TECHNISCHE UNIVERSTÄT MÜNCHEN
FAKULTÄT ELEKTROTECHNIK

Diplomarbeit

Thema	Dynamic Membership in Real-Time Multicast Groups
Vorgelegt von	Eberhard Müller-Menrad
Geboren am	5. April 1968 **in** Stuttgart

zur Erlangung des akademischen Grades

Diplomingenieur (Dipl.-Ing.)

Erstellt in	The Tenet Group International Computer Science Institute and University of California at Berkeley, USA
Eingereicht am	Lehrstuhl für Prozessrechner Professor Dr.-Ing. Georg Färber Technische Universität München
im	November 1993

Contents

ABSTRACT **5**

1. INTRODUCTION **6**

1.1 Introduction to Multimedia Networking 6
1.2 Real-time Communication 7
1.2.1 Packet-switching or Circuit-switching Technology? 7
1.2.2 Connection-oriented or Connectionless Service? 8
1.3 Related Work 9

2. THE TENET APPROACH TO REAL-TIME COMMUNICATION **10**

2.1 Components of the Tenet Approach 10
2.2 The First Tenet Scheme (Scheme 1) 11
2.2.1 The Real-time Channel and Resource Reservation 11
2.2.2 Traffic Description and Performance Specification 11
2.2.3 The Channel Set Up Procedure 12
2.2.4 Flow Control 14
2.2.5 Scheduling 14
2.3 The Prototype Tenet Protocol Suite (Suite 1) 16
2.4 The Second Tenet Scheme (Scheme 2) 17

2.4.1 Support for Multicasting 17
2.4.2 The New Ideas 18
2.4.3 Importance of Dynamic Membership 21
2.4.4 Goal of Our Work 21

3. DYNAMIC MEMBERSHIP IN MULTICAST GROUPS 23

3.1 Basic Design Considerations 23
3.2 Scenario of a Conference 24
3.3 The JOIN Operation 27
3.3.1 The JOIN Service Primitive 27
3.3.2 Design Alternatives 27
3.3.3 The JOIN Algorithm 29
3.3.3.1 Finding a Candidate Attachment Node 31
3.3.3.2 Consideration of Performance Guarantees 34
3.3.3.3 Attachment of the New Node 44
3.3.4 Concurrency Control 49
3.4 The LEAVE Operation 53
3.4.1 The LEAVE Service Primitive 53
3.4.2 The LEAVE Algorithm 53
3.4.2.1 Freeing Resources at a Leaf Node 54
3.4.2.2 Freeing Resources at the Branch Node 55
3.5 Tree Reorganization 57

4. CONCLUSION 60

ACKNOWLEDGEMENTS 62

REFERENCES 63

APPENDIX 66

List of Figures

1. Components of the Tenet Approach 10
2. The Prototype Tenet Protocol Suite 17
3. Specification of a Target Set 19
4. Resource Sharing 21
5. Physical Topology of a Network 24
6. Scenario of a Conference in Scheme 2 26
7. Notation for the AttachJoiningNode Algorithm..... 30
8. Local Delay Bounds at Node *n* 36
9. Table of Ordered Tupels [$d_{act_{n,i,l}}$; $d_{n,i,l}$] 39
10. Scenario after Nodes D4 and D6 have joined 49
11. Conflict Matrix of JOIN and ESTABLISH Operations 51
12. Scenario after D3 has left 56
13. Scenario after Tree Reorganization 59

Abstract

With the availability of high-speed networks and high-performance workstations, it is now possible to build distributed multimedia applications. Multimedia traffic demands real-time guarantees on the delivery of traffic and its performance parameters. A real-time service provides its client with the ability to specify their performance requirements and to obtain guarantees on the satisfaction of these requirements which have to be met by the network. We address the available options for providing such guarantees. The Tenet Group at ICSI and UC Berkeley is investigating the design and implementation of real-time network protocols. The approach of the Tenet Group to providing real-time guarantees is briefly described in this paper.

An important feature of the service offered by a network is the support for multicast communication. In this thesis, we introduce the key features of the Tenet approach to real-time multicasting and address the problem of dynamic membership in real-time applications such as conferences in particular. We present a model for real-time multicast channels and explain the approach to dynamic membership of destination nodes in a multicast communication group in this environment. This thesis introduces scalable algorithms for finding a good attachment point to a multicast tree and for the release of resources in the network when a node leaves a multicast group. In order to preserve guarantees given to existing channels, admission tests are needed. These computations are described in detail. Finally, we discuss reorganization issues of multicast trees in order to maintain the efficiency of a multicast tree.

Chapter 1
Introduction

1.1 Introduction to Multimedia Networking

Multimedia offers an enormous variety of applications, thus becoming increasingly important in the future. Being a powerful tool for the communication of data and new ideas it will characterize and influence most forms of business communications, as well as personal and educational applications. Multimedia can be defined as follows:

> "Multimedia is a class of computer-driven interactive communication systems which create, store, transmit, and retrieve textual, graphic, and auditory networks of information." [12]

Generally, multimedia is the current term used to encompass a variety of applications involving many different types of media, such as motion video, audio, still images, image streams, and traditional data. With its wealth of presentation and editing options, multimedia can accelerate learning, heighten productivity, and encourage creativity. The most common multimedia applications will probably be education, video conferencing, training, presentation, and compound document mail, but also specific applications such as scientific visualization, medical imaging, and remote collaborative work will be possible.

In order to realize all those possibilities some problems have to be solved: Multimedia information generates huge amounts of data and hence requires very large bandwidths, thus, high speed networks and powerful computers are necessary. Some applications like multimedia mail can tolerate deferred transmission for store-and-forward operations, but others like video

conferencing have rather strict requirements in terms of delay-sensitive delivery and tight performance control in order to minimize distortion. Networks supporting multimedia communications have to meet these requirements [15, 16].

Thus, in the area of computer networking, multimedia applications can be considered a subset of the larger class of real-time applications [11]. Therefore the problem of real-time communication is addressed in the remainder of this section.

1.2 Real-time Communication

Real-time communication is characterized by a specification of the performance of the communication, i.e. communication with guaranteed *Quality of Service* (QoS), and by network algorithms and protocols to meet this specification. Guarantees are needed by all those systems where timely of data arrival is crucial to its usefulness, e.g. systems that process and transmit digital audio, digital video, or any combination thereof with other types of information. The Quality of Service may be composed of parameters such as delay, variance in delay (delay jitter), throughput, and probability of data loss.

Real-time applications can be divided into two categories: those which require transmission of data units at regular intervals (continuos media clients), and those which generate data for transmission at relatively arbitrary times (message oriented clients). An example of the former is video conferencing in which video frames are sent in regular intervals, for the latter mail service with guaranteed delivery latency [19, 22]. For the support of all of those applications efficient networks offering real-time services are needed. The efficiency of a network is usually measured with respect to the above parameters.

1.2.1 Packet-switching or Circuit-switching Technology?

Concerning the subnet structure of the network, the available options for providing real-time services in a wide-area context are circuit-switching and packet-switching [3].

Circuit-switching networks can provide real-time support by dedicating a circuit of fixed bandwidth to each connection for the entire duration of the connection. It is the standard method for providing real-time performance guarantees in the current commercial environment, but it does

not achieve an efficient usage of network resources because an application must reserve the maximum bandwidth which it intends to use. This is not particularly convenient when the traffic includes a large data or text component, i.e. when the traffic is bursty (which can be expected for multimedia traffic). Additionally, circuit-switching is inappropriate for short transmissions, where the connection set-up creates excessive overhead.

Packet-switching is better suited for this type of traffic, but current packet-switching networks cannot provide bounded delays and bounded delay jitter, and therefore cannot guarantee the required levels of performance to the clients [5, 10].

1.2.2 Connection-oriented or Connectionless Service?

A separate issue from the subnet structure is the service that is offered by the network layer. It can be connection-oriented or connectionless.

Connectionless service does not require explicit establishment of a connection but to guarantee real-time service on top of a connectionless service will be very difficult. A packet delay is hard to control if each packet going from a given sender to a given receiver can follow a different route, and some of these routes may become congested while real-time packets are in transit. Also, the reservation of resources that is needed to achieve guaranteed performance is hard to realize in this case.

The connection establishment, which may be slow, is needed for connection-oriented service, but implementing connections and using packet-switching technology makes sense when the subnet is trying to provide a highly robust service [20].

For the reasons mentioned above the problem of how to support multimedia and real-time applications in a packet-switching networks is currently an area of intensive research activity, and among others the Tenet Group at the International Computer Science Institute, Berkeley, California, tries to guarantee performance in a connection-oriented packet-switching internet environment.

1.3 Related Work

Other projects have also developed multicast protocols. A prominent example for a connectionless approach is the multicast IP protocol of the Internet [2, 4]. IP multicasting is an extension to the standard IP network level protocol supporting multicast traffic. An IP datagram is transmitted to a *host group*, a set of hosts identified by a single IP destination address (*class D address*). The source does not know who and where the current destinations are. Before any datagrams destined to a particular group can be received the new destination must join the multicast by specifying the multicast IP address for which they want to receive packets. Multicast packets are routed like all other IP packets, with receivers accepting traffic addressed to joined groups in addition to the normal host address. The multicast IP protocol has been implemented on many different architectures and is now part of several new operating system releases for workstations. Multicast IP does not use resource reservation and datagrams are transmitted only with "best-efforts" reliability, i.e. the datagram is not guaranteed to arrive intact at all members of the destination group or in the same order relative to other datagrams.

The Stream Protocol, Version II (ST-II) [21] is an experimental internetwork-layer stream protocol developed for the Internet. It is connection-oriented and supports multicasting. Unlike multicast IP it guarantees end-to-end bandwidth and delay. The guarantee is provided by requiring bandwidth reservation at channel establishment time and ensuring that the requested bandwidth is available throughout the lifetime of the connection. When a new channel is established, the QoS parameters for bandwidth and delay are negotiated between sender, network (routers on the path), and receivers. For this purpose ST-II decouples control from data delivery. The details of routing, admission tests, resource reservation within a node, and so on, are left to the implementor. ST-II allows dynamic joining of new destinations, but the new receiver must first communicate directly with the sender using a control protocol to exchange relevant information such as the group address. The sender then requests ST-II to add the new receiver with the connection completed only if there is sufficient bandwidth to process the new destination. This solution obviously does not scale well.

None of the approaches mentioned above allows dynamic membership of nodes under real-time conditions. ST-II comes closest to the Tenet approach but in the current version all join operations have to be initiated by the source [21] and thus not scale well. Chapter 2 of this thesis describes the Tenet approach to real-time communication and defines the goal of our work. Chapter 3 explains algorithms for dynamic membership in real-time multicast groups in the Tenet environment. Chapter 4 concludes the thesis.

Chapter 2
The Tenet Approach To Real-Time Communication

2.1 Components of the Tenet Approach

Concerning the components of the Tenet Group's approach, we distinguish between *approach*, *scheme*, and *suite* [11]. Approach refers to the general ideas and principles that have guided the Tenet Group's work; scheme refers to a set of algorithms implementing the ideas, and suite refers to the actual goal, network protocols. Figure 1 shows the relationships among the concepts mentioned in this paper.

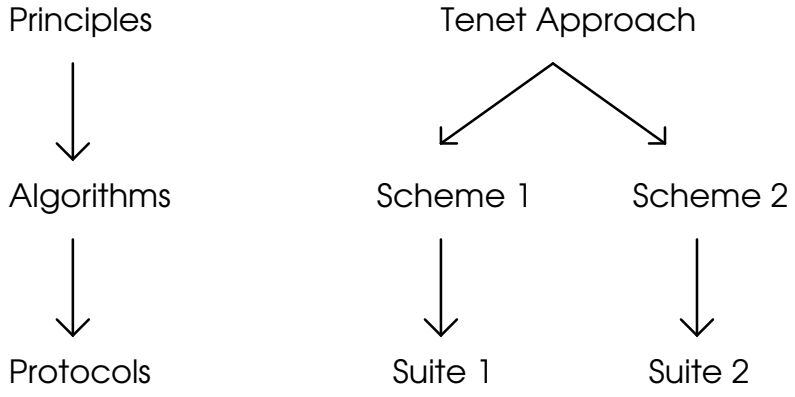

Figure 1: Components of the Tenet approach

2.2 The First Tenet Scheme (Scheme 1)

2.2.1 The Real-time Channel and Resource Reservation

The communication abstraction on which the first scheme is based is the *real-time channel*, which is defined as a simplex unicast end-to-end connection with performance guarantees and traffic restrictions. A real-time channel connecting a source with a destination is associated with a set of nodes and links within the network through which the real-time data for this connection pass [11]. In the nodes the transmitted data can be stored and then forwarded to the next node. The assumption that is made about the links between the store-and-forward nodes in order to enable real-time guarantees is that there be a finite bound for the link delay of each packet.

In order to achieve guaranteed performance the client (in the role of either a sender or a receiver) and the server (the network) have to enter into a contract specifying the limits on the client's traffic characteristics and the Quality of Service that the server has to provide. The client promises not to exceed its specific load, and the network promises to meet its guarantees. Thus, the traffic model characterizes a source while the Quality of Service is observed by a destination. The negotiation of this contract is done during the channel establishment. In order to being able to meet the guarantees given to the channel, the network has to reserve resources, which is also done at channel establishment time.

2.2.2 Traffic Description and Performance Requirements

There are two sets of parameters used by the client to describe his traffic characteristics and his performance requirements [6, 18].

The first set can be defined as traffic requirements. It includes the minimum interarrival times for the transmitted packets, x_{min}, the minimum value of the average packet interarrival time, x_{ave}, the interval I on which these values have been computed and the maximum packet size s_{max}. The client must not violate the limitations on his traffic for all the duration of the communication, if he wants to obtain guaranteed real-time service.

The second set contains the performance requirements. These requirements are expressed in terms of transmission delay, delay jitter, and reliability. The first type of requirement can consist of an absolute bound on the source-destination delay, D_{max}. In this case the channel is called

deterministic. Another way of expressing a requirement on the expected delay is in terms of a maximum value D_{max} to be satisfied by the packets with a probability greater than Z_{min} (*statistical* channel). In addition a limit J_{max} on the variance of delay (jitter-bound) with a maximum value of D_{max} (*bounded jitter* channel) may be specified. In this case we distinguish between nodes that can control jitter (*J-type*) and nodes that are not able to control jitter (*NJ-type*). The reliability of a channel is specified in probabilistic terms. It is given as the minimum probability W_{min} that packets are not lost due to buffer overflows in the nodes [7].

2.2.3 The Channel Setup Procedure

In the Tenet scheme a channel must be established before data can be transferred. When a client (i.e. a source) wants to set up a connection, he specifies the performance requirements and describes its traffic characteristics for this channel. The network takes these parameters, chooses a route for the channel and divides the end-to-end traffic and performance requirements into local requirements for each node on the route. An establishment message is sent from the source to the destination. Besides trying to establish a connection, several channel admission tests are performed at each node in order to verify that guarantees can be offered without violating the performance promised to other clients.

We distinguish between the computations performed at each node along the route of the channel (*Server Tests for Admission Control*), and the computations performed at the destination (*Destination Tests for Admission Control*) [9]. The offered guarantees are based upon the node's processing power, buffer capacity, and line bandwidth, as well as the already existing real-time channels and their performance requirements.

The server tests are as follows:

- The *deterministic test* is required when the channel to be established is deterministic (i.e. Z_{min}=1). It consists of verifying that enough processing power is available in the node to accommodate the additional channel without impairing the guarantees given to others.

- The *statistical test* is performed for the establishment of both statistical and deterministic channels if at least one statistical channel is already passing through that node. The goals of this test are to determine whether the probability for a delay is

below its maximum, and to provide the destination host with information necessary to compute the actual probability Z if the new channel is statistical. For the sake of simplicity only deterministic channels (i.e. deterministic guarantees) will be considered in this thesis.

- The *delay bound test* is needed in all cases and determines the minimum delay bound to be assigned to the channel.

- The *buffer space test* has to verify that enough buffer space for the new channel is available at that node in order to allow the packets to spend a maximum amount of time equal to the minimum local delay bound the node can offer to the new channel.

At each node along the path, the best performance which can be guaranteed is computed, and tentative resource reservations are made. Resources that are reserved are CPU power, buffer space and scheduling resources (also referred to as service priority).

In order to make the establishment fast, the establishment message is restricted to only one round trip. Thus, the last point along the path where a decision for a channel request can be made is at the destination node. The destination determines in the destination tests whether the accumulated guarantees meet the requirements, and if so, completes the channel establishment. This is done by determining the amount by which performance requirements have been exceeded, and deciding which node may reduce their guarantees, and by how much. A confirmation message informing the nodes about the performance bounds assigned to them is sent back along the same path. Nodes reduce their resource reservation, and the client is notified of successful channel establishment. Since the admission tests are of particular interest in the approach to dynamic membership the details of these tests and computations are explained in the appendix.

The relaxation of resources during the reverse pass increases the availability of the network for future requests, and enables a balanced distribution of the resources along the path. This is needed as the local channel establishment algorithms attempt to acquire as much of the resources as possible on the forward pass. Note that the resources are not reserved for the channel's exclusive use; the channel has only a higher priority for using them, but, in keeping with the principles of packet-switching, they can be used by non-real-time traffic whenever packets from the real-time channel are not present.

If any test at a node fails, the channel cannot be established along that route, and a corresponding message is sent back to the source along the same path. When a node receives an unsuccessful establishment message on its way back, it frees all the resources that were reserved during the forward trip.

Internetworking is supported by hierarchically structuring the establishment message. It contains a frame for each level of the internetworking hierarchy, and when the message enters a subnetwork, a new frame is created that contains the traffic description and performance requirements for this level. As a message leaves the subnetwork, the frame is popped off [7].

2.2.4 Flow Control

Flow control regulates the quantity of data placed on the network per unit time interval. In this way, it increases network efficiency by reducing the number of dropped packets and avoids the need for frequent retransmission. Flow control is also needed to verify that a client does not exceed its traffic load specified at channel establishment time. In case of such misbehavior appropriate countermeasures must be taken in order to satisfy the quality of service guaranteed to other clients of the real-time service.

In the Tenet scheme, flow control is rate-based, i.e. the sender controls its sending packet rate on the basis of its knowledge of the characteristics of the receiver and of those of the channel's path. Rate-based flow control is more appropriate than window-based flow control because it does not require acknowledgements. At channel establishment time, the receiver checks whether it will be able to accept packets at the rate declared by the sender.

A solution of the problem of misbehaving clients consists of providing distributed rate control by increasing the deadlines of the offending packets, so that they will be delayed in heavily loaded nodes, where they would otherwise seriously interfere with the operation of other channels. In case of limited buffer space, some of them might even be dropped because of buffer overflow [10].

2.2.5 Scheduling

In a packet-switching network, packets from different connections will interact with each other at each node (also referred to as switch). Without proper control, these interactions may adversely affect the network performance and violate the guarantees given to the clients. This interaction can be eliminated by using an appropriate rate-based scheduling discipline at the switches. Thus, there are two levels of control: the connection admission control at the connection level (as described in 2.1.3), and the service discipline at the packet level. A rate-based service discipline provides a client with a minimum service rate independent of the traffic characteristics of other clients. Such a discipline, operating at each switch in the network, manages the resources reserved at channel establishment time: bandwidth, buffer space, and schedulability.

In order to effect the control at the packet level, i.e. controlling the order in which packets are serviced, several service disciplines can be used, such as Delay-EDD (Earliest-Due-Date), Virtual Clock, Fair Queuing, the Stop-and-Go protocol, the Hierarchical Round Robin Discipline (HRR), Jitter-EDD, and Rate-Controlled Static-Priority (RCSP) Queuing [8, 23, 24]. Since some of the tests to be performed during channel establishment or in order to verify if dynamic membership is feasible are dependent on the scheduling discipline implemented in that node, the discussion in this paper is restricted to the case of Earliest-Due-Date scheduling in its two version: Delay-EDD and Jitter-EDD.

Classic EDD scheduling is deadline-based, i.e. each packet is assigned a deadline, and the packets are sent in order of increasing deadlines. Delay-EDD is an extension where the server negotiates a contract with each source. If the source (i.e. the client) obeys its traffic description specified at channel establishment, then the server will provide a delay bound. The two parameters x_{min} and x_{ave} are used to characterize bursty traffic and to offer statistical guarantees. The server uses x_{ave} to do bandwidth allocation, and x_{min} to ensure that delay bounds can be met. The key lies in the assignment of deadlines to packets. The server sets a packet's deadline to the time at which it should be sent had it been received according to the contract. This is just the expected arrival time added to the delay bound at the server. By reserving bandwidth at the peak rate, Delay-EDD can assure each channel a hard delay bound [24].

In the Tenet approach this policy is modified in the way that, in case of a conflict, priority is given to deterministic over statistical channels. All of the other tasks of a node have a lower priority; local tasks are preemptable by real-time packets. Thus, the scheduler maintains at least three queues in each node: one for deterministic packets, one for statistical packets, and the third

for all other types of packets and all local tasks. The ordering of the first two queues is by increasing deadlines [10].

The Jitter-EDD [8] extends Delay-EDD in order to provide delay-jitter bounds. This implies a bound on the minimum as well as on the maximum delay. After a packet has been served at each server, it is stamped with the difference between its deadline and its actual finishing time. A regulator at the entrance of the next switch holds the packet for this period before it can be scheduled. Thus, the original traffic pattern is restored in each node and the required minimum and maximum delay guarantees can be met. In conjunction with appropriate connection admission control, this policy allows to provide throughput, delay, delay jitter, and loss free guarantees in a connection-oriented packet-switching environment.

2.3 The Prototype Tenet Protocol Suite (Suite 1)

The functions needed to enable a network to offer real-time service to applications are the following ones [11]:

(1) real-time data transfer at the internetwork layer
(2) end-to-end transport of real-time data
(3) channel set-up
(4) data transfer control

In the prototype Tenet suite, these four tasks are implemented as separate protocols as shown in Figure 2.

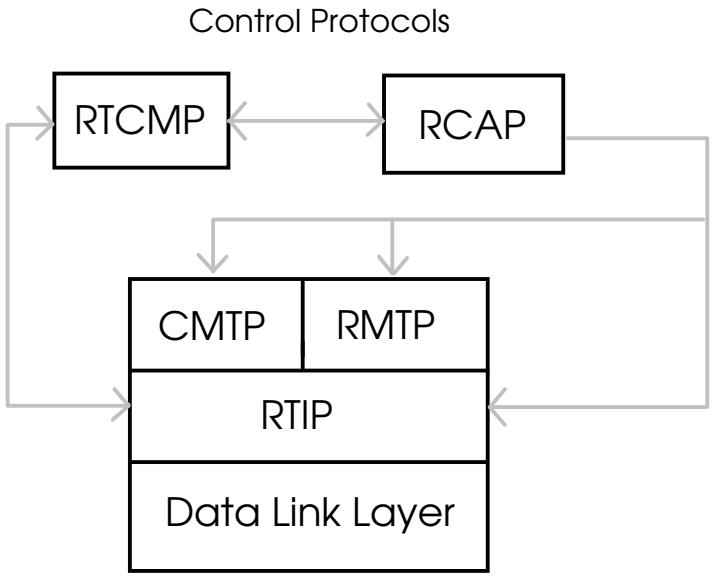

Figure 2: The Prototype Tenet Protocol Suite

- The Real-time Channel Establishment Protocol (RCAP) provides management services, such as channel establishment, teardown, and modification. It takes the client requests containing traffic descriptions and performance requirements and sets up real-time channels in the internetwork [1, 14].

- The data transfers at the internetwork layer are done by the Real-Time Internet Protocol (RTIP), which schedules data packets according to the resource reservations made by RCAP [25].

- Two protocols are designed for the transport layer: the Real-time Message Transport Protocol (RMTP) for message-oriented clients (e.g. urgent messages and mail service with guaranteed delivery latency), and the Continuos-Media Transport Protocol (CMTP), which is appropriate for applications like audio and video.

- The protocol that will control data transfers is called the Real-Time Control Message Protocol (RTCMP) [13].

2.4 The Second Tenet Scheme (Scheme 2)

2.4.1 Support for Multicasting

Scheme 2 is the new Tenet scheme for real-time communication on which the next Tenet suite (Suite 2) will be based. One of the main goals of this scheme is scalability, i.e. the support for a range of applications from those consisting of a few participants to those serving thousands. The support of applications with large-scale group communication, e.g. conferencing, requires multicast communication. Multicasting, i.e. the transmission of a packet from a source to a set of destination hosts, is an important service for improving the efficiency and robustness of distributed systems and applications. A conference could also be set up by specifying 1-to-1 channels from each source to each destination but this solution requires huge amounts of resources and long establishment times. Thus, efficient multicasting becomes increasingly important, especially in the context of packetized audio and video because of the very high bandwidth demands of these media.

2.4.2 The New Ideas

The key features being included in Scheme 2 are a new model of the client-service interface, real-time multicast channels, dynamic membership, dynamic channel management, and the channel group abstraction to support resource sharing at the network level.

QoS Specification as a Range

In the new model of the client-service interface, the Quality of Service requirements are specified as ranges since an application is likely to have some desired QoS requirements while it could tolerate some deviation at the same time. By specifying ranges at the channel establishment the need for negotiations is reduced and a higher probability of the acceptance of that channel is achieved. The range is expressed by specifying the desired value and the tolerable deviation.

1xN Multicast Channel

In order to achieve the goal of multicasting, the Scheme 2 paradigm is one of a 1xN channel, i.e. a single source and multiple destinations. From this abstraction the support of many

models of conferencing is feasible, from a simplex unicast channel to a complex conference that combines many channels into a multiple source - multiple destination conversation.

Target Set and Dynamic Membership

In the medium to large scale applications sources have to address many destinations and do not want to specialize on a per-destination basis. Respectively, destinations are not interested in individual sources but want to announce what data they wish to receive and what QoS they require. The entity enabling such behavior is called a *Target Set:* a set of destinations that receive the same data each specifying its QoS requirement. Thus, semantically a target set unites the nodes that have a common interest to receive certain data with guaranteed performance. Figure 3 illustrates the specification of a Target Set.

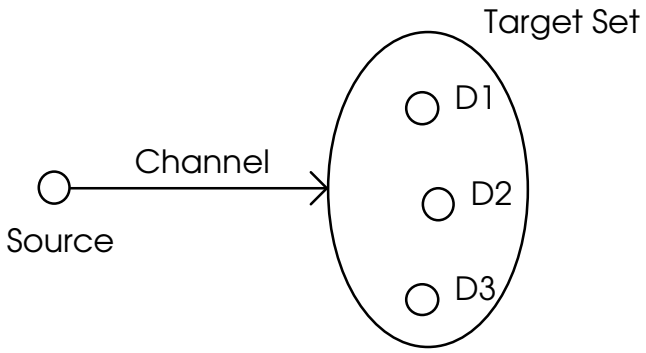

Figure 3: Specification of a Target Set

Sources do not address destinations but Target Sets. Thus, a new source joins an ongoing conference by establishing a channel to an existing Target Set. However, Target Sets have loose semantics: for any given channel addressing the Target Set, some destinations may be rejected at channel establishment time. Still, the invariant exists that no destination can receive from a channel if it is not in the Target Set of that channel. Destinations can be members of multiple Target Sets and may specify a different QoS requirement for each one. Destinations must be able to dynamically join and leave Target Sets. Since multiple channels may address a Target Set, a join of this Target Set must imply an attempt to join all those channels.

On-Demand and Secondary Channels

In order to reduce routing and establishment effort, sources speaking up rarely can be marked *on-demand*. In this case the channel establishment is not fully completed at channel establishment time and the channel cannot be used immediately. Before the transmission of data, the channel has to be enabled which completes the establishment. Thus, a part of the channel establishment can be done together with other channels and the complexity of the on-demand channel establishment is reduced. Channels for sources accepting any fixed QoS they can get can be marked *secondary*. This allows the network to use any existing route to the destinations in a Target Set.

Dynamic Channel Management (DCM)

Dynamic channel management increases the availability and flexibility of a packet-switching network by providing it with two powerful capabilities. The first capability is the ability to modify the traffic and performance parameters of a real-time channel without violating the client's contract. This modification can be made transparently to the client. The second capability is the ability to transparently reroute a real-time connection without violating the guarantees given to the clients [17].

Sharing of Resources

Application-specific relationships can be used in order to reduce the amount of resources that must be reserved for that application. Most of the target applications such as conferences and panel discussions do not allow an arbitrary number of persons speaking at the same time, i.e. concurrently active sources. For the network, this fact implies a bound on the aggregate traffic of this group. If the network is informed about these relationships resources can be shared by that group and the utilization of resources is improved. Such groups are called *Sharing Groups* and can be defined as a set of channels with an aggregate traffic characterization. The sum of all sources sending data over channels specified by a Sharing Group will not violate the specified traffic model. Figure 4 illustrates the idea of resource sharing.

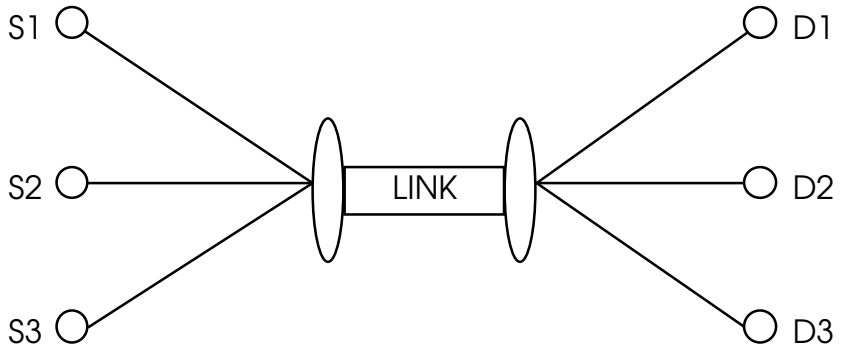

Figure 4: Resource sharing

The sources S1 through S3 send data over the link. If, e.g., only one active source is allowed at the same time, only the resources for one connection must be reserved in that link and the switches, and the required amount of resources for this application is substantially decreased.

The Database

In the case of multicast communication in Scheme 2 more information must be stored in the network than in the case of unicast communication. For example, the network has to maintain the membership of Target Sets and Resource Sharing Groups, store information about multicast trees and results of operations, etc. All this information will be stored in a database (e.g. in an X.500 directory). Since Scheme 2 is still in the design phase only the need for such a database but no details are known.

2.4.3 Importance of Dynamic Membership

Dynamic membership is one of the main characteristics of many important applications such as conferences. Some participants of a conference may be late comers, i.e. they do not show up at the time the conference is set up. Additionally, there may always be some unknown potential participants, i.e. persons who are not known at the establishment of a conference but who want to join as soon as they are informed about it. On the other hand, some participants may prefer leaving a conference before its end.

As we have seen in the explanation of the new ideas in Scheme 2 and the characteristics of a conference, the realization of dynamic membership in real-time protocols supporting applications such as conferencing is absolutely necessary and therefore an important feature in the Tenet approach. The remainder of this thesis consists of a definition of the goal of our work, an explanation of the details of the problem and our approach to the support of such a dynamic membership.

2.4.4 Goal of Our Work

The general goal of this work is to enable a dynamic membership for destinations in Target Sets in the Tenet Scheme 2 environment. This implies the support of a dynamic join as well as a dynamic leave.

These operations have more specific goals: they should be as cheap as possible, fast, efficient, scalable, and maybe even with guaranteed success of the operation. Since some of these goals are contradictory a trade-off has to be made.

A leave operation must always succeed. In order to provide a good service it is essential that a destination is able to leave a conference when it decides to do so.

Concerning the join the following solution was chosen: In order to minimize the complexity guarantees for the success of this operation are not given. The operation and the algorithm implementing them should be relatively cheap in terms of code complexity, computation costs and required network resources but should not have a very bad performance in terms of probability of success.

CHAPTER 3
DYNAMIC MEMBERSHIP IN MULTICAST GROUPS

3.1 Basic Design Considerations

Dynamic membership in Scheme 2 always refers to a Target Set and not to single channels. Since destinations are interested in receiving certain data that may be sent by multiple sources using multiple channels it does not make sense to join a single channel. If a subset of nodes is only interested in some of the channels' data the creation of another Target Set is required.

Dynamic membership only refers to nodes in the role of a destination, i.e. clients who want to receive data. If a client wants to be a source and send data, the establishment of a channel is required, a different operation. In the case of a dynamic join we must consider the facts that basically every destination may want to join a conference and that destinations are always associated with a QoS specification, i.e. destinations want to receive data with a guaranteed QoS. Thus, the performance requirements of the new destinations must be met.

In the case of a dynamic leave we must consider the problem of how to reclaim resources, i.e. resources reserved for the participation of that node in the Target Set it is leaving should be available for other applications after the leave.

Finally, the most important point that we must take into consideration is the fact that we do not want to disrupt existing connections. In either case, dynamic join or leave, the transmission of data to the destinations must not be interrupted, and the guarantees given to these destinations before the join or leave must be met.

3.2 Scenario of a Conference

Since multimedia interaction between users usually takes the form of conferences let us consider the scenario of a multimedia conference as an example for our model of real-time multicast channels. We distinguish between the underlying network, i.e. the physical topology of the links, and the topology of a multicast tree. The underlying network consists of a set of nodes (routers) interconnected by links. 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. Routers are denoted as diamonds in Figure 5 which shows our example of the underlying network.

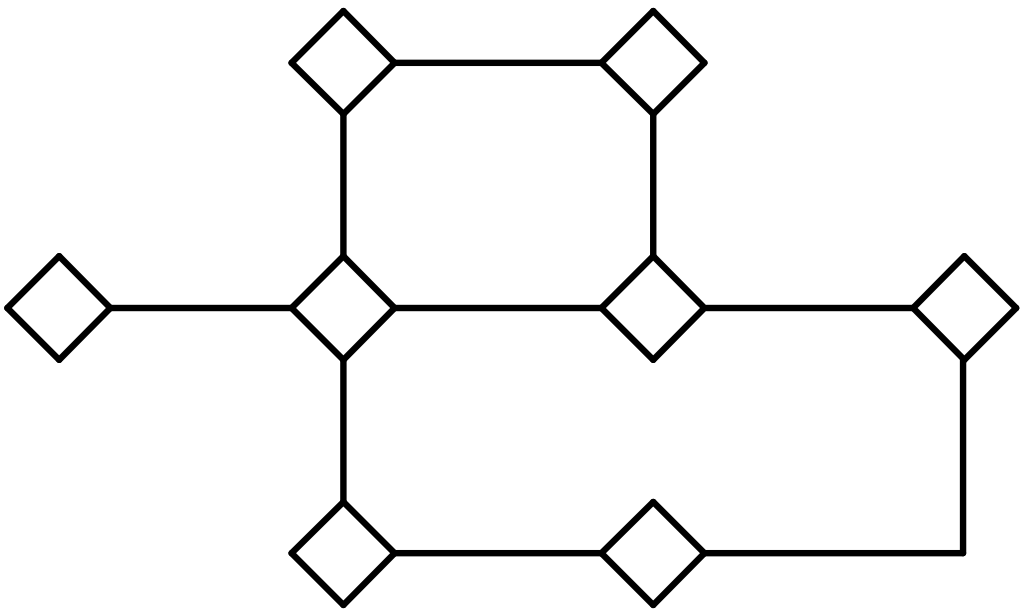

Figure 5: Physical Topology of a Network

Concerning the topology of multicast trees we further distinguish between multicast tree nodes (denoted as dots in Figure 6) and application nodes (denoted as circles in Figure 6). 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 (i.e., the amount of reserved resources) of the node 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 (as explained in 2.2.1). We will see that these tables are also instrumental in finding good attachment points for new nodes.

An end system is always composed of two different nodes (a multicast tree node and an attached application node) since each end system must also contain a routing function (similar to each host containing an IP implementation in the INTERNET). 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. Applications receiving data are called destinations, applications sending data are called sources. The circle node is associated with the application that wants to send (in case of a source) or receive (in case of a destination) data, the diamond with the router that actually sends the data to the destinations or receives the data and forwards it to the upper layers. A circle node corresponds to layers 4 to 7 of the OSI reference model, a diamond forwarding the packets corresponds to layers 1 to 3.

Note that we can only change the topology of the circles; the topology of the link level routers is usually set up separately by operator or long-term network management algorithms. It is neither affected by multicast connection setup and teardown nor by joining or leaving nodes.

Figure 6 shows our simple scenario of a conference in the Scheme 2 environment (e.g. a panel discussion with two panellists and three audience members). The important elements in this scenario are the sources S_i, the destinations D_i, two multicast channels (1xN channels), and a Target Set. Semantically, sources send data to a Target Set (as explained in 2.3.2) but, of course, the sources are connected to the destinations by the two 1xN multicast channels. The Target Set

contains the nodes D2, D3 and D5. One of the routers is designated as node *n* because it is of particular interest in one of the following chapters.

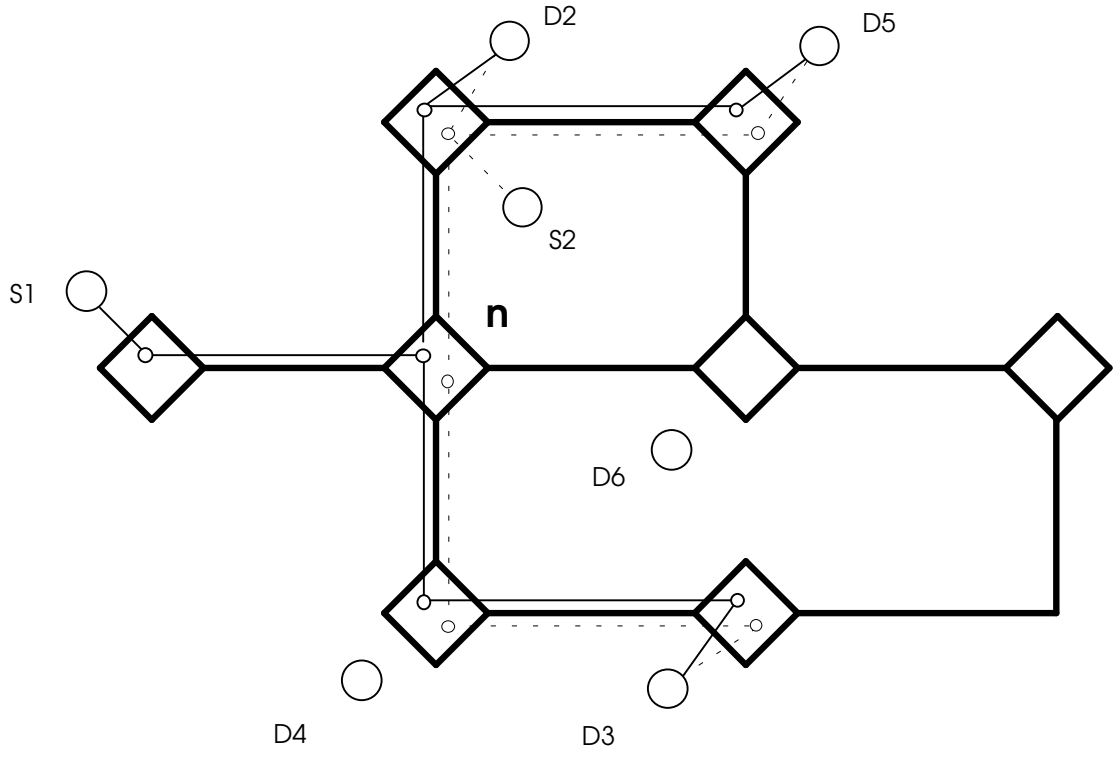

Figure 6: Scenario of a Conference in Scheme 2

The events we will discuss in our example are as follows:

- The new destinations D4 and D6 want to become members of this conference, i.e. they want to receive the same data as the other destinations.
- One of the destinations in the Target Set wants to leave it.

3.3 The JOIN Operation

3.3.1 The JOIN Service Primitive

The join operation adds a new destination node to a Target Set. A complete join operation requires two separate steps to be taken: the inclusion of the new destination node in the database specifying the members of that Target Set (as explained in 2.4.2) and the connection of the new destination node to the channels currently sending to the Target Set. The service primitive that effects the join takes the following form:

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

NodeId is the address of the new destination that wants to join the Target Set specified by TargetSetId. In the Quality of Service Specification (QoS) the new destination can specify the QoS it wants to obtain. The ALL-OR-NOTHING or PARTIAL-OK option allows the user to specify what the outcome of the operation will be if the network fails to connect the node to all existing channels.

If the PARTIAL-OK option is chosen the new destination accepts not being connected to all channels sending to the Target Set. In case of the ALL-OR-NOTHING option the join operation will not be executed if the connection to at least one channel fails, either because the Quality of Service Specification of the new node cannot be met or because the joining would violate performance guarantees given to other channels.

3.3.2 Design Alternatives

Success Vector

A design alternative would be to issue the JOIN operation without an ALL-OR-NOTHING or PARTIAL-OK option. The network would return a *success vector* indicating to the application which channels could be connected to, 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 not.

This alternative has major drawbacks as follows: Even if the connection to one of the channels fails the network continues to try to establish connections to the remaining channels. Since multiple nodes will be tested for every channel in order to verify if a connection is feasible, and resources must be reserved at every node for every channel, this is a very expensive operation. If the client is not willing to accept a partial rejection it is useless to continue after the connection to one of the channels failed, and resources are wasted. Additionally, the transport layer interface has to be crossed twice which is expensive, and the network has to maintain state information for uncompleted JOINs which is hard to clean up in the case of a failure. If the user states in advance what the outcome of the operation should be in case of partial rejection these drawbacks can be avoided.

Enabled/Disabled Destinations

Another design alternative are on-demand destinations, similar to on-demand channels (as explained in 2.3.2). This alternative implies two different states of destinations (enabled and disabled) and requires an option in the JOIN service primitive to choose the desired state as well as two service primitives (ENABLE(Destination-Id,TargetSet-Id) and DISABLE(Destination-Id,TargetSet-Id)) to effect the transitions. A disabled destination will not receive data sent to the Target Set.

Several possibilities exist for the difference between the two states which can be explained for the transition from the enabled to the disabled state. If we want to guarantee that a destination being enabled gets back the same QoS it had before the resources reserved for this destination must be available at the (unknown) point of time of the ENABLE call. This has the drawback that these resources must remain reserved and are not available for other real-time channels. Thus, the only possible use of these resources is for non real-time traffic or for the destination itself. In order to make the resources available for non real-time traffic there is no need to explicitly disable a destination and free the resources reserved for it. The Tenet scheme allows non real-time traffic to use resources reserved for a real-time channel as soon as there is no traffic on that channel. Therefore, it is sufficient to cut the traffic flow on the branch connecting the destination with the multicast tree.

An efficient use of resources can only be achieved by accepting the risk of getting a worse QoS than the one promised to the destination at the previous enabled state. In this case the resources become available for other real-time channels. In the disabled state only logical information (such as routing information) is stored and physical resources (such as bandwidth and

processing power) are reserved when enabling a node. Thus, the network can use resources efficiently, and there is no need to compute a new route for the destination at every ENABLE call. Additionally, a destination can join a Target Set before it actually wants to receive the data and express its interest in the data sent to the Target Set. In this way channels established to the Target Set between the JOIN of the destination and the point of time when it is enabled are able to consider this destination in their routing algorithm. This routing information can even be optimized before enabling the destination.

But it must be emphasized that all the benefits which can be gained are on the network's part and no advantages are achieved for the user. It is questionable whether the user would ever disable a destination. Therefore, this alternative is not considered here.

3.3.3 The JOIN Algorithm

In addition to including the new destination node into the data base specifying the Target Set, the JOIN operation must attempt to include the new destination into the multicast trees of all existing channels currently sending to the Target Set. This implies finding an optimal way of including the new node into each of the existing multicast trees. Since the multicast channels are mutually independent the optimal attachment point can be chosen separately for each multicast tree. The attachment algorithm presented below can be applied to each multicast tree separately. In any case the new node will only be attached to a tree if

1. all performance guarantees given to the 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 will use the following notation which is illustrated for our example by Figure7:

- n_j denotes the joining destination node
- n_c denotes a candidate attachment node
- p_c denotes the path from n_j to n_c
- n_μ denotes the nodes on p_c ($\mu = 1,...,u$)
- n_ν denotes the nodes on the path from the source to n_c ($\nu = 1,... v$)

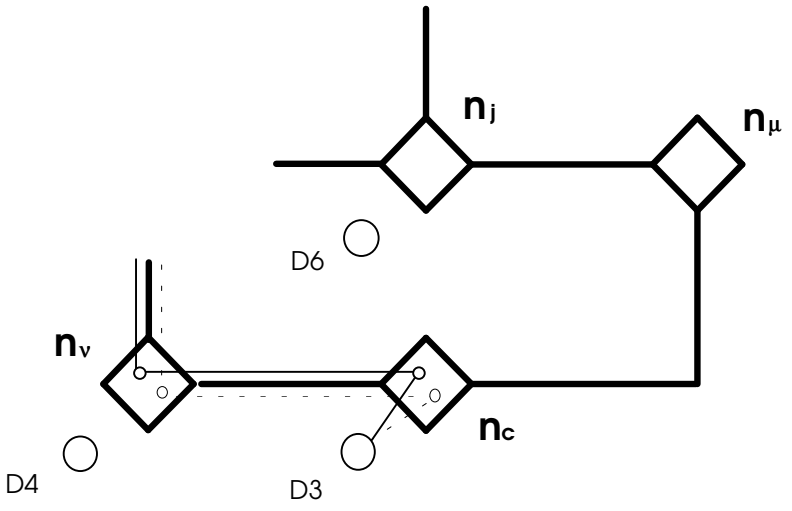

Figure 7: Notation for the AttachJoiningNode Algorithm

The AttachJoiningNode algorithm takes the following form:

ALGORITHM AttachJoiningNode (n_j)

```
begin
REPEAT
        find-candidate-node(nj ; nc,pc)          /* select subset of nodes to
                                                 /*reduce search space
                                                 /*outcome: candidate node nc
                                                 /* and path pc leading to it
        multicast-admission-tests(nc)            /* enough resources at
                                                 /* attachment node ?
        for     each node nμ  on pc between nj and nc do
                unicast-admission-tests(nμ )     /* enough resources on the
                                                 /* path pc ?
        if path pc okay then
                destination-tests(nμ )           /* can QoS requirements of nj
                                                 /* be met?
        if      okay then
                compute-cost(nj,nc ; cc)         /* cost cc for attaching
                                                 /* new node nj to nc
UNTIL all candidate nodes are found
find nc,opt such that cc,opt = min (cc)          /* minimum attachment cost
attach(nj,nc,opt)
end
```

The AttachJoiningNode algorithm contains six subroutines. The *find-candidate-node* subroutine will be described in 3.3.3.1; the *multicast-admission-tests* subroutine will be explained in detail in 3.3.3.2. The *unicast-admission-tests* determine whether a node on the path from the joining node to the attachment candidate can handle the additional (guaranteed) traffic. These tests are identical to the server tests for admission control for the establishment of unicast channels in Tenet Scheme 1 and the detailed formulas are explained in the appendix. The *destination-tests* subroutine considers the QoS requirements of the new destination and will also be described in 3.3.3.2. The *compute-cost* subroutine will use the same optimization criteria as the corresponding cost function in the channel establishment algorithm; the discussion of this topic is beyond the scope of this thesis. The *attach* subroutine will be explained in 3.3.3.3.

3.3.3.1 Finding a Candidate Attachment Node

As mentioned above, we distinguish between the underlying network and the topology of multicast trees and we cannot change the topology of the underlying network. Concerning the position of the new destination that wants to join we distinguish between two cases:

1. The new destination is located at a router already forwarding data for the multicast connection. In this case the only potential attachment node is the local router. In our scenario in Figure 6 destination D4 can only be attached to both multicast channels at the local router.

2. The multicast tree is not passing through the local router, i.e. the shortest distance between the new destination and the multicast tree is one or multiple hops. This case is illustrated by the position of destination D6 in Figure 6. In this case the optimal inclusion point can be anywhere in the existing tree. Concerning the inclusion of the new node in the multicast tree two alternatives exist:

   - The optimal solution is to recompute a new multicast tree including all existing destinations and the new one. This solution is closely related to the establishment of a multicast channel. It results in an optimal multicast tree but it will be a very expensive operation and a disruption of service could result for other nodes during the restructuring operation, thus violating the guarantees given to the existing channels. Disruption can be avoided if a Dynamic Channel Management scheme is in place.

- If suboptimal attachment is acceptable a possibility is to consider attachment as a leaf node to the existing multicast tree. The new node can be attached to an existing leaf node or to an existing intermediate node. We use the term *leaf node* for a node with no descendants and the term *intermediate node* for a node in the middle of a multicast tree. 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 and possibly its subtree(s) (in case of a violation), but no other branches of the multicast tree. Again an optimization algorithm must be executed to find the best attachment point.

In order to keep the JOIN operation simple it is only a layer 4 operation, i.e. a new host joining the Target Set does not cause a restructuring of the multicast tree. 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 Set.

Thus, the only way of including a destination node in a multicast tree is to attach it as a leaf node to one of the multicast tree nodes of that channel. Our goal concerning the *find-candidate-node* subroutine is to find a good attachment node where the new destination can be connected to a multicast tree. The first consideration concerning this algorithm is the question 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 costs. The second solution will maintain better network performance, and we therefore prefer to evaluate several attachment alternatives.

On the other hand we must take into consideration that in a large scale conference with many participants, eventually distributed all over the world, many multicast tree nodes are potential attachment nodes. It would be too expensive to consider all potential attachment nodes. In order to keep the JOIN operation scalable the first step of the algorithm is the computation of candidate nodes for the attachment. In this way a reasonable subset of nodes can be chosen and actually taken into consideration for the admission tests. This focusing on only a subset of nodes is especially important since at each potential attachment node several computations have to be executed as described in 3.3.3.2. Unfortunately most networks do not maintain geographical

information in their nodes; therefore we cannot use a "nearness" criterion. The only information typically available is the hop distance between nodes.

We propose two solutions. Note that both of them do not require the sources to do the computations. This is important since the source of a multicast tree is a potential bottleneck anyway and such a solution will 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 route first hits the multicast tree (this can be any of the multicast tree nodes or the source itself) becomes a candidate node. 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. A shorter route (i.e. a route with less hops) from the new destination to these neighbor nodes than the one computed before might exist; therefore these routes are re-computed.

Expanding Circle Search: The new destination checks all its neighboring nodes whether they are 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 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. 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 number 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 tree will show which is better.

The following example that is based on Figure 6 illustrates the two different algorithm. We now have the following possibilities: in order to include the new destination D6 into channel 1 and channel 2 we must attach it to a multicast tree node (i.e. a dot) of the respective channel at one of the routers that contain a multicast tree node of the respective channel. As mentioned above in the *Expanding-Circle-Search* these routers should be reached without using routers of the network which are already part of the multicast tree (e.g. it does not make sense to attach the new destination D6 to the router at D2 since this router can only be reached by using the router at D5 or router *n*; it is a preferable to attach it directly to the router at D5 or the router *n*). Thus, the possible attachment points for both channels in our scenario are the router at D3, the router at D5, and the router *n*.

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 the cost function will be computed for all candidate nodes, and the best attachment node will be selected, as described below.

3.3.3.2 Consideration of Performance Guarantees

As mentioned in the basic design considerations, in any case we have to ensure that the guarantees given to existing channels are not violated and that the performance requirements of the new node are met. The new node will only be attached to a multicast tree if these goals can be achieved. Otherwise the connection of the new destination to this channel is not possible. Since network resources are always finite, performance guarantees always imply resource reservation. Therefore admission tests must be performed whenever a new resource is requested.

This problem is similar to the channel establishment in Tenet scheme 1 (as explained in 2.2.3). In Scheme 1 admission tests verify if a new (unicast) channel can be established in the network without violating the guarantees given to the existing channels. These *unicast-admission-tests* are used along the path p_c (see 3.3.3) and are also the basis of the *multicast-admission-tests*.

In order to verify if a candidate node (i.e. a potential attachment node) is able to handle the attachment of the new destination without violating the guarantees of the channels already passing through that node, *multicast-admission-tests* have to be executed at this node. These tests differ from the *unicast-admission-tests* since a new link and not a new channel must be served by the node. We must verify whether enough resources are available at that node to serve the new link. Additionally, the minimum local delay bound that can be offered for scheduling the packet for the new link has to be computed.

We are not able to specify the details of all multicast admission test until we know exactly what happens in a node in Scheme 2 and how the guarantees given to a destination in the node's subtree (i.e. a destination receiving its data over a channel which uses that router) can be met. The exact algorithms within a node are related to many other functions in Scheme 2, such as channel establishment, routing, DCM, and resource sharing. These other functions are currently still in the design phase.

Using our scenario as shown in Figure 6 we can give an example for our performance considerations. The two multicast channels use the same router *n* to forward data to the respective destinations. i.e. *n* receives packets from Channel 1 and Channel 2. The destinations D2, D3, and D5 are receiving data from both channels and are thus in the subtree of router *n*. Each destination has specified its QoS requirements, either at the channel establishment or at a JOIN operation if it has joined the conference later. In this example Target Sets are not considered since only the admission tests at node n are of interest. Therefore, all destination may receive data from Channel 1 with a different QoS than data of Channel 2 since it is possible that the two channels send to different Target Sets and the destinations have specified different performance requirements for each of the Target Sets.

We assume that (similar to Scheme 1) node *n* is not aware of any end-to-end performance requirements of a channel. For example, the end-to-end delay bounds $Dmax_j$ from the source to the respective destinations D_j are unknown but node *n* knows the different 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 the link 2 towards destinations D2; $d_{n,1,4}$ for the packets on channel 1 towards D4, and $d_{n,2,4}$ for the packets on channel 2 to be sent over the outgoing link 4 towards D4. Figure 8 illustrates the local delay bounds known to node *n*.

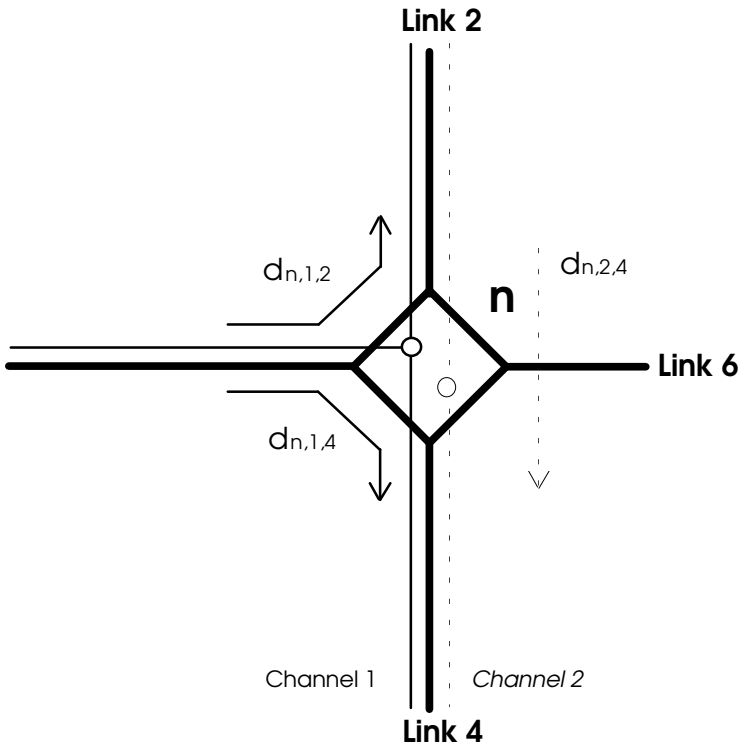

Figure 8: Local Delay Bounds at Node *n*

Other local parameters characterizing a channel *i* and each of its outgoing link *l* in a node *n* can be derived similarly. They are similar to the parameters relevant to real-time unicast channels as described in [9], but augmented by a link identifier *l* where appropriate:

- $t_{n,i}$: maximum service time for a packet of channel *i* at node *n*
- $w_{n,i}$: local minimum probability that a packet is not lost due to buffer overflow (the clients reliability requirement W_{min} is mapped onto local parameters $w_{n,i}$; see 3.3.3.3 for details)

Admission Test for CPU Power

When a packet arrives on one of the channels with more than one outgoing link at the node, one or more copies of it have to be made since the channels split and separate packets have to be sent over the outgoing links used by the branches of that channel. Additionally, the packet

must be sent out over the outgoing links. These operations require a certain amount of CPU power. Simulations of multicast channels have not been done so far, and details about the required amount of CPU power are not yet known.

The following assumption is made in order to simplify this problem: 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 ($t_{n,i,OUTPUT}$). The time $t_{n,i}$ required for processing a packet of the channel i at the node n is

$$t_{n,i} = t_{n,i,INPUT} + f_{n,i} \cdot t_{n,i,OUTPUT} \quad (1)$$

where $f_{n,i}$ is the fanout of channel i at node n. 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 ***computing 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 usage U_n of the server at the node n by the amount required for the new link (which is equal to increasing $f_{n,i}$ by 1)

$$U_n := U_n + \frac{t_{n,i,OUTPUT}}{x_{min,i}} \quad (2)$$

where $x_{min,i}$ is the minimum interarrival time of the packets (the worst case for utilization). The actual test consists of verifying whether U_n is still smaller than 1:

$$U_n < 1 \quad (3)$$

If this inequality is satisfied enough computing power is available to serve the new link and the computing power test is passed.

In our scenario as shown in Figure 8 both channels are running through node n. The fanout of channel 1 at node n is equal to 2 ($f_{n,1} = 2$), the fanout of channel 2 is equal to one ($f_{n,2} = 1$). We assumed that the Source-Directed Search will consider node n as a candidate for attaching the new destination D6 to channel 1. If enough CPU power is available at node n so that D6 can be attached to channel 1 at node n over link 6 can be verified by incrementing the fanout of channel 1 at node n from two to three and doing the computations of the computing power test.

Admission Test for Delay Bound

Obviously the availability of sufficient CPU power does not imply that all delay bounds will be met; delay bounds must be tested independently. Therefore the next step is the ***delay bound computation***. Its goal is to determine the minimum local delay $dmin_{n,i,l_j}$ that can be offered by the server of the node n to the new link of the channel i where l_j is the outgoing link towards the joining destination n_j. This is the minimum value of d_{n,i,l_j} that makes the packets on the link l_j of the channel i schedulable without violating the guarantees given by node n to other channels.

The verification that the assignment of $dmin_{n,i,l_j}$ to the new link does not cause any violations of the local delay bounds of the other packets scheduled by this node can be achieved 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 and represents the maximum delay a packet is allowed to experience at that node: $d_{actn,i,l}$ is the local delay bound used as an operating point by the node, i.e.; the node will never exceed $d_{actn,i,l}$ in its current operations. Storing these two bounds enables the creation of a table that contains tupels of the local delay bound and the value of the actual delay. This allows us to change the operating point of the node without violating guarantees given to the user.

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

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

If the scheduling discipline used 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 $d_{actn,i,l}$ and $d_{n,i,l}$ increasing order of $d_{actn,i,l}$. The table for node n in our example in Figure 8 would look like the table shown in Figure 9. It contains the three tupels of local delay bounds known to node n; β indicates the position of a tupel in the table.

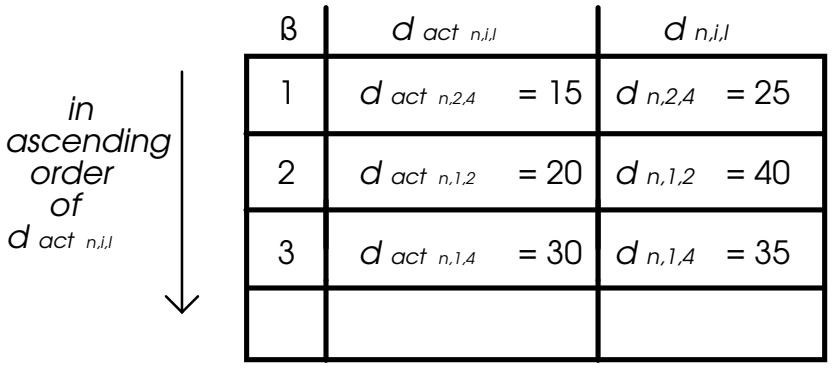

Figure 9: Table of Ordered Tupels [$d_{n,i,l}$; $d_{actn,i,l}$]

The minimum local delay $d_{n,i,lj}$ that can be offered by the node is the value of the first position in the list where the delay bound of the new link can be inserted without causing any of the links of the channels to violate their delay bounds. The insertion of the local delay bound for the new link somewhere in this table can cause increasing values of $d_{actn,i,l}$ of the links below it since the packet for the new link is scheduled before those. This is possible as long as the new operating points $d_{actn,i,l}$ do not exceed the bounds $d_{n,i,l}$ for any link (see equation 4). The positions above the insertion need not to 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 $d_{actn,i,l}$ below the insertion point exceeds its bound $d_{n,i,l}$

$$d_{actn,i,l} > d_{n,i,l} \qquad \forall \beta > \kappa \tag{5}$$

where κ is the position of the insertion.

In our scenario the local delay bounds known at node n are $d_{n,1,2}$, $d_{n,1,4}$, and $d_{n,2,4}$ (as shown above) and the respective actual local delays. The minimum local delay bound $d_{n,i,lj}$ that must be determined in order to attach destination D6 to channel 1 over link 6 is $d_{n,1,6}$.

Admission Test for Buffer Space

As mentioned above 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 as many copies as outgoing links of that channel exist and then schedules each copy separately according to its specific local delay bound. The drawback of this solution is the fact 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 respective local delay bounds are known to the node, 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. Other links are served in the same way. When the last delay bound is about to expire the packet being still in the buffer is sent. 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 at node *n* $w_{n,i}$ 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 assigned to the new link is greater than all other delay bounds assigned to that channel at the node. Thus, the probability that an attachment is rejected due to insufficient buffer space is reduced. Moreover, the ***buffer space test*** for availability of additional buffer space is not always required. Thus, the overall costs of a JOIN operation can be reduced.

If we take the second alternative (i.e. copy on output) the buffer space test starts with checking

$$dmin_{n,i,lj} > \max_{l}(d_{n,i,l}) \qquad (6)$$

where $dmin_{n,i,lj}$ is the minimum local delay bound assigned to the outgoing link of the new destination (which has been computed at the *delay bound computation*) and $\max_{l}(d_{n,i,l})$ is the maximum local delay bound assigned to any outgoing link of channel i at node n. If this condition is false the buffer already assigned at this node is sufficient to accommodate the new link. Otherwise new buffer space has to be acquired.

The available buffer space B at node n is

$$B \leq B_n - \sum_{k=1}^{c_n} b_{n,k} \tag{7}$$

when B_n is the total amount of buffer space at the node n, c_n is the number of channels already running through that node, and $b_{n,i}$ is the buffer space assigned to channel i.

This value is assigned to $b_{max_{n,i}}$, which is the maximum value of buffer space available for channel i at node n

$$b_{max\,n,i} := B \tag{8}$$

and the maximum available buffer space is reserved for the additional buffer space $b_{add_{n,i}}$ for channel i to which the attachment is attempted:

$$b_{add\,n,i} := b_{max\,n,i} \tag{9}$$

The next step is the computation of the maximum amount of time $dmax_{n,i}$ the packets of channel i are allowed to stay in the buffer.

$$dmax_{n,i} = \frac{b_{max\,n,i} \cdot x_{min,i}}{s_{max,i}} - \sum_{\nu=m}^{\nu-1} (d_{n\nu,i,l} - t_{n\nu,i}) \tag{10}$$

where n_ν denotes the nodes along the path from the source to the attachment node (ν =1,...v). The first term computes the maximum amount of time taking the worst case into consideration by using the maximum packet size ($s_{max,i}$) and the minimum interarrival time ($x_{min,i}$) of the packets of channel i. The second term considers the jitter that may be experienced by the packets at previous nodes. In case of a jitter-controlled channel (i.e. $J_{max,i}$ is defined), m expresses the position of the previous node able to control jitter since at each node of this type the original traffic pattern is

restored. If $J_{max,i}$ is undefined, no jitter-control is in effect, and the full path from the source to this node is considered by assigning m the value of 1. Note that at each node n_v in equation 10 the link index l is a function of the new destination n_j (l = f(n_j). This function determines the outgoing link towards n_j.

The actual buffer space test consists of verifying if the amount of time the packets of channel i are allowed to stay in the buffer is greater or equal to the amount of time the packets sent over the new link exceed the amount of time the packets were stored so far:

$$d_{max\,n,i} \quad \geq \quad (dmin_{n,i,l_j} - \max_{l}(d_{n,i,l})) \tag{11}$$

If this inequality is satisfied, enough buffer space is available to store packets of that channel even for an amount of time which is equal to the greatest delay bound of all outgoing links (which is the one assigned to the new link).

In addition to verifying that enough resources are available at the attachment node, we must take the QoS requirements of the new destination into consideration. At the new destination, the *destination-tests* subroutine determines whether the accumulated guarantees 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 $J_{max,i}$

These tests are similar to the destination tests for admission control at channel establishment in Scheme 1 (as explained in the appendix). We must take into consideration that in this case the end-to-end performance requirements along the complete route from the source of the channel to the new destination, and not only the path p_c from the attachment node to the new destination must be considered. The destination tests are now described in detail.

Destination Test for Delay Bound (*D*-Test)

This test determines whether the end-to-end delay a packets experiences along the path from the source of channel i to the new destination exceeds the maximum delay bound $D_{max,i}$ that the new destination is willing to accept. The end-to-end delay is composed of two terms: $D_{prop,i}$

and $D_{min,i}$. The first term is the sum of the propagation delays of all links from the source to the new destination

$$D_{prop,i} = \sum_{\nu=1}^{v} dprop_{n\nu,i} + \sum_{\mu=1}^{u} dprop_{n\mu,i} \quad (12)$$

where the first sum considers the path from the source to the candidate node, and the second sum the path from the candidate node to the destination. $D_{min,i}$ is the minimum end-to-end delay that a packet may experience at the nodes along the path from the source to the new destination

$$D_{min,i} = D^*_{nc,i} + dmin_{nc,i,l} + \sum_{\mu=1}^{u} dmin_{n\mu,i,l} \quad (13)$$

where $D^*_{nc,i}$ is the total delay a packet experiences along the route from the source to the candidate node n_c (each node n in a multicast tree is aware of the delay $D^*_{n,i}$ a packet of channel i experiences between the source of this channel and node n); $dmin_{nc,i,lj}$ is the minimum local delay bound that has been computed in the *multicast-admission-tests* for the attachment node n_c. The sum in equation 13 considers the minimum local delay bounds $dmin_{n\mu,i,l}$ that have been computed in the *unicast-admission-tests* for the nodes n_μ on the path from n_c to the new destination n_j. Again, at each node n_μ in equation 13 the link index l is a function of the new destination n_j (l = f(n_j). This function determines the outgoing link towards n_j. The actual *D*-Test is equivalent to the one performed in Tenet Scheme 1 and consists of verifying whether total delay that can be offered is smaller or equal to the delay requirement:

$$D_{min_n,i} \leq D_{max,i} - D_{prop,i} \quad (14)$$

Destination Test for Jitter (*J*-Test)

This test is only computed if a delay jitter bound $J_{max,i}$ is defined. For this test the values of the local delay bounds $d_{n,i,l}$ on the path p_c are needed. So far only the value of $dmax_{n,i,l}$ is known for each node; this is the amount of time the packets of channel i are allowed to stay in the buffer as computed in equation 10. In order to reduce the cost of the JOIN operation we first compute the *J*-Test considering $dmax_{n,i,l}$. If the previous J-type server is on the path p_c between the attachment node and the new destination we compute the jitter accumulated in the worst case as follows

$$J_{nj,i} = \sum_{\mu=m}^{u} (dmax_{n\mu,i,l} - t_{n\mu,i}) \quad (15)$$

where m is the position of the previous J-type server. If the previous J-type server is not on the path p_c but somewhere between the source and the attachment point we compute

$$J_{nj,i} = \sum_{\nu=m}^{v} (d_{n\nu,i,l} - t_{n\eta,i}) + \sum_{\mu=1}^{u} (dmax_{n\mu,i,l} - t_{n\mu,i}) \quad (16)$$

considering the (known) delay bounds along the path between the J-type server and the attachment node, and the values of $dmax_{n,i,l}$ between the attachment node and the new destination. We then verify if the accumulated jitter is below the bound specified by the new destination.

$$J_{nj,i} \leq J_{max,i} \quad (17)$$

If this test is passed the network can satisfy the jitter requirement in any case. In the case of a violation of inequality 17 we compute the local delay bounds $d_{n,i,l}$ at the nodes along the path p_c and re-compute the *J*-Test with these delay bounds. The computation of these delay bounds is explained in 3.3.3.3.

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. The cost of the attachment to n_c over p_c is computed.

3.3.3.3 Attachment of the New Node

The AttachJoiningNode algorithm 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 $p_{c,opt}$ are reserved.

The first alternative has the drawback that most of the preliminary reservations have to be released explicitly. This is less robust in case of network failures. The second alternative has the drawback of a small risk that a parallel JOIN or ESTABLISH allocates the resources between the admission test and the actual reservation when a decision for an attachment node was made. Simulations and 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 low enough to be acceptable, and that 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 re-computes 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 (using the computations described above).

The actual values for the local probability of buffer overflow $w_{n,i}$, the local delay bounds $d_{n,i,l}$, and the local parameters of the buffer space $b_{n,i}$ for the nodes n_μ on the path $p_{c,opt}$ and for the attachment node $n_{c,opt}$ are computed. The *unicast-admission-tests* compute the maximum amount of resources available at each node n_μ along $p_{c,opt}$; the *multicast-admission-tests* compute the maximum amount of resources at $n_{c,opt}$. In general the QoS requirements can be met with smaller amount 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 3.3.3.2 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 user. But the intention of storing this slack is to enable the accommodation of future joins, and not to provide a better QoS for the joining destination node. Therefore only parameters at the attachment node $n_{c,opt}$ and at nodes along the path $p_{c,opt}$ are relaxed. The computations take the following form:

d-Computation

If the network can satisfy the delay requirement of the new destination by considering the delay from the source to the attachment node and the values of $dmax_{n,i,l}$ along $p_{c,opt}$

$$D_{max,i} \geq D^*_{n_{c,opt},i} + dmax_{n_{c,opt},i,l} + \sum_{\mu=1}^{u} dmax_{n_\mu,i,l} \quad (18)$$

then the value of $dmax_{n,i,l}$ is assigned to the local delay bounds $d_{n,i,l}$ at the attachment node and all the nodes on $p_{c,opt}$.

$$dmax_{n_{c,opt},i,l} := d_{n_{c,opt},i,l} \quad (19)$$

$$dmax_{n_\mu,i,l} := d_{n_\mu,i,l} \qquad (\mu = 1,\ldots u) \quad (20)$$

If the delay requirement of the new destination cannot be met by considering the values of $dmax_{n,i,l}$, i.e.

$$D_{max,i} < D^*_{n_{c,opt},i} + dmax_{n_{c,opt},i,l} + \sum_{\mu=1}^{u} dmax_{n_\mu,i,l} \quad (21)$$

the delay bounds are computed by relaxing the minimal delay bounds that could be offered at the attachment node $n_{c,opt}$

$$d_{n_{c,opt},i,l} = dmin_{n_{c,opt},i,l} + \frac{1}{u+1}(D_{max,i} - D_{min,i}) \quad (22)$$

and at the nodes n_μ along $p_{c,opt}$.

$$d_{n_\mu,i,l} = dmin_{n_\mu,i,l} + \frac{1}{u+1}(D_{max,i} - D_{min,i}) \qquad (\mu = 1\ldots u). \quad (23)$$

w-Computation

The computation assigns the respective values to the local reliability requirements $w_{n,i}$. These parameters are needed for the computation of the buffer space (see below). If the new destination does not accept any loss of packets due to buffer overflow, i.e.

$$W_{min,i} = 1 \quad (24)$$

the network has to set these parameters at the attachment node and all nodes on $p_{c,opt}$ to 1.

$$w_{n_{c,\,opt},\,i} \;=\; 1 \tag{25}$$

$$w_{n_{\mu},\,i} \;=\; 1 \qquad\qquad (\mu = 1,\ldots u) \tag{26}$$

If the requirements of the new destination are less stringent and the destination accepts that the reliability of the connection is lower than 1, i.e.

$$W_{min,\,i} \;<\; 1 \tag{27}$$

the relaxation is distributed over the nodes n_{μ} along $p_{c,opt}$ and the attachment node $n_{c,opt}$. In this case the local parameters $w_{n,i}$ are computed as follows:

$$w_{n_c,\,i} \;=\; W_{min,\,i}^{\frac{1}{(u+1)}} \tag{28}$$

$$w_{n_{\mu},\,i} \;=\; W_{min,\,i}^{\frac{1}{(u+1)}} \qquad (\mu = 1,\ldots u) \tag{29}$$

b-Computation

We first compute $\Delta_{n,i}$, which is the maximum amount of accumulated jitter at node n that may be experienced by a packet along its way from the source to the respective node. This value is computed for the attachment node

$$\Delta_{n_{c,\,opt},\,i} \;=\; \sum_{\nu=m}^{v} (d_{n\nu,\,i,\,l} - t_{n\nu,\,i}) \tag{30}$$

where m is the position of the previous J-type server, and all nodes along the path $p_{c,opt}$. If the previous J-type server is between the attachment node and the new destination we compute $\Delta_{n,i}$ for the nodes as follows:

$$\Delta_{n,\,i} \;=\; \sum_{\mu=m}^{n} (d_{n_{\mu},\,i,\,l} - t_{n_{\mu},\,i}) \tag{31}$$

If the previous J-type server is not on the path $p_{c,opt}$ but somewhere between the source and the attachment point we compute

$$\Delta_{n,i} = \sum_{\nu=m}^{v} (d_{n\nu,i,l} - t_{n\eta,i}) + \sum_{\mu=1}^{n} (d_{n\mu,i,l} - t_{n\mu,i}). \quad (32)$$

The buffer space actually required by each node n_μ on the chosen path $p_{c,opt}$ is computed as follows:

$$b_{n\mu,i} = s_{max,i} \left\lceil \frac{d_{n\mu,i,l} + \Delta_{n\mu,i}}{x_{min,i}} w_{n\mu,i} \right\rceil \quad (33)$$

The additional amount of buffer space $b_{add_{n,i}}$ actually required by the attachment node $n_{c,opt}$ for serving the new link is

$$badd_{n,i} = s_{max,i} \left\lceil \frac{d_{n,i,l} - \max_{l}(d_{n,i,l})}{x_{min,i}} w_{n,i} \right\rceil \quad (34)$$

This is the amount of buffer space that must be assigned to channel i at the attachment node in addition to the buffer space reserved for this channel so far. Considering jitter in this computation is not required since this is done in the computation of the buffer space required for that channel at establishment time. The total amount of buffer space assigned to the channel when a new link is attached to it is the sum of the buffer space reserved for the channel so far $b_{n,i}$ and the additional buffer space $b_{add_{n,i}}$ required for the new link:

$$b_{n,i} := b_{n,i} + b_{add_{n,i}} \quad (35)$$

At each node along $p_{c,opt}$, the scheduling resources and buffer space reserved on the forward trip are relaxed to the computed values.

Let us now come back to our example in Figure 6, and let us consider the attachment for node D6. Candidate nodes for attaching D6 were the router *n*, the router at D3, and the router at D5. The result of the computations might be such that D6 is attached to channel 2 at D5 and to channel 1 at D3 (it is possible that router *n* already serving multiple outgoing links is not able to accommodate new links). Figure 10 illustrates this solution.

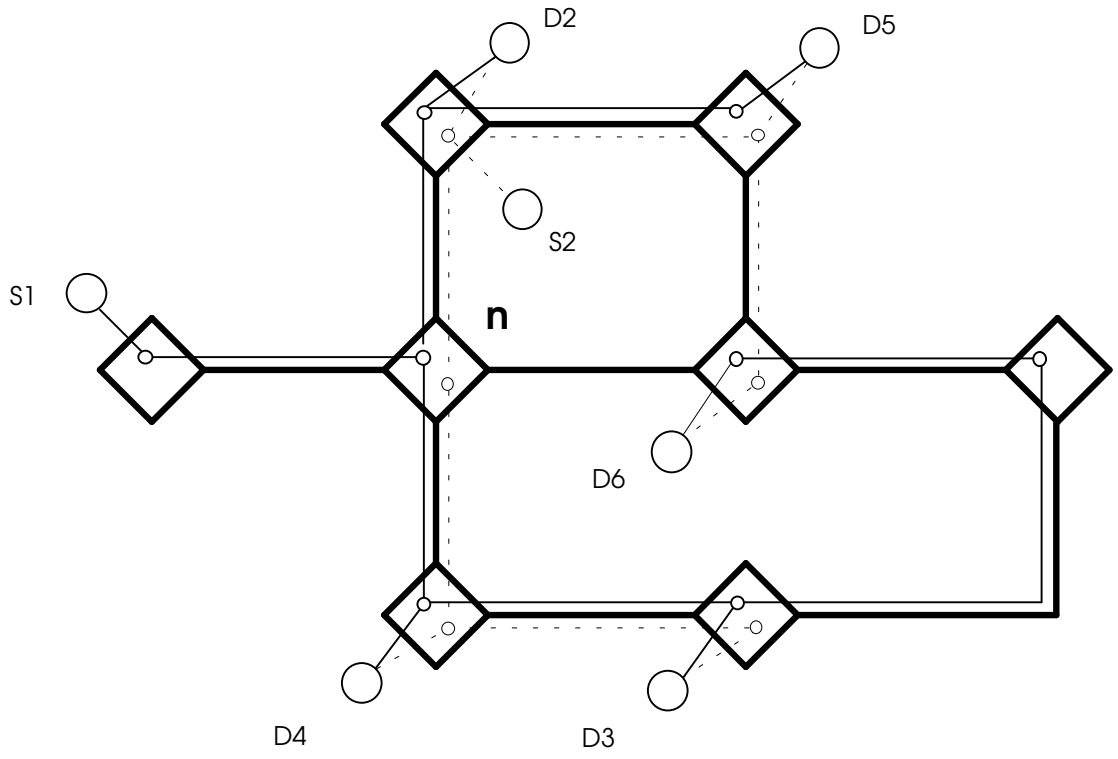

Figure 10: Scenario after D4 and D6 have joined

3.3.4 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 user always get a consistent view of the Target Set and the channels sending to it. Inconsistencies can arise when a new node joins the Target Set, and a new channel is being established at the same time. We must take precautions 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, and 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 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

(a) either all operations involved have specified the PARTIAL-OK option

(b) or at least one of the operations involved have specified ALL-OR-NOTHING and all members of the Target Set are reached by all channels after completion of all operations.

A possible implementation for this problem is obviously serialization, i.e. conflicting operations are always executed in a serial order. This could easily be implemented by locking the Target Set while a JOIN or an ESTABLISH operation is executed, thus not allowing other operations to be executed. But serialization does not scale well. In case of a large conference the problem of the Target Set being locked most of the time could arise. In order to minimize the locking time and to maximize accessibility to the Target Set let us consider the different cases described in Figure 11 showing how the different operations might coincide with each other and in what cases problems arise.

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 destinations 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 simultaneous ESTABLISH(PARTIAL-OK) and JOIN(PARTIAL-OK) operation since both operations do not expect a consistent outcome (i.e. inclusion of all destinations into all channels). Thus, problems arises only in the cases indicated by a flash in Figure 11, and the Target Set must only be locked in those specific cases.

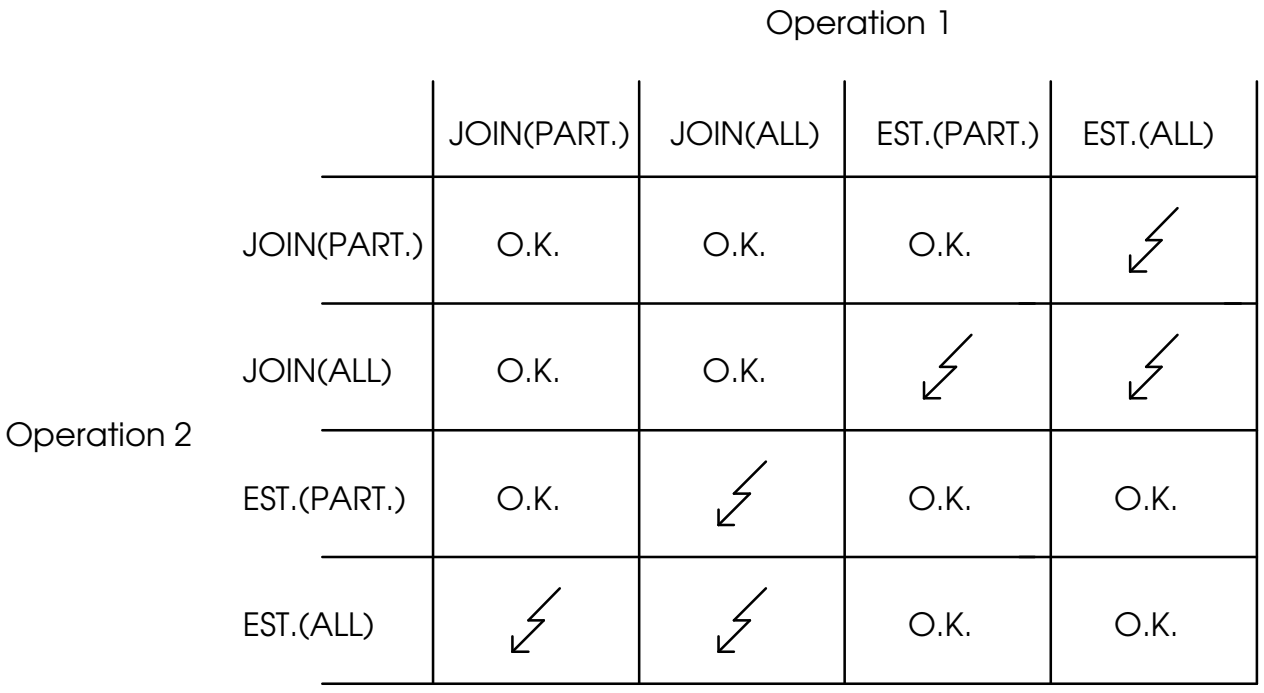

Figure 11: Conflict Matrix for JOIN and ESTABLISH Operations

Thus, we can implement concurrency control by using two different types of locks: a JOIN lock is activated by a JOIN operation and does not allow an ESTABLISH operation to be executed; 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.

In the case of a JOIN(PARTIAL-OK) call the locking time of the ESTABLISH lock can further be minimized in the following way: The very first step of the JOIN(PARTIAL-OK) operation is the inclusion of the new destination in the data base. After this step the connection of this new destination to all the channels is attempted. The ESTABLISH lock must only be active for the time it takes to include the new destination in the data base which is a short operation in comparison to connecting the new destination to the channels sending to the Target Set. Thus, even if the JOIN operation is not fully completed when ESTABLISH is called the ESTABLISH operation can be executed, and the new destination will be included in that channel. This minimization is especially useful since the collision of JOIN and ESTABLISH operations are more

likely to occur in a large scale conference in which it will be difficult anyway to be connected to all channels, i.e. where usually the PARTIAL-OK option will be chosen in a JOIN call.

But we have to consider that it only makes sense to include the new destination in the database as the first step in a JOIN operation if the PARTIAL-OK option is chosen. In the case of ALL-OR-NOTHING it is not clear if the JOIN operation succeeds until the successful connection to the last of the channels sending to the Target Set. In this case the inclusion in the database should be the last step and the Target Set should be locked for any ESTABLISH operation in order to avoid establishing a channel to a destination that will eventually not become a member of the Target Set. Thus, a JOIN(ALL-OR-NOTHING) operation must lock the Target Set for any ESTABLISH call for the whole operation. Respectively, an attempt to connect a new destination to all existing channels may miss the new channel being established if the JOIN lock is not activated.

As a general conclusion, the ALL-OR-NOTHING option will provide the user with stronger semantics than PARTIAL-OK: 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. We must support such loose semantics, i.e. a destination can be a member of a Target Set even if it is not connected to all the channels sending to the Target Set, in order to provide scalability. In a large scale conference with thousands of channels it will be extremely difficult, if not impossible, to connect a new destination node to all the channels. An application example for ALL-OR-NOTHING could be a small multimedia conference, an example for PARTIAL-OK would be an audio/video multicast of a convention or a seminar.

In addition to the global conflicts discussed above two parallel JOIN operations can collide while computing admission tests (as mentioned in 3.3.3.3). In our approach we avoid conflicts between two JOINs by re-evaluating the admission tests when the eventual attachment path has been selected (as described in 3.3.3.3). In this way we do not need to serialize the JOIN operations by making the JOIN locks incompatible in the conflict matrix which would not scale well.

3.4 The LEAVE Operation

3.4.1 The LEAVE Service Primitive

The leave operation removes a destination from a Target Set. In order to complete a leave operation two separate steps must be taken. The node is disconnected from all channels it is currently connected to and the node is removed from the data base where the Target Set is specified. The service primitive effecting the leave takes the following form:

LEAVE (NodeId,TargetSetId)

NodeId is the address of the destination leaving the Target Set specified by TargetSetId. As before we distinguish between routing nodes and host nodes. Just like the JOIN operation, the LEAVE is a layer 4 operation. It only refers to multicast nodes. Since routing nodes refer to layer 3 and have to forward packets all the time (thus, not only during the participation in the conference) they are not considered in the LEAVE operation. As mentioned in the basic design considerations a LEAVE operation always succeeds and might fail only if an invalid NodeId or TargetSetId is specified.

3.4.2 The LEAVE Algorithm

The LEAVE algorithm first disconnects from all channels sending to the specified Target Set. It then removes the node from the Target Set. As with the JOIN operation we disconnect from each channel separately, executing the same DisconnectLeavingNode algorithm. For each channel we can distinguish two cases:

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

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

Let us assume that destination node L is leaving. We use the term *leaf node* for a node with no descendants and the term *branch node* for the first upstream node that contains a branch point of the tree or a local destination. Similar to the JOIN operation p_L denotes the path from the

leaving destination to the branch node; n_i denotes the nodes on path p_l. The DisconnectLeavingNode algorithm takes the following form:

ALGORITHM DisconnectLeavingNode (n_L,ChannelId)

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

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 the concurrence problem discussed for ESTABLISH and JOIN operations 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 disconnected from all channels.

3.4.2.1 Freeing Resources at a Leaf Node

We discuss the case of a leaf node first. The subroutine *free-leaf-node-resources* executes the following steps:

- The CPU requirement of channel *i* is set to 0.
- All buffers reserved for channel *i* are released.

- The scheduling resources are released by removing the local delay bounds $d_{actn,i,l}$ and $d_{n,i,l}$ from the corresponding table of the outgoing link.

Thus all resources reserved at the node are released and the node is completely removed from the multicast channel.

3.4.2.2 Freeing Resources at the Branch Node

A branch node cannot be removed completely because it is still needed to forward data of the channel to other destinations. 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:

$$f_{n,i} := f_{n,i} - 1 \tag{14}$$

- 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 is the maximum of all local delay bounds of channel i at the branch node (l_L is the outgoing link at the branch node towards the leaving destination). Thus, the condition for a relaxation of the buffer space reserved for this channel is

$$d_{n,i,l_L} = \max_{l}(d_{n,i,l}) \tag{15}$$

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 after the disconnection of the leaving node. The exceeding buffer space $b_{exn,i}$ by which the reserved buffer space for channel i at node n can be reduced is

$$b_{ex\,n,i} := s_{max,i} \frac{d_{n,i,l_L} - \max_{l}(d_{n,i,l})}{x_{min,i}} w_{n,i} \tag{16}$$

and the required buffer space can be relaxed to

$$b_{n,i} := b_{n,i} - b_{ex\,n,i} \quad (17)$$

- The scheduling resources are released by removing the local delay bounds $d_{act_{n,i,l}}$ and $d_{n,i,l}$ from the relevant table for the outgoing link (as above).

To illustrate our LEAVE algorithm consider destination D3 in Figure 10. In this example we want to disconnect D3 from channel 1 and channel2. For channel 2 D3 is a leaf node. All resources reserved for channel 2 at D3 are freed. The branch node for channel 2 is D4 since D4 has a local receiver. Only the resources related to forwarding a packet of channel 2 on the link towards 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 is shown in Figure 12.

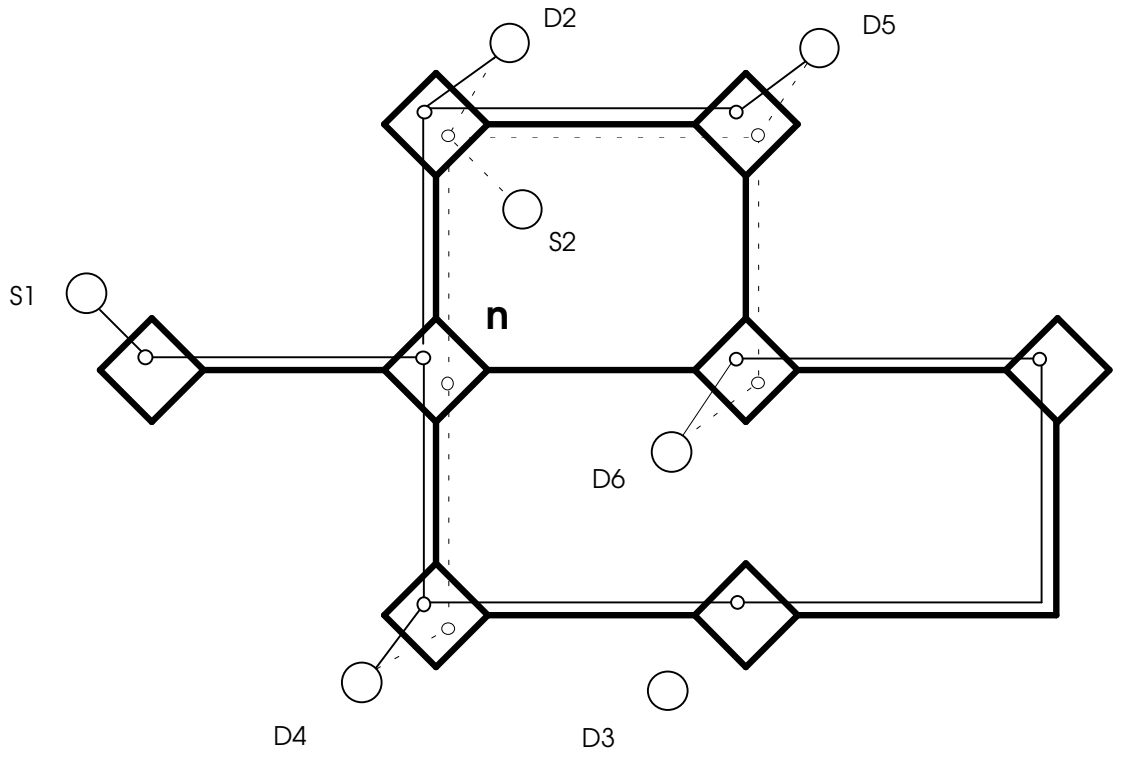

Figure 12: Scenario after D3 has left

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. After a LEAVE the topology of the multicast tree may become less efficient. For example, after the LEAVE of an intermediate node the structure of the multicast tree might not be optimal any more since this tree was only chosen to consider the destination which is now leaving. In our example D6 is still connected to channel 1 via two intermediate routers. Connecting via the router at D5 or router *n* could be more efficient for the network. In general the tree can degrade substantially after multiple LEAVE operations and an optimization by restructuring the multicast tree might be desirable. We will discuss tree reorganization issues in the following section.

3.5 Tree Reorganization

The basis for a discussion of tree reorganization issues is the definition of an optimal multicast tree. It can be defined as follows:

Consider a graph $G = (R,L)$ 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. $S \in L$ 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 $C_{n,i,l}$ associates 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 link. An *optimal multicast tree* is a tree such that

1. no router *r* in the multicast tree has a utilization $U_r > 1$

2. the cost of this tree 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 3.4.2. 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 multiple alternative subtrees and their cost functions. They are thus able to find better and more efficient 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].

The problem now being identified is when to reorganize a multicast tree. Generally we feel that a reorganization after LEAVE operations is more important than after a JOIN operation since any of the nodes in a multicast tree might leave. In the case of a JOIN operation the AttachJoiningNode algorithm presented in 3.3.3 ensures good network performance by using a best-fit solution. Simulations and actual measurements have to evaluate when a tree reorganization is actually desirable.

Note that generally a reorganization of a multicast tree is not intended in order to improve the QoS of the current destinations. These destinations still receive data with the QoS that was promised to them either at channel establishment time or when they joined a Target Set. The goal of the reorganization of a multicast tree is to improve the network performance and thus to increase the probability of being able to accommodate future JOINs.

Coming back to our example in Figure 12 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 the router of D5. This is shown in Figure 13.

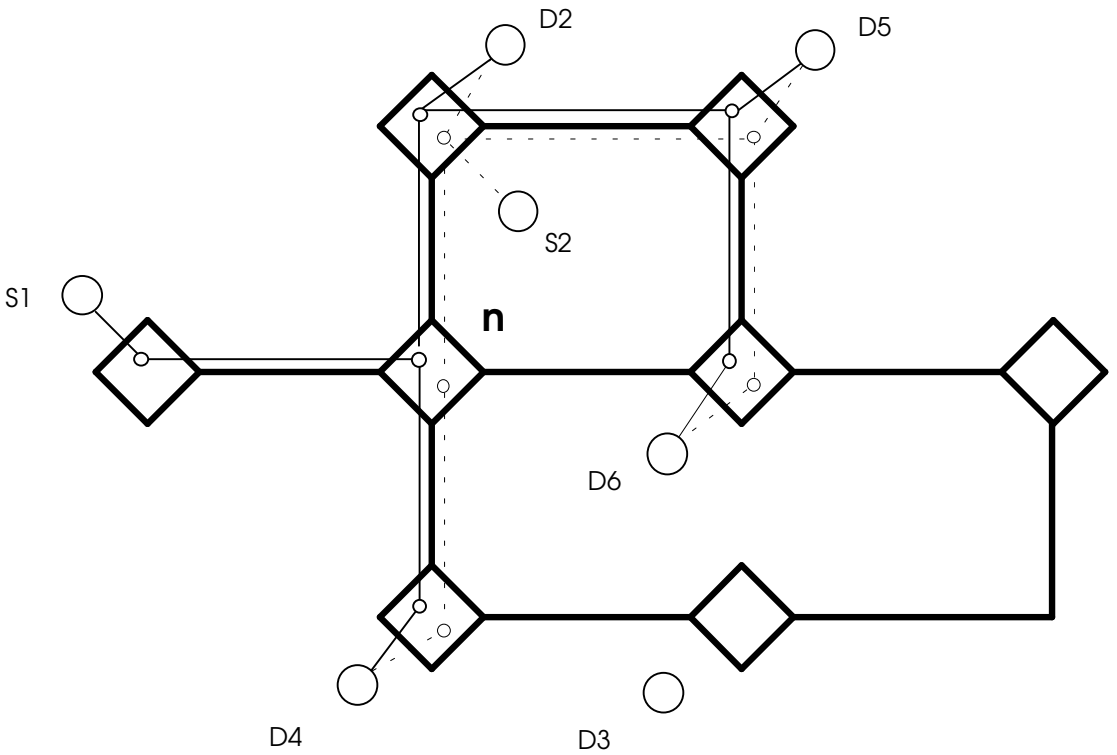

Figure 13: Scenario after Tree Reorganization

CHAPTER 4
CONCLUSION

The general background of this thesis was the increasing importance of multimedia networking. Many multimedia applications require real-time performance guarantees that must be offered by the network. The available options for providing real-time services in a wide-area context have been described, and the approach of the Tenet group to real-time communication has been explained in detail. This approach is connection-oriented and based on packet-switching internetworks.

The current research activity of the Tenet group concentrates on support for real-time multicasting. We have introduced the key features of the new Tenet scheme. As an important part of the support for multicasting the goal of this work was to enable dynamic membership of destination nodes in multicast groups. We have introduced the notion of a Target Set as a set of nodes with a common interest, and presented a model for real-time multicast communication in internetworks. The model distinguishes between the topology of links and the topology of multicast channels. Based on this model, we have presented the JOIN and LEAVE service primitives, their semantics, and algorithms implementing.

Compared to a traditional network a real-time multicast environment introduces several new challenges: finding an optimal attachment point for a joining node, extending admission tests, concurrency control in order to maintain consistency, releasing the appropriate amount of resources when a node is disconnected from a multicast tree, reorganizating a tree in order to maintain efficiency, and maintaining performance guarantees even when the tree topology changes.

We have proposed solutions to these problems, with a special emphasis on scalability. Future work will concentrate on performance evaluation of the algorithms presented in this paper.

ACKNOWLEDGEMENTS

My six-month stay in Berkeley has been an unforgettable working experience for me. I want to thank everybody who supported my studies in Berkeley, in particular

* Prof. Färber and Prof. Ferrari who offered me the great opportunity to work in the excellent research environment at the International Computer Science Institute

* Prof. Effelsberg who worked on the same project for valuable hints and excellent discussions

* all members of the Tenet group who I really enjoyed working with for many useful discussions

* Renee Reynolds for being a great help in organizational matters

Most of all I would like to thank my parents for their support throughout my studies and Sabine for accepting my decision to study abroad.

REFERENCES

[1] A. Banerjea and B. Mah: The Real-Time Channel Administration Protocol. In *Proceedings of the Second International Workshop on Network and Operating System Support for Digital Audio and Video*, Heidelberg, Germany, November 1991.

[2] D. Cheriton and S. Deering: Host Groups: A Multicast Extension to the Internet Protocol. Internet RFC 966, Stanford University, USA, December 1985.

[3] D. Comer: Internetworking with TCP/IP. Prentice Hall, Englewood Cliffs, New Jersey, USA, 1988.

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

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

[6] D. Ferrari: Client Requirements for Real-Time Communication Services. IEEE Communication Services, 28(11), USA, November 1990.

[7] D. Ferrari: Realtime Communication in an Internetwork. Technical Report TR-92-001, International Computer Science Institute, Berkeley, California, USA, January 1992.

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

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

[10] D. Ferrari and D. Verma: A Scheme for Realtime Channel Establishment in Wide-Area Networks. IEEE Journal On Selected Areas In Communications, vol.8, no. 3, USA, April 1990.

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

[12] D. Gayeski: Making Sense of Multimedia: Introduction to Special Issue. Educational Technology, pages 9-13, USA, May 1992.

[13] C. Lowery: Protocols for Providing Performance Guarantees in a Packet-Switching Internet. TR-91-002, International Computer Science Institute, Berkeley, California, USA, January 1991.

[14] B. Mah: A Mechanism for the Administration of Real-Time Channels. Technical Report UCB/CSD-93-735, University of California at Berkeley, California, USA, 1993.

[15] J. McQuillan: An Introduction to Multimedia Networking. Business Communications Review, pages 74-79, USA, November 1991.

[16] J. McQuillan: Multimedia Networking: An Applications Portfolio. Data Communications International, pages 85-94, USA, September 1992.

[17] C. Paris, 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, USA, 1993.

[18] J. Ramaekers and G. Ventre: Quality-of-Service Negotiation in a Real-Time Communication Network. Technical Report TR-92-023, International Computer Science Institute, Berkeley California, USA, 1993.

[19] C. Szyperski and G. Ventre: A Characterization of Multi-Party Interactive Multimedia Application. Technical Report TR-93-006, International Computer Science Institute, Berkeley California, USA, February 1993.

[20] A. Tanenbaum: Computer Networks. Second Edition, Prentice Hall, Englewood Cliffs, New Jersey, USA, 1988.

[21] C. Topolcic: Experimental Internet Stream Protocol, Version 2 (ST-II). Internet RFC 1190, CIP Working Group, USA, October 1990.

[22] B. Wolfinger and M. Moran: A Continuos Media Data Transport Service and Protocol for Realtime Communication in High Speed Networks. In *Proceedings of the Second International Workshop on Network and Operating System Support for Digital Audio and Video*, Heidelberg, Germany, November 1991.

[23] H. Zhang and D. Ferrari: Rate-Controlled Static-Priority Queuing. Technical Report TR-92-003, International Computer Science Institute, Berkeley, California, USA, February 1992.

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

[25] H. Zhang, D. Verma, and D. Ferrari: Design and Implementation of the Real-Time Internet Protocol. In IEEE Workshop on the Architecture and Implementation of High Performance Communication Subsystems, Tucson, Arizona, USA, February 1992.

APPENDIX

Admission Tests for Channel Establishment in Scheme 1

Name	Condition	Computation	Test	Record	Forward
Deterministic Test	$z_{min,i}=1$	$U_{det,n} := U_{det,n} + \frac{t_{n,i}}{x_{min,i}}$ where $U_{det,n}$ is initialized to 0	$U_{det,n} < 1$	$U_{det,n}$	-
Delay Bound Computation	-	Let $d_{n,1} \le d_{n,2} \le \ldots \le d_{n,c_n}$ for $k = 1,2,\ldots,c_n$ if $d_{n,h} - \sum_{l=1}^{h} t_{n,l} - T_n - t_{n,c+1} \ge 0$ for $h = k,k+1,\ldots,cn$ then $dmin_{n,i} = \sum_{l=1}^{k} t_{n,l} + T_n + t_{n,c_n+1}$	-	$dmin_{n,i}$	$dmin_{n,i}$
Buffer Space Test	-	$bmax_{n,i} := B$ where $B \le B_n - \sum_{k=1}^{c_n} b_{n,k}$ $b_{n,i} := bmax_{n,i}$ $dmax_{n,i} = \frac{bmax_{n,i} \cdot x_{min,i}}{s_{max,i}} - \sum_{k=m}^{n-1} (dmax_{k,i} - t_{k,i})$ (i) if $J_{max,i}$ is undefined: m=1 (ii) if $J_{max,i}$ is defined: m = position of the previous J-type server	$dmax_{n,i} \ge dmin_{n,i}$	$b_{n,i}$ $dmax_{n,i}$	$dmax_{n,i}$

Table A.1: Server Tests for Admission Control

Name	Computation	Test	Forward
D-Test	$D_{prop,\,i} = \sum_{n=1}^{N} dprop_{n,\,i}$ $D_{min,\,i} = \sum_{n=1}^{N} dmin_{n,\,i}$	$D_{min,\,i} \le D_{max,\,i} - D_{prop,\,i}$	-
w-computation	if $W_{min,i} < 1$: $w_{n,\,i} := W_{min,\,i}^{1/N}$ $(n=1,...N)$ if $W_{min,i} = 1$: $w_{n,\,i} := 1$ $(n=1,...N)$	-	$w_{n,i}$ $(n=1,...N)$
d-computation	if $\sum_{n=1}^{N} dmax_{n,\,i} \le D_{max,\,i}$ $D'_{max,\,i} = \sum_{n=1}^{N} dmax_{n,\,i}$ $d_{n,\,i} := dmax_{n,\,i}$ $(n=1,...N)$ if $\sum_{n=1}^{N} dmax_{n,\,i} > D_{max,\,i}$ $D'_{max,\,i} := D_{max,\,i}$ $d_{n,\,i} = dmin_{n,\,i} + \frac{1}{N}(D_{max,\,i} - D_{min,\,i})$ $(n=1,...N)$	-	$d_{n,i}$ $(n=1,...N)$ $D'_{max,i}$ $\Delta_{n,\,i} = \sum_{k=m}^{n} (d_{k,\,i} - t_{k,\,i})$ (i) if $J_{max,i}$ is undefined: m=1 (ii) if $J_{max,i}$ is defined: m = position of the previous J-type server
J-Test	if $J_{max,i}$ defined: $J_{N,\,i} = \sum_{k=m}^{N} (d_{k,\,i} - t_{k,\,i})$ $Dprop_{n,\,i} = \sum_{k=1}^{n} dprop_k$ $(n=1,...N)$ $D^*_{n,\,i} = D_{prop_{n,\,i}} + \sum_{k=1}^{n-1} d_{k,\,i}$ $(n=1,...N)$	$J_{N,\,i} \le J_{max,\,i}$	$D^*_{n,i}$ $(n=1,...N)$
b-computation	$b_{n,\,i} = s_{max,\,i} \left\lceil \frac{d_{n,i} + \Delta_{n,i}}{x_{min,i}} w_{n,i} \right\rceil$ $(n=1,...N)$	-	$b_{n,i}$ $(n=1,...N)$

Table A.2: Destination Tests for Admission Control