, the best spacing (time wise) between the p_r TFs would be: $\lfloor \frac{k}{f} \rfloor$. Thus, equation (3) compares the maximum difference between the spacing imposed by each feasible schedule to the best possible spacing and chooses the schedule which minimizes this difference.

Complexity of Link Scheduling

The size of the feasible scheduling set FS is determined by two parameters: p_r and $k - f$: enumeration can be obtained by $C(k - f, p_r)$ (i.e., number of ways to pick a subset of p_r out of $k - f$ distinct objects). Since the enumeration involves factorial operation, if p_r and $k - f$ are close to each other, it is possible to choose the best allocation by enumeration. However, for large factorial computations FS can be too large to enumerate; we propose a min-max algorithm to solve this problem with acceptable complexity.

3.2.2 Min-Max Algorithm

The input of the algorithm is a bit vector BV_{in} (Figure 12.a) which represents at least $k - f \geq p_r$ available TFs for a VR. The algorithm returns a bit vector BV_{out} (Figure 12.b) that indicates which TFs should be reserved for the multicast group.

The algorithm shown in Figure 13 builds a Line-Graph (LG) from the information carried by BV_{in}. Define a *median* of a LG as a node to which maximum *distance* from any other node in the graph is minimum. The *distance* between two nodes is defined as the sum of the link *costs* on the path between them as explained in Figure 13.

The algorithm takes p_r steps and at each step finds a median of an updated (reduced) graph as shown in Figure 14. Each median node corresponds to a TF selected by the algorithm for optimal scheduling. The location of median nodes in BV_{out} is identified by zero while all the other bits are set to 1. Each time a median node is chosen the LG is divided into two connected components (called segments). A new median is chosen from the segment with maximum total cost computed during the selection of the previous median node (see Figure 14). The selected segment is called the *active segment*; initially the entire LG is considered a single active segment.

3.2.3 Objective Function for Load Balancing

In this section we relax the assumption that a TF is used to transmit a single packet. In this case different amounts of bits can be reserved during TFs. Balancing the load over the TFs allows a larger number of VRs to be allocated on the network since it decreases the blocking probability. Thus, we introduce a new objective function which aims at balancing load when scheduling TFs for a VR. This objective function minimizes the differences among the amounts of non reserved bits in each of the k TFs of a TC. The optimal schedule is the one which satisfies

$$\min_{\sigma \in FS} \left\{ \sum_{i=1}^{p(\sigma)_r} \left(\beta(\sigma)_i - \overline{\beta(\sigma)} \right)^2 \right\} \tag{4}$$

where $\beta(\sigma)_i$ is the number of non reserved bits during the i^{th} TF, *after* the schedule σ has been applied. The variable $\overline{\beta(\sigma)}$ is the average amount of non reserved bits over the whole TC. The two objective functions (3) and (4) can be in conflict and thus a tradeoff between them must be found.

Algorithm: *MinMax_Link_Schedule*
Input: VR_r; BV_{in}; p_r
Output: BV_{out} with exactly p_r zeros to indicate
which TFs will be used by VR_r
Preprocessing: construct a weighted line-graph such that
nodes represent available TFs
a link connects two consequent TFs
the weight (cost) on a link is the difference between
RIDs of the end point TFs
mark the line-graph as "active" segment
Begin
1 Set all the bits to 1 in BV_{out}
Begin FOR $i \leq p_r$ DO
2 Choose the median of the active segment and reset the bit
in BV_{out} identified
by the median's index
3 Compute the total cost of each "active" segment
4 Store the costs and end points of the segments in list
5 Choose the segment with maximum total cost from this
list and make this segment "active"
End
6 **Return(**BV_{out}**)**
End

Figure 13: A TF Choice Algorithm to Minimize Jitter in a VR

4 Adaptive Bandwidth Sharing

The adaptive bandwidth sharing concerns a single VR and does not affect any other VR embedded in the network. Active stations actually share the allocated bandwidth by sending their own packets during some of these TFs. Both passive stations and transit nodes just forward packets during the reserved TFs. The set of active stations changes dynamically as, for example, in a video-conference call the set of active speakers changes. The stations participating in a multicast group run a signaling protocol that allows each of them to know the actual set of active stations[3].

As the number a_r of active stations changes, the share of bandwidth available to each station changes and they have to (1) adapt their transmission rate to it (Section 4.1) and (2) agree on which TF each station is using (Section 4.2).

4.1 Rate Adaptation

Through one of the algorithms described in Section 4.2, each active station knows the amount of bandwidth it can use for transmission. When the bandwidth available

[3] This signaling protocol is out of the scope of this paper; some possible candidate protocols are available for the purpose.

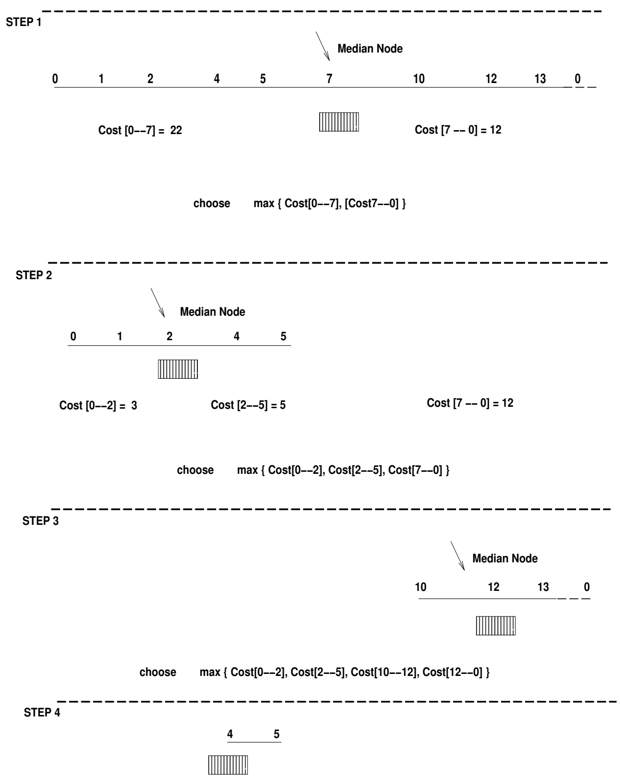

Figure 14: Example of Execution of the Algorithm to Choose 4 TFs.

to an active station changes, the application is notified in order to change the rate at which data is produced.

Some applications require some time in order to make this change. For example, in video-conferencing with compression data is produced through a process starting with capturing video frames, going through a number of processing steps, and ending with quantization and run-length encoding of the video information. The compressed data is buffered and retrieved from the buffer at a constant bit rate. Changing the amount of data produced by this process implies tuning the parameters that regulate some of these processing steps. The earlier the involved processing step, the longer the time for having the change take effect at the network interface.

When a station becomes passive, it notifies other stations and immediately stops transmitting. Each active station realizes that its share of the bandwidth is larger and notifies the application to increase the data generation rate. Even though the increment of the bandwidth share is immediately available to the active nodes, it takes some time before the change in the application generation rate reaches the network interface; i.e., the transmission rate of active nodes does not increase instantly.

When a station becomes active, it notifies other stations. However, it does not start transmitting immediately. All the previously active stations immediately notify their application to lower the data generation rate. Since it takes some time before the actual transmission rate of all the active stations decreases, the new active station waits for its bandwidth share to be available before starting transmitting. We assume that newly active stations wait a fixed time (in the order of one second) before transmitting. This is needed in order to assure that no packet is dropped in either the network interface of active stations or inside the network, due to unavailability of communications resources.

This waiting time is acceptable as the time scale in which stations change their activity status is usually long. For example, in a video-conferencing application a station becoming active implies that somebody is asking the right to speak. Thus, waiting should not be of concern since the new speaker will usually have to wait before getting the permission to speak.

4.2 Bandwidth Sharing

Given the set of the p_r TFs reserved to a multicast group r, the bandwidth sharing problem consists in having each active station using a subset of them to send its own packets[4]. The subset used by each active station should be built satisfying the following criteria:

1. each node gets a fair share of the bandwidth MB_r allocated to the multicast group;
2. the end-to-end jitter experienced by packets is minimum;
3. the unused bandwidth (i.e., the TFs during which no multicast data is transmitted) is minimized[5].

4.2.1 Round Robin Allocation

The allocated bandwidth can be shared fairly among the active stations by having each active station using each TF in a round robin fashion. Active stations transmit their packets in any *free* or unused TF. When a station receives a packet originated by itself, it must not use the next TF to send another packet of its own. This allows downstream stations to use the TF, in a round robin manner, for their own transmissions. The fraction of unused bandwidth due to this phenomenon is:

$$\frac{1}{a_r + 1}$$

This unused bandwidth is analogous to the unused bandwidth due to the token walk time in token ring.

[4]Whenever dealing with TFs in this section, we mean TFs reserved to the VR.

[5]Even though unused bandwidth can be used to transmit best-effort traffic, the VR should try and use as much as possible of the bandwidth allocated to it.

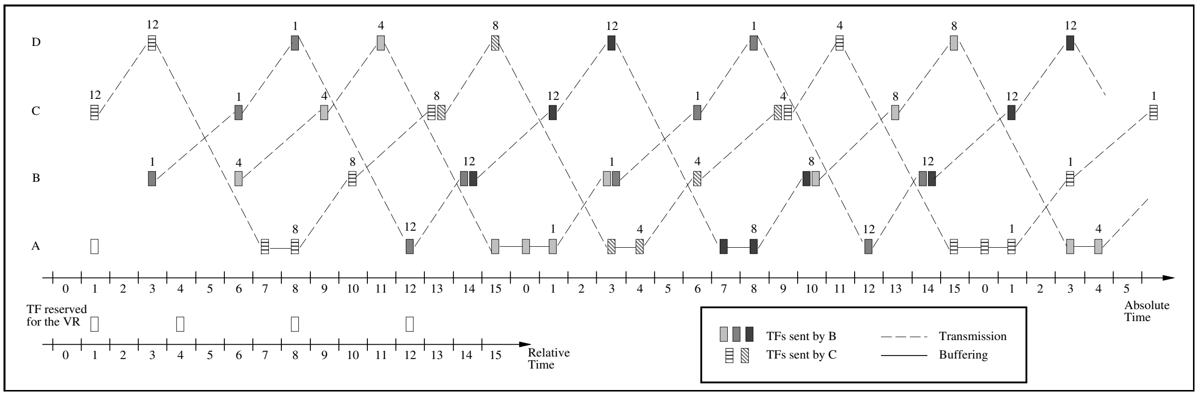

Figure 15: Fixed Allocation of TFs

4.2.2 Fixed Allocation

A fixed allocation scheme can be used to reduce the amount of bandwidth unused due to the bandwidth sharing procedure. According to the fixed allocation scheme, active stations come to an agreement on which TFs each of them is using[6]. When a station receives back a packet sent by itself, it can use the next TF for sending its data. Figure 15 shows an example in which 4 TFs (RID 1, 4, 8, 12) have been reserved for the VR depicted in Figure 5 and nodes B and C are active. Each active node is using two TFs to transmit its data.

Note that the RIDs of the two TFs each node is using change over time because the buffering node A buffers and forwards packets in a different TF. Moreover, some bandwidth is still unused because in each moment only 3 TFs out of the 4 reserved are used. This is due to the behavior of the buffering node, the distribution of the TFs in the TC, and the relationship between the lengths of VR and TC. This situation can be accepted because with this behavior the buffering node delays packets for a short time as shown in the example of Figure 15.

In the example shown in Figure 15 the phenomenon is particularly harmful as the unused TF is always one of the 2 assigned to station C which is getting only half of the fair bandwidth share.

This unfairness becomes smaller as the number of TFs assigned to each station becomes larger. This unfairness can be avoided by applying more complex bandwidth sharing schemes or buffering policies in the buffering node. Note that when dealing with unused bandwidth in the following discussion, the contribution given by the foregoing phenomenon is not taken into account.

The fixed allocation of TFs to active stations is possible only if $a_r \leq p_r$, otherwise a different scheme (e.g., round robin allocation) must be applied.

Nevertheless, one of the main fields of application for real-time group multicasting is large scale video-conferencing, where it is possible to have a large number of listeners

[6]This can be done through some signaling mechanism that is out of the scope of this paper.

(i.e., large p_r) and a small number of active speakers (i.e., small a_r). Only the active speakers can transmit to the group.

When $p_r \bmod a_r = 0$, fixed allocation of TFs lets each active station use $\frac{p_r}{a_r}$ TFs per TC, i.e., active stations share the bandwidth fairly and do not leave unused any fraction of it. The jitter introduced depends on the distribution of TFs in the TC and the allocation of TFs to active stations.

When $p_r \bmod a_r \neq 0$, p_r can be written as

$$p_r = I \cdot a_r + N, \text{ where } I = \lfloor \frac{p_r}{a_r} \rfloor \text{ and } N = p_r \bmod a_r$$

Each active node uses the fixed allocation criteria on I TFs; for the remaining N TFs several approaches can be followed.

1. The N TFs are allocated to the active nodes in a round robin fashion. As a result, with this approach some bandwidth is not used and a large jitter is introduced because usually a node sends I packets per TC and seldom it sends $I + 1$ packets. This exceptional packet received between two regular packets can sharply reduce the packet inter-arrival time, thus yielding a large variation of the end-to-end delay (increase of the delay jitter).

2. N of the a_r active nodes fixedly use $I + 1$ TFs for sending their packets. The bandwidth is not shared fairly, but all the allocated bandwidth is used and the bandwidth sharing scheme does not contribute to the jitter.

3. The N TFs are left unused; the multicast group does not use some of the bandwidth allocated to it, but the bandwidth allocation scheme does not contribute to jitter and it is very simple.

5 Scalable Growth of the Multicast Group

This section discusses how new node can join the multicast group in a scalable manner. The solution extends the virtual ring topology by including one or more *trees*, as shown in Figure 16. Exploitation of trees is particularly suited to include in the multicast group remote and isolated nodes. The discussion is divided into: (i) join as a receiving node, and (ii) join as a transmitting (active) node.

5.1 Join as a Receiving Node

Suppose node I wants to joint the multicast group as a listener, namely, it wants to receive all the information transmitted over the virtual ring. This can be done by the following steps:

Step 1: Find a path to a node on the virtual ring, say node J, as shown in Figure 16, or to a node on an existing path that is already connected to the virtual ring.

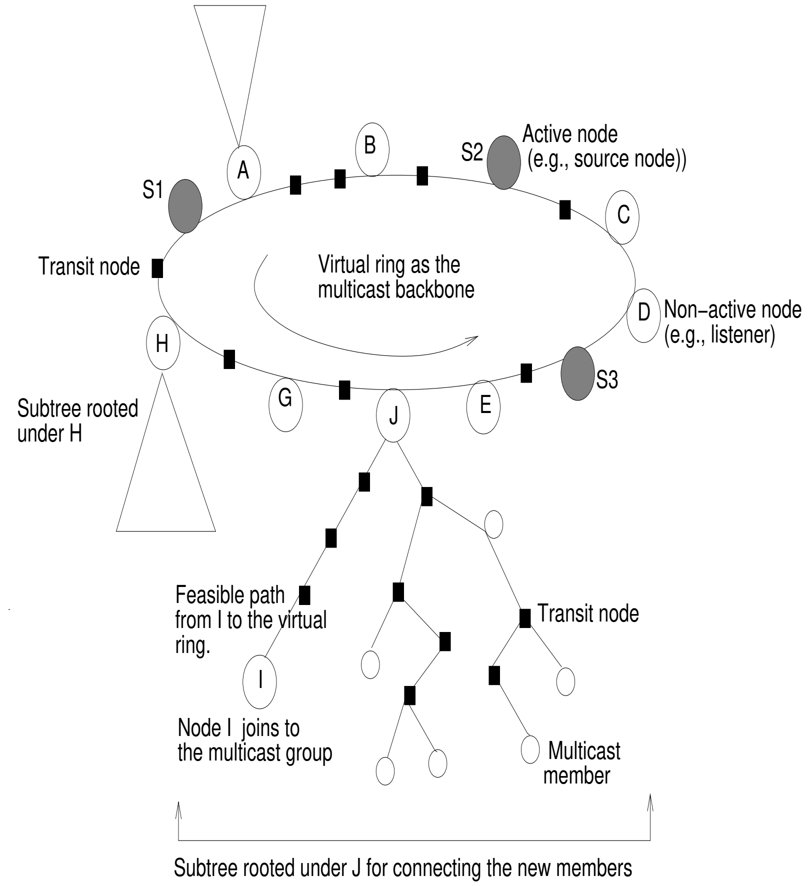

Figure 16: Example: Scalable join operation to include new members into a multicast group

Note that this node can be part of the group or a transit node (as was discussed in Section 2).
Step 2: Find for each link on this path p_r available TFs, such that, there are feasible schedules for the p_r TFs on this path (as was discussed in Section 3).
Step 3: If no such schedule is feasible, then go back to Step 1.

We note that the path connecting node I to the multicast group has two simple properties: (i) it may become a path of a subtree rooted at J, and (ii) the capacity from the virtual ring down the subtree to all receiving nodes on its leaves is the same as the virtual ring capacity.

Thus, adding receiving nodes is simple and scalable. For example, assume there is a virtual ring for a multicast group in North America and a user in Europe wants to listen to the group it can do it, as described before, and without affecting the existing group operation.

5.2 Join as a Transmitting Node

Let's assume that node I, shown in Figure 16, is already a receiving node. However, the user at node I after listening to the group for a while wants to "voice" her or his opinion. Next we discuss the steps needed in order to enable this user to transmit to the group.

Step 1: Check that the subtree is connected to a node on the virtual ring that is a member in the group and not a transit node. If it is a transit node then make this node a part of the multicast group by making it run the real-time multicast protocol.

Step 2: Assume, as shown in Figure 16, that node I is connected to node J on the virtual ring. Node I will signal node J that it wants to become active and node J will signal to the group. Since capacity has not been allocated upward in the subtree, node I should allocate on the path to node J the same capacity reserved to the multicast group on the virtual ring. **Step 3:** When it is possible, node J will signal back to node I: (i) that it can start transmitting, and (ii) what fraction of the virtual ring capacity it can use.

Node I uses a fraction of capacity reserved upward on the subtree rooted by node J

The above join and transmitting operation has the following potential problems that do not exist when the transmitting node resides on the virtual ring:

1. Node J should buffer the packets it receives from node I before it can transmit them into the virtual ring. This can add up to one virtual ring round trip delay to every packet.
2. Multiple active nodes on the same subtree are possible. However, a feasible schedule which ensures allocation of enough capacity for each path connecting an active node to the virtual ring is needed. We note that this is a difficult problem similar to finding a feasible schedule for multiple rings.

We currently investigate performance trade offs for adaptive real-time group multicast over a virtual ring versus over a tree. Recently, it is shown in [1] that constructing a multicast ring requires approximately %25 more links than a multicast tree. However, for group multicasting each direction of a tree must be shared while a ring can be unidirectional. Thus, we investigate how to quantify the buffering and bandwidth requirements with the delay and jitter bounds that can be obtained.

6 Conclusions

The combination of virtual rings and timing information is interesting and intriguing. In this work we have shown how useful this combination can be for *adaptive real-time multicast*, while in a previous work we have shown that it is also useful for *reliable multicast* for bursty data processing applications [10].

The proposed approach to adaptive real-time multicast is characterized by having a *deterministic behavior*. This is important for real-time applications in two ways:

(1) it minimizes the *bound* on the delay variation across each virtual ring, and (2) it ensures no loss due to congestion. The no loss property is important for real-time video applications using compression. In a compressed video stream a loss of a single packet can result in the inability to properly display multiple consecutive pictures, which can annoy the viewer.

There are several intriguing issues for further research. The scheduling and allocation of time frames to the various multicast groups (virtual rings) can be improved by further investigating how to reduce the blocking probability. Blocking is defined as the impossibility of allocating enough bandwidth for a new multicast group while some capacity is still available, but not in the proper order of time frames. Another issue concerns the unused capacity assigned to a virtual ring due to the bandwidth adaptation schemes proposed in this work.

References

[1] S. Chen, O. Günlük, and B. Yener. Optimal packing of group multicastings. *Proc. of IEEE INFOCOM'98*, 1998.

[2] P. H. Dana. Global positioning system overview. http://www.utexas.edu/depts/grg/gcraft/notes/gps/gps.html.

[3] D. Ferrrari, A. Banerjea, and H. Zhang. Network support for multimedia. a discussion of the tenet approach. *Computer Networks and ISDN Systems*, 26:1267 – 1280, 1994.

[4] A. Gupta, W. Heffner, M. Moran, and C. Szyperski. Network support for realtime multi-party applications. In *Fourth International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV'93)*, Lancaster, England, November 1993.

[5] A. Gupta, W. Howe, M. Moran, and Q. Nguyen. Resource sharing for multi-party real-time communication. In *IEEE INFOCOM'95*, Boston, MA, April 1995.

[6] C-S. Li, Y. Ofek, A. Segall, and K. Sohraby. Pseudo-isochronous cell switching in ATM networks. In *IEEE INFOCOM'94*, pages 428–437, 1994.

[7] C-S Li, Y. Ofek, and M. Yung. "Time-driven priority" flow control for real-time heterogeneous internetworking. In *IEEE INFOCOM'96*, 1996.

[8] Y. Ofek. Integration of voice communication on a synchronous optical hypergraph. *INFOCOM'88*, 1988.

[9] Y. Ofek. Generating a fault tolerant global clock using high-speed control signals for the MetaNet architecture. *IEEE T. on Communications*, pages 2179–2188, May 1994.

[10] Y. Ofek and B. Yener. Reliable concurrent multicast from bursty sources. *IEEE JSAC*, 14(3):434–444, April 1997.

[11] A. K. Parekh and R. G. Gallager. A generalized processor sharing approach to flow control in integrated services networks: The sigle-node case. *IEEE/ACM Transactions on Networking*, 1(3):344–357, June 1993.

[12] A. K. Parekh and R. G. Gallager. A generalized processor sharing approach to flow control in integrated services networks: The multiple node case. *IEEE/ACM Transactions on Networking*, 2(2):137–150, April 1994.