Dynamic Join and Leave for Real-Time Multicast

Wolfgang Effelsberg, Eberhard Müller-Menrad
International Computer Science Institute
1947 Center Street
Berkeley, California 94704
effelsberg@pi4.informatik.uni-mannheim.de

TR-93-056

October 1993

Abstract

Many new applications in networks require support for multicast communication. In addition, continuous data streams such as audio and video require real-time performance guarantees to ensure quality of service. We introduce a model for real-time multicast channels and present a set of scalable algorithms for the dynamic joining and leaving of destination nodes in this environment. In particular we present an algorithm for finding a good attachment point to the multicast tree. We also describe detailed admission tests that preserve the guarantees given to existing channels. Our algorithm for a leaving node specifies in particular the resources to be released in the network. We also discuss tree reorganization issues.

1 Introduction

Modern networks have much higher data rates than traditional networks. These high data rates enable new kinds of applications, in particular multimedia applications. However, the inclusion of multimedia data streams into data networks poses new problems: the network must provide transmission in real-time, and many new applications require multicast from one sender to multiple receivers.

Some new applications comprise only a few participants and are short-lived. But others can have hundreds or thousands of participants, and can last for hours or days; examples are telecasts from conferences or remote teaching. In this context the question arises how nodes can join or leave an existing real-time multicast session. This is the topic of our paper.

The background of our work is the Tenet project at ICSI and UC Berkeley. It defines its basic *Tenet Approach* to real-time communication and a sequence of *Tenet Schemes* in which this approach is realized in steps [6, 7, 9]. Tenet Scheme 1 provides real-time unicast channels over internetworks with resource reservation and well-defined admission tests for new channels. Scheme 1 has been implemented. The work presented here is part of Tenet Scheme 2, where the major new feature is support for multicast communication.

Other projects have also developed multicast protocols. A prominent example is the multicast IP protocol of the Internet [3, 4, 5]. It defines an address space for host groups and describes extensions to IP implementations for packet duplication and routing. The source does not know who and where the current destinations are. New destinations can join a multicast by specifying the multicast IP address for which they want to receive packets. The multicast IP protocol has been implemented on many different architectures and is now part of several new operating system releases for workstations. It is popular for workstation conferencing and seminar broadcasting. Multicast IP does not use resource reservation, and does not provide performance guarantees for multimedia data streams.

The ST-II protocol (Internet Stream Protocol Version 2, [19]) was also developed as an Internet protocol suite extension. It is a connection-oriented network layer protocol supporting multicast. Unlike multicast IP, it guarantees end-to-end bandwidth and delay. The guarantee is based on bandwidth reservation at channel establishment time: when a new channel is established, the QoS parameters for bandwidth and delay are negotiated between sender, network (routers on the path) and destinations. The ST-II standard concentrates on the definition of control messages for this purpose; the details of routing, admission tests, resource reservations within a node, and so on, are left to the implementor. ST-II allows dynamic joining of new destinations, but only via the source. The new node has to inform the source by means of a control message, and the source will re-establish the channel including the new member. This solution obviously does not scale well. Several research laboratories have implemented ST-II successfully [20, 18, 12, 13].

The OSI community is also working on a new generation of transport protocols. The concept of a *Quality of Service* requested by the user and guaranteed by the transport service is an important part of these new protocols [2]. Support for multi-peer connections is also included in the new proposal. Details of multicast algorithms or operational implementations have not yet been reported.

None of the approaches mentioned above allow dynamic joining and leaving of nodes under real-time conditions. ST-II comes closest to our approach, but in the current version all join operations have to be managed by the source [19]. Thus, dynamic joins do not scale well. We consider scalable dynamic join and leave operations to be an important requirement for multimedia applications in digital networks.

In Section 2 of this paper we introduce our model for real-time multicast communication; it is largely based on Tenet Scheme 2. Sections 3 and 4 discuss algorithms for the dynamic joining and leaving of nodes during the lifetime of a multicast tree. Section 5 concludes the paper.

2 A Model for Real-Time Multicast Communication

2.1 Links, Multicast Trees and Applications

We discuss multicast communication in the context of a point-to-point topology with typical internetworks in mind. The underlying network consists of a set of nodes interconnected by *links*. These can be implemented in various ways: They can be physical links (e.g. leased lines), ATM connections, connections within a connection-oriented packet-switching network, "tunnels" between routers in an internet (where the intermediate nodes are invisible to the multicast tree nodes), and so on. Thus, the multicast algorithms presented in this paper can be applied to many different types of physical networks, and in particular internetworks.

Each link must be able to provide performance guarantees such as delay and delay jitter bounds for packets routed over the link. These guarantees can be deterministic or stochastic (i.e. “with a probability of 98% the delay will be less than 2 ms“). If an existing network is unable to provide such guarantees, it must be enhanced by new node functions. Obviously real-time multicast can only be achieved if the basic links offer guaranteed performance. For the sake of simplicity we will only consider deterministic guarantees in this paper.

A *multicast tree* (or *multicast channel*) is a $1 \times n$-connection between one source and n destinations ($1 \leq n \leq N$ where N is the number of nodes in the network). Some applications will establish only one multicast tree (e.g. for information distribution), others will establish several trees (e.g. a workstation conference or a newsgroup), and in yet others each receiver will also be a sender. In our model, $M \times N$ communication is always based on M multicast channels. From the user's point of view these channels are independent; in order to keep multicast support simple and general, the coordination and control of the M multicast channels is not built into the network, but left to the higher layers of the protocol hierarchy. For example floor passing [1] or multicast ordering [14, 15] are handled in higher layers.

The nodes in a multicast tree perform the following operations:

- They route packets and forward them to output links (switching). This can include the duplication of packets on several output links if the node contains a branching point of the multicast tree.
- They maintain local resource utilization tables in order to determine the current load of the node (i.e., the amount of resources that has already been reserved) and the acceptability of new requests.
- They accept tree management commands to add or drop new output links for a multicast channel.

The motivation for maintaining resource utilization tables is that resource reservation is a prerequisite for guaranteed real-time performance. We will see that these tables are also instrumental in finding good attachment points for new nodes.

The resource utilization tables can be used for realtime and non-real-time communication. In the case of real-time multicast the table is first used for admission tests. A new multicast tree can only be established through existing nodes, and a new node can only be attached to an existing multicast tree under the condition that the performance guarantees given to existing multicast connections will not be violated. In both real-time and non-realtime communication the information about resource utilization is used for the initial routing of an *optimal* multicast tree, and locating the *optimal* attachment point when joining a new node to an existing tree.

Notice that we have to distinguish between the topology of the links, as defined above, and the topology of a multicast tree. Many multicast trees can be routed over the same link topology concurrently as long as all the QoS requirements of all channels can be met.

The *application* in our sense comprises the upper layers using the real-time multicast network. The service interface between our network and the application corresponds to a transport layer service in the OSI reference model, i.e., we provide end-to-end multicast with performance guarantees. Each end system must also contain a routing function (similar to each host containing an IP implemetation in the internet).

Figure 1 shows an example of a router topology without any connections. We represent routers as diamonds.

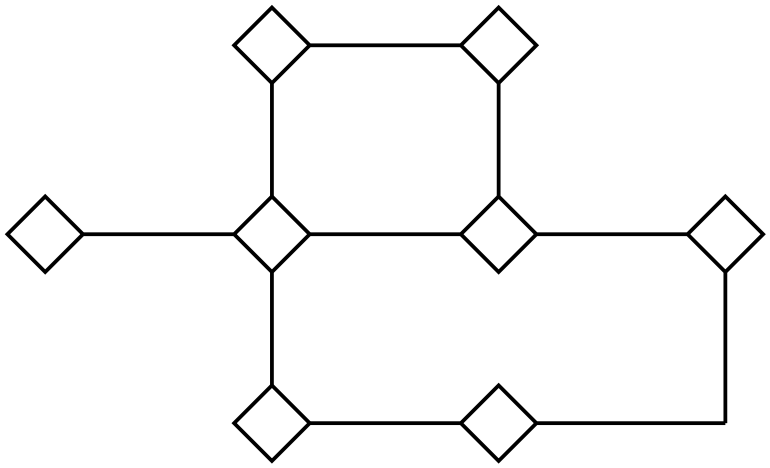

Figure 1: Example Topology of Routers

Figure 2 shows the same topology with two multicast connections. Multicast tree nodes are represented as dots, and applications as circles. $S1$ and $S2$ are the two sources, the Di denote the destinations. Destinations $D4$ and $D6$ are not yet participating in the multicasts. We assume that both sources are sending to the same set of destinations. Note that our multicast tree management can only change the topology of the dots, i.e. of the multicast tree nodes. The topology of the link level routers is usually set up separately by operator commands or long-term network management algorithms; it is neither affected by multicast connection setup and teardown nor by joining or leaving nodes.

2.2 Target Sets

Experience shows that it is very inconvenient for many applications to have to deal with every multicast channel separately. Therefore we introduce the concept of a *target set*. Semantically a target set is a set of nodes interested in a particular topic. When a new channel is established, the application can specify the desired target set rather than listing all destination nodes explicitly. Similarly, a destination node can specify that it wants to join a target set, implicitly expressing its interest in joining all multicast trees currently sending to this target set. The network itself has to maintain the target set membership in a database (e.g. in an X.500 directory). We feel that target sets are a powerful abstraction, providing a user-friendly interface to multi-peer applications and optimization clues to the network itself. We are aware of

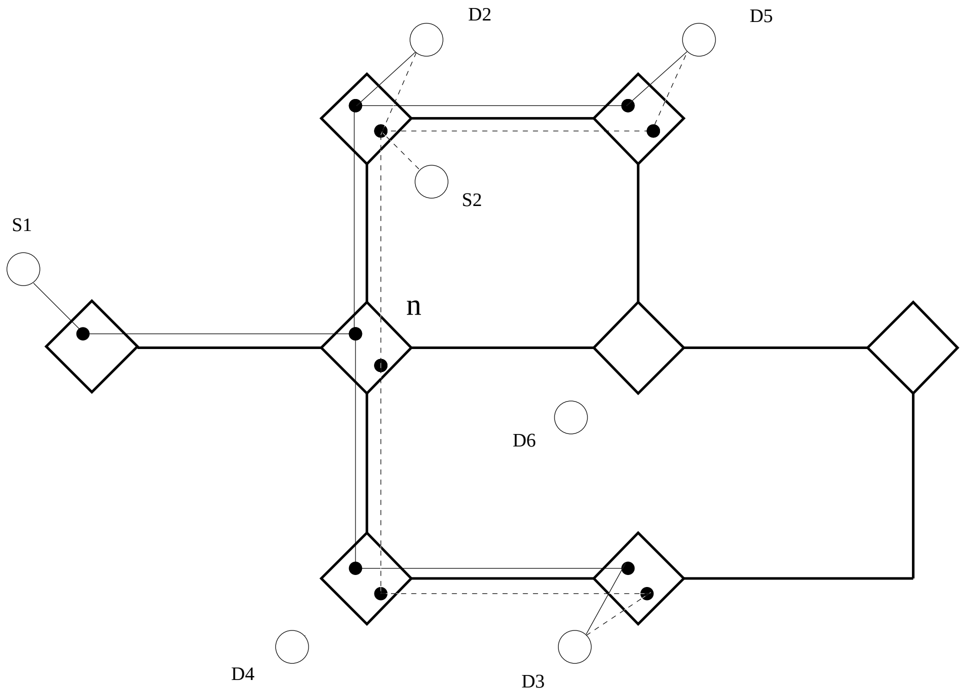

Figure 2: Routers with two Multicast Trees

the fact that this implies the maintenance of more state information in the network, but it also allows the system to keep this information consistent on behalf of all users.

Within a target set each destination can specify its own QoS requirements. Destinations are free to dynamically join or leave target sets, and a destination can be a member of multiple target sets. In the latter case, the same destination may specify different QoS requirements for each target set.

Sources send to target sets, thereby fully abstracting from individual destinations. Multiple sources may send to the same target set, in which case the data are multiplexed at the destinations. A higher layer service is free to add an application-specific protocol to demultiplex the data based on varying criteria, such as the id of the source or some dynamically changing classification keys. It is also free to enforce a sender selection discipline (such as floor passing), or to implement protocols for multicast ordering [14]. In our example in Figure 2, both sources are sending to the same target set consisting of destinations $D2$, $D3$ and $D5$.

If a source wants to send to a target set, the source asks for establishment of a channel to that target set. As part of the channel specification, the source presents its traffic characteristics (similar to an ATM source). Thus, a multicast channel connects a source to exactly one target set. Just as destinations are free to dynamically join and leave target sets, sources are free to dynamically establish and tear down channels. When a new multicast channel is established, a tree routing algorithm similar to [22, 21] is invoked. It determines a tree topology from the source to all nodes currently in the target set. The routing algorithm takes the link topology into account and computes admission tests for all nodes in the path; this ensures that guarantees that have been previously given are not violated by the new channel. The details of the channel establishment procedure are beyond the scope of this paper.

3 The JOIN Operation

3.1 Basic Design Considerations

As mentioned above, a dynamic JOIN operation always joins a new destination node to a target set, not to a single channel. This is in accordance with the semantics of multicast channel establishment, where a new sender always attempts to connect to all nodes of a target set. If a subset of nodes is only interested in some of the channels, then our model requires the creation of another target set for this purpose.

The dynamic JOIN operation will, in any case, be a relatively complicated and costly operation because the performance guarantees given to existing channels must not be violated by the joining node. Therefore the integration of the new node into each of the existing multicast trees sending to the target set requires extensive admission tests.

3.2 The JOIN Service Primitive

The JOIN operation adds a new node to a target set; this is a database update operation. According to our semantics, an attempt will then be made by the network to connect the new node to all the channels currently sending to the target set. The initiating application can specify what the outcome of the JOIN operation will be if the network fails to connect the node to all existing channels.

The JOIN service primitive takes the following form:

```
JOIN(Nodeid,TargetSetId,QoS,PARTIAL-OK | ALL-OR-NOTHING)
```

The last parameter allows the user to specify the desired outcome in case a connection to one or more of the active channels is not possible. If `PARTIAL-OK` is specified, the operation will be successful even if the new node cannot be connected to all channels currently sending to the target set. If `ALL-OR-NOTHING` is specified, the operation will be undone as soon as the connection to one of the channels fails.

In any case the network will return (or store internally for later explicit retrieval) a *success vector* indicating to the application which channels it could be successfully connected to. Knowledge of this success vector will enable the application to implement additional semantics where necessary.

A design alternative would be to issue the JOIN operation without the last parameter. The network would return a success vector immediately, indicating the channels that can be joined, and the performance parameters available for each of them. The application could then decide in a second step whether the JOIN should be completed or undone. We feel that this alternative has major drawbacks: The transport layer interface has to be crossed twice, which is expensive. The network has to maintain state information for uncompleted JOINs, which is hard to clean up in the case of a failure. In addition, the complete success vector has to be computed even if the network can determine early that a join to some channels is not possible. If the user states in advance what the outcome should be in case of partial success, these disadvantages can be avoided.

The right to execute the JOIN operation will be controlled by the network management subsystem, in particular by the authorization and authentication components. This is generally true for all operations issued against our transport layer interface.

3.3 The JOIN Algorithm

In addition to including the new node into the target set in the database, the JOIN operation must attempt to attach the new node to all existing channels currently sending to the target set. This implies finding an optimal attachment point for each of the existing multicast trees.

Since the multicast channels are mutually independent, the optimal attachment point can be chosen separately for each channel. The attachment algorithm presented below can be applied to each multicast tree separately.

The new node will only be attached to a tree if

1. all performance guarantees given to existing nodes in the tree are still valid after attaching the new node
2. the performance requirements of the new node can be met.

For the discussion of the `AttachJoiningNode` algorithm we use the notation introduced in Figure 3.

The `AttachJoiningNode` algorithm is shown in Figure 4.

The algorithm `AttachJoiningNode` contains six subroutines. The *find-candidate-node* subroutine will be described in 3.3.1. The *unicast-admission-test* determines whether

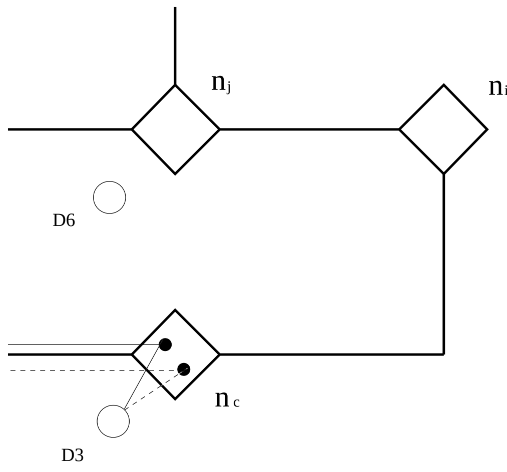

Figure 3: Notation for `AttachJoiningNode` Algorithm

a node on the path from the joining node to the attachment candidate can handle the additional (guaranteed) traffic. It is identical to the server tests for admission control for Tenet unicast channels [10]. The *multicast-admission-test* subroutine will be described in detail in 3.3.2. The *destination-test* subroutine considers the QoS requirements of the new destination and will also be described in 3.3.2. The *compute-cost* subroutine will use the same optimization criteria as the corresponding cost function in the channel establishment algorithm, and is not discussed in detail here. The *attach* subroutine will be explained in 3.3.3.

We will now discuss the functions *find-candidate-set* and *multicast-admission-test* in detail.

3.3.1 Finding a Candidate Attachment Node

In the simplest case the destination is located at a router already forwarding data for the multicast connection. In this case the only node considered as a candidate node is the local router. In our scenario in Figure 2 destination $D4$ can only be attached to both multicast channels at the local router.

If the multicast tree is not passing through the local router the optimal attachment point for the new destination can be anywhere in the existing tree. An optimal new tree could also involve tree restructuring where the new node becomes an intermediate node for other subtrees. This tree restructuring is similar to the tree balancing algorithms known from tree data structure (e.g. B-trees). Thus, if the new tree must be an *optimal* tree, the tree attachment algorithm will be very expensive to com-

```
ALGORITHM AttachJoiningNode ($n_j$)

begin
REPEAT                                          /* select subset of nodes from
                                                /* multicast tree in order to reduce
                                                /* search space
   find-candidate-node($n_j$;$n_c$,$p_c$)          /* new node $n_c$ for candidate set, and
                                                /* path $p_c$ leading to it

   multicast-admission-test($n_c$)               /*enough resources at attachment node?
   for each node $n_i$ on $p_c$ between $n_j$ and $n_c$ do
      unicast-admission-test($n_i$)              /* enough resources on the path?
   if path $p_c$ okay then
      destination-test($n_j$)                    /* can QoS requirements of $n_j$ be met?
   if okay then
      compute-cost($n_j$,$n_c$;$c_c$)               /* cost $c_c$ for attaching
                                                /* new node $n_j$ to $n_c$

UNTIL all candidate nodes found
find $n_{c,opt}$ such that $c_{c,opt} = \min(c_c)$      /* minimum attachment cost
attach($n_j$, $n_{c,opt}$)
end
```

Figure 4: Algorithm AttachJoiningNode

pute, and a disruption of service could result for other nodes during the restructuring operation.

If suboptimal attachment is acceptable, a possibility is to consider attachment only as a leaf node to the existing multicast tree. Again, an optimization algorithm must be executed to find the best attachment point. The new node will become a leaf node, leaving the existing tree structure unchanged. Attachment as a leaf node avoids complex tree restructuring. The attachment only influences the performance parameters of the attachment node and possibly its subtree(s), but no other branches of the multicast tree.

The new node can be attached to an existing leaf node or to an existing non-leaf node. Since only the subtree(s) of the attachment node are influenced by the new node, attaching it to an existing leaf node often minimizes the number of nodes affected. If the new node is attached to an existing leaf node, then the performance guarantees given to all other nodes in the tree will not be violated because their traffic pattern does not change, and these nodes do not have to perform additional work. On the other hand, attaching new nodes only to existing leaf nodes can lead to a very unbalanced tree over time.

If the new node is attached to an existing non-leaf node, the performance parameters will have to be recomputed for this node. This alternative offers intermediate balancing qualities and intermediate overhead. We therefore propose an algorithm attaching the new node *as* a leaf node, but not necessarily *to* a leaf node.

In any case the suboptimal tree will only exist until the channel is torn down. When a new channel is established to the same target set, the initial tree routing algorithm will include all nodes currently in the target.

Our goal is to find a good attachment point to which the new destination can be connected as a leaf node. The first consideration concerning this algorithm is whether its outcome will be first-fit or best-fit. With first-fit we accept the first node that is able to handle the attachment and to meet the QoS requirements of the new destination as the attachment node. With best-fit we choose among many possible attachment nodes the one which offers minimal cost. The second solution will maintain better network performance, and we thus prefer to evaluate several attachment alternatives.

For large multicast trees it would be prohibitive to consider all nodes as potential attachment nodes. In order to keep the JOIN operation scalable the first step of our JOIN algorithm is the computation of a set of candidate nodes. In this way a reasonable subset of nodes can be taken into consideration for the admission tests. Unfortunately most networks do not maintain geographical information in their nodes; therefore we cannot use a real "nearness" criterion. The only criterion typically available is the hop distance between nodes.

We propose two simple solutions. Note that neither of them requires the sources to do computations. This is important since the source of a multicast tree is a potential bottleneck anyway, and such a solution would not scale well.

Source-Directed Search A candidate node is determined by following a route from the new destination to the source of the multicast tree (the nodeid of the source is always known to the network). The standard routing algorithm of the network can thus be used to find a path from the destination to the source. The node at which this path first hits the existing multicast tree becomes a candidate node. In the worst case this is the source itself. In order to have more alternatives for optimization, a pre-specified number of neighbor nodes within the multicast tree can also be considered as candidate nodes (e.g. one level up and one level down the tree).

Expanding Circle Search The new destination checks all its neighboring nodes whether they are already part of the respective multicast tree. It starts with the nodes at one-hop distance and spreads out in every direction, incrementing the number of hop distances of the nodes taken into consideration. The hop-(n+1)-expansion is only considered when the hop-(n)-expansion has not yet reached the multicast tree. This is repeated until a pre-specified number of eligible nodes already in the multicast tree is found. These become candidate nodes, and the path p_c to each one is stored.

This alternative can be optimized if we require that all nodes in a multicast tree store their hop distance from the source, i.e. the number of hops along their route to the source. This is very easy to do. The nodes with a minimal hop distance from the source are more likely to meet the QoS requirements of the new destination since parameters such as delay and delay-jitter typically have lower values at nodes in the upper part of a multicast tree. Among the nodes found by the algorithm described above, the node with the minimal hop distance from the source is determined, and only nodes with a hop distance not exceeding the minimal one by a pre-specified limit are admitted as candidate nodes.

It seems that there is no obvious advantage of one alternative over the other. Only experience with real applications and the size and structure of their multicast trees will show which is better.

The following example based on Figure 2 illustrates the two different algorithms. Destinations $D2$, $D3$ and $D5$ are receiving data from both multicast channels; they form the current target set. Each destination has specified its QoS requirements, either at channel establishment time or when joining later. In order to include the new destination $D6$ into channels 1 and 2 we must attach it to a multicast tree node (i.e. a dot) of the respective channel at one of its routers. Possible attachment points for both channels in our scenario are the router at $D5$, the router at n, and the router at $D3$. Note that it is not necessary to attach the new destination to all channels at the same router. For our example we assume that the *Source-Directed Search* technique will consider router n as a candidate for channel 1, and either the router at $D5$ or router n as candidates for channel 2, depending on the unicast routing algorithm of the network. The *Expanding Circle Search* method will start with routers at one-hop distances, i.e. the routers at $D5$ and router n. If the search limit is two hops, the router at $D3$ will also be considered a candidate. The admission tests and cost functions will be computed for all candidate nodes, and the best attachment node will be selected in the way described below.

3.3.2 Admission Tests

Any real-time network gives performance guarantees to its users. Since network resources are always finite, performance guarantees always imply resource reservation. New resources in a node can only be reserved under the condition that existing channel guarantees are not violated. Therefore, admission tests must be performed whenever a new resource is requested. This is not only true for multicast, but also for unicast connections.

Extensive research was done in the Tenet group at ICSI and UC Berkeley on admission tests for unicast channels. The reader is referred to earlier Tenet publications for a detailed discussion of these [11, 7]. A complete set of formulae for admission tests for real-time unicast channel establishment can be found in [10]. Admission tests for multicast channels are an extension to those for unicast channels. In the following we assume that the reader is familiar with the unicast case.

The tests for a JOIN differ from the ones executed at channel establishment time since only a new branch and not a new tree must be served by the node. We must verify whether enough resources are available at that node to add the new branch. In our `AttachJoiningNode` algorithm we have used the term *multicast-admission-test* to distinguish them from the unicast admission tests.

We assume that node n is not aware of any end-to-end performance requirements of a channel; it is only aware of local parameters. For example, the end-to-end delay bound $D_{max,j}$ from the source to the respective destinations Dj is unknown; node n knows the local delay bounds $d_{n,i,l}$ for scheduling the arriving packets on channel i to the respective links l. In our scenario these are as follows: the local delay bound $d_{n,1,2}$ for the packets on channel 1 to be sent over link 2 towards destination $D2$, $d_{n,1,4}$ for the packets on channel 1 to $D4$, $d_{n,2,4}$ for the packets on channel 2 to $D4$. Figure 5 illustrates the local delay bounds known to node n.

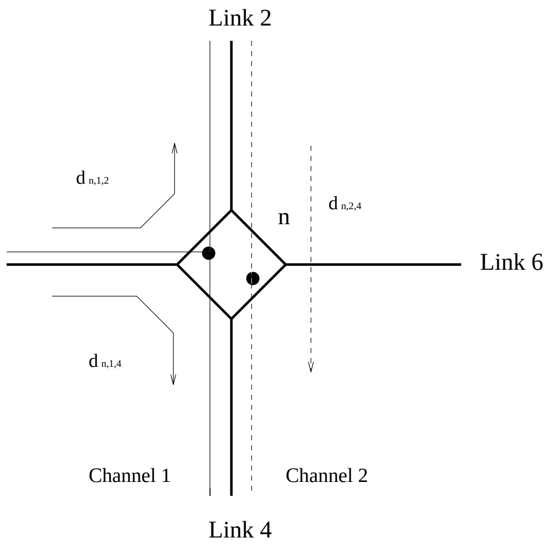

Figure 5: Local Delay Bounds in a Routing Node

Other local parameters that characterize a channel i and each of its outgoing links l in node n can be derived similarly. They are similar to the parameters relevant to real-time unicast channels as described in [10], but augmented by a link identifier l where appropriate:

$t_{n,i}$: maximum packet service time,
$w_{n,i}$: local minimum probability of buffer overflow.

Admission Test for CPU Power

When a packet arrives on one of the channels, a copy of it has to be made at node n since the channels split, and separate packets have to be sent over the two links used by the branches of each channel. Additionally, the packet must be sent out over the outgoing link. These operations require a certain amount of CPU power.

The following assumption is made in order to simplify our computations: a value is assigned to the amount of CPU time needed to process a packet at its arrival ($t_{n,i,input}$) and when it is sent over the outgoing link ($t_{n,i,output}$). We further assume that channel i at node n has a fanout of $f_{n,i}$, i.e. each incoming packet is duplicated onto $f_{n,i}$ outgoing links. Then the time $t_{n,i}$ required for processing a packet of channel i at node n is

$$t_{n,i} = t_{n,i,input} + f_{n,i} * t_{n,i,output} \tag{1}$$

Since the amount of CPU time needed to serve the packets of a channel depends on the number of outgoing links of that channel at that node, a CPU power test considering the increasing number of outgoing links has to be made when attempting to attach a new destination to the node. This test has to determine whether the addition of a new link would cause the worst-case utilization of the server to exceed the maximum utilization bound of 1. This is achieved by increasing the utilization U_n of node n by the amount required for the new link (which is equivalent to increasing $f_{n,i}$ by 1):

$$U_n := U_n + \frac{t_{n,i,output}}{x_{min,i}} \tag{2}$$

where $x_{min,i}$ is the minimum interarrival time of the packets (the worst case for utilization), and then testing

$$U_n \leq 1. \tag{3}$$

Admission Test for Delay Bounds

Obviously the availability of sufficient CPU power does not imply that all delay bounds will be met; delay bounds must be tested independently. We determine the minimum local delay $dmin_{n,i,l+1}$ that can be offered by node n to the new link of channel i where $l+1$ is the index locally assigned to the new link. This is the minimum value of $d_{n,i,l+1}$ that makes the packets on link l of the channel i schedulable without violating the guarantees given by node n to other channels.

The admission test for delay bounds is computed as follows: We distinguish two types of local delay bounds: $d_{n,i,l}$ is the local delay bound as derived from the user's QoS delay guarantee; $dact_{n,i,l}$ is the local delay bound used as an operating point by the node, i.e., the node will never exceed $dact_{n,i,l}$ in its current operations. The node stores these two bounds. This allows us to change the operating point of the node without violating promises given to the user.

The minimum value of $dmin_{n,i,l+1}$ is the minimum local delay that does not cause any violation, i.e. the minimum one for which all of the following inequalities are satisfied:

$$dact_{n,i,l} \leq d_{n,i,l} \qquad \forall i, l \tag{4}$$

If the scheduling discipline in the node is the very widely used EDD (Earliest-Due-Date) scheduling, the delay bound computation can be implemented by sorting the tuples of $dact_{n,i,l}$ and $d_{n,i,l}$ in increasing order of $dact_{n,i,l}$. Then the minimum local delay $d_{n,i,l+1}$ that can be offered by the node is the value of the first position in the list where the new link can be inserted without causing any of the other links to violate their respective delay bound. The insertion of the delay bounds for the new link somewhere in this table can cause increasing values of $dact_{n,i,l}$ of the links below it since the packet is scheduled before those. This is possible as long as the new operating points $dact_{n,i,l}$ do not exceed the bounds $d_{n,i,l}$ for any link (see equation 4). The positions above the insertion need not be reconsidered since they are scheduled as before.

Whether the new link can be admitted can now be verified by testing if any of the values $dact_{n,i,l}$ below the insertion point exceeds its bound $d_{n,i,l}$.

In a multicast node it is necessary to copy an incoming packet in order to serve multiple outgoing links. Two alternatives exist:

1. At the arrival of a packet at node n, the node immediately creates copies and then schedules each copy separately according to its specific local delay bound. The drawback of this solution is that additional buffer space is needed since all copies must be stored separately.

2. Node n keeps the arriving packet in the input buffer. The local delay bounds are known to node n, and when the first copy (i.e. the one with the smallest local delay bound) is due, a new copy of the packet is made and sent over the outgoing link. When the next delay bound is about to expire the next copy is created and sent, and so on. Thus, even if multiple links are served by the node for one channel, buffer space for one packet only is needed at that node. Of course, this buffer space must be reserved for an amount of time equal to the maximum local delay bound of a packet of that channel. Since at all times only

one copy of the packet is stored in the buffer the local minimum probability of buffer overflow $w_{n,i}$ is only specified per multicast channel and not per link.

In the second alternative additional buffer space is only required for a new link if the local delay bound for the new link is greater than all other delay bounds for that channel at that node. Thus, the probability that an attachment is rejected due to insufficient buffer space is reduced. Moreover, the numerical buffer space test for availability of additional buffer space is not always required. Thus the overall costs of a JOIN operation can be reduced.

In a similar way, an admission test is performed for buffer space. The buffer space required is a consequence of the local delay bounds and the accumulated jitter. Again, the main difference between a unicast channel and a multicast channel is that the test has to consider each outgoing link l. The details of all these admission tests can be found in [16].

In addition to verifying that enough computing power and buffer space are available at the *attachment node*, we must also take the QoS requirements of the *new destination node* into account. At the new destination, the *destination-test* subroutine determines whether the accumulated performance parameters from the source to the destination meet the requirements. For each performance bound of channel i a separate test has to be made:

- the D-Test for the end-to-end delay bound $D_{max,i}$
- the J-Test for the end-to-end delay jitter bound $J_{max,i}$

The details of these tests can be found in [16]. If all admission tests described in this section are passed, the path p_c and the attachment node n_c are able to meet all QoS requirements of the new destination, and the costs of the attachment to node n_c over the path p_c are computed.

3.3.3 Attachment of the New Node

The algorithm `AttachJoiningNode` computes multiple candidate attachment nodes n_c and the paths p_c leading to them. When these candidate nodes are defined, we have two possibilities:

1. For each successful admission test a preliminary resource reservation is made. This will block other parallel JOIN or ESTABLISH operations from acquiring the same resources (an ESTABLISH operation attempts to establish a channel and allocates resources for that purpose).

2. No reservations are made until the attachment node has been decided. Then, only the resources on the chosen path are reserved.

The first alternative has the drawback that most of the preliminary reservations have to be released explicitly. This is less robust in the case of network failures. The second alternative has the drawback of a small risk that a parallel JOIN or ESTABLISH(PARTIAL-OK) acquires the resources between the admission test and the actual reservation when a decision for an attachment node is made. Simulations or actual measurements must evaluate which alternative is preferable. We feel that the risk of not being able to allocate the resources in the second alternative is acceptable, and this is the better solution. In order to assure that the required resources are still available after the attachment decision is made, the `attach` subroutine recomputes the admission tests for the attachment along the selected path $p_{c,opt}$ from the attachment node to the new destination, and allocates the required resources.

Similar to the unicast channel establishment, all admission tests compute the maximum amount of resources available at each node along $p_{c,opt}$. In general the QoS requirements can be met with smaller amounts of resources. Therefore a relaxation mechanism similar to the one at the establishment of a multicast channel is used.

The relaxation is distributed over the path. As mentioned in Section 3.3.2, we propose to store the two delay bounds $dact_{n,i,l}$ and $d_{n,i,l}$ in each node. This allows the network to change the actual operating point without violating the delay QoS promised to the users.

At the new destination the actual values for the minimum probability of buffer overflow $w_{n,i}$, the local delay bounds $d_{n,i,l}$ and the local parameters of the buffer space for the nodes n_i on the path from the attachment node n_c to the new destination n_j are computed. In this computation, the end-to-end performance requirements of the complete route from the source to the new destination must be considered, but only parameters at nodes along $p_{c,opt}$ are relaxed. The details of these computations can be found in [16].

Let us now come back to our example in Figure 2, and let us consider the attachment for node $D6$. Candidate nodes for attaching $D6$ were the router n, the router at $D5$, and the router at $D3$. The result of the computations for joining destination $D6$ might be such that $D6$ is attached to channel 2 at $D5$ and to channel 1 at $D3$. Figure 6 illustrates this solution.

3.4 Concurrency Control

In a large network we have to consider the case where several update operations on the same target set can occur in parallel. The network has to ensure that the users always have a consistent view of the target set and the channels sending to it.

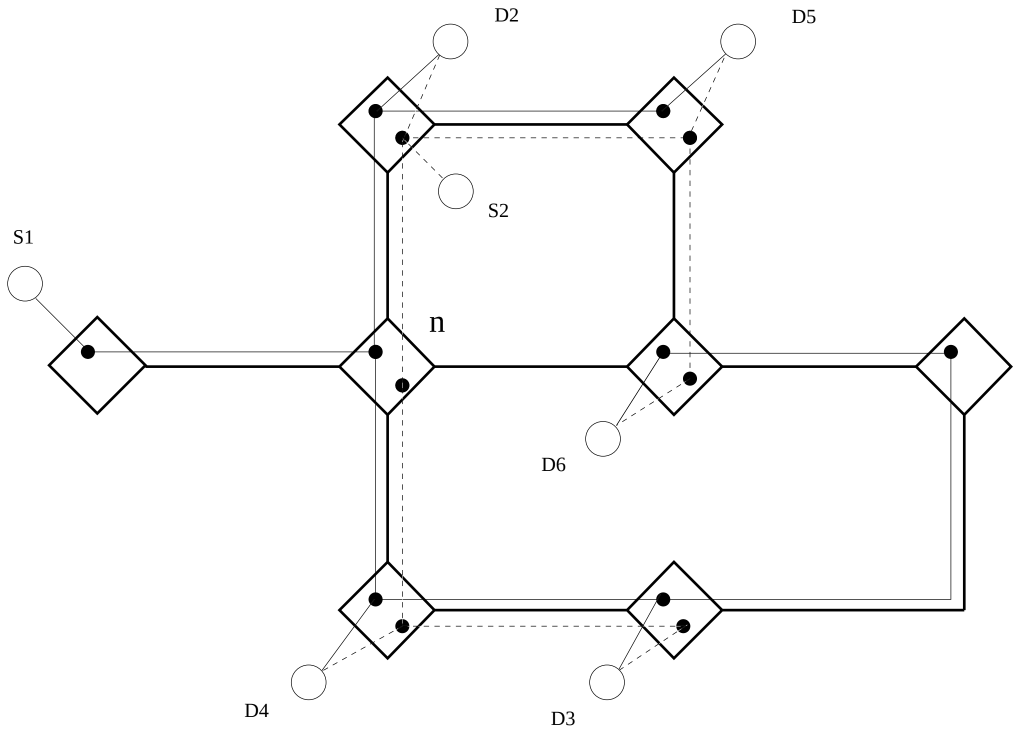

Figure 6: Nodes $D4$ and $D6$ have joined

Inconsistencies can arise when a new node joins the target set, and a new channel is being established at the same time. Then we must take precautions to make sure that the new node will be included in the new channel.

Conflicts can occur between simultaneous ESTABLISH and JOIN operations. In both operations we must distinguish between an ALL-OR-NOTHING and a PARTIAL-OK requirement. Similar to the JOIN operation discussed above, an ESTABLISH(ALL-OR-NOTHING) operation tries to establish a channel to all destinations in a target set, an ESTABLISH(PARTIAL-OK) operation accepts a channel that is not able to reach all destinations in the target set. If a JOIN operation is in progress while a new channel is being established, the channel establishment may miss the new destination without realizing it. Respectively, a JOIN operation may miss a new channel if the establishment of that channel is unfinished.

To be more precise we define a correctness criterion for concurrent operations on multicast trees:

The outcome of a concurrent execution of operations on multicast trees is correct if

- *either all operations involved have specified the PARTIAL-OK option*

- *or at least one of the operations involved has specified ALL-OR-NOTHING and all members of the target set are reached by all channels after completion of concurrent operations.*

A possible way to implement this criterion is obviously serialization, i.e. conflicting operations are always executed in serial order. This could easily be implemented by locking the target set while a JOIN or an ESTABLISH operation is being executed. But serialization does not scale well: In a large conference the problem of a target set being locked most of the time could arise.

However, the degree of concurrency can be increased without endangering consistency by closer observation of the conflict patterns between JOIN and ESTABLISH operations. Table 1 shows the conflict matrix for ESTABLISH locks and JOIN locks.

	JOIN(PART)	JOIN(ALL)	ESTABLISH(PART)	ESTABLISH(ALL)
JOIN(PART)	OK	OK	OK	No
JOIN(ALL)	OK	OK	No	No
ESTABLISH(PART)	OK	No	OK	OK
ESTABLISH(ALL)	No	No	OK	OK

Table 1: Conflict matrix for JOIN and ESTABLISH locks

We take into consideration that concurrent channel establishments, even those trying to include all destinations in the multicast trees, do not conflict with each other since the number of destination nodes in the target set is unchanged. Similarly, two JOIN operations can be executed concurrently, no matter which option has been chosen, as long as no new channels are established in parallel. We also observe that there is no need to lock the target set in case of a simultaneuos ESTABLISH(PARTIAL-OK) and JOIN(PARTIAL-OK) operation since both operations do not expect a consistent outcome. Thus, problems arise only in the cases indicated in Table 1, and the target set must only be locked in those specific cases.

Thus we can implement concurrency control by using two different types of locks on target sets: a JOIN lock is activated by a JOIN operation and does not allow an ESTABLISH operation to be executed in parallel; respectively, an ESTABLISH lock is activated by an ESTABLISH call and locks the target set for JOIN operations. This allows much more concurrency than exclusive locks for all update operations.

As a general conclusion, the ALL-OR-NOTHING option will provide the user with stronger semantics than the PARTIAL-OK option: All nodes of a target set are always reached by all channels sending to that target set. But the PARTIAL-OK option scales much better for large networks. An application example for ALL-OR-NOTHING could be a small multimedia conference, an example for PARTIAL-OK would be an audio/video multicast from a convention or a seminar.

In addition to the global conflicts discussed above, two parallel JOIN operations can collide while computing admission tests. In our approach we avoid conflicts between two JOINs by re-evaluating the admission tests when the eventual attachment path has been selected. In this way we do not serialize the JOIN operations by making the JOIN locks incompatible in the conflict matrix (see Table 1).

4 The LEAVE Operation

4.1 Basic Design Considerations

The leave operation removes a destination node from a target set and disconnects it from all channels that send to that target set. The resources reserved for the links to this destination node are freed. As with the JOIN operation we disconnect from each channel separately, executing the same disconnect algorithm.

The LEAVE operation is much simpler than the JOIN operation because a leaving node will never decrease the available resources of another node; this is true for CPU and buffers. It will thus never endanger performance guarantees given to other channels.

For each channel we can distinguish two cases:

1. The leaving node is a leaf in the multicast tree.
2. The leaving node is an intermediate node.

If the leaving node is an intermediate node then a tree reorganisation might be desirable.

4.2 The LEAVE Service Primitive

The LEAVE service primitive takes the following form:

```
LEAVE(NodeId,TargetSetId)
```

`NodeId` is the address of the destination leaving the target set specified by `TargetSetId`. In our model we distinguish between routers and nodes in the multicast tree. Just like the JOIN the LEAVE is a layer 4 operation; it only refers to multicast nodes and not to routers.

4.3 The LEAVE Algorithm

The LEAVE algorithm first disconnects the leaving destination from all channels that send to the specified target set. It then removes the node from the target set.

The DisconnectLeavingNode algorithm is executed for each channel separately. We use the term *leaf node* for a node with no descendants and the term *branch node* for the first upstream node that contains either a branch point of the tree or a local destination. Let us assume that destination node L is leaving. Similar to the JOIN operation p_L denotes the path from the leaving destination to the branch node. The algorithm is shown in Figure 7.

```
ALGORITHM DisconnectLeavingNode($n_L$,ChannelId)

begin
if $n_L$ is a leaf node                                 /* start with leaving node
   then
      free-leaf-node-resources($n_L$)
      for each node $n_i$ on $p_L$ do                    /* process path $p_L$ to branch node
         if $n_i$ has no other links                     /* path node was only needed for $n_L$
         then
            free-leaf-node-resources($n_i$)
free-branch-node-resources
end
```

Figure 7: Algorithm DisconnectLeavingNode

The two subroutines *free-leaf-node-resources* and *free-branch-node-resources* will be discussed in detail below.

In the second step the node is removed from the target set. This is simply a database update operation. Notice that that concurrency problem disussed for ESTABLISH and JOIN above does not occur here: no matter in which order other update operations and a LEAVE are executed, the node will always eventually be removed from the target set and all channels.

Freeing Resources at a Leaf Node

The subroutine *free-leaf-node-resources* executes the following steps:

- The CPU requirement of channel i in the node is set to 0.
- All buffers reserved for channel i are released.
- The scheduling resources are released by removing the local delay bounds $dact_{n,i,l_L}$ and d_{n,i,l_L} from the table of the corresponding outgoing link.

Thus, the node is completely removed from the multicast channel.

Freeing Resources at the Branch Node

A branch node cannot be removed completely because it is still needed to forward data on the channel to other nodes.

The subroutine *free-branch-node-resources* executes the following steps:

- The CPU requirement of channel i in the node is released by decrementing the fanout $f_{n,i}$ of channel i by one.
- A relaxation of the buffer space requirement of channel i at the branch node is feasible if the local delay bound d_{n,i,l_L} of the removed link l_L is the maximum of all local delay bounds of channel i at the branch node. Thus, the condition for a relaxation of the buffer space reserved for this channel is

  $$d_{n,i,l_L} = \max_{l}(d_{n,i,l}) \tag{5}$$

  where l_L is the outgoing link of channel i towards the leaving destination. If this equation is satisfied the packets of this channel must only be stored for an amount of time equal to the maximum of the remaining local delay bounds.
- The scheduling resources are released by removing the local delay bounds $dact_{n,i,l_L}$ and d_{n,i,l_L} from the relevant table for the outgoing link (as above).

To illustrate our LEAVE algorithm consider destination $D3$ in Figure 6. In this example we want to disconnect $D3$ from channel 1 and channel 2. For channel 2 $D3$ is a leaf node. All resources at $D3$ are freed. The branch node for channel 2 is $D4$; $D4$ has a local receiver. Only the resources related to forwarding a packet on the link to $D3$ are released. For channel 1 $D3$ is an intermediate node. Since local delivery does not consume network resources in our model no resources can be freed at $D3$. The new situation after disconnecting $D3$ from both channels is shown in Figure 8.

As mentioned above our algorithm does not change the topology of the multicast tree at a branch node; the traffic along the remaining branches of the multicast tree flows as before. The topology of the multicast tree can become less efficient. In our example $D6$ is still connected to channel 1 via two intermediate routers; connecting via the router at $D5$ or router n could now be more efficient for the network. In general the tree can degrade substantially after multiple LEAVE operations. We will discuss tree reorganization issues in the next section.

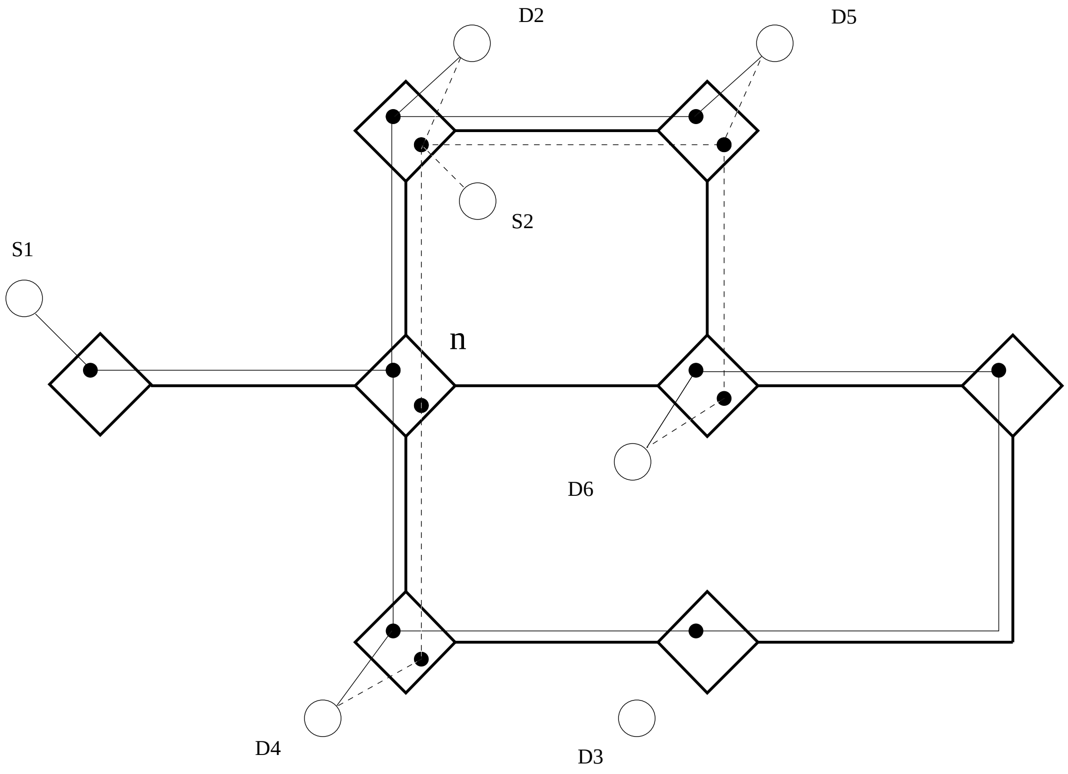

Figure 8: Scenario after $D3$ has left

4.4 Tree Reorganization

An optimal multicast tree can be defined as follows:

Consider a graph $G = (R, L)$ of routers where R is the set of routers and L is a set of links connecting them. Each router $r \in R$ has a utilization $U_r, 0 \leq U_r \leq 1$.

A multicast tree $M = (N, \{(n, i, l)\})$ is defined as a set of nodes N and links (n, i, l) leaving multicast node n on channel i towards multicast node l, passing over a link of L. $S \in N$ is the source, and $D \in N$ is a destination. The multicast tree starts at node S and reaches out to all destinations in $\{D\}$. Each link (n, i, l) has a cost $C_{n,i,l}$ associated with it. This cost depends on

1. the cost of the physical link it traverses
2. the cost of allocating resources at router r containing multicast node n.

The cost of a multicast tree M is the sum of the costs of its links.

An *optimal multicast tree* is a tree such that

1. no router r in the multicast tree has a utilization $U_r > 1$.
2. its cost is minimal.

By restriction to the well known NP-complete Steiner-Tree problem, it can be shown that finding an optimal multicast tree is NP-complete. It is therefore very unlikely that an algorithm can be found that will maintain an optimal tree at all times.

A heuristic and scalable solution is

- to reorganize the tree only on demand and not at each JOIN or LEAVE operation, and
- to restrict reorganization to subtrees.

A possible heuristic uses the notion of branch node introduced in Section 4.3. We determine all multicast nodes in all subtrees of the branch node, and recompute new subtrees using the subroutines of the `AttachJoiningNode` algorithm. These subroutines evaluate the cost functions of alternative trees and are thus able to find better local solutions.

It is particularly difficult to reorganize a multicast tree in a real-time network because performance guarantees could be violated during the reorganization. The only clean solution is to reserve the resources on the new tree first, switch over in an atomic operation, and then release the resources along the old tree. Obviously additional resources are required during the reorganization process. Detailed algorithms for dynamic channel management can be found in [17].

In our example we can reorganize the tree for channel 1 after destination $D3$ has left. We assume that the current path from $D4$ to $D6$ via two intermediate nodes is no longer desirable. The tree reorganization sketched above would reorganize the subtree of the branch node at $D4$. It could find a better attachment point for $D6$ at $D5$. This is shown in Figure 9.

5 Conclusions

We have presented a model for real-time multicast communication in internetworks. The model distinguishes between the topology of links and the topology of multicast channels. We have also introduced the notion of a target set as a set of nodes with a common interest. Channel establishment, JOIN and LEAVE are defined for target sets.

Based on this model we have described the JOIN and LEAVE service primitives, their semantics, and algorithms implementing them. Compared to a traditional network a

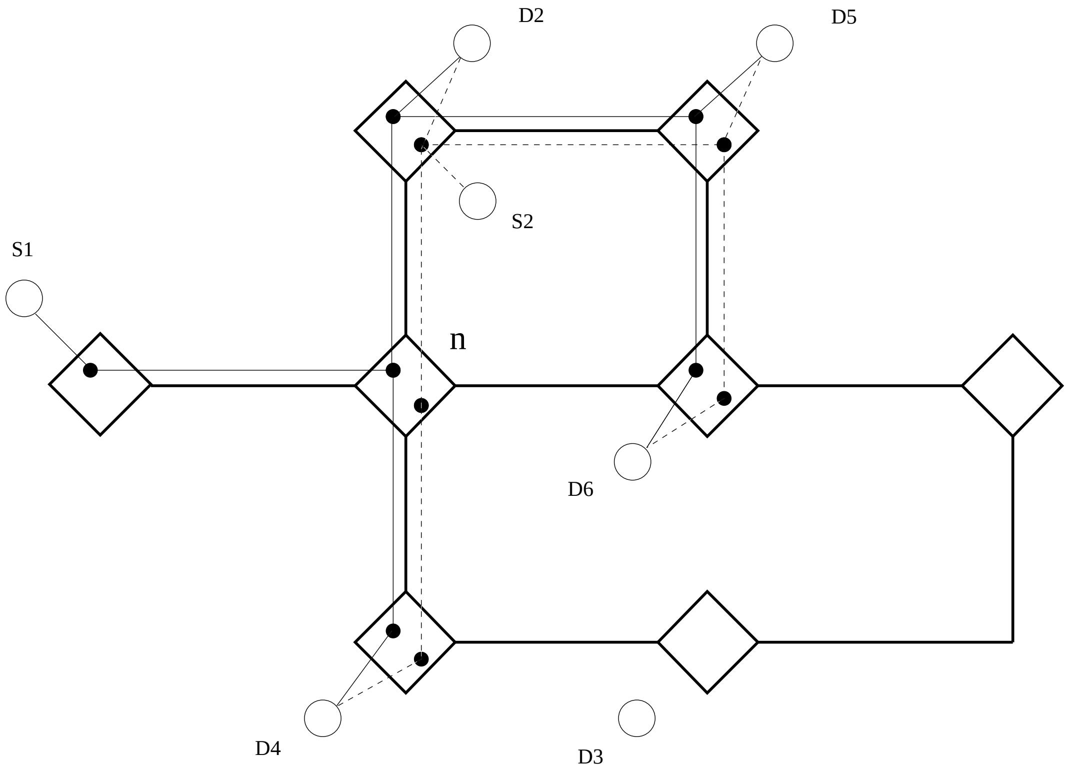

Figure 9: Scenario after Tree Reorganization

real-time multicast environment introduces several new challenges: finding an optimal attachment point for a joining node, maintaining performance guarantees even when the tree topology changes, extending admission tests, designing concurrency control in order to maintain consistency, and reorganizing a tree after LEAVE operations. We have proposed solutions to these problems, with a special emphasis on scalability.

Future work will concentrate on performance evaluation of our algorithms.

Acknowledgment

We would like to thank Domenico Ferrari and the members of the Tenet group in Berkeley for many valuable hints and excellent discussions. The International Computer Science Institute hosted us for several months, and we gratefully acknowledge the excellent research environment there.

References

[1] K. Birman, R. Cooper: The ISIS Project: Real Experience with a Fault Tolerant Programming System. ACM Operating Systems Review, Vol. 25, No. 2, 1991, pp. 103-107

[2] A. Danthine, Y. Baguette, G. Leduc, L. Leonard: The OSI 95 connection-mode transport service: the enhanced QoS. Proc. Fourth International Conference on High Performance Networking, Liege, Belgium, December 1992, Vol.C-14, pp. 235-252

[3] S.E. Deering, D.R. Cheriton: Host Groups: A Multicast Extension to the Internet Protocol. Internet RFC 966, December 1985

[4] S.E. Deering: Host Extensions for IP Multicasting Internet RFC 1112, August 1989

[5] S.E. Deering: Multicast Routing in a Datagram Internetwork. PhD Thesis, Stanford University, California, 1991

[6] D. Ferrari: Realtime Communication in Packet-Switching in Wide-Area Networks. Technical report TR-89-022, International Computer Science Institute, Berkeley, California, May 1989

[7] D. Ferrari: Realtime Communication in an Internetwork. Journal of High-Speed Networks, Vol. 1, No. 1, 1992, pp. 79-103

[8] D. Ferrari: Distributed Delay Jitter Control in Packet-Switching Internetworks. Journal of Internetworking: Research and Experience, vol. 4, n.1, 1993, pp.1-20

[9] D. Ferrari, A. Banerjea, H. Zhang: Network Support for Multimedia. Technical Report TR-92-072, International Computer Science Institute, Berkeley, California, November 1992

[10] D. Ferrari: A New Admission Control Method for Real-Time Communication in an Internetwork. in S.H. Son, Ed., Principles of Real-Time Systems, Prentice Hall, 1994

[11] D. Ferrari, D.C. Verma: A Scheme for Real-Time Channel Establishment in Wide-Area Networks. IEEE Journal On Selected Areas In Communications, Vol.8, No. 4, 1990, pp. 368-379

[12] D. Hehmann et al: Implementing HeiTS: architecture and implementation strategy of the Heidelberg High-Speed Transport System. Proc. Second International Workshop on Network and Operating System Support for Digital Audio and Video, Heidelberg, Germany, November 1991, Lecture Notes in Computer Science, Springer-Verlag, 1992, p. 33-44.

[13] R.G. Herrtwich: An architecture for multimedia data stream handling and its implication for multimedia transport service interfaces. Proc. of the Third Workshop on Future Trends of Distributed Computing Systems Taipei, Taiwan, April 1992, IEEE Comput. Soc. Press, 1992. p. 269-75.

[14] E. Mayer: An Evaluation Framework for Multicast Ordering Protocols. Proc. ACM SIGCOMM '92 Conference. Communications Applications, Architectures and Protocols, Baltimore, MD, USA, 17-20 Aug. 1992). Computer Communication Review, Oct. 1992, vol.22, (no.4), pp. 177-187.

[15] E. Mayer: Concurrent multicast checkpointing. Proc. Upper Layer Protocols, Architectures and Applications. IFIP TC6/WG6.5 International Conference, Vancouver, BC, Canada, 27-29 May 1992, pp. 321-335

[16] E. Müller-Menrad: Dynamic Membership in Real-time Multicast Groups Diplomarbeit, Lehrstuhl für Prozessrechner, Technische Universität München, 1993 (in English)

[17] C. Parris, D. Ferrari: A Dynamic Connection Management Scheme for Guaranteed Performance Services in Packet-Switching Integrated Services Networks Technical Report TR-93-005, International Computer Science Institute, Berkeley, California, 1993

[18] C. Partridge, S. Pink: An implementation of the revised Internet Stream Protocol (ST-2). Internetworking: Research and Experience, March 1992, Vol. 3, No. 1, pp. 27-54.

[19] C. Topolcic: Experimental Internet Stream Protocol, Version 2 (ST-II) Internet RFC 1190, October 1990

[20] C. Topolcic: ST II. Proc. First International Workshop on Network and Operating System Support for Audio and Video, Berkeley, CA, November 1990. Available from International Computer Science Institute, Berkeley, California, USA

[21] D.C. Verma: Routing Reserved Bandwidth Multipoint Connections. Proc. ACM SIGCOMM 93, San Francisco. ACM Computer Communication Review, Vol 23, No. 4, 1993. pp. 96-105

[22] B.M. Waxman: Routing of Multipoint Connections. IEEE Journal on Selected Areas in Communications, Vol. 6, 1988, pp. 1617-1622

[23] B. Wolfinger, M. Moran: A Continous Media Data Transport Service and Protocol for Realtime Communication in High Speed Networks. Proc. Second International Workshop on Network and Operating System Support for Digital Audio and Video, pages 171-182, Springer Verlag, Heidelberg, Germany, November 199o, pp. 171-182

[24] H. Zhang, S. Keshav: Comparison of Rate-Based Service Disciplines. Proceedings of ACM SIGCOMM 1991, Zurich, Switzerland, September 1991, pp.113-122