Doctoral Thesis

A multipoint communication architecture for end-to-end quality of service guarantees

Author(s):
Bauer, Daniel

Publication Date:
1997

Permanent Link:
https://doi.org/10.3929/ethz-a-001823181

Rights / License:
In Copyright - Non-Commercial Use Permitted

This page was generated automatically upon download from the ETH Zurich Research Collection. For more information please consult the Terms of use.
A MULTIPONT COMMUNICATION ARCHITECTURE FOR END-TO-END QUALITY OF SERVICE GUARANTEES

A dissertation submitted to the

SWISS FEDERAL INSTITUTE OF TECHNOLOGY ZÜRICH

for the degree of
Doctor of Technical Sciences

presented by

DANIEL BAUER

Dipl. Informatik-Ing. ETH
born Februar 13, 1968
citizen of Oberbüren (SG)

accepted on the recommendation of
Prof. Dr. Bernhard Plattner, examiner
Prof. Dr. Thomas Plagemann, co-examiner

1997
Berichte aus der Kommunikationstechnik

Daniel Bauer

A Multipoint Communication Architecture for End-to-End Quality of Service Guarantees

Shaker Verlag
Aachen 1997
Bauer, Daniel:
A Multipoint Communication Architecture for End-to-End Quality of Service Guarantees / Daniel Bauer. - Als Ms. gedr. -
Aachen: Shaker, 1997
(Berichte aus der Kommunikationstechnik)
Zugl.: Zürich, ETH, Diss., 1997
ISBN 3-8265-2638-4

Copyright Shaker Verlag 1997
Alle Rechte, auch das des auszugsweisen Nachdruckes, der auszugsweisen
oder vollständigen Wiedergabe, der Speicherung in Datenverarbeitungs-
anlagen und der Übersetzung, vorbehalten.

Als Manuskript gedruckt. Printed in Germany.

ISBN 3-8265-2638-4
ISSN 0945-0823

Shaker Verlag GmbH • Postfach 1290 • 52013 Aachen
Telefon: 02407 / 95 96 - 0 • Telefax: 02407 / 95 96 - 9
Internet: www.shaker.de • eMail: info@shaker.de
The developments in information technology of the last years have led to major advances in high-speed networking, multimedia capabilities for workstations and also distributed multimedia applications. In particular, multimedia applications for computer supported cooperative work have been developed that allow groups of people to exchange information and to collaborate and cooperate joint work. However, existing communication systems do not provide end-to-end guarantees for multipoint communication services which are needed by these applications.

In this thesis, a communication architecture is described that offers end-to-end performance guarantees in conjunction with flexible multipoint communication services. The architecture is implemented in the Multipoint Communication Framework (MCF) that extends the basic communication services of existing operating systems. It orchestrates endsystem and network resources in order to provide end-to-end performance guarantees. Furthermore, it provides multipoint communication services where participants dynamically join and leave.

The communication services are implemented by protocol stacks which form a three layer hierarchy. The topmost layer is called multimedia support layer. It accesses the endsystem's multimedia devices. The transport layer implements end-to-end protocol functions that are used to forward multimedia data. The lowest layer is labelled multicast adaptation layer. It interfaces to various networks and provides a multipoint-to-multipoint communication service that is used by the transport layer. Each layer contains a set of modules that implement a single protocol function. Protocol stacks are dynamically composed out of modules. Each protocol uses a single module on each layer.

Applications specify their service requirements as Quality of Service (QoS) parameters. MCF maps these QoS parameters to the above mentioned layers, where they are used to calculate the needed resources. A resource manager
reserves memory, CPU and multimedia devices in the endsystem. Access to the CPU is provided by a real-time scheduler for periodic tasks, which executes the protocols. The reservation of network resources is delegated to the network resource manager. MCF orchestrates endsystem and network resources in order to provide a guaranteed service covering the whole path from multimedia device to multimedia device.

The evaluation of MCF shows that the proposed architecture results in an easy to use and efficient solution. The dynamic composition of protocol stacks offers high flexibility and allows applications to transport any multimedia data over any network. Resource reservations provide the performance guarantees needed for continuous media such as audio or video.

In der vorliegenden Dissertation wird eine Kommunikationsarchitektur vorgestellt, welche Laufzeitgarantien für Mehrpunktkommunikationsdienste zur Verfügung stellt. Diese Architektur wurde im Rahmen des 'Multipoint Communication Framework' (MCF) implementiert. Das Rahmenwerk erweitert die grundlegenden Kommunikationsdienste existierender Betriebssysteme, es verwaltet Endsystem- und Netzwerkressourcen um Laufzeitgarantien geben zu können. Die angebotenen Kommunikationsdienste sind mehrpunktfähig, wobei sich die Dienstbenutzergruppe dynamisch ändern kann, neue Teilnehmer können hinzukommen und bestehende können sich entfernen.


# Table of Contents

Abstract iii

Kurzfassung v

Table of Contents vii

List of Figures xi

1. Introduction 1
   1.1 Motivation ................................................. 1
   1.2 Problem Statement .................................... 4
   1.3 Outline ................................................ 7

2. Multicasting - An Overview 9
   2.1 Terminology ............................................. 10
   2.2 End-to-End Multicast Connectivity ....................... 11
      2.2.1 Multicasting in Shared Medium LANs ............. 12
      2.2.2 Multicasting in Datagram Networks ............... 13
      2.2.3 Multicasting in Fast Packet Switching Networks .. 19
   2.3 Multicast Transport Protocol Principles ................. 21
      2.3.1 Multicast Association Management .................. 21
      2.3.2 Multicast Error Control ............................ 21
      2.3.3 Multicast Flow Control ............................. 25
      2.3.4 Multicast Congestion Control ....................... 26
      2.3.5 Multicast Ordering .................................. 27
   2.4 Summary .................................................. 32
### Table of Contents

3. **Quality of Service in Communication Systems** 33
   3.1 Terminology .......................................................... 34
   3.2 QoS and Resource Concept ......................................... 35
     3.2.1 Application Requirement Specification ...................... 36
     3.2.2 QoS Parameter Mapping and Resource Calculation ....... 37
     3.2.3 Resource Management ........................................... 40
     3.2.4 QoS Negotiation .................................................. 42
     3.2.5 QoS Re-negotiation .............................................. 44
   3.3 Summary .............................................................. 44

4. **Related Work in Multipoint, Multimedia Communication** 47
   4.1 Multicast Network and Transport Protocols .................... 48
     4.1.1 Internet Stream Protocol (ST-2+) .......................... 48
     4.1.2 Tenet Protocol Suite 2 ........................................ 50
     4.1.3 Integrated Services in the Internet
           Architecture and RSVP ........................................ 52
     4.1.5 Multicast Extensions of the OSI95
           QoS Negotiation Scheme ....................................... 56
   4.2 Communication Frameworks ....................................... 58
     4.2.1 QoS-Architecture .............................................. 59
     4.2.2 Omega .......................................................... 61
     4.2.3 Da CaPo++ .................................................... 63
     4.2.4 QoS Framework ................................................ 65
   4.3 Comparison and Conclusion of the Related Work ............. 66

5. **Multipoint Communication Framework Design** 69
   5.1 Concepts ..................................................................... 70
     5.1.1 Environment ...................................................... 71
     5.1.2 Multipoint Communication ..................................... 71
     5.1.3 Quality of Service Concept ................................... 73
   5.2 Data Transport in MCF ............................................. 77
     5.2.1 Protocol Layering ............................................... 77
     5.2.2 Protocol Composition .......................................... 79
     5.2.3 Data Paths and Module Interface ............................ 81
     5.2.4 Protocol Execution ............................................. 84
   5.3 QoS and Resource Administration in MCF ....................... 95
     5.3.1 Application Requirement Specification ....................... 95
| 5.3.2 QoS Parameter Mapping and Resource Calculation | 97 |
| 5.3.3 Admission Control and Resource Reservation | 104 |
| 5.3.4 Local QoS Downgrading for Heterogeneous Receivers | 106 |
| 5.3.5 QoS Re-negotiation | 108 |
| 5.4 Summary | 111 |

6. **MCF Implementation** 113

6.1 Implementation Overview 114

6.1.1 Component Overview 115

6.1.2 MCF Tasks 117

6.2 Application Programming Interface (API) 119

6.2.1 API Services 119

6.2.2 API Protocol 122

6.3 Transport Subsystem 122

6.3.1 The Rate-Monotonic Scheduler 122

6.3.2 Timer Management 126

6.3.3 Buffer Management 126

6.4 Control and Management Subsystem 128

6.4.1 Join Session 128

6.4.2 QoS Re-negotiation 133

6.5 Multimedia Support Modules 138

6.5.1 Audio Modules 139

6.5.2 Calculation of the Playback Point 141

6.6 Transport Modules 144

6.6.1 Segmentation and Reassembly 144

6.6.2 Multicast Error Control 146

6.7 Multicast Adaptation Modules 148

6.8 Critical Assessment and Summary 151

6.8.1 Shortcomings of the ATM Implementation 151

6.8.2 GMS Integration 152

6.8.3 Support for Heterogeneous Receivers 153

6.8.4 Summary 153

7. **Evaluation** 155

7.1 Usability as a Communication Framework 156

7.1.1 Supported Application Classes 156

7.1.2 Audio Conference Application 157

7.1.3 Perceived Quality of MCF's Services 160
Table of Contents

7.2 Performance Measurements ........................................ 163
  7.2.1 Scheduling Overhead ........................................ 164
  7.2.2 Protocol Execution Times .................................. 166
  7.2.3 QoS Re-negotiation Performance ......................... 167

7.3 Scalability ..................................................... 171
  7.3.1 Scalability in the Number of Receivers ................. 171
  7.3.2 Scalability in the Number of Senders .................. 172
  7.3.3 QoS Re-negotiation and Scalability ..................... 173

7.4 Critical Assessment ............................................ 174

8. Conclusions and Outlook ........................................ 177
  8.1 Review of Claims ............................................. 178
  8.2 Suggestions for Future Work ................................ 182
    8.2.1 Improvement of MCF ..................................... 182
    8.2.2 Future Research Topics ................................ 183

References .......................................................... 185
Acknowledgements .................................................... 197
Curriculum Vitae ..................................................... 198
<table>
<thead>
<tr>
<th>Figure Number</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Communication framework architecture</td>
<td>4</td>
</tr>
<tr>
<td>2.1</td>
<td>Multicast protocol stack</td>
<td>10</td>
</tr>
<tr>
<td>2.2</td>
<td>Multipoint-to-multipoint multicasting in LANs</td>
<td>12</td>
</tr>
<tr>
<td>2.3</td>
<td>Three types of delivery trees [Deering 91]</td>
<td>15</td>
</tr>
<tr>
<td>2.4</td>
<td>Point-to-multipoint multicasting in ATM</td>
<td>20</td>
</tr>
<tr>
<td>2.5</td>
<td>Hierarchical error control scheme</td>
<td>23</td>
</tr>
<tr>
<td>2.6</td>
<td>Multicast model [Hadzilacos et al. 94]</td>
<td>27</td>
</tr>
<tr>
<td>2.7</td>
<td>Layering model [Hadzilacos et al. 94]</td>
<td>28</td>
</tr>
<tr>
<td>2.8</td>
<td>Implementation of causal and total order</td>
<td>31</td>
</tr>
<tr>
<td>2.9</td>
<td>Ordering relationships</td>
<td>31</td>
</tr>
<tr>
<td>3.1</td>
<td>QoS and resource concept</td>
<td>35</td>
</tr>
<tr>
<td>3.2</td>
<td>QoS mapping and resource calculation</td>
<td>39</td>
</tr>
<tr>
<td>3.3</td>
<td>Example of a QoS mapping function</td>
<td>40</td>
</tr>
<tr>
<td>3.4</td>
<td>Negotiation situations</td>
<td>43</td>
</tr>
<tr>
<td>4.1</td>
<td>Tenet and Internet protocol stacks [Banerjea et al. 94]</td>
<td>51</td>
</tr>
<tr>
<td>4.2</td>
<td>Three-way handshaking [Mathy et al. 94]</td>
<td>57</td>
</tr>
<tr>
<td>4.3</td>
<td>Incompatibility in selected values [Mathy et al. 94]</td>
<td>58</td>
</tr>
<tr>
<td>4.4</td>
<td>QoS-Architecture [Campbell et al. 94]</td>
<td>60</td>
</tr>
<tr>
<td>5.1</td>
<td>MCF environment</td>
<td>72</td>
</tr>
<tr>
<td>5.2</td>
<td>The GMS architecture [Wilde et al. 96]</td>
<td>73</td>
</tr>
<tr>
<td>5.3</td>
<td>Multimedia- and control data in MCF</td>
<td>74</td>
</tr>
<tr>
<td>5.4</td>
<td>MCF protocol stacks</td>
<td>78</td>
</tr>
<tr>
<td>5.5</td>
<td>Example of a composed protocol stack</td>
<td>80</td>
</tr>
<tr>
<td>5.6</td>
<td>Main and back path in flows</td>
<td>82</td>
</tr>
<tr>
<td>5.7</td>
<td>Module interface</td>
<td>83</td>
</tr>
<tr>
<td>5.8</td>
<td>TSS Runtime Environment [Keller 96]</td>
<td>84</td>
</tr>
<tr>
<td>5.9</td>
<td>Task scheduling</td>
<td>85</td>
</tr>
<tr>
<td>5.10</td>
<td>Packet scheduling</td>
<td>86</td>
</tr>
<tr>
<td>5.11</td>
<td>Packet and task scheduling</td>
<td>87</td>
</tr>
<tr>
<td>5.12</td>
<td>Task model [Leung et al. 80]</td>
<td>88</td>
</tr>
<tr>
<td>5.13</td>
<td>MMS module as leaky bucket</td>
<td>89</td>
</tr>
<tr>
<td>5.14</td>
<td>Admission test for non-pre-emptive RM [Keller 96]</td>
<td>91</td>
</tr>
<tr>
<td>5.15</td>
<td>Assignment of scheduling priorities</td>
<td>92</td>
</tr>
<tr>
<td>5.16</td>
<td>Module precedence graph for flow sending on back path</td>
<td>93</td>
</tr>
</tbody>
</table>
Figure 5.17: QoS parameter space ..................................................96
Figure 5.18: QoS mapping and resource calculation .........................98
Figure 5.19: Splitting of the end-to-end delay ................................100
Figure 5.20: Example profile database entry ...................................103
Figure 5.21: Filters for local QoS downgrading ...............................107
Figure 6.1: Implementation environment .......................................114
Figure 6.2: Component interactions ..............................................117
Figure 6.3: Scheduling classes in SVR4 UNIX® ...............................118
Figure 6.4: Example priorities of MCF tasks .................................119
Figure 6.5: Session and flow data structures .................................120
Figure 6.6: API implementation ..................................................122
Figure 6.7: Scheduler’s event queue [Keller 96] ...............................123
Figure 6.8: Considering the UNIX® dispatch latency [Keller 96] ........125
Figure 6.9: Scheduling of modules ...............................................125
Figure 6.10: Buffer management ..................................................127
Figure 6.11: Flow data structure .................................................128
Figure 6.12: Module data structure ..............................................129
Figure 6.13: Control protocol stack .............................................133
Figure 6.14: QoS parameter specification for backward mapping ....136
Figure 6.15: End-to-end delay for audio flows ..............................140
Figure 6.16: Audio frame queuing and mixing ...............................142
Figure 6.17: Calculation of the playback point .............................144
Figure 6.18: Reassembly state machine .......................................146
Figure 6.19: Round-trip time calculation [Bauer et al. 96] ..............148
Figure 6.20: ATM multicast conversion .......................................149
Figure 7.1: Audio conference application ....................................158
Figure 7.2: Components on the end-to-end path ................................161
Figure 7.3: Dispatch latency and scheduling overhead ..................165
Figure 7.4: QoS backward mapping ...........................................168
Figure 7.5: QoS switching execution time distribution .....................169
Figure 7.6: Perceived QoS re-negotiation time .............................170
Chapter 1

Introduction

1.1 Motivation

Computers have been developed for the processing of digital information. Today, digital information can represent not only text, numbers and graphics, but also video and audio. The integration of these multiple media into a single, multimedia workstation allows for a more vivid presentation of information to the user than ever possible before. In conjunction with computer networks, multimedia workstations opened a new field of applications. In particular, multimedia applications for Computer Supported Cooperative Work (CSCW) gained a lot of interest. CSCW systems enable groups of people to exchange information, to coordinate tasks as well as to collaborate and cooperate joint work [Lubich 95]. Networked workstations may not only be used to support working groups, some visionaries such as Nicholas Negro-
ponte anticipate computers to become the ubiquitous communication tools that eventually integrate telephones as well as TV sets into a single unit [Negroponte 95].

Research in the area of CSCW resulted in several experimental systems. The best known example is the Internet Multicast Backbone application suite (MBONE applications). It consists of a set of tools that are used to distribute audio, video and graphical data (e.g. slides) of meetings and conferences to a world-wide audience [Casner et al. 92]. The tools send data over the multicast backbone of the internet, hence the name MBONE application suite. In the JVTOS [Dermler et al. 93] and the MMC [Altenhofen et al. 93] project, conferencing systems have been developed that allow a user to attend virtual meetings using his workstation. These systems emulate face-to-face meetings. Participants interact using audio and video, each participant sees and hears his partners. Furthermore, the integration on a workstation allows a participant to access and distribute any information stored on his workstation to his fellow participants. Such conferencing systems aim at reducing travelling costs and allow for more spontaneous meetings and thus increase workgroup efficiency [Lubich 95].

The above mentioned CSCW systems are examples of multiparty, multimedia applications, since a whole group of participants use multimedia data, namely texts, graphics and in particular audio and video, to communicate. The experience gained in using JVTOS showed that a satisfying quality for audio and video can only be achieved if the underlying endsystems and networks fulfil a set of prerequisites.

Audio and video are referred to as continuous media or real-time media, since data continuously change over time. Real-time media are represented as streams of periodic data units. In multiparty, multimedia applications, real-time media are generated in a workstation - or endsystem - and conveyed over the network to a set of destination endsystems. In the destination endsystems, the data units must be presented continuously to the user. For example, an audio stream that has been generated using 44100 samples per second must also be played back at a rate of 44100. The rate can only be maintained, if the involved endsystems as well as the network offer a throughput guarantee [Anderson 93]. For interactive communication used in virtual face-to-face meetings, guaranteed throughput alone is not sufficient.
Interactivity is only possible if the delay experienced by the transferred data does not exceed a certain limit [Nahrstedt et al. 95 A]. The processing of real-time media thus asks for end-to-end performance guarantees with respect to delay and throughput. Throughput and delay limits depend on the type of media (e.g. audio or video) that is transported as well as on its quality. Both of these parameters are part of the application requirements used by the application to specify the behaviour of the communication.

Simultaneous communication with several partners requires that data can be sent to groups of receivers. Consequently, endsystems and networks have to support multipoint communication. Since applications may distribute data to large audiences, it is mandatory that multipoint communication is efficient even for large numbers of receivers.

Today’s state-of-the-art networks offer basic support for the transportation of real-time media. A user of an ATM network may request a certain Quality of Service (QoS) for his transport connections. QoS in ATM consists of a throughput guarantee which can be requested by the user as well as an upper limit for the delay. At the same time, ATM networks offer point-to-multipoint multicasting, where data is sent to multiple receivers in a single operation.

In contrast to modern networks, endsystem support for multiparty multimedia application is still insufficient. Recently, communication frameworks have been developed that support multimedia communication among two partners [Gopalakrishnan et al. 95 A], [Nahrstedt 95 C]. These frameworks integrate communication protocols and multimedia presentation into a single system, as depicted in Figure 1.1. By using real-time schedulers, a minimum throughput and a maximum delay in the endsystems is guaranteed. In combination with network QoS guarantees, these frameworks provide a transport service with end-to-end guarantees ranging from multimedia device to multimedia device. Thus, they extend the notion of quality of service to comprise endsystems as well as network guarantees. However, these communication frameworks do not support multipoint-to-multipoint applications, their basic target is support for point-to-point multimedia applications, only.
1.2 Problem Statement

In order to provide endsystem support for multipoint multimedia applications, a new approach is required. The reason is that multiparty applications exhibit fundamental differences compared to traditional point-to-point applications:

- Multiparty applications allow groups of participants to communicate. The group is subject to change, new participants may join and existing ones leave, without interrupting the communication for the rest of the group. This behaviour will be denoted as multipoint dynamics.

- Some multiparty applications are used to disseminate data to large audiences. These applications require that the communication is scalable with respect to the number of receivers. One aspect of scalability is that the time needed for sending data is independent of the number of receivers. In general, the resources used by the sender such as storage or bandwidth must not be affected by the number of receivers.

- The set of participants in a multiparty application is expected to be heterogeneous, with differing endsystems and network accesses. For example, endsystems may guarantee different throughput. As a result, not all endsystems are able to process the same amount of data. For multimedia data such as video, less throughput means less quality. However,
it is preferable that each receiver obtains the best quality that his endsystem is able to support.

Existing communication frameworks do not support multiparty applications. Their design is based on assumptions concerning point-to-point applications:

- Before any data is being exchanged, the two parties agree on the end-to-end guarantees. This is done by negotiating the quality of service. The negotiation is usually steered by the party that initiated the communication. After the partners have agreed, the communication is set up and data is transmitted from the sender to the receiver.

  In order to support multicast dynamics, the design of a framework must be able to accommodate for joining and leaving participants while data are being exchanged. Since any participant may leave the communication, no participant can be used for steering the negotiation, as it is done in point-to-point frameworks.

- In point-to-point applications, data are sent to a single receiver only. Heterogeneity, therefore, does not exist. Consequently, point-to-point frameworks offer no provisions for heterogeneous receivers.

- The algorithms and mechanisms used in point-to-point frameworks were designed to accommodate exactly one sender and one receiver. The communication protocols used, for instance, store state information for each receiver and are thus not scalable.

This thesis addresses the design of a Multipoint Communication Framework (MCF) that offers a multipoint-to-multipoint transport service with end-to-end performance guarantees as needed by multiparty, multimedia applications. It is claimed that the design fulfils the requirements of these applications which consists of the following issues:

- **Multipoint-to-Multipoint Communication**
  MCF offers multipoint-to-multipoint communication, where several senders distribute data to a group of several receivers.

- **Multipoint Dynamics**
  The design supports multipoint dynamics, i.e. dynamic join and leave of participants during the communication.
- **End-to-End Performance Guarantees**
  MCF provides performance guarantees for multimedia data. This means, for example, that a constant frame rate for video transmission can be maintained.

- **Scalability**
  MCF supports applications that disseminate data to a large number, i.e. hundreds, of receivers.

- **Transparency**
  Applications specify their requirements in a transparent way. Applications do not have to know any mechanisms or algorithms used in the framework. They need to specify the characteristics of the media, only. The quality for a video streams, for example, is specified by the frame rate and the picture quality, but not the throughput. Similarly, the application is not aware of how multicasting is done in the network. The framework offers an abstraction for multipoint communication that is mapped to the underlying network.

- **Adaptable Media Quality**
  Applications may change the quality of the real-time media during the communication without having to stop and restart the communication.

- **Heterogeneity**
  The frameworks allows for heterogeneous receivers. Receivers that are unable to process the offered amount of data may reduce the volume, albeit at the expense of a degraded quality. Likewise, some receivers may tolerate a larger delay than specified by the application.

- **Flexibility and Extensibility**
  Applications may distribute any real-time media over any network that is supported by MCF. The composition of media type and network is done at run time, such that applications don't have to be adapted. Furthermore, MCF can be extended to support new media as well as new networks without affecting existing applications.
1.3 Outline

The foundations of MCF are multipoint communication and performance guarantees for real-time media. Both topics have been active research areas for the last years, although independently of each other. Consequently, the two topics are introduced separately. Chapter 2 introduces multicast networking. It gives an overview of multicasting by examining each level of a multicast communication stack. Chapter 3 presents the essentials of QoS in multimedia communication systems. It describes the basic methods needed for providing guaranteed QoS, void of any current implementation.

Related work is presented in Chapter 4. It describes communication protocols and frameworks that at least partially support multipoint, multimedia communication.

The design of the developed multimedia communication framework is described in Chapter 5. Chapter 6 describes a prototype implementation of the design. The evaluation of the design as well as performance measurements of the prototype follow in Chapter 7. Chapter 8 draws the conclusions of the work.
Chapter 2

Multicasting - An Overview

Multicast communication is a core technology that is both used and provided by MCF. This chapter gives an overview of multicasting. Multicasting is introduced following the layers of a simplified protocol stack shown in Figure 2.1. The lower half of the protocol stack provides end-to-end multicast connectivity. Two approaches are presented, namely datagram networks based on shared medium LANs as well as fast packet switching networks. MCF itself is based on end-to-end multicast connectivity, which is enriched with control and ordering mechanisms.

The chapter is organized as follows: the first section introduces the used terminology. The following section then describes multicasting in shared medium LANs. Datagram networks and in particular multicast routing is presented next. Fast packet switching is introduced using the Asynchronous Transfer Mode (ATM) as an example. Multicast error control is characterized next, together with aspects of multicast flow and congestion control. A more theoretical overview of multicast ordering principles concludes the chapter.
2.1 Terminology

The terminology used in conjunction with multicasting is not standardized. In this chapter and throughout the thesis, the following terminology will be used:

**Definition 2.1:** *Multipoint communication* refers to communication where data is exchanged simultaneously among a set of parties. A *multipoint application* is an application which is used by several participants to exchange data simultaneously.

Multipoint communication is an abstract term which is completely free of any performance aspects. Multicasting, in contrast, is associated with efficiency. Multipoint communication can be implemented using several unicasts or a single multicast. Of course, the use of multicasting leads to more efficient, less resource consuming systems.

**Definition 2.2:** A *unicast* is the action of transmitting data to a single receiver.

**Definition 2.3:** A *broadcast* is the action of transmitting data to all receivers using a single, atomic operation. The term ‘all’ may only comprises all receivers attached to a single network or subnetwork.
2.2 End-to-End Multicast Connectivity

**Definition 2.4**: A multicast is the action of transmitting data to a set of receivers using a single, atomic operation. Multicast communication or short multicasting describes the exchange of data using multicast operations.

**Definition 2.5**: A multicast group is the set of receivers that is reached by a multicast. The multicast group forms a single logic entity. Receivers may be part of several multicast groups simultaneously.

Multicasting itself comes in two flavours:

**Definition 2.6**: In point-to-multipoint multicasting, a single sender distributes data to a multicast group. In multipoint-to-multipoint multicasting, several senders send data to a multicast group. Membership changes in the multicast group affect all senders. In both cases, the sender(s) may be receiver at the same time.

In multicasting, the term "connection" is misleading, since in the multipoint-to-multipoint case, no single connection can be identified. Therefore, the term multicast association is introduced:

**Definition 2.7**: A multicast association is the state where one or several senders and receivers are ready to exchange data.

The members of a multicast association may change dynamically. Two operations are defined for changing memberships of a multicast association:

**Definition 2.8**: The join operation is used by senders and receivers to become part of a multicast association. In point-to-multipoint multicasting, the join operation for receivers may be initiated by the sender. This operation, were the receiver passively waits until the sender triggers the join operation, is called sender-initiated join. The receiver-initiated join is initiated by the receiver that wants to become a member of the multicast association. The leave operation is the opposite of join, it is used to leave a multicast association.

### 2.2 End-to-End Multicast Connectivity

This section describes two means for providing end-to-end multicast connectivity. The first section introduces multicasting in shared medium LANs. End-to-end multicasting is provided by datagram networks that interconnect these shared medium LANs, as described in the Section 2.2.2. The second
approach that provides end-to-end multicasting are fast packet switched networks such as ATM.

2.2.1 Multicasting in Shared Medium LANs

In a shared medium LAN, all endsystems are attached to the same physical medium. Data are exchanged using connectionless communication where each data frame carries its own address. A data frame that is sent on the LAN passes all attached endsystems. Several endsystems may copy the same frame from the LAN. Thus, multipoint-to-multipoint multicasting can be easily implemented in shared medium LANs. Figure 2.2 shows the situation where two senders multicast data to three receivers, whereas three endsystems are not taking part in the multicast association.

![Figure 2.2: Multipoint-to-multipoint multicasting in LANs](image)

IEEE 802.3 (CSMA/CD), 802.4 (token bus) and 802.5 (token ring) LANs possess inherent multicast capabilities. Data frames carry so called Medium Access Control (MAC) addresses, which are separated in unicast and multicast addresses. These addresses are distinguished by a single bit, called the group/individual bit. Each network interface carries a unique MAC address that is used for unicasting. The interface board will generate a software interrupt for all frames carrying the unique address which causes the device driver to copy the frame into the endsystem. For multicasting, several implementations exist. In the most simple case, the interface adapter generates a hardware interrupt for each frame that carries a multicast address. The network layer software then decides whether the frame is processed further or thrown away, depending on the multicast group memberships of the endsystem. This implementation is rather inefficient, since a software interrupt is
generated for every multicast frame that is sent on the LAN. More advanced
network adapters can be instructed to accept multicast frames on a per ad-
dress basis. Typically, these adapters are able to store between 16 and 32 ad-
dresses. Only frames that are sent to one of these addresses are forwarded to
the endsystem. Most of today's network adapters implement multicasting in
the latter way, which results in a very efficient multicasting. A third kind of
implementation uses so called hash buckets. Instead of storing a MAC ad-
dress on the network adapter, hash values are used. The adapter will generate
a software interrupt for every multicast address that matches one of the hash
values. Although this solution still copies unwanted frames, it is more effi-
cient than the first solution and less complex to implement than the second
one.

2.2.2 Multicasting in Datagram Networks

The only standardized implementation of multicasting in a datagram net-
work is IP multicasting. IP multicasting delivers the multicast capabilities of
shared medium LANs to the internetwork layer [Deering 89], [Deering 91].
As within shared medium LAN multicasting, IP multicasting uses group ad-
dresses for addressing a set of receivers. Senders do not need to know the in-
dividual addresses of the receivers. IP multicasting is strictly receiver
oriented. Receivers are allowed to join and leave a multicast group at any
time. Neither senders nor other receivers are notified of changes in multicast
group memberships.

If a multicast datagram is sent on a shared medium LAN, the internetwork
layer maps the IP multicast address to a LAN multicast address. The same
mapping happens if a receiver joins a multicast group: the IP layer computes
the MAC multicast address and instructs the network adapter to accept
frames carrying this address. If a receiver leaves a multicast group, it stops
accepting frames for this address.

The internetwork layer uses routers to connect LANs. In order to route mul-
ticast packets correctly, routers must know group memberships. The Internet
Group Management Protocol (IGMP) [Deering 89] is used for this purpose.
Each subnetwork has an elected router that broadcasts periodically group
membership query packets on the subnetwork. One member responds by
multicasting a group membership report packet. Every router thus knows
that a particular group is present on the subnet. On receiving the membership report, the other group members do not reply to the query. If no report is received for the group after a small number of queries, the routers conclude that a group is no longer present. To reduce join latency, each endsystem sends an unsolicited membership report after joining a group.

**Broadcast Routing Using a Spanning Tree**

The task of multicast routing is to find a delivery tree that reaches all receivers. A solution is to establish a *spanning tree*. A spanning tree is a subset of the overall network topology such that two nodes in the network are only connected by a single path. Once a spanning tree has been established, routers forward multicast datagrams on all outgoing branches of the spanning tree except on the branch from which the datagram was received. The spanning tree algorithm implements broadcasting, datagrams are forwarded to all receivers, regardless of group memberships. Obviously, implementing multicasting using broadcast routing with a spanning tree results in waste of bandwidth, since datagrams are delivered to all nodes, irrespective of their group membership.

**Reverse Path Broadcast Routing**

The use of a single spanning tree for a given network topology will likely result in bandwidth bottlenecks, since all datagrams travel over the same tree. A better solution is to use a spanning tree for each sender, an example of which is shown in the first tree in Figure 2.3. Such a spanning tree represents a broadcast tree that delivers packets to all attached subnetworks. A source rooted broadcasting tree is realised by an algorithm called reverse path forwarding, which was originally developed by Y. Dalal and R. Metcalf [Dalal et al. 78] and revised by S. Deering [Deering 91]. Reverse path forwarding is based on unicast routing. Each endsystem receives multicast datagrams on the same path as it receives unicast datagrams. The algorithm works as follows: if a datagram arrives via the link that would be used to reach the source of the datagram, a copy is forwarded on all other links. If a datagram is received on another link than the shortest path towards its source, it is discarded. By using this scheme, a loop-free broadcast tree for each source is formed.
Reverse Path Truncated-Broadcast Routing

Truncated-broadcast routing is an extension of the previously described broadcast routing. A truncated broadcast tree limits the number of unwanted deliveries, as shown in Figure 2.3. It is used, for example, in older versions of the Distance Vector Multicast Routing Protocol (DVMRP) [Waitzman et al. 88], which itself is based on unicast distance vector routing. Each router is able to recognize which of its attached subnetworks are leaf subnetworks. A leaf is not used by any router to reach the source. Routers can detect leaf subnetworks by listening to datagrams generated by the split horizon technique. The absence of such datagrams indicates a leaf subnetwork. In order to accomplish truncated-broadcast delivery, the routers behave as follows: a datagram has to arrive via the nearest subnetwork towards the source. A copy of the datagram is then forwarded on all subnetworks except on leaf subnetworks that do not contain members for the particular multicast group. The location of members of multicast groups is made known through the Internet group management protocol.
Reverse Path Multicast Routing

The actual version of DVMRP implements a routing scheme called reverse path multicast routing. In this scheme, a multicast delivery tree is constructed by pruning a broadcast delivery tree such that it only reaches the subnets that contain receivers of the particular multicast group (see Figure 2.3). Since there is potentially a different multicast tree for every combination of source and group, the costs for pre-computing this tree is not at all scalable. The distance vector multicast routing algorithm follows therefore an \textit{on-demand pruning} of trees. When a multicast datagram reaches a router that has no information about the source and the group, the datagram is delivered using the truncated-broadcast algorithm. Each router with no child subnetworks or no destinations on its leaf-subnetwork sends a \textit{prune message} for the particular (source, group)-pair back towards the source. Datagrams are no longer delivered to links on which a prune message has been received. A router that receives prune messages from all its subnets forwards this prune message towards the source. In this way, information about the absence of group members propagates back up the tree towards the source. Those prune messages have a limited lifetime so that routers do not have to store this messages forever. Thus, any branch that is pruned from the tree grows back. The next multicast datagram triggers new prune messages, as long as there are still no members of the group on that branch. In order to reach low join latency, routers may send \textit{graft messages} to cancel previously sent prune messages. Graft messages are sent on the following occasions:

- An endsystem becomes member of a pruned multicast group.
- A new subnetwork with members on it is attached to the router.
- A new router is attached on a subnetwork.
- A graft message is received from a subordinate router.

Graft messages are acknowledged by the receiving router, whereas prune messages are not. This is because the loss of a graft message may results in the loss of data, the loss of a prune message results only in a short waste of bandwidth.

A serious problem is the waste of bandwidth due to the first multicast datagrams. Those datagrams travel through the whole network, before prune messages can be sent. There is a potential danger of a \textit{broadcast storm}.
Reverse path multicast routing has also been implemented using link state routing (MOSPF) [Moy 94]. The extension defines a new link state that is used to hold information about group memberships. The advantage of using link state routing as opposed to vector-distance is that routers can perform reverse-path forwarding and pruning based on the "network map", i.e. the link state database. Thus, the first datagram does not need to be flooded. The flooding and pruning strategy used in DVMRP has two major disadvantages. The first datagram that is sent to a group is broadcast to all routers, even to those that are not involved in the multicast traffic of that particular group. This wastes resources in terms of bandwidth, processing and storage capabilities. The second disadvantage is that information must be stored on a per source basis. As a consequence, the number of senders affects the scalability of DVMRP.

Core Based Trees

As opposed to DVMRP, Core Based Tree (CBT) routing uses only a single delivery tree per group instead of one per sender [Ballardie et al. 93]. The delivery tree is a spanning tree for the receivers of the group which is shared among all senders. One router builds the core of the tree, hence the name. Non-core routers are attached to the core router. The tree spans all receivers, the senders need not be part of the tree. Receivers join the tree by sending a join request towards the core. The first router that is part of the delivery tree will add the new branch to the delivery tree. A sender that wishes to multicast a datagram, unicasts the datagram to one of the routers in the tree. The datagram is forwarded along the branches of the delivery tree. Since the delivery tree is in fact a spanning tree, datagrams can not loop.

Core based trees maintain a hard-state and do not adapt to network changes. This is in contrast to the source based trees used in DVMRP, which are built on demand. These trees are said to be data-driven and have a higher overhead than core-based trees.

The advantages of CBT are:

- Routers do not need to keep state information for all senders. The algorithm therefore is more scalable in respect of the number of senders.
- The receivers can choose which of the routers they wish to join to, e.g.
they may control the direction of growth of the tree. It is possible to implement load balancing, e.g. those routers that are not heavily loaded are used.

- No pruning of branches is needed, since the only delivery tree is adapted whenever a receiver joins the group.

The disadvantages of core based trees include:

- A shared tree is never optimal for all senders and introduces additional delay.
- The tree does not automatically adapt to network and topology changes. Therefore, control messages are needed to test reachability.
- With numerous active senders, the core routers might become “hot-spots” or “bottlenecks”.

**Protocol Independent Multicast**

The network working group of the IETF is currently working on a proposal called *Protocol Independent Multicast* (PIM) [Deering et al. 95 A], [Deering et al. 96 B]. The proposed architecture describes an efficient multicast routing that is independent of any particular routing protocol. The architecture contains two protocols: one used for *dense multicast groups* and one for *sparse multicast groups*. The dense mode is suited for multicast groups whose members are widely represented or where ample bandwidth is available. The sparse mode, in contrast, is intended for groups with few members which are distributed over a large area.

The dense mode protocol [Estrin et al. 96] uses reverse-path forwarding to implement truncated tree broadcasting, much as DVMRP and MOSPF. In contrast to DVMRP which is based on distance-vector routing, dense mode routing uses only the standard unicast routing tables, irrespective of how they were computed.

The sparse mode protocol [Deering et al. 96 A] works similar to core based tree routing. Sparse mode multicast groups are associated with so called *rendezvous points*, the core routers in CBT. Receivers join the rendez-vous points, senders unicast their datagrams to the rendez-vous points where they are distributed to all receivers. In PIM sparse-mode routing, however, a re-
receiver is also allowed to send join messages directly towards a sender. This will eventually result in a reverse-path forwarding tree for the source.

### 2.2.3 Multicasting in Fast Packet Switching Networks

Fast packet switching is a concept that places minimal functionality into the network. The outstanding technology used is *Asynchronous Transfer Mode* (ATM), which will be used to show the multicasting concepts of fast packet switching networks.

An ATM network consists of ATM switches that are interconnected by point-to-point links. Endsystems are attached by point-to-point links to ATM switches. Unlike shared medium LANs, ATM networks are connection oriented and provide bandwidth guarantees. ATM offers point-to-multipoint multicasting where a single sender is connected to multiple destinations. The ATM cells are replicated in the ATM switches where the connection splits in two or more branches (see Figure 2.4). ATM multicast connections are unidirectional only. The reason is that ATM cells do not carry endsystem addresses but only a link-local connection identification in form of a *Virtual Connection Identifier* (VCI) and a *Virtual Path Identifier* (VPI). If multicast connections were bidirectional and the sender received several cells from different receivers, it would not be possible to correctly reassemble them, since there would be no way of telling which cell belongs to which receiver [Alles 95]. For the same reason, ATM does not offer multipoint-to-multipoint virtual channel connections, since a receiver could not correctly reassemble incoming cells, since the cells of different senders would carry the same VCI/VPIs. However, it is possible to build a multipoint-to-multipoint abstraction by overlapping several point-to-multipoint virtual channel connections, as it is done in the proprietary signalling protocol CMAP [Cox et al. 94].

ATM uses a signalling protocol to establish, maintain and clear network connections. Two standards for the user-network interface exists, namely version 3.1 (UNI 3.1) of the ATM Forum [Forum 94] and ITU's Q.2931 [Q.2931 95]. For point-to-point connection setups, UNI 3.1 and Q.2931 work identically. They both set up point-to-point connections, select and assign VCI/VPI values for the connection on each link and request resources reservation according to the QoS parameters given in the call setup.
The point-to-multipoint connection setup is available in the UNI 3.1 protocol, but not in Q.2931. The ITU is currently specifying a signalling protocol with multicast support [Q.298X 94]. Multipoint connections can only be set up by the sender. The sender uses the ATM unicast address to connect to the first receiver. Additional receivers are then added to this connection. Each connection request is routed just like a point-to-point connection request. The only difference is that connections travelling on the same link are merged into a single connection. For the multipoint connection, a single QoS is assured, e.g. all receivers experience the same QoS. If the resources on a branch to a receiver do not suffice for guaranteeing the QoS, the receiver can not be added.

ATM does not offer group or multicast addresses. An endsystem identifies an established point-to-multipoint connection using a connection identifier, namely the VCI/VPI pair. VCI and VPI values have only local significance per link and are not globally unique.

Currently, the ATM Forum is working on the UNI 4.0 protocol specification [Forum 95]. The preliminary draft of UNI 4.0 offers slightly extended point-to-multipoint capabilities. In addition to the multipoint call setup used in UNI 3.1, UNI 4.0 offers leaf initiated join capabilities. With this capabilities, a receiver can join a multipoint connection without the sender being involved.
2.3 Multicast Transport Protocol Principles

Multicast transport protocols fulfill a set of different tasks. An initialization phase is used to establish the multicast association. During the data transfer phase, mechanisms for error correction, flow and congestion control are used. Furthermore, multicast transport protocols may also provide ordered packet delivery. The following subsections give an overview of these functions and mechanisms.

2.3.1 Multicast Association Management

In a centralized approach, a single entity is responsible for the administration of the entire multicast association. In many centralized point-to-multipoint protocols, this entity is the sender. It keeps track of the state of all receivers. An example of such a protocol is ST-2+ [Delgrossi et al. 95], [Topolcic 90]. A multicast association, called flow in ST-2+, is initiated by the sender which connects to the initially known receivers. Late coming receivers send a join request to the sender, which admits or rejects the newcomers. In the multipoint-to-multipoint capable MTP [Armstrong et al. 92], an entity called web master is responsible for administering the multicast association. Senders as well as receivers join the multicast association by sending a join request to the web master, which admits or rejects the requests.

In decentralized multicast protocols, the members of a multicast association are not necessarily aware of each other. The join operation is a local matter only that initialized the locally needed components. Vital administration information such as connection parameters is distributed by the senders in regular intervals [Floyd et al. 95] or even in every packet header [Holbrook et al. 95].

2.3.2 Multicast Error Control

In a centralized approach, a single host is responsible for assuring reliability for the entire multicast association. This single host is usually the sender. It has to keep track of the state of all receivers and, if necessary, start error recovery. Consequently, control information is exchanged continuously between sender and receivers. The advantage of a centralized controller is
global knowledge. A failure of a link or a receiver is noticed by the controller. This is important for applications that need to know whether data has been transmitted to all receivers. Therefore, a centralized controller can operate in the all-or-nothing paradigm, which means that a message either reaches all participants or none. A generalization of the all-or-nothing paradigm is the so called \textit{k-reliability}, defining that a message is delivered correctly to at least \( k \) receivers of a specified group. Unfortunately, centralized error control limits scalability by two factors. Firstly, the controller has to store information about every member of the multicast association. Secondly and more important, control information has to flow between the controller and the receivers. This control information increases as the number of receivers increases and thus reduces scalability.

Distributed error control overcomes the drawback of the centralized approach not being scalable. Error control is done by each receiver separately, the senders merely distributes data. This leads to additional flexibility, since each receiver decides independently on how errors are to be recovered. The focus is no longer a central error control, but every individual receiver. The disadvantage of the distributed approach is a loss of global knowledge. It is not possible to guarantee that data has been delivered correctly to all receivers, since if a link or a receiver runs out of order, the sender will not notice it.

\textbf{Sender Initiated Error Control}

The sender initiated approach is a straightforward extension of well-known unicast error control methods, such as go-back-n or selective retransmission [Mase et al. 83], [Gopal et al. 84]. A single sender multicasts data packets to all receivers. Errors are detected by a combination of acknowledgements and timers. Senders keep track of the state for each receiver. Sender-initiated error control schemes are centralized and provide global knowledge. If a packet for a particular receiver gets lost, the sender retransmits it. Properties of these methods are well understood, at least in case of unicast communication. They provide reliable communication, where senders know the state of every receiver. This is important for applications like distributed database systems where the sender has to know whether all receivers received data correctly. Unfortunately, the sender-initiated approach has some major drawbacks. Firstly, it does not scale well to a large number of participants. Since every participant has to acknowledge received packets, the sender receives a
great number of acknowledgements almost simultaneously. This effect is called *packet implosion* and it can cause a breakdown of the sender. Even if the sender can cope with all acknowledgements, it has to keep state information for every receiver. Therefore, memory and CPU usage increases as the number of receivers grows. Hence, sender-initiated error control is applicable for a small number of receivers only.

**Hierarchical Error Control**

Sender-initiated error control can be extended to a hierarchical scheme as depicted in Figure 2.5. Participants are arranged in a tree-like hierarchy defining the sender as root [Hofmann 96]. There are two possibilities to perform error control. In the first variant, the sender multicasts data packets to its children. Children acknowledge data and forward them to their children where the process is repeated. This method offers better scalability than direct sender-initiated error control. However, no end-to-end error control is performed. If packets are corrupted or lost inside a node after they have been acknowledged, this situation can not be recovered since the sender has discarded them.

Within another approach, after reception of data each node propagates packets to its children and waits for an acknowledgement. Only if all acknowledgements have arrived, the node acknowledges the packet to its parent. End-to-end error control and scalability can be achieved using this approach, but only to the cost of a relative high acknowledgement delay.

*Figure 2.5: Hierarchical error control scheme*
**Receiver Initiated Error Control**

Receiver-initiated error control overcomes scalability limitations of sender-initiated methods [Pingali et al. 94]. Receivers are responsible for detecting errors and requesting retransmissions of erroneous or lost packets. Lost packets are detected using sequence numbers. Each packet contains a sequence number. If a receiver detects a gap in received sequence numbers, it requests lost packets from the sender or possibly from other receivers by sending a *Retransmission Request* (RRQ). If the RRQ is multicast to the group, other receivers that experienced an identical packet loss will recognize it and suppress their RRQ. In addition to data packets, the sender issues periodically so called keep-alive packets. Keep-alive packets contain the last valid sequence number. This enables receivers to detect lost packets, even if the sender has no more data packets to transmit.

In the receiver-initiated approach, the load of senders is reduced. Therefore, the approach scales better. However, receiver-initiated error control suffers from a loss of global knowledge as all distributed error control schemes. Another problem arises according to the distribution of transmitting keep-alive packets. On one hand, if keep-alive packets are sent very often, the overhead of the protocol is significant. On the other hand, if keep-alive packets are sent rarely, the delay for detecting errors is increased considerably.

**Forward Error Correction**

Either sender- and receiver-initiated approaches require the retransmission of erroneous packets, which introduces delay. For audio- and video-streams sent over networks with a high bandwidth-delay product, a retransmission delay is unacceptable. *Forward Error Correction* (FEC) schemes overcome this problem [Biersack 92]. Errors are corrected by introducing redundancy into the original packet stream. Depending on the kind of redundancy, bit errors and one or several continuous packet losses can be corrected. Error correction takes place at the receiver. Since no feedback exists between sender and receivers, FEC is completely scalable.

However, FEC increases the need for extra bandwidth according to added overhead in terms of explicit redundant data. Forward error correction does not completely avoid delay. If a packet gets lost, the receiver might has to wait for further packets, until suitable information is available to reconstruct
the missing packet. Furthermore, in extreme cases it is not possible to recover from packet losses, since bursty errors exist. There is a trade-off between redundancy that is introduced and the number of errors that can be corrected as well as the delay that is needed to recover from an error. As with receiver-initiated error control, FEC is distributed and does not allow for sender-based detection, whether data has been received by all participants correctly.

2.3.3 Multicast Flow Control

Flow control avoids that receivers are overloaded with incoming packets. Flow control is needed since processing of incoming packets may take place at a slower rate than sending the packets. This leads eventually to buffer overflow at a receiver's side.

Multicast flow control has to take into account that only a single or a few of all receivers in a multicast group may be overloaded. Each method that is used handles this problem in its own way.

A sliding window based flow control mechanism, as it is used in TCP, is applicable where sender initiated error control schemes are applied. The send window size is only advanced if acknowledgements of all receivers are received. Thus, the sender is slowed down to the data rate of the slowest receiver.

Another solution is to limit the rate at which senders transmit their data to a fixed rate. Receivers that can not cope with this rate are not allowed to join the multicast association. In fixed rate flow control, the rate can not be changed during the lifetime of a multicast association. The advantage of this simple solution is that it does not require any feedback from the receivers. Although overload situations are avoided and a certain data rate is guaranteed, slow receivers may not be able to participate.

An adaptive rate based flow control is used in MTP [Armstrong et al. 92]. A central entity, the web master, controls the flow of all senders. Receivers may adjust the rate by sending control messages to the web master and therefore 'vote' on the rate. Receivers that can not cope with the selected rate have to leave the multicast association.
The above mentioned schemes all slow down the senders. For continuous media, however, it is also possible to speed up receivers by throwing away some packets. The quality will be degraded, but only for slow receivers.

2.3.4 Multicast Congestion Control

Congestion control is used to avoid a breakdown of the network, a so called congestion collapse. The network breaks down, if the sum of the load of the connected stations exceeds the total bandwidth the network can carry. If the network offers admission control in association with resource reservation and traffic policing, it is not possible for congestion to occur. Therefore, congestion control is only needed in best effort networks.

Congestion control in a multicast transport protocol has of course the same goal as congestion control in unicast transport. Several solutions exist. Given a large number of recipients, it is possible that congestion occurs only in a small part of the network and that only few of the receivers suffer from congestion. If the sender slows down, the problem will be solved as in the unicast case. Unfortunately, all other recipients receive data at a slower rate. Another solution is that the affected receivers are excluded from the multicast group, which is a somewhat crude solution to the congestion problem.

As with flow control, the output of the sender has to be reduced, since the network can no longer carry the offered load. TCP uses a congestion control that is tightly coupled to the error control scheme. Whenever the sender detects a congestion because of increased delay or increased packet losses, the send window is halved. After the congestion has passed, TCP uses a so called slow-start algorithm. The send window is increased step by step and not doubled as in the congestion avoidance phase. Increasing the window slowly avoids an oscillation between congestion and no traffic.

The congestion control scheme used in TCP is also applicable for multicast protocols that use some kind of sliding window protocol. The sender not only detects congestion but also identifies the receivers that are affected by the congestion. If a congestion is detected, the transport layer informs the application about the situation. The application then decides what to do, i.e. whether to slow down or to remove the congested receivers from the multicast group.
A more scalable solution, called probabilities acknowledgement, is proposed in [Bolot et al. 94]. It is applied in [Holbrook et al. 95] and works as follows: Each receiver acknowledges an incoming packet with a certain probability. The probability is computed by the sender in such a way that packet implosion does not occur and that a fixed number of acknowledgements is to be expected. If the number of acknowledgements is much lower than the expected number, then the sender assumes that there is a congestion and slows down. The probability is computed by estimating the number of total receivers. The sender does this by repeatedly requesting a state report from the multicast group. Each receiver answers with a certain probability that is specified in the request. The sender uses the number of answers to compute a correction of an initial estimation. The sender increases the answer probability with every request and repeats the process until the iteration converges.

Within this scheme, the state of the network is only known to a certain degree. Local congestion might remain unnoticed. However, congestion that affects a large part of the network is detected.

### 2.3.5 Multicast Ordering

Fault-tolerant distributed systems require more complex communication primitives than the reliable delivery of data. In particular, the order of delivery of messages must occur according to certain rules. Several order relations have been identified to be crucial for distributed systems. This section gives the definition of these relations. The model used as basis for the relations is shown in Figure 2.6. It consists of a number of senders and receivers as well as the two operations multicast and deliver (see Figure 2.7).
message is multicast to a multicast group. The receivers of the group deliver the message to the application. Receivers may be part of several multicast groups at the same time. Multicast groups may be served by several multicast senders at the same time. Senders may also receive messages, in which case they become members of multicast groups. For each message, the operator \( \text{sender}(m) \) denotes the sender of the message and \( \text{group}(m) \) the destination multicast group. The definitions in this section are based on [Hadzilacos et al. 94].

### Reliable Multicast

Multicast ordering is done on top of reliable multicast. Reliable multicasting is provided by an error control mechanism, as discussed in the last section. For the purpose of the following discussion, reliable multicasting is defined as follows:

**Definition 2.9:** Reliable multicast is a multicast that satisfies the following three properties:

- **Validity:** If message \( m \) is multicast then it will be eventually delivered by a receiver, or \( \text{group}(m) \) is empty.

- **Agreement:** If a message \( m \) is delivered by a receiver, then all receivers in \( \text{group}(m) \) eventually deliver \( m \).

- **Integrity:** A receiver in \( \text{group}(m) \) delivers \( m \) at most once, and only if \( m \) was previously multicast to \( \text{group}(m) \).

Reliable multicast does not define any order of message delivery. First In First Out (FIFO) order requires that messages are delivered in the same order.
as they are sent. FIFO order comes in two flavours, depending on whether messages are sent to a single multicast group or to several groups.

FIFO Order

**Definition 2.10:** Local FIFO order is a reliable multicast that satisfies the following requirement:

- If a sender multicasts \( m \) before \( m' \) and \( \text{group}(m') = \text{group}(m) \), then no receiver delivers \( m' \) unless it has previously delivered \( m \).

**Definition 2.11:** Global FIFO order is a reliable multicast that satisfies the following requirement:

- If a sender multicasts \( m \) before \( m' \), then no receiver delivers \( m' \) unless it has previously delivered \( m \).

The difference between local FIFO and global FIFO can be seen in the following example. Assume two multicast groups \( G_1 = \{A, B\} \) and \( G_2 = \{B, C\} \). Sender \( S \) multicasts \( m_1 \) to \( G_1 \) and then \( m_2 \) to \( G_2 \). Receiver \( B \) receives both messages. In local FIFO, the delivery order of the messages remains unspecified, whereas in global FIFO \( B \) has to deliver \( m_1 \) before \( m_2 \).

An implementation of local FIFO order can easily be done by using sequence numbers. For global FIFO, a sequence number for each multicast group has to be used in every message, e.g. each message contains a vector of sequence numbers. Receivers order the messages by building the sum of the sequence numbers for those groups they are member of.

Causal Order

FIFO order defines the ordering of messages that are sent by a single sender. However, operation on messages from different senders often depend on each other.

**Definition 2.12:** An operation \( e \) causally precedes operation \( f \), denoted \( e \rightarrow f \), if:

- the same process executes both \( e \) and \( f \), in that order, or
- \( e \) is the operation multicast\( (m) \) and \( f \) is the operation deliver\( (m) \), or
- there is an operation \( h \), such that \( e \rightarrow h \) and \( h \rightarrow f \).
Note that the causal precedence relation $\rightarrow$ defines a partial order.

Causal order defines an order relation that takes into account the causal precedence of operations. As with FIFO order, there exists a local and a global order relation.

**Definition 2.13:** Local causal order is local FIFO order that satisfies the following requirement:

- If the multicast of a message $m$ causally precedes in $\text{group}(m)$ the multicast of a message $m'$, then no receiver delivers $m'$ unless it has previously delivered $m$.

**Definition 2.14:** Global causal order is global FIFO order that satisfies the following requirement:

- If the multicast of a message $m$ causally precedes the multicast of a message $m'$, then if a receiver in $\text{group}(m)$ delivers $m'$, it delivers $m'$ only if it has previously delivered $m$.

Note that for global causal order, $\text{group}(m)$ may be different from $\text{group}(m')$.

Causal order can be implemented using a vector of timestamps, called logical clock [Schiper et al. 89]. This algorithm is used in the ISIS toolkit [Birman et al. 91] where it is part of a selection of reliable multicast protocols. Another solution of the causal order problem is shown in Figure 2.8. Messages are sent to a mediator in FIFO order where they are forwarded to the appropriate multicast group. The mediator serializes the messages and forwards them in the same order as they are received. Using the definition of causal precedence and causal order, it can be shown that this simple solution implements causal order. This kind of mechanism is used, for example, in the MTP protocol [Armstrong et al. 92]

**Total Order**

The last order relation presented here is total order. Total order states that all receivers of a message must deliver it in the same relative order. Total order is not an extension of causal or FIFO order, rather it is orthogonal to causal and FIFO order.
Definition 2.15: Local total order is satisfied if:
- If message \( m \) and \( m' \) are delivered by the receivers \( A \) and \( B \) and \( \text{group}(m) = \text{group}(m') \), then \( A \) delivers \( m \) before \( m' \) if and only if \( B \) delivers \( m \) before \( m' \).

Definition 2.16: Global total order is satisfied if:
- \( m < m' \): if and only if any receiver delivers \( m \) and \( m' \) and \( m \) is delivered before \( m' \).
- The relation \(<\) is acyclic.

The solution for causal order shown in Figure 2.8 also solves the total order problem. [Hadzilacos et al. 94] shows that a set of modular algorithm exists such that each type of multicast order can be extended to the next stronger type. In Figure 2.9, these algorithms are represented as arrows.
2.4 Summary

Multicasting has first been implemented in shared medium LANs, where multicasting and broadcasting have been used to implement basic network services such as address resolution. However, only by developing multicast routing algorithms in the network level, these capabilities have become accessible to higher layers. Today, IP and ATM multicasting provide a basic end-to-end multicast service. Both IP and ATM multicasting are standardized and used in commercial products. IP multicasting is also well established, the multicast backbone that is used in the Internet already connects thousands of endsytems.

The development of suitable multicast transport protocols has been the focus of many research projects. Although major results have been achieved mainly in the area of error control algorithms, only a few protocols have been standardized. Furthermore, the deployment of these protocols is insignificantly small. The reason for this might be that none of these protocols is flexible enough to be usable for all situations. This lead to the current situation where transport protocol mechanisms are part of multipoint applications which directly access IP or ATM multicasting.

MCF itself is based on end-to-end multicast services, in particular IP and ATM multicasting. Multicast transport protocol mechanisms are part of MCF, where they may be accessed by any application.
Chapter 3

Quality of Service in Communication Systems

The concept of Quality of Service (QoS) plays an important role in the design of advanced communication systems. This chapter introduces the abstract principles of QoS concepts as they are applied in today's communication systems. Furthermore, it defines terms and methodologies that are used to describe the design of MCF. In the first section of this chapter, the terminology is introduced, the second section defines methods that are applied in QoS based communication systems.
3.1 Terminology

The term Quality of Service (QoS) was first introduced to describe characteristics of low level data transmission in communication systems. With the appearance of multimedia systems, the meaning of QoS was broadened to cover all system components in a distributed multimedia system:

Definition 3.1: Quality of Service represents the set of those quantitative and qualitative characteristics of a distributed system necessary to achieve the required functionality and performance of an application. Functionality and performance include both the presentation of data to the user and general user satisfaction [Vogel et al. 95].

The Quality of Service is expressed using Quality of Service parameters:

Definition 3.2: A Quality of Service parameter describes a specific attribute of the multimedia system using a typed value. A QoS parameter is a structure consisting of an attribute name and of a typed value. An attribute describes a single aspect of the functionality or the performance of a distributed multimedia system.

Due to the very nature of multimedia applications, a vast set of different QoS exist. Applications have to specify the QoS they want to use such that the distributed multimedia system is able to provide the respective functionality and performance. This leads to the following definition:

Definition 3.3: Application requirements are a set of QoS parameters that are used by the application to describe the QoS that it requests of the underlying communication framework.

Multimedia communication systems are composed of several parts or layers, where each part offers a service. A service is defined as follows:

Definition 3.4: A service comprises the aggregate of all capabilities and features, which are available at an interface between two layers, or in general, between two objects that are correlated [Stiller 95]. The object that offers the service is called service provider.

A multimedia communication system needs resources to perform services. Steinmetz et al. [Steinmetz et al. 95 A] define resources as follows:
**Definition 3.5:** A *resource* is a system entity required for manipulating data. Each resource has a set of distinguishing characteristics.

- *Active resources* provide a service, whereas *passive resources* denote system capabilities required by active resources.
- A resource can either be used *exclusively* by one process or *shared* between a set of processes.
- A resource is called *single resource* when it exists only once in a system, otherwise it is called *multiple resource*.

### 3.2 QoS and Resource Concept

Before an application can use a multimedia communication system (MCS), it has to specify its requirements. A MCS consists of at least two endsystems and one or several intermediate systems, as shown in Figure 3.1.

![Figure 3.1: QoS and resource concept](image)

The communication systems form a layered hierarchy. Each layer provides a service either to the application or the layer on top of it. The services are described by layer specific QoS parameters. Therefore, this model is also known as QoS layering [Nahrstedt 95 C]. Since the service offered by a layer is based on other layer's services, the application requirements can only be satisfied if every layer meets the requested QoS [Anderson 93].
The MCS uses application requirements for two purposes:

- It selects and configures the layers such that the functionality requested in the application requirements is satisfied.
- In order to guarantee the requested performance, the MCS reserves the resources that are needed by each component.

Processing of QoS parameters inside a MCS is done in several, distinguishable steps which are explained in the following sections.

### 3.2.1 Application Requirement Specification

Due to the diversity of applications and media, there also exists a large variety of QoS attributes. Table 3.1 shows a few example attributes and their corresponding value ranges [Nahrstedt 95 C].

<table>
<thead>
<tr>
<th>QoS Attribute</th>
<th>Value Ranges</th>
</tr>
</thead>
<tbody>
<tr>
<td>Audio sample size</td>
<td>8-bit, 16-bit</td>
</tr>
<tr>
<td>Audio sample rate</td>
<td>8 KHz, 11 KHz, 22 KHz, 44.1 KHz</td>
</tr>
<tr>
<td>Audio playback point (delay)</td>
<td>100 - 150 ms</td>
</tr>
<tr>
<td>Video frame rate</td>
<td>5-60 fps</td>
</tr>
<tr>
<td>Video frame width</td>
<td>0-720 pixels</td>
</tr>
<tr>
<td>Video frame height</td>
<td>0-576 pixels</td>
</tr>
<tr>
<td>Video colour resolution</td>
<td>8 bit/pixel, 16 bit/pixel, 24 bit/pixel</td>
</tr>
<tr>
<td>Video aspect ratio</td>
<td>4:3, 16:9</td>
</tr>
<tr>
<td>Video compression ratio</td>
<td>1:1, 2:1, 50:1</td>
</tr>
<tr>
<td>Audio / video sync skew</td>
<td>+/- 80 ms</td>
</tr>
</tbody>
</table>

*Table 3.1: Example of QoS parameters*

Users often do not need a very precise QoS, since differences in the service quality are tolerated by the human perception of the presented media. Therefore, the application requirements specify ranges of values for QoS attributes. This is possible for QoS parameter values that can be expressed as ordered scalars. The communication system is said to meet the application
requirements if for each attribute the QoS that the system provides lies within the specified range.

Beside QoS parameters, application requirements also contain a parameter named type of service or service commitment. It defines how QoS parameters are to be interpreted by the MCS. Three types of services are distinguished [Ferrari 90]:

- **Within best effort service**, the service provider consents to the requested QoS values without giving any guarantees. No resources are reserved, the QoS can only be maintained as long as sufficient resources are available.

- **Statistical guarantees** or **soft guarantees** are provided when resources are reserved using a statistical model. As a result of this, the requested QoS may be temporarily violated. Resource reservation strategies used for soft guarantees are also called optimistic reservation models, they lead to a good utilisation of the available resources. Depending on the actual reservation model, statistical guarantees range from 'nearly best effort' to 'almost deterministic'.

- **Deterministic guarantees** or **hard guarantees** are the strongest form of guarantees. Resources are exclusively reserved based on worst case assumptions. In contrast to soft guarantees, the application requirements are only violated in case of system or component failure.

### 3.2.2 QoS Parameter Mapping and Resource Calculation

The service provided by each layer is described using layer specific QoS parameters. The topmost layer uses the QoS parameters of the application requirements. To offer a service, each layer (except the lowest layer) uses services from the next lower layer as well as resources. In order to meet a requested QoS, each layer decides which QoS it needs from the lower layers. Therefore, the QoS parameters that are passed by a QoS request to layer N+1 are mapped to QoS parameters that are requested of layer N. Additionally, the QoS parameters are also used to calculate the amount of resources that are needed by each layer. The following notation is used to describe the mapping:
• $\overrightarrow{Q}_N$ is a vector of QoS parameters that describe the service of layer $N$. It is used to request a service quality of layer $N$.

• $\overrightarrow{Q}_N[x]$ denotes a QoS parameter value for the QoS parameter attribute $x$. For example, the QoS vector of an audio application contains the element $\overrightarrow{Q}_N[\text{sampling rate}]$ which may have a value of 44100.

• $\overrightarrow{R}_N$ is a vector describing the amount of resources needed by layer $N$ for providing the QoS described by $\overrightarrow{Q}_N$.

• $\overrightarrow{R}_N[x]$ represents the amount of resource $x$ that is needed by layer $N$. For example $\overrightarrow{R}_N[\text{memory}]$ describes the memory needed by layer $N$.

• $\overrightarrow{M}_N()$ is the QoS mapping function of layer $N$ that maps a QoS vector of layer $N$ to a QoS vector of layer $N+1$.

• $\overrightarrow{F}_N()$ is the resource calculation function that calculates the amount of resources that is needed by layer $N$.

Figure 3.2 illustrates QoS mapping in the forward direction, i.e. from the application towards the network. For each layer $N$, a mapping function $\overrightarrow{M}_N()$ of the form $\overrightarrow{Q}_N = M_{N+1}(\overrightarrow{Q}_{N+1})$ is defined, where $\overrightarrow{Q}_{N+1}$ is a vector containing the input QoS parameters and $\overrightarrow{Q}_N$ is the resulting vector with the output QoS parameters. Figure 3.3 shows an example of the mapping of audio QoS parameters to parameters used at the transport layer. In general, each element $\overrightarrow{Q}_N[x]$ depends non-linearly on all elements of vector $\overrightarrow{Q}_{N+1}$. This means that there is no scalar function $m$, such that the following holds: $\overrightarrow{Q}_N[x] = m_{N+1}(\overrightarrow{Q}_{N+1}[x])$. Instead, $\overrightarrow{Q}_N[x]$ always depends on the whole vector $\overrightarrow{Q}_{N+1}$: $\overrightarrow{Q}_N[x] = (M_{N+1}(\overrightarrow{Q}_{N+1}))[x]$.
3.2. QoS and Resource Concept

The calculation of resource requirements depends on available resources, which may be different for each end-system. For instance, the execution time for a particular function inside a layer depends on the hardware, e.g. on the speed of the CPU, the bus architecture, the cache memory and other hardware components. Information about the hardware is therefore stored in a hardware profile. The profile is used to calculate the resource requirements. Additionally, the type of service, e.g. statistical guarantee or deterministic guarantee is also taken into account in the calculation. Resource calculation functions, therefore, take the form of \( \overrightarrow{R_N} = \overrightarrow{F_N}(\overrightarrow{Q_N}, \hat{P}, T) \), with \( \overrightarrow{Q_N} \) being the set of QoS parameters, \( \hat{P} \) the hardware profile information and \( T \) the type of service. The result of the calculation is a vector \( \overrightarrow{R_N} \) that contains the usage figures for the needed resources of layer \( N \).
QoS mapping is also needed in the reverse direction, i.e. from the lowest layer towards the application. If resources become unavailable or if the application decides to change the amount of resources that it is using, then the new QoS as it is provided to the application has to be calculated. The mapping in the backward direction, however, is not deterministic. If, for example, the bandwidth is reduced for an audio stream, it is possible to reduce the sampling rate or the sampling size or the number of channels. Therefore, the backward mapping is only possible if the QoS attributes are ordered by preference or importance, e.g. the parameters are weighted. By considering this weight, a backward mapping yields a unique result. The reverse mapping function takes the form $Q_{N+1} = M'_N(Q_N, \vec{W}, R_N)$, where $R_N$ denotes the new resource usage figures, $Q_N$ the QoS parameters of the lower layer and $\vec{W}$ the weights for the parameters $Q_{N+1}$.

### 3.2.3 Resource Management

Resource management consists of the following tasks:

- *Admission control* and *resource reservation*
- Resource access management for exclusive resources, often denoted as *QoS maintenance* or *QoS enforcement*
- Incorporation of resource changes
When an application first accesses a service, MCSs calculate the amount of resources that are needed to satisfy requested QoS. Admission control then decides whether the resources can be reserved. Admission control is conducted by applying an individual *admission test* for each resource, as shown in Table 3.2. If admission tests are successful, the resources are reserved. The actual reservation depends on the type of the resource. Shared resources are reserved by marking a part of the resource, e.g. a memory block is marked as reserved. Exclusive resources such as the CPU can only be used by one component at a time. For these resources, an access discipline must be provided, e.g., an access schedule that decides which component is allowed to use the resource.

<table>
<thead>
<tr>
<th>Resource Type</th>
<th>Admission Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>A scheduling test decides whether the tasks are locally schedulable and can meet the required deadline.</td>
</tr>
<tr>
<td>Memory</td>
<td>The admission test for memory decides whether enough physical memory is available.</td>
</tr>
<tr>
<td>Bandwidth</td>
<td>The bandwidth test determines whether the available bandwidth is sufficient. The bandwidth test also takes into account the rate at which packets are produced, since the packet rate is also a limitation of network access, due to per packet overhead.</td>
</tr>
<tr>
<td>Multimedia devices</td>
<td>Multimedia devices are exclusive resources. The admission test checks whether devices are free and whether they can provide the requested quality.</td>
</tr>
<tr>
<td>End-to-end delay</td>
<td>The end-to-end delay test examines the processing delay in endsystems and the network latency.</td>
</tr>
</tbody>
</table>

*Table 3.2: Admission tests*
While multimedia data are being exchanged, resource management is responsible that components can access resources if required. Exclusive resources are managed by a scheduler that controls the access. A typical example of an exclusive resource is the CPU which is often controlled by a real-time scheduler. Shared resources such as memory do not need an access discipline, since each component uses only a fraction, e.g. a memory block, of the resource which is accessible without any limits.

The change of the QoS leads to changes in the needed resources. There are two aspects of resource changes. If the QoS is increased, additional resource are needed. For a decrease in QoS, parts of the resources must be freed, since the may be needed for another service. In either way, resource changes are tightly coupled with QoS re-negotiation.

### 3.2.4 QoS Negotiation

The purpose of *QoS negotiation* is to orchestrate the layers of the communication systems in such a way that application requirements can be satisfied without wasting resources. After a successful QoS negotiation, each involved layer offers the QoS that is requested by reserving the resource needed for the QoS. If a single layer is unable to provide the requested QoS, the overall QoS can not be provided and the negotiation fails.

QoS negotiation is a complex process that involves parameter distribution, parameter mapping, admission control and resource reservation. Due to the complexity of the operation, QoS negotiation takes many forms. The OSI95 architecture describes a classification of QoS negotiation schemes that differentiate in the involved parties and the way QoS parameters are negotiated by the parties [Danthine et al. 93 A], [Danthine et al. 94]. Negotiation schemes are applied in two basic situations, as depicted in Figure 3.4.

*Layer-to-layer negotiation*, which is used within an endsystem or intermediate system, involves a caller and a callee that negotiate QoS parameters using a two-way handshake (1) and (2). In the case of *peer-to-peer negotiation*, the caller and the callee are not on the same system. Besides the caller and callee, the negotiation involves a service provider which must also agree on the negotiated QoS parameter values. The caller requests a certain QoS from the service provider (1). A maybe modified version of the request is forwarded
to the callee (2), which selects a value and returns it to the service provider (3) who indicates the value to the caller (4). For both layer-to-layer as well as peer-to-peer negotiation, three types of negotiations are distinguished:

**Definition 3.6:** *Unilateral negotiation:* The caller proposes QoS parameter values that can be accepted or rejected by the callee. Negotiation is reduced to a "take it or leave it" approach. In case of peer-to-peer negotiation, not only the callee but also the service provider may reject the QoS request.

**Definition 3.7:** *Bilateral negotiation:* The caller proposes QoS parameter values that can be accepted, accepted with modifications or rejected by the callee. In case of peer-to-peer negotiation, the service provider is only allowed to accept or reject the QoS request. The service provider is not allowed to modify the request.

**Definition 3.8:** *Triangular negotiation* involves a caller, a service provider and a callee. The QoS parameter may be accepted, modified or rejected by both the service provider and the callee.

QoS parameter values are typed and thus take only values of a defined base set. If the base set defines an ordering relation, QoS parameter values can be compared. This means that for any two different values, the stronger of the two can be chosen. Stronger not necessarily means larger, for a delay QoS parameter, stronger means smaller, for a frame rate QoS parameter, stronger means larger. The type of a QoS parameter value has an influence on how parameters can be modified by service providers or callees. For ordered values such as scalars, the following mechanisms are defined [Danthine et al. 93 A]:

**Definition 3.9:** *Information exchange:* The target value may be weakened without any restrictions.
Definition 3.10: **Bounded target**: The target value may be weakened, but not below a lowest acceptable value. The lowest acceptable value must be specified by the caller.

Definition 3.11: **Contractual value**: The target value may be strengthened, but not above a certain bound. The bound for strengthening must be provided by the caller.

For a single QoS parameter, a bounded target as well as a contractual value may be defined at the same time.

Parameters whose base set does not define an ordering relation, no range can be defined. Thus, a parameter may not be weekended or strengthened. Consequently, callee and service provider may only accept or reject such a value. Examples of such values are encoding formats, such as linear, alaw, and µlaw encoding for audio.

### 3.2.5 QoS Re-negotiation

QoS re-negotiation is used to change the QoS without aborting and re-establishing the communication. QoS re-negotiation may be triggered by a user or by the framework upon detecting a violation of the offered QoS. QoS violation may only occur for statistical service guarantees.

QoS re-negotiation works similarly to QoS negotiation, i.e. it also consists of parameter distribution, parameter mapping, and admission control. Resource reservation, however, is more complex, since the resource management must allocate additional resources or release parts of already allocated resources. Ideally, changes in resources do not interrupt the service, i.e. the resources are adapted “on the fly”. However, since data keeps flowing during the QoS re-negotiation, the resource manager must decide when the changes in resources may take place. A possible solution is to introduce checkpoints in the flow which trigger the resource reservation changes.

### 3.3 Summary

This chapter presents an abstract quality of service model of existing communication systems. The model defines architectural elements which are
needed in communication systems to provide services according to the needs of applications. In the presented model, applications specify their requirements using QoS parameters. These parameters are mapped to internal representations such that each component of the communication system is able to calculate the resources which are needed to provide the requested service. A resource management controls access to resources and guarantees that existing services are not affected by new service requests. QoS negotiation is used to orchestrate the resource management in the involved systems and QoS re-negotiation is used to adapt an existing communication service to a changed environment.

The presented model reflects current research results. It tries to describe the essential elements of existing communication systems. However, research in the area of communication systems will not be finished before some time. The model therefore has to be viewed as temporary, only. It offers a high level abstraction of the current state of the art by introducing a common terminology which will also be used in the following chapters.
Chapter 4

Related Work in Multipoint, Multimedia Communication

This related work chapter examines communication protocols and frameworks that have been developed for the purpose of multipoint, multimedia communication. The systems discussed here have been selected based upon two criteria. Firstly, they use or implement any form of multicast communication. Secondly, the quality of the offered communication service can be specified using QoS parameters. The chapter roughly follows historical developments that have taken place. The works presented are divided into two groups. The first group consists of single protocols or mechanisms which are necessary, but not sufficient for supporting multipoint, multimedia applications. In the second group, communication frameworks are discussed that offer QoS guarantees not only for data transport but also for protocol
processing and possibly multimedia presentation. These frameworks may make use of protocols and mechanisms presented in the first group. MCF, as a multipoint communication framework, belongs to the second group. Consequently, it is compared to the frameworks discussed in the second group.

### 4.1 Multicast Network and Transport Protocols

Multicast network and transport protocols have set the basis for the development of communication frameworks which are discussed in Section 4.2. In particular, the network protocols may be directly used by communication frameworks. As such, the basic interest lies in the kind of services these protocols offer. The presentation of the protocols and mechanisms focuses on the following points:

- A classification of the protocol or mechanism according to the OSI reference model as well as a short summary of the functionality.
- The type of multicast communication, e.g. point-to-multipoint or multipoint-to-multipoint.
- The multicast association establishment, join and leave of participants.
- QoS specifications, QoS negotiation and re-negotiation.
- QoS enforcement.

#### 4.1.1 Internet Stream Protocol (ST-2+)

ST-2+ is a combined internet and resource reservation protocol that can be used instead of IP [Delgrossi et al. 95], [Mitzel et al. 94], [Delgrossi et al. 93 A], [Herrtwich et al. 92]. ST-2+ is the result of a development that started with ST in the late 1970's. A revised version of the protocol was specified by the IETF in RFC 1190 [Topolcic 90]. In 1993, the ST-2 working group was formed by the IETF, which resulted in the specification of ST-2+.

Like IP, ST-2+ consists actually of two protocols, ST for the data transport and the Stream Control Message Protocol (SCMP) for all control functions. For the purpose of this discussion, the name ST-2+ will include both protocols. ST-2+ is a complete network layer protocol that combines data han-
dling and resource reservation setup. The protocol is connection oriented, it offers duplex unicast connections and simplex point-to-multipoint connections. For establishing connections, ST-2+ interfaces to a routing function which is outside the scope of ST-2+. As within IP, data transport in ST-2+ is not reliable.

The stream concept is the most important part of ST-2+. Streams are represented as trees with the sender as root, receivers as leaves and intermediate systems as nodes inside the tree. The initial receivers as well as new receivers are added to the tree by the sender, which sends connect messages to them. Multicast association establishment in ST-2+ is thus strictly sender oriented. Receivers may leave a tree by sending a disconnect message towards the sender. The sender, however, may also send disconnect messages to receivers.

Each stream is associated with QoS parameters. The format of these parameters is transparent to ST-2+, the QoS parameters are transported by ST-2+, but not interpreted. An example specification uses the following QoS parameters: message size, message rate and end-to-end delay together with the service types statistical guarantee or deterministic guarantee.

QoS negotiation is coupled with connection setup. The negotiation is a sequence of triangular negotiations that involves the sender, the receivers as well as any number of intermediate systems. ST-2+ itself does not reserve any resources, it merely conveys the QoS parameters to local resource managers. These managers are also responsible for QoS enforcement. The initial QoS parameters are sent from the source towards the receivers. Each intermediate system tries to reserve the resources requested in the connection setup while possibly modifying the initial QoS parameters. The receivers either accept without modification, modify or reject the setup and send it back to the source. On the back path, the QoS parameters are used in the intermediate systems to adjust their reservations to the actual state. The source receives answers of all targets, computes the overall QoS and propagates it to all destinations.

The QoS parameters of an existing stream may be changed by the source. The source transmits the new QoS parameters towards the receivers. The intermediate systems try to adjust to the new situation. On success, the param-
eters are propagated further down the tree or otherwise a reject message is sent back to the source. Receivers may also either accept or reject the modification and return the appropriate answer back to the sender.

ST-2+ defines group relationships among the streams. The purpose of groups is resource sharing. For example, ST-2+ allows for bandwidth sharing inside the streams of a group. The classical example for bandwidth sharing is an audio conference application, where only a single speaker produces audio data at one time. Therefore, only bandwidth sufficient for a single source must be allocated.

4.1.2 Tenet Protocol Suite 2

The Tenet Protocol Suite 1 has been developed at the University of California at Berkeley and the International Computer Science Institute [Ferrari et al. 94], [Banerjea et al. 94]. It provides guaranteed real-time services over packet switched networks based on simplex, unicast connection. The Tenet Protocol Suite 2 is an extension that offers support for point-to-multipoint connections. The protocol family consists of an internet protocol called Real-Time Internetwork Protocol (RTIP), of two transport protocols, the Real-Time Message Transport Protocol (RMTP) and the Continuous Media Transport Protocol (CMTP). Besides these protocols that are used for the data transport, the signalling protocol Real-Time Channel Administration Protocol (RCAP) is used for setting up connections. It includes admission tests as well as resource reservation. Tenet is designed to co-exist with non-real-time protocols such as TCP/IP, as depicted in Figure 4.1. In order to provide real-time guarantees, protocols of Tenet must run on a data link layer that offers service guarantees.

RCAP is responsible for establishing point-to-multipoint connections, called channels [Bettati et al. 95]. Channels are later on used by the connection oriented RTIP protocol. They are initially set up by the sender. During the set-up, QoS negotiation takes place and resources are reserved in the intermediate systems. This is done by sending a setup message that contains QoS parameters along the multicast delivery tree to the receivers. Admission control is done on the forward direction, whereas on the backward direction the actual resources are reserved. If an admission test on a branch to a receiver fails, the receiver is excluded from the channel. Besides this sender orient-
ed connection setup, RCAP also supports receiver initiated joins and leaves. In the receiver initiated join, a setup message is sent towards the nearest intermediate system of the channel, thus reserving the resources needed for the additional receiver. A pure receiver-initiated setup is achieved if the initial target set of the sender is empty.

QoS parameters are formed of the two parts traffic characteristics and performance requirements. The traffic characteristics describe the bandwidth and the traffic pattern generated by the application. A whole set of traffic models are supported. An example of a traffic characteristic describes the maximum packet size, the average period as well as minimum inter-packet time and minimum periodic inter-packet time. The performance requirements are expressed using end-to-end delay, delay jitter, and maximum packet loss. Performance requirements are specified as ranges. Admission control succeeds, if the intermediate system is able to support the defined traffic within the performance range.

The QoS defined for a channel is enforced by RTIP. RTIP implements a rate control and a scheduling module. The rate control module shapes the incoming traffic according to the traffic characteristics that are valid for the channel. Additionally, it monitors the incoming packets and decides when they have to be forwarded by the scheduler. For any packets that arrive at the RTIP layer, it ensures that they are forwarded within the specified delay.

RCAP is capable of sharing resources among several associated channels [Gupta et al. 95]. Channels that share resources are grouped in so called sharing groups. Sharing groups are useful for applications where only a subset of senders is active simultaneously. For sharing groups, bandwidth ad-
mission and scheduling tests are done on the requirements of the whole group which are lower than the sum of all individual channel requirements.

RCAP offers QoS re-negotiation through a mechanism called *dynamic traffic management* [Heffner 95]. The semantics of dynamic traffic management allow a sender to change the QoS if all current receivers are able to support the changed QoS. The re-negotiation is done using a two-phase commit protocol. In the first phase, it is tested whether all intermediate systems and all receivers support the new QoS. In the second phase, the new QoS is committed or aborted, respectively.

Besides the services presented here, RCAP additionally supports other services such as advance reservation of resources, third party coordination which allows so called network resource booking agents to act on behalf of other entities and resource partitioning, which allows network administrators to assign resources to user groups.

### 4.1.3 Integrated Services in the Internet Architecture and RSVP

Integrated Services Internet Architecture is a proposal that integrates best effort services of the Internet with services that provide QoS guarantees [Braden et al. 94]. In order to offer performance guarantees, a communication subsystem consisting of packet scheduler, classifier, admission control and reservation protocol has been proposed. The tasks of these components are defined as follows:

- **Packet Scheduler**
  The packet scheduler is responsible for QoS enforcement. It forwards packets of different flows according to the QoS installed for the flows. It is used at the network layer in the endsystems as well as in the intermediate systems, where it interfaces to the datalink layer. The packet scheduler may use data link layer dependent bandwidth allocation mechanisms, if available.

- **Classifier**
  For each incoming packet, the classifier decides to which flow the packet belongs. This is necessary, since no connections are set up.
Packets of the same flow are treated equally by the packet scheduler, e.g. they use the same reserved resources.

- **Admission Control**
  Admission control is used in the endsystems and intermediate systems to decide whether enough resources are available to admit a new flow.

- **Resource Setup Protocol**
  The resource setup protocol is used to create and maintain a flow-specific state in the endsystems and intermediate systems, e.g. it conveys the information that is needed by the systems to reserve the required resources.

With these components at hand, applications can request delay bounds as well as bandwidth guarantees. Currently, specifications of different service types are being developed, i.e. guaranteed QoS [Shenker et al. 96] and committed rate QoS [Baker et al. 96].

A likely candidate to be used as a resource setup protocol in the Internet is the *Resource ReSerVation Protocol* (RSVP) [Zhang et al. 93], whose specification is being finalized [Braden et al. 96]. RSVP is used to reserve resources in intermediate systems. Data is transported using IPv4 [Postel 81] or IPv6 [Deering et al. 95 B]. RSVP is targeted at a multipoint-to-multipoint environment with support for heterogeneous receivers. Reservations in RSVP are always initiated by receivers, since receivers are the ones that experience the QoS. Senders multicast data regardless whether resources have been reserved or not.

Reservation requests consist of a *flowspec* and a *filterspec*. The flowspec contains the QoS parameters that are used to reserve the resources. Flowspecs are not interpreted by RSVP, they are transparently forwarded to the intermediate systems. The format of the QoS parameters depends on the used services [Wroclawski 96]. For guaranteed QoS service, the flowspec defines the requested bandwidth by peak data rate and maximum packet size as well as the delay. The filterspec defines which packets are entitled to use the reserved resources. In protocols such as ST-2+ and RCAP, reservations are done per connection. RSVP, however, operates in a flow-oriented environment. A flow is defined as set of packets that are sent by possibly many senders to a single multicast group using the same transport protocol.
Flowspec and filterspec allow for flexible resource management, since the filterspec can be changed independently of the flowspec. RSVP distinguishes among shared or distinct reservations as well as explicit and wildcard sender selection. This gives a total of four reservation styles, out of which, however, only three are useful:

- **Wildcard-Filter**
  In the wildcard filter style, the resources are shared among the packets of all senders, with no limitations on the number of senders, i.e. the resources are used by packets of every sender.

- **Fixed-Filter**
  In the fixed filter style, resources are reserved for each designated sender. Resources are not shared between packets of different senders, since each sender may have its individual flowspec.

- **Shared-Explicit**
  The shared-explicit reservation style defines a flowspec that can be used by packets of a designated set of senders. Packets of senders that do not belong to the set can not use the resources.

Each RSVP sender periodically transmits so called *path messages* towards the receivers. Path messages describe the traffic characteristics of the sender's data. They are also used to inform the receivers about the state of the reservations on the path to the sender. Receivers learn about new senders by inspecting path messages. They send reservation requests towards the sources, reserving the needed resources in the intermediate systems. RSVP takes a so called *soft state* approach to managing reservations. Reservations must be refreshed periodically, otherwise they are deleted. So the soft state approach increases the robustness and avoids resource waste in error situations. At the same time, the soft state approach eliminates the need for QoS re-negotiation phases. Finally, it must be noted that RSVP does not provide QoS guarantees for all packets. The first packets that are sent may be transported before a reservation took place. The same problem exists if packets are re-routed.

### 4.1.4 Xpress Transport Protocol Version 4 (XTP V4)

XTP is a general purpose transport protocol [XTP Forum 95]. The protocol offers the combined functionality of TCP and UDP, but includes multicast...
transport, multicast group management, transport layer priorities and traffic descriptions for Quality of Service negotiations. XTP runs on IP, native ATM or directly on IEEE 802 LANs and uses the respective multicast capabilities.

The design principle of XTP was to offer mechanisms rather than any particular policy. Error, flow and rate control can be chosen independently on a per connection basis, these mechanisms are fully orthogonal. XTP offers a simplex point-to-multipoint multicast transport mechanism which can be configured in several ways.

Multicast connection setup in XTP is a combination of a sender initiated and receiver initiated mechanism. Before a connection is set up, the receivers are set into the listening state. The sender or multicast master connects to the receivers by sending a FIRST packet. The FIRST packet contains QoS parameters, both what the sender can maintain and what it wishes to use. The QoS parameters are chosen out of various formats offered by XTP. The formats range from 'no specification' up to a format where MTU size, delay, throughput, the type of traffic guarantees and the class of reliability can be specified. Receivers have three possibilities:

- Accept the traffic specification.
- Accept the traffic specification with modifications.
- Reject the traffic specification and leave the group.

The transmitter receives the answers of all participating receivers. XTP does not specify how the common QoS is chosen, this is implementation dependent. The implementation described in [de Rezende et. al 95] offers three possibilities:

- The QoS of the sender must not be altered.
- The receivers are allowed to use local packet filters for downgrading, however, the QoS of the sender is not altered.
- The multicast master chooses a QoS according to an application specified algorithm (e.g. the 'weakest' specification). This specification is distributed to the participating receivers.

The join operation is used to add a new receiver to an existing multicast association. This operation is receiver oriented: a receiver that wants to join the
connection sends a JOIN message to the multicast sender. In return, the new receiver gets the current QoS parameters. The new receiver is only allowed to accept or reject the traffic specifications, a negotiation is not possible.

According to the XTP protocol specification, traffic specifications can be re-negotiated by either the multicast sender or by any of the receivers. However, the specification does not describe how this is to be done. In the implementation of [de Rezende et. al 95], only the multicast master can initiate a traffic re-negotiation. This re-negotiation works similar to the initial negotiation done during the connection setup.

4.1.5 Multicast Extensions of the OSI95 QoS Negotiation Scheme

An extension of the OSI95 QoS negotiation scheme is described in [Mathy et al. 94]. It defines triangular negotiation schemes for a simplex, point-to-multipoint connections. All participants have to be known at connection setup time, no late joiners are allowed. QoS negotiation is done at connection setup.

Depending on the scope, two classes of QoS parameters are defined. The first class comprises the parameters whose scope is the whole connection. These parameters are classified as connection wide. The second class are those parameters whose scope is limited to a single receiver, they are denoted receiver selected. A typical example of a receiver selected parameter is the delay which is different for each receiver. The throughput, on the other hand, is an example of a connection wide parameter. The purpose of the QoS negotiation is to find a common value for all connection wide parameters. The receiver selected parameters don’t have to be negotiated.

Connection wide QoS parameters are seen as a structure containing an attribute name and three values called compulsory, threshold and maximum quality. The structure defines a contract between the service users and the service provider. In a contract, not all of the three values have to be used. However, if all three are used, the maximum quality value has to be “stronger” than the threshold which itself has to be “stronger” than the compulsory value. The semantics of the three values are as follows:
• **compulsory**
  The service provider monitors the value and aborts the service when it cannot achieve the requested service.

• **threshold**
  The service provider monitors the value and signals to the users when the requested service is not achieved.

• **maximum quality**
  The service provider monitors the value and "avoids occurrence of interactions with the service users that would give rise to a violation of the selected value" [Danthine et al. 93 A]. The maximum quality value defines an upper limit to a QoS parameter value. If the limit is violated, the service provider tries to correct the problem without notifying the service user.

QoS negotiation for parameters that are associated with a compulsory or threshold value are negotiated using a range consisting of the minimal acceptable value and a target value. The negotiation itself is done using a three way handshaking as depicted in Figure 4.2. The service user defines the minimal acceptable value and the target value and requests a connection setup. The service provider softens the target value depending on the quality it is able to provide and indicates the connection setup request to the called users. Each called user may is offered a different target value. Every called user se-
lects a single value out of the offered range and returns it to the service provider. The service provider then computes a single value for the connection-wide parameter and confirms it to the service user and the called users.

It is not always possible to compute a connection wide value that satisfies all participants. Since each called user chooses a value according to its own requirements, it is possible that a response of one user lies outside the indicated range of some other users. Figure 4.3 gives an example of such a situation. User 2 selects a value which lies in the indicated range, but outside the range that was indicated to user 1.

![Incompatibility](image)

*Figure 4.3: Incompatibility in selected values [Mathy et al. 94]*

### 4.2 Communication Frameworks

Communication frameworks have been developed to support multimedia applications. As such, they offer performance guarantees not only for the transportation, but also for data processing in the endsystem. The purpose of this section is to give an overview of communication frameworks and their degree of support for multipoint, multimedia application. The degree of support is evaluated using criteria that have been identified in the first chapter:
4.2. Communication Frameworks

- **Type of multicast communication**
  Does the framework support point-to-multipoint or multipoint-to-multipoint multicasting?

- **Multicast dynamics**
  Does the framework allow for late joiners? Can new members join an existing multicast association? Can existing members leave a multicast association?

- **Heterogeneous receivers**
  Does the framework allow receivers to experience individual QoS? How is this achieved?

- **Transparent application requirements**
  Is the translation of abstract, media-specific application requirements to an internal representation supported? To which degree does the framework support QoS mapping?

- **QoS re-negotiation**
  Is it possible to re-negotiate the initially selected QoS?

- **Endsystem QoS maintenance**
  How is the QoS maintained in the endsystems? Does the framework provide QoS on the basis of application-to-application or on the basis of multimedia-device-to-multimedia-device?

### 4.2.1 QoS-Architecture

The QoS-Architecture of Lancaster University (QoS-A) is a layered architecture of services for QoS management of continuous media flows in multi-service networks [Campbell et al. 94], [Coulson et al. 94]. It supports a wide range of QoS for multimedia applications. QoS-A integrates QoS configurable protocols in both the endsystem and the network. The QoS-A is divided in layers and planes:

- The *protocol plane* consists of the protocol stack that is divided into the layers physical, data, network, transport, and distributed platform layer. The protocol plane itself is divided into two parts: a user part and a control part. The user part is used for application data, which is assumed to be unidirectional multimedia data. Multimedia data must be provided by the application, QoS-A does not interface directly to mul-
timedia devices. For the control data, a bidirectional, reliable transfer with low latency is provided.

- The QoS maintenance plane has one QoS manager per layer. Each manager is responsible for monitoring and fine-tuning the components on their layer. Fine-tuning is done transparently (e.g. by increasing some buffer sizes), whereas coarse corrections such as re-negotiation are delegated to the flow management plane.

- The flow management plane is responsible for flow establishment, QoS negotiation and QoS re-negotiation. Additionally, the flow management implements QoS mapping.

QoS-A aims at providing an enhanced transport service for point-to-point applications in the first place. It provides limited support for point-to-multipoint applications. QoS-A requires that all receivers are known at start-up time. Additionally, all receivers experience the same QoS, no heterogeneity is allowed. These limitations simplify flow establishment which can be done similar for multicasting and unicasting. QoS negotiation is done during flow setup. It is based on the concept of resource server. Resource servers manage network resources of endsystems and intermediate systems inside a management domain, e.g. inside a part of the network. The sender issues a flow establishment message to the resource server, which conducts admission tests for the network resources inside a domain. If the test is successful, it multicasts the resulting QoS to the receivers, which conduct admission control on the transport layer. On the orchestration and distributed platform layer, no resources are reserved and thus no admission control needs to be
carried out. The resource server collects confirmations of the receivers. If the admission test was successful on all endsystems, the flow is established. QoS re-negotiation is carried out similar to the initial QoS negotiation. It is initiated by the sender or any receivers by sending a maintain message to the resource server, which tries to incorporate the changes.

In the QoS-A, the application requirements are described using frame size, frame rate as well as end-to-end delay. For commonly used media types such as standard video, HiFi audio and voice audio, predefined applications requirements exists. The flow management plane contains resource calculation function which are used to calculate the amount of buffer space as well as the scheduling parameters. A QoS mapping function translates the QoS parameters to network QoS parameters such as bandwidth and delay bounds.

The QoS-A maintains a flow scheduler at the transport level. It schedules frames according to a pre-determined endsystem delay. Frames are queued according to class and deadline. The admission controller limits the number of flows and thus reserves endsystem resources. A monitor examines the current state of the framework. Small performance degradations are corrected transparently by fine-tuning the scheduler or the buffer management.

The QoS-A has been implemented on top of the real-time capable micro-kernel operating system Chorus. Chorus has been extended by so called real-time handlers which allow to call application code by the system's real-time scheduler. Thus, protocol execution times can be guaranteed.

### 4.2.2 Omega

Omega is an architecture for orchestrating resources in communicating endsystems [Nahrstedt et al. 95 B]. The design is based on a set of layers, where the services at each layer are described using QoS parameters. It consists of an application layer and a transport layer, each implementing a single protocol. The application layer interfaces either directly to multimedia devices or alternatively to the application. It uses and controls the transport layer, which is responsible for connection setup, error correction, and rate control. The transport layer itself is based on a network layer that provides end-to-end data connectivity with QoS guarantees. An important entity inside the Omega architecture is the QoS Broker. The QoS Broker is responsible for es-
establishing connections, carrying out QoS negotiation, mapping QoS parameters from layer to layer and orchestrating resources.

The QoS Broker supports point-to-point, point-to-multipoint, and multipoint-to-point communication. The multipoint-to-point communication is used for a receiver that wants to receive data from several senders simultaneously, it is not further discussed here. For the point-to-multipoint communication, a sender distributes data to a fixed group of receivers, e.g. the participants must be initially known. QoS negotiation is part of the connection setup. Its purpose is to reserve resources in the endsystems and the network. The sender initiates the negotiation. In the terminology of the QoS Broker, the sender is the buyer that wants to buy resources, whereas the receivers are the sellers of resources. The negotiation is done on a layer-per-layer basis and contains the following steps:

- An admission test in the application layer checks, whether enough resource are available on the local endsystem.
- The receivers are prompted to conduct an admission test in their application layer and report the results back to the buyer. Receivers may use local filters for downgrading media streams.
- An endsystem admission test on the transport layer is first conducted locally and then on all receiving endsystems, analogous to the application layer.
- Finally, the buyer delegates the admission test for network resources to the network layer, which is outside the scope of the QoS Broker.

The layered negotiation requires that QoS parameters are mapped from layer to layer. After the negotiation on a layer is completed, the maximum possible QoS that may be supported by all receivers for this layer is mapped to the next lower layer. The parameters on the application layer are media specific. The QoS-Broker offers a tuning service that allows the user to watch a series of audio/video clips through a graphical user interface. During the presentation, the user is allowed to change the so called perceptual QoS by adjusting parameters such as playback rate, frame size etc. The users selection is then used as the application requirements. The application layer as well as the transport layer map the QoS parameters to resource usage figures, which are used for the admission control of endsystem resources.
QoS re-negotiation is supported during the transmission. Due to the fact that the required resources depend non-linearly from all QoS parameters, the QoS Broker only allows to change a single parameter at once. The re-negotiation as such is then carried out similarly to the initial QoS negotiation.

Omega enforces endsystem QoS using a real-time scheduler. The scheduler controls the tasks in the application and transport layer, which are non-pre-emptive and periodic. On each invocation, a task carries out a single step in the processing of the protocol for a certain connection. The scheduler itself as well as the two communication layers are realised as a single, fixed-priority real-time process on top of UNIX SVR4. QoS Broker's scheduler is aware of the dependencies that exist between application tasks and transport tasks. It calculates a semantically correct schedule for all connections based on the application requirements. The calculation of the schedule is based on execution time measurements that are stored off-line in a profile database. Since the scheduler's calculation is based on worst-case figures, no monitoring or fine-tuning is needed during the transmission.

4.2.3 Da CaPo++

Da CaPo is a communication framework intended for point-to-point multimedia applications [Plagemann et al. 94 B], [Plagemann 94 C], [Vogt et al. 93]. Da CaPo protocols cover the OSI layers three to seven and are configured at run time. Applications specify their requirements in form of weighted QoS parameters. Da CaPo uses QoS parameters of both peers to calculate a conforming protocol [Plagemann et al. 94 A]. A configured protocol bridges the functionality gap that exists between the application and the underlying network. It consists of modules that implement protocol mechanisms. Da CaPo offers a high flexibility due to the fact that any imaginable protocol mechanism can be integrated, e.g. the classical protocol mechanisms such as error control, segmentation and reassembly, as well as network security mechanisms such as encryption, decryption and authentication or multimedia support mechanism that interface directly to multimedia devices.

Da CaPo++ [Stiller et al. 96] extends the Da CaPo framework with a set of application components that run on top of the core communication framework. These components extend the communication to an application framework for multimedia applications. They include building blocks such as
picture-phone, X-window sharing and elements for the integration of multimedia streams into web servers and web browsers. The Da CaPo core system itself is extended with security and multicast capabilities.

Da CaPo++ offers unidirectional point-to-multipoint multicasting based on any multicast capable network infrastructure. Da CaPo++ supports dynamic join and leave of receivers. A new receiver joins by requesting the protocol definition of the configured protocol from the sender. Each receiver instantiates the same protocol. Consequently, no heterogeneity is supported.

The Da CaPo as well as the Da CaPo++ framework let the application specify application requirements in form of attribute names, weight functions and knock-out values. The configuration process selects the modules such that the weighted attribute values are maximized. The configuration process does not examine the semantics of the attributes, it operates on pattern matching only. Therefore, new attributes can be as easily integrated into the framework as new modules are being developed. QoS mapping functions are implemented in the modules and applied during the configuration process. For multicast connections, the protocol configuration is carried out by the sender and then distributed to joining receivers.

In order to maintain the requested quality for a connection, Da CaPo and Da CaPo++ monitor the configured protocol and compare the values with the application requirements. If the quality falls below a threshold value, the protocol is reconfigured. This means that new modules or new mechanisms are selected such that the requested quality can be maintained. If the reconfiguration is not successful, the application is aborted as soon as the values fall below a predefined knock-out value.

QoS re-configuration is either triggered by the monitor or by the user. In both cases, a new protocol is computed. QoS re-negotiation for multicasting in Da CaPo++ may be triggered only by the sender. The new protocol is computed based on the changed application requirements and then distributed to the receivers.
4.2. Communication Frameworks

4.2.4 QoS Framework

Washington University’s QoS Framework aims at providing QoS guarantees in endsystems [Gopalakrishnan et al. 95 A]. The framework consists of four components: QoS specification, QoS mapping, QoS enforcement and protocol implementation model. As such, the framework does not provide communication services to the application, it merely supports the development of applications that by themselves implement communication protocols. Consequently, peer-to-peer QoS negotiation and re-negotiation is outside the scope of the QoS Framework, the framework merely supports layer-to-layer negotiation.

Applications specify their requirements using one of three classes. The isochronous class offers the attributes frame size, frame rate and delay, the burst class offers total size and bandwidth and the low delay class offers message size, delay and message rate. The QoS parameters are used to calculate endsystem resource figures and network bandwidth. Endsystem parameters include scheduling parameters and buffer space requirements.

The main focus of the framework is on QoS enforcement, which is done using so called real-time signals. Real-time signals are a replacement for real-time threads. The processing of real-time data is done in handlers which are triggered by real-time signals. These real-time upcalls are scheduled using a non-pre-emptive rate-monotonic scheduler that was added to the UNIX kernel. Protocols are implemented at the application level, with the protocol functions being executed in the handlers. A shared memory mechanism is used for moving the data directly from the network driver to the application level protocol.

Since protocols are implemented in application space, it is also possible to implement multicast communication protocols. However, as with the other protocol implementations, the QoS negotiation is left to the application and not directly supported in the framework.
4.3 Comparison and Conclusion of the Related Work

Table 4.1 gives an overview of the communication frameworks discussed in this chapter. The QoS Framework takes a special place among the presented approaches. All other frameworks provide a transport service to the application or even transport multimedia data directly between multimedia devices. The QoS Framework, however, merely enables the implementation of application layer multimedia protocols with endsystem guarantees. However, it is an interesting approach that can be used to build any multipoint, multimedia application. Therefore, the type of the multicast communication, multicast dynamics as well as QoS re-negotiation depend on the application that is implemented on top of the QoS Framework.

<table>
<thead>
<tr>
<th>Framework</th>
<th>Type of Multicast Communication</th>
<th>Multicast Dynamics</th>
<th>Heterogeneous Receivers</th>
<th>Transparent Application Requirements</th>
<th>QoS Re-negotiation</th>
<th>Endsystem QoS Maintenance</th>
</tr>
</thead>
<tbody>
<tr>
<td>QoS-A</td>
<td>Point-to-multipoint</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes, real-time handlers</td>
</tr>
<tr>
<td>Omega</td>
<td>Point-to-multipoint</td>
<td>Yes</td>
<td>Yes</td>
<td>Only easing of QoS</td>
<td></td>
<td>Yes, meta-scheduling</td>
</tr>
<tr>
<td>QoS Framework</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Yes</td>
<td></td>
<td>Yes, real-time handlers</td>
</tr>
<tr>
<td>Da CaPo++</td>
<td>Point-to-multipoint</td>
<td>Yes</td>
<td>No</td>
<td>Yes, Sender initiated</td>
<td></td>
<td>Monitoring, no guarantees</td>
</tr>
<tr>
<td>MCF</td>
<td>Multipoint to multipoint</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes, meta-scheduling</td>
</tr>
</tbody>
</table>

Table 4.1: Summary of related work
4.3. Comparison and Conclusion of the Related Work

All presented approaches provide basic support for point-to-multipoint multicasting. The Omega architecture also supports multipoint-to-point communication, e.g. a receiver that wants to receive data simultaneously from several senders. However, the most general case of multipoint-to-multipoint communication is not supported by any of the frameworks.

Multicast dynamics is only supported in Da CaPo++. This is a surprising fact, since network level and resource reservation protocols such as ST-2+, RTIP/RCAP and RSVP support multicast dynamics.

Support for heterogeneous receivers are provided by Omega using filters in the endsystem. It is assumed that support for heterogeneous receivers will increase as soon as more suitable encoding techniques for audio and video are applied. The current filters are used mainly for video data, where whole video frames are dropped. However, more sophisticated encoding schemes allow not only for a reduction of the frame rate, but for a graceful degradation of the picture quality, e.g. sharpness, number of colours etc.

The QoS specification in all of the presented frameworks is suitable for continuous media data streams. The most sophisticated approach can be found in the Omega framework, which presents test video and audio clips to the user in order to let him specify the QoS parameters interactively.

QoS re-negotiation is also supported in all of the mentioned frameworks.

The frameworks solve endsystem QoS enforcement using three different approaches. The QoS-A and the QoS Framework use real-time handlers based on the operating systems real-time scheduler. Whereas QoS-A is implemented on top of a real-time operating system, the QoS Framework itself extends the UNIX kernel with a real-time scheduler. The Omega architecture relies on the real-time capabilities of UNIX SVR4. It implements a deadline driven real-time scheduler 'inside a single process. The UNIX scheduler acts as a metascheduler for Omegas scheduler. The Da CaPo++ approach uses a monitor that allows the framework to react upon QoS violations.

The summary of the related work shows that no communication framework exists that is able to provide the whole range of functionalities required by multipoint, multimedia applications. It is the claim of MCF to provide all the necessary functions.
Chapter 5

Multipoint Communication Framework Design

This chapter presents the design of the Multipoint Communication Framework (MCF). The design consists of two parts, the multipoint communication and the quality of service architecture. Both parts reflect fundamental concepts which are needed to fulfil the application requirements which have been identified in Chapter 1. The first section presents those basic concepts. Section 5.2 presents the communication architecture and Section 5.3 explains MCF’s QoS handling.
5.1 Concepts

MCF provides support for multipoint applications that require performance guarantees. The requirements, which have been identified in Chapter 1, determine the concepts on which MCF is based. In the following sections, the foundations for the multipoint communication service and the quality of service handling are presented.

The multipoint communication service has been designed to comply to the following requirements:

- **Multipoint-to-Multipoint Communication**
  Data are distributed by several senders to groups of several receivers.

- **Multipoint Dynamics**
  Participants may join and leave during the communication.

- **Scalability**
  Data may be distributed to hundreds of receivers.

The quality of service concept is a result of the following requirements:

- **End-to-End Performance Guarantees**
  Performance is guaranteed on the whole path from data source to data sink.

- **Transparency**
  Applications specify the required service quality according to the quality of the media that is perceived on the receivers. Thus, applications do not have to know internal mechanisms or algorithms used inside the framework.

- **Adaptable Media Quality**
  Applications may change the quality of the service during the communication using QoS re-negotiation.

- **Heterogeneity**
  Receivers that do not possess enough resources to process incoming data with the required quality may locally downgrade the perceived quality by using filters on the data.
5.1. Concepts

5.1.1 Environment

MCF extends the basic communication services offered by the operating system and adds multipoint communication with performance guarantees. The resulting service enables applications to distribute multimedia streams among a group of workstations. In order to do so, MCF interacts with multimedia devices, endsystem resources and the basic communication services of the operating system. Figure 5.1 shows how MCF is embedded in a workstation environment. MCF consists of two parts, the transport subsystem and the control and management subsystem. The transport subsystem executes the protocols which are used to transport multimedia and also control data. It fetches continuous media data from multimedia devices and forwards them over the network. Continuous media data are not processed by applications, they merely control the behaviour of MCF. Applications, however, may themselves produce data which are then forwarded over the network. The control and management system builds the control interface to the application, it steers the transport subsystem and carries out QoS handling.

5.1.2 Multipoint Communication

Multicast associations in MCF are called flows. Flows provide multipoint-to-multipoint communication with dynamic join and leave capabilities. Flows enable senders to distribute data to a group of receivers. If a receiver joins a flow, it will receive data from all senders that send to the flow. Similarly, if a receiver leaves a flow, it will stop receiving data from any of the senders. If a sender joins a flow, it will automatically reach all attached receivers.

A participant that wants to join a flow needs a description of the flow. This flow description contains an identification of the protocols, the QoS parameters and the network address that are used. Participants that join a flow need to know where the flow description is accessible, a well-known access point is needed. Since participants of a flow may constantly change due to multicast dynamics, they can not be used as access points. For the administration of flow descriptions, a distributed information base, called Group and session Management System
Chapter 5. Multipoint Communication Framework Design

Figure 5.1: MCF environment

(GMS), is used [Wilde et al. 96], [Wilde 97]. The first step in setting up a flow is storing its description in the GMS. This operation is called flow creation. After a flow has been created, its description can be accessed from the GMS by any endsystem running MCF. The join operation consists of fetching the desired flow description and instantiating the flows according to this description.

GMS is modelled as a directory service. The architecture of GMS is shown in Figure 5.2. Flow descriptions are stored in the GMS System Agents (GSA). The GSAs form a fully distributed database that communicate using the GMS System Protocol (GSP) [Wilde 96 B]. MCF accesses the GMS using a GMS User Agent (GUA). The communication between GUAs and GSAs is carried out over the GMS Access Protocol (GAP) [Wilde 96 A].

MCF uses three different communication types, as shown in Figure 5.3 and described below:
5.1. Concepts

Figure 5.2: The GMS architecture [Wilde et al. 96]

- Multimedia data are transported using unidirectional flows. The participants execute a multimedia protocol stack that transports data from the senders to a group of receivers. The protocols allow that a participant joins a multimedia flow simultaneously both as sender and receiver.

- Control data, which is needed to carry out QoS re-negotiation, is transported using a separate flow. The control flow is unidirectional only. Participants join the control flow both as senders and receivers such that control data is exchange symmetrically among all participants. The participants execute a control protocol stack to transport control data.

- Data exchange with the GMS system is done by the GMS user agent which uses bidirectional point-to-point connections to the ‘nearest’ GMS system agent. The GMS connections are used to access flow descriptions which are stored in the GMS system.

5.1.3 Quality of Service Concept

The main goal of the quality of service concept in MCF is to offer end-to-end performance guarantees. End-to-end guarantees mean that data from a source to one or more sinks are transported with given guarantees, e.g. that data arrive at the sinks before a given deadline. Continuous media streams origin from a multimedia device such as a digital camera or a microphone attached to an audio device. Data are output on another device such as a frame buffer or a loudspeaker attached to
an audio device. Consequently, end-to-end for continuous media spans the path from source device to sink device and thus also includes the presentation of data to the user. For other media types such as texts and graphics which are fetched from the application, the end-to-end guarantee provided by MCF covers the transport of the data from application interface to application interface. In these cases, the quality which is offered to the user depends on how the application presents data to the user.

MCF offers hard guarantees for the data transport. End-to-end data transport consists of protocol execution in the senders and receivers as well as transportation over a network. The guarantees are provided by
exclusively reserving all resources that are needed for each flow. Endsystem resources are reserved by MCF, whereas the reservation of network resources is delegated to the network. The endsystem resources managed by MCF are CPU, memory and multimedia devices. Access to the CPU is controlled by a real-time scheduler that works on periodic tasks. The scheduler guarantees that protocol execution for each flow meets the required deadlines. Memory is reserved by assigning blocks of physical memory to the exclusive use of a flow. Devices are also exclusively assigned to flows.

It is assumed that the operating system provides adequate support for the reservation of endsystem resources. MCF's real-time scheduler requires access to the CPU within a guaranteed maximum time. Such guarantees are provided for example by the fixed-priority real-time scheduling1 in UNIX® SVR4 [Goodheart et al. 93]. Furthermore, the reservation of memory must be done on physical memory rather than on virtual memory, since access times to virtual memory can not be guaranteed due to swapping or demand paging.

MCF assumes that the network provides bandwidth and delay guarantees. The network must be able to offer a requested bandwidth from a sender to a group of receivers. Additionally, it has to provide an upper limit on the delay. MCF also assumes that the network does not weaken given guarantees.

The reservation mechanisms applied in MCF are completely transparent to the applications. Applications do not specify the resources that should be reserved but rather describe the characteristics of the data that is transported. Application requirements are therefore always specific to the media type. For an audio flow, the application requirements consist of QoS parameters specifying sampling rate, sampling precision, number of audio channels (i.e. mono or stereo) as well as end-to-end delay. MCF maps the QoS parameters specified in the application requirements to internal representations which are then used to calculate the needed resources. In particular, the application specified QoS

---

1. UNIX® SVR4 real-time scheduling may not be directly used for scheduling multimedia protocols, as has been shown in [Nieh 93 et al.]
parameters are also used to derive the service quality which has to be requested from the network.

During the communication, the QoS of flows can be changed by the application using QoS re-negotiation. QoS re-negotiation is used to change the QoS of several flows simultaneously. Applications assign priorities to flows and QoS parameters. As a result of the re-negotiation, the 'most appropriate' QoS for each flow that is supported by all participants is selected. QoS re-negotiation is mainly useful if resources are scarce. Applications use QoS re-negotiation to control how resources are assigned to flows, as shown in the following example. A conferencing application uses a video and audio flow to implement video conferencing and also provides a shared whiteboard application that is used to exchange drawings and graphics among the participants. At the beginning of a virtual meeting, mainly high quality video and audio are being used. In a later stage, the shared whiteboard and high quality audio should be used. The application then carries out QoS re-negotiation, where the audio flow and the flow used for the whiteboard are assigned a higher priority than the video flow. As a result of the re-negotiation, the quality of the video will be reduced, thus freeing resources which then are used for the whiteboard application. If plenty of resources are available, QoS re-negotiation is not needed. Then, all flows will be executed with the highest quality possible.

The QoS selected for a flow is valid for all participants. In some cases, however, applications may allow that individual receivers use a lower QoS if they do not possess enough resources. In such a case, the QoS specified in the application requirement may be locally downgraded. Downgrading only makes sense for continuous media. There, some data may be filtered out while the remaining part still produces a usable result. Filtering thus reduces the amount of data that has to be processed in the receiver and thus reduces the amount of resources that have to be allocated for the processing.
5.2 Data Transport in MCF

Data are transported using a set of protocols. The transport subsystem (TSS) provides an environment for running both multimedia and control protocol stacks. For the design of the TSS, the following design goals have been applied:

- **Flexibility**
  Any media type that is supported by MCF may be transported over any supported network.

- **Extensibility**
  New media types and support for new networks may be integrated into MCF without having to change existing applications.

- **Performance Guarantees**
  The TSS guarantees an upper limit on the execution time of protocols.

- **Efficiency**
  Protocol execution is efficient enough for transporting continuous media streams such as video or audio.

5.2.1 Protocol Layering

In order to provide a flexible and extensible transport environment, no static protocol stack can be used. Therefore, the TSS allows for dynamic composition of protocol stacks. Each protocol stack is divided into three layers. Each layer implements a protocol that provides its own type of service. In the multimedia protocol stack, the three layers are labelled *multimedia support, transport* and *multicast adaptation*. The control protocol stack differs from the multimedia protocol stacks only in the topmost layer which is called *control processing layer*. Figure 5.4 shows the layering for data and control protocol stacks. Both stacks are being executed by the transport subsystem using the same mechanisms.
The layers in MCF protocol stacks have the following functionality:

- **Multimedia Support**
  The multimedia support layer builds the interface to multimedia devices or alternatively to the application. The multimedia layer is source and also sink for all data that is transported. As a source, the layer generates a stream of multimedia frames at a fixed rate. Multimedia frames are containers that are used to carry any type of multimedia data. Each frame carries a timestamp, such that it can be presented correctly at the sinks. As a sink, the layer is responsible for the presentation of the multimedia data.

- **Control Processing**
  The control processing layer implements the QoS re-negotiation protocol. The re-negotiation protocol is based on QoS handling and resource functions that are provided by the control and management system.

- **Transport**
  The transport layer implements end-to-end protocol functions which enhance the basic communication service offered by the network. Examples of such functions are segmentation and reassembly or error control functions such as forward error control or receiver initiated error control. The error control functions are, for example, needed in the control protocol stack, since the QoS re-negotiation protocol needs a reliable transport service.

- **Multicast Adaptation**
  The multicast adaptation layer builds the interface to the underlying network. It provides multipoint-to-multipoint connectivity.
to the transport layer. If the underlying network only supports point-to-multipoint multicasting, the multipoint-to-multipoint connectivity is emulated using several point-to-multipoint connections.

### 5.2.2 Protocol Composition

Each layer consists of a set of *modules* that implement the layers' services. For each media type, one or more\(^1\) modules exists in the multimedia support layer. On the transport layer, modules implement transport protocol functions. The multicast adaptation layer contains a module for each network type. A protocol stack is composed using a single module per layer. Thus, a protocol stack is defined by a module triple. Figure 5.5 shows an example of a module triple that is used for live audio transmission. On the multimedia support layer, the audio module is selected which interfaces to the microphone and the loudspeaker. On the transport layer, the segmentation and reassembly module is chosen which forwards or receives its data over the ATM/AAL5 module on the multicast adaptation layer. Modules are designed following an object-oriented approach, modules on each layer offer the same interface and are completely exchangeable. The protocol stack is set up by dynamically instantiating the modules.

Modules which constitute a flow's protocol are defined by the application that creates the flow. The module triple is part of the flow description which is stored in the GMS. Defining a module triple requires basic knowledge of transport protocols and each module's functionality. Application programmers without this knowledge may use pre-defined protocol stacks, called *flow templates*. For each supported media type, a flow template exists that contains a pre-defined stack. Flow templates are stored on the GMS (see [Wilde 97]).

### Autoconfiguration

After a protocol stack has been composed, the modules need to adapt themselves to each other. This process is called *autoconfiguration*.

---

1. For each multimedia device, a separate module is needed. For example, an audio module may fetch data from microphone while another module uses the line-in audio device.
During the autoconfiguration, modules request the properties of the module directly below it, i.e. the multimedia support module queries the transport module, the transport module requests properties of the multicast adaptation module. An examples of autoconfiguration work as follows: The transport module offers segmentation and reassembly. In order to segment the multimedia frames into packets, the transport module needs to know the Maximum Transfer Unit (MTU) that the multicast adaptation module is able to transport. During autoconfiguration, the transport module requests the MTU size of the multicast adaptation module. By considering the MTU, the transport module generates the largest possible packets. Autoconfiguration is also needed to disable unnecessary functionality, as in the following example: The transport module implements FIFO ordered packet delivery by sorting incoming packets. If the multicast adaptation module interfaces to a network that already provides ordered packet delivery, then the transport module disables its ordering algorithm and directly forwards packets to the multimedia support module.

The functionality of autoconfiguration is limited by the information that is exchanged among the modules. This information, which describes the properties of modules, needs to be independent of any par-
5.2. Data Transport in MCF

ticular module. Consequently, only general information such as the MTU or the ordering relation can be exchanged.

5.2.3 Data Paths and Module Interface

Flows transport multimedia frames or application generated data from senders to receivers. Besides this payload data, flows are also used to transport intra-flow control data. Unlike payload data which is transported from senders to receivers only, intra-flow control data are sent in both directions. An example of intra-flow control data are negative acknowledgements in a receiver-initiated error control scheme which are transported from a receiver back to a sender or keep-alive packets which are transported from a sender to its receivers. The directions of the data flow are called main path and back path respectively. Figure 5.6 shows the main path from sender to receivers and the back path from receiver to sender. The main path is used for both payload and intra-flow control data which are multicast from a single sender to all receivers. On the back path, control data are unicast from a receiver to a single sender.

Payload data origin on the multimedia transport layer, whereas intra-flow control data are generated and also consumed on the transport layer, only.

Multimedia frames and control packets that are sent on the main path carry an identification of the sender. This identification might be a network layer address or a connection identifier, depending on the network that is being used. Identifiers are used by receivers to send control data on the back path to a particular sender.

Inside endsystems, data are processed and forwarded by modules. Each module offers a simple interface for the processing of data, for both sending direction and receiving direction (see Figure 5.7). The interface provides a set of module functions that are used to access the module's protocol function. The interface is the same for all modules. However, not all module functions are needed by all modules on all layers. For example, multimedia support modules do not generate data on the back path.
Data in a flow are processed by calling the module functions of each module on each layer. The sequence of these calls is determined by the modules themselves. Each module that is called carries out the required processing and then triggers\(^1\) the next module call. The semantics of the offered module functions is as follows:

- **Request Main**
  The module is requested to process data in the main path. On the sending side, the module prepares data for sending over the network, on the receiving side, data are processed and forwarded towards the multimedia device or the application.

- **Indication Main**
  The module is informed that the next module on the main path has completed the processing of previously forwarded data. In-

---

\(^{1}\) The actual module call is done by the real-time scheduler.
5.2. Data Transport in MCF

dication main is used if a call to request main on one layer leads to several calls of request main on the next layer. An example is the segmentation of multimedia frames, where a call of request main for the transport module leads to several calls of request main for the multicast adaptation module. Each time the multicast adaptation module finishes processing, indication main of the transport module is called such that the next fragment can be forwarded.

On the back path, control messages, which are small in size, are transported by a single call per layer. Consequently, no indication back exists.

- **Request Back**
  The module is requested to process data on the back path. On the receiving side, the module prepares data for sending back to a sender, on the sending side, data are forwarded to the transport layer or processed on the transport layer.

```
Request main

Sending side

Indication main

Request back

Module

Receiving side

Request main

Indication main
```

Figure 5.7: Module interface

MCF's data path and module interface design has been derived from concepts that have been introduced in the Da CaPo framework [Plagemann 94 C]. Protocol stacks in Da CaPo contain an arbitrary number of modules, whereas MCF protocol stack always contain exactly three modules. MCF therefore is able to use a simpler design. In particular, control data in MCF are only produced on the transport layer, where they are also consumed. This simplifies the calculation of a correct module call sequence.
5.2.4 Protocol Execution

Protocols are executed by the runtime environment which is part of the transport subsystem (TSS). This environment is shown in Figure 5.8. It consists of a scheduler that activates protocol processing by calling the module functions, a timer management that notifies modules if a timer expires and a buffer management which implements zero-copy data forwarding among the modules.

![Figure 5.8: TSS Runtime Environment [Keller 96]](image)

The most prominent part of the transport subsystem is the scheduler. The scheduling algorithm has been chosen based on the following criteria:

- The model on which the scheduling algorithm is based has to fit the requirements of the transport subsystem.
- An admission test for the algorithm exists that decides whether a new flow can be admitted. A new flow may be admitted only if all flows in the system can be processed within their deadlines.
- The algorithm must allow for a simple and efficient implementation.
Scheduling Model and Requirements of the Transport Subsystem

Scheduling in communication systems can be done either on tasks that execute protocols or on packets that are being selected for a service. Both approaches can be used to guarantee an upper bound on the delay. A task scheduler guarantees that tasks are being processed within a given deadline, whereas a packet scheduler guarantees that packets gain access to a service within a given time.

MCF uses a task-based real-time scheduler that executes instantiated protocols such that a guarantee on the processing delay can be given. The scheduler executes module functions which are used to gain access to protocol functions. The main focus of the scheduler is to control the access to the CPU such that tasks are executed within their deadlines (see Figure 5.9). Each task processes a single packet or multimedia frame upon each invocation. By guaranteeing a delay bound for the execution of tasks, MCF’s scheduler also guarantees a delay bound for the processing of packets.

Packet schedulers are used at queuing points (i.e. between layers) to select packets for a service. They are most often used in the data link layer for selecting packets of different flows that are then forwarded on the physical layer (see Figure 5.10). In the data link layer, packet schedulers fulfill the following purposes [Bennett et al. 96 A]:

- Allow for bandwidth reservations and bandwidth guarantees on a per flow basis.
- Provide a guaranteed queuing delay for traffic which is constrained by a leaky bucket.
• Implement a 'fair' access to the network for flows without bandwidth reservations and traffic constrains.

Both packet scheduling and MCF's task scheduling can be used to process packets within a specified delay bound. However, the two methods are designed for different scenarios. Packet scheduling is intended to control access to a single service, for example a transmission link. Packet schedulers assume that packets of the same length are served within the same time. In MCF, packets of different flows are processed by different modules and thus experience a different service time. Therefore, packet schedulers are not applicable for protocol processing in MCF.

Nevertheless, packet schedulers are needed to implement protocol functions in the multimedia support layer. Multimedia support modules in receiving flows are used to forward multimedia frames to multimedia devices. These devices offer a single service, for example the playback of audio samples. Packet schedulers are used to control the playback time of multimedia frames by smoothing out delay variation effects that have been introduced by the network or by the task schedulers. These variations or jitter effects arise since packets may be transmitted or processed faster than the guaranteed delay bound. Packet schedulers in the multimedia support layer delay packets until they reach the guaranteed end-to-end delay. Thus, by guaranteeing an upper bound on the delay and by delaying packets which are too fast, a correct timing is provided as it is needed for continuous media [Anderson 93].

Figure 5.11 shows how packet and task scheduling are being used in endsystems that communicate using MCF. The left side of the picture
5.2. Data Transport in MCF

Figure 5.11: Packet and task scheduling shows an endsystem that sends audio and video data using two MCF flows. The task scheduler, shown as circular arrow, executes the modules which fetch multimedia data from camera and microphone and carry out the protocol processing. After being processed inside MCF, the packets are forwarded over a network that provides bandwidth and delay guarantees. A packet scheduler, for example an ATM cell scheduler on an ATM network adapter, controls the access to the network. This packet scheduler is not part of MCF. On the receiving endsystem, the multicast adaptation layer fetches packets from the network layer. As in the sending endsystem, MCF’s task scheduler executes the modules. On the multimedia support layer, a packet scheduler forwards the processed packets to the multimedia devices such as the audio and video device. This packet scheduler decides on the correct playback time of the multimedia frames.

In order to be able to guarantee deadlines, an admission test is used which decides whether enough CPU resources are available for new tasks. Schedulers that provide an admission test work on periodic tasks. For periodic tasks, the execution times can be predicted and thus
a decision can be made whether all deadlines can be met in the future. The task model used for real-time scheduling algorithms is shown in Figure 5.12. Each task is characterized by its starting time \( s \), the worst case execution time \( e \), its deadline \( d \) and the period \( p \).

![Task model](image)

**Figure 5.12: Task model [Leung et al. 80]**

MCF uses a *non-pre-emptive rate monotonic* task scheduler. The rate monotonic (RM) scheduler belongs to the class of fixed priority scheduling algorithms. In the RM algorithm, each task keeps its priority as long as the set of tasks which are scheduled does not change. RM assigns the highest priority to the task with the shortest period.

In the following paragraphs, the reasons which lead to the decision of using the non-pre-emptive rate monotonic algorithm are presented. As candidate schedulers, pre-emptive and non-pre-emptive rate monotonic and *earliest deadline first* (EDF) algorithms have been considered. The evaluation is based on results which have been found by Liu and Layland which studied pre-emptive RM and EDF algorithms [Liu et al. 73] as well as by discussions of non-pre-emptive RM and EDF algorithms done by Steinmetz and Nahrstedt [Steinmetz et al. 95 A] and an admission test for the non-pre-emptive RM algorithm found by Nagarajan and Vogt [Nagarajan et al. 93].

Continuous media data such as audio and video are represented as stream of frames which require periodic processing. Protocols that handle continuous media data are therefore very well suited to be executed by a real-time scheduler that works on periodic tasks. However, the TSS is also used to transport other media types such as graphics or texts or control data in the control flow. In general, these data are pro-
5.2. Data Transport in MCF

duced in bursts. By providing an input buffer in the multimedia support module, the burstyness can be smoothed out. The input buffer together with the periodic scheduling form a leaky bucket as shown in Figure 5.13. Thus, a periodic stream of packets results that can be processed similar to multimedia frames.

![Diagram of data bursts, multimedia support module, and periodic scheduling]

Figure 5.13: MMS module as leaky bucket

Scheduling is either pre-emptive or non-pre-emptive. Pre-emptive schedulers are theoretically capable of scheduling a larger set of tasks than non-pre-emptive ones. An implementation of a pre-emptive scheduler, however, has to take into account the following facts:

- Pre-emptive task switching requires that for each task its local state of execution, i.e. registers and program counter, must be saved and restored. Since this operation takes time, task switching is not done at every possible time, but only within a certain granularity. In the Solaris\textsuperscript{TM} operating system, this granularity ranges from 40 milliseconds to 200 milliseconds [Sun 95 A].

- Each task that is pre-empted must be capable to resume its execution. This means that it must not be interrupted during a critical operation.

- Variables and data structures that are shared among several tasks lead to mutual exclusion problems whose solution may cause priority inversions.

In MCF, a non-pre-emptive scheduler is used for the following reasons: Firstly, a non-pre-emptive scheduler allows for a simple implementation with low overhead, whereas pre-emptive scheduling is more complex. Secondly, the execution times of modules is small compared to the deadlines. For example, the execution times of all modules of
Chapter 5. Multipoint Communication Framework Design

the prototype implementation is smaller than 5 millisecond on an average fast workstation, as will be shown in Chapter 7. This is as small or smaller than the time granularity used in existing pre-emptive schedulers. Nevertheless, some module such as the compression of video images in software might have large execution times. Modules that have large execution times can be processed in several steps. These modules voluntarily have to yield the CPU before having completed their task. Since the modules yield the CPU themselves, no inconsistencies occur. Furthermore, the complexity of storing the current state of execution is capsulated inside these modules such that the scheduler can be kept simple and efficient.

Scheduling Admission Tests

A key requirement for a real-time scheduling algorithm in MCF is a suitable admission test. The admission test for non-pre-emptive EDF is valid only for tasks whose deadlines are longer than the period: $d_i = p_i + E$, where $E$ is the execution time of the longest task in the system [Steinmetz et al. 95 A]. For protocol processing, though, this precondition can not be met, since each multimedia frame must be processed within its period. For the class of fixed priority algorithms such as the rate monotonic scheduling algorithm, Nagarajan and Vogt found an admission test which is valid for a set of $n$ non-pre-emptive, independent, periodic tasks with $0 \leq e_i \leq d_i \leq p_i$ [Nagarajan et al. 93]. The admission test requires that all tasks are ordered according to their priority such that task $m$ has the highest priority and task $l$ the lowest. Then, a schedule for non-pre-emptive rate monotonic exists if for all tasks $i$ the following holds:

$$\max_{1 \leq j \leq n} (e_j) + \sum_{j = i + 1}^{n} e_j \cdot \left(\left\lceil \frac{d_i - e_j}{p_j} \right\rceil + 1 \right) + e_i \leq d_i$$

(5.1)

Figure 5.14 is a graphical representation of the admission test formula. It shows that a task can be executed within its deadline if the longest task in the system, tasks with higher priority and the current task fit into the deadline.
The preconditions of the admission test for the rate monotonic algorithm are fulfilled by MCF’s tasks. The first precondition, which requires that deadlines are smaller than periods, is met since modules in a flow process packets and frames within the period of the flow. Thus, the deadline of these modules is smaller than the period. The second precondition requires that tasks can be ordered according to their priority. The priority is given by the rate-monotonic scheduler which assigns the highest priority to the task with the shortest period. Tasks with the same period are assigned an arbitrary priority. This priority assignment scheme requires that tasks are independent of each other, since the priority assignment does not consider inter-task dependencies. However, modules of the same flow have to be executed in the semantically correct sequence. In the sending direction, the multimedia support module must be executed prior to the transport module which in turn must be executed before the multicast adaptation module. This sequence is called module precedence graph. Modules in the same flow which depend on each other experience the same period. This fact is being used by MCF’s priority assignment which is done as follows:

- The priority of modules is given according to the period of the flow. A smaller period means a higher priority.
- The priority of modules in the same flow is given according to the precedence graph. Priorities are assigned according to the sequence of execution.
- Modules of different flows that have the same period are given priorities according to their instantiation time. A flow which is instantiated earlier has a higher priority.

This priority assignment reflects inter-module dependencies. At the same time, the assignment scheme conforms to the rate-monotonic algorithm.

![Diagram](image)

**Figure 5.15: Assignment of scheduling priorities**

Figure 5.15 shows an example of how scheduling priorities are assigned to modules in two flows. Flow A is used for sending, the precedence graph reflects that the multimedia support module must be called first, then the transport module and finally the multicast adaptation module. Flow B receives data using a reassembly module in the transport layer that reassembles two packets into a single multimedia frame. For this flow, the multicast adaptation module must be called first, then the transport module, then again the multicast adaptation module and the transport module. After having reassembled the multimedia frame, the multimedia support module is called. Since flow A
has a shorter period than flow B, all its modules have a higher priority than the modules in flow B. The resulting priorities are shown as numbers below the modules.

The module precedence graph depends on the end-to-end protocol implemented by the transport module. Multimedia support and multicast adaptation modules process a single packet or multimedia frame per protocol function. Transport modules, however, may require several passes until a multimedia frame is processed. An example is segmentation and reassembly, which results in a precedence graph as shown for flow B in Figure 5.15. The module precedence graph also reflects if the transport module sends intra-flow control data on the back path\(^1\). In these cases, the multicast adaptation module that sends data on the back path is included in the precedence graph. Figure 5.16 shows an example of a module precedence graph for a receiving flow that implements reassembly and sends intra-flow control data on the back path. The first two invocations of the multicast adaptation module and the transport module are used to receive multimedia frames that consists of at most two fragments. Next, the multicast adaptation module that sends the control data on the back path is placed. The multimedia support module that puts the multimedia frames onto the multimedia device is the last element in the module precedence graph.

![Module precedence graph for flow sending on back path](image)

1. Intra-flow control data may only be sent by transport modules, not by multimedia support or multicast adaptation modules.
Simple Implementation

The non-pre-emptive rate monotonic algorithm allows for a simple implementation. Since tasks are never pre-empted, the scheduler does not need to save and restore the execution state of tasks. Instead, tasks are executed by calling high level procedures, e.g. C functions. Additionally, non-pre-emptive scheduling simplifies the implementation of protocols. When using non-pre-emptive scheduling, task switches occur at well defined places. Thus, modules do not need to solve the mutual exclusion problem.

Timer Management

Some transport modules such as the multicast error control transport module require non-periodic timers. Timer management is integrated into the real-time scheduling. Each time a new task has to be scheduled, the scheduler tests whether a timer for a module has expired. The scheduler notifies modules about time-outs by calling the module’s timer handling. The timer handling function inherits the priority of the module that installed them. This means that the timer handling function is called only if no higher priority task is eligible for scheduling.

Modules that use timers have to consider the execution time of the timer handling function in the CPU resource calculation.

Buffer Management

The buffer management provides two services. Firstly, it offers memory for packets that are processed by a flow. During the resource calculation, each module calculates the maximum number of packets that it stores simultaneously. For example, an error control module may hold a log buffer containing packets for possible retransmission, whose size it has to calculate. For each flow, the buffer management allocates a pool of buffers during the resource reservation. The second service of the buffer management is to provide mechanisms for forwarding packets between modules. The buffer management provides a data structure with buffers for header and data part. This data structure also contains a reference counter, such that packets can easily be forwarded per reference.
5.3 QoS and Resource Administration in MCF

QoS and resource administration is conducted by the control and management subsystem (CMS). For this task, the CMS uses control functions which are part of each module. QoS and resource administration consists of several parts. The first element is the specification of application requirements in the form of QoS parameters. These parameters are mapped to an internal representation which is then used to calculate the required resources. Admission tests as the third part checks whether sufficient resources are available before they are being reserved. The QoS re-negotiation, as the last element, is used to change the quality of service of existing flows.

5.3.1 Application Requirement Specification

Applications specify the quality of a flow's service using QoS parameters. Application requirements consist of a description of the media quality that is transported and of media independent parameters. The media quality is described using several QoS parameters, such as frame rate and frame size in the case of video or sampling rate and sampling depth in the case of audio. The media independent parameters are used to describe aspects of the multipoint communication service. They consist of end-to-end delay, maximum number of senders and receivers as well as the maximum number of simultaneously active senders and receivers.

When a flow is created, the application defines a range for each QoS parameter value. By doing so, a QoS parameter space is defined. This space defines boundaries for the actual QoS. Once defined, the QoS parameter space can not be changed. The QoS of the flow is defined by a so called QoS vector which must be part of the QoS parameter space. This vector is called working point. It consists of a single value for each QoS parameter. The initial working point is supplied by the creating application together with the QoS parameter space. The working point can be changed using QoS re-negotiation within the boundaries defined by the QoS space. An example of a simplified QoS parameter space for a flow transporting a video stream is shown in
Figure 5.17. The QoS parameters consist of end-to-end delay, size of the video image and frame rate. By defining a range for each of these QoS parameters, a three-dimensional QoS parameter space is defined. The working point is a vector in this space that consists of a value $d$ for the delay, $s$ for the image size and $f$ for the frame rate ($Q[d, s, f]$).

Besides the working point, the application also defines the range of downgrading for heterogeneous receivers. The area for downgrading defines how much the quality might be reduced by receivers that use packet filters. In the example shown in Figure 5.17, the quality might be reduced by increasing the delay or decreasing the frame rate. The image size, however, must not be changed. Downgrading is mainly useful for non-interactive applications that distribute continuous media streams such as video or audio. The area for downgrading enables the creating application to define for which QoS parameters and to which extent receivers are allowed to downgrade the QoS defined by the working point. If no downgrading is allowed, an empty area has to be specified.
5.3.2 QoS Parameter Mapping and Resource Calculation

QoS parameter mapping and resource calculation are preparatory steps needed for admission control and resource reservation. QoS mapping translates application requirements to internal representations. The mapping is needed due to the layered approach of MCF's protocols. The services offered by each layer are configured using layer specific QoS parameters. The working point specified by the application describes the service quality requested from the multimedia support module. The service needed from the transport module is derived from the working point by translating the QoS parameters to the transport layer. The transport module itself maps the QoS parameters to the multicast adaptation layer. After QoS mapping, the resources needed by each layer are calculated. Using the QoS mapping and resource calculation model introduced in Section 3.2.2 on page 37, the QoS mapping in MCF can be described as follows:

- $\overrightarrow{Q_{MMS}}$ is the working point that describes the QoS parameters at the multimedia support layer. It consists of a media dependent and a media independent part. The parameters of the media dependent part are determined by the multimedia support module. A module transporting audio data accepts QoS parameters for the sampling rate, the sampling size and the number of channels. For video, frame rate and image size can be specified. The media independent parts of the vector describe the end-to-end delay, the delay split factors, the maximum number of senders and receivers as well as the maximum number of simultaneously active senders and receivers. The delay split factors and the maximum number of participants are needed for the resource calculation, as will be explained in the following sections.

- $\overrightarrow{Q_{TP}}$ describes the service at the transport layer, which is independent of the used protocols. The vector elements describe frame rate, frame size and frame ordering. The parameters of $\overrightarrow{Q_{TP}}$ are derived from $\overrightarrow{Q_{MMS}}$.

- The service at the multicast adaptation layer is defined by the QoS vector $\overrightarrow{Q_{MCA}}$. Its elements are packet rate and packet size.
These QoS attributes are independent of any protocol.

- The resource vectors $R_{\text{MMS}}$, $R_{\text{TP}}$, $R_{\text{MCA}}$ describe the resources needed by each layer. Each vector contains the amount of memory and CPU resources needed. Memory is specified by the number of bytes used, CPU usage is given by the flow’s period and deadline as well as by the worst-case execution time of each module. The $R_{\text{MMS}}$ vector includes also the multimedia devices that are reserved by the multimedia support module. Likewise, the $R_{\text{MCA}}$ vector at the multicast adaptation level includes the bandwidth that has to be reserved in the network.

- $M_{\text{MMS}}()$ is the QoS mapping function of the multimedia support layer that maps $Q_{\text{MMS}}$ to $Q_{\text{TP}}$. The mapping function on the transport layer is denoted as $M_{\text{TP}}()$.

- $F_{\text{MMS}}()$, $F_{\text{TP}}()$, $F_{\text{MCA}}()$ are the resource calculation functions that calculate the resource needed by each layer.

Figure 5.18 gives an overview of MCF’s QoS mapping and resource calculation.

![Diagram](image)

*Figure 5.18: QoS mapping and resource calculation*
5.3. QoS and Resource Administration in MCF

QoS Mapping Functions

MCF distinguishes mapping functions at the multimedia support layer and at the transport layer. Mapping functions are part of the modules, since they depend on the service that is provided by each module.

The QoS mapping at the multimedia support layer translates media specific QoS descriptions in transport level QoS parameters. This mapping depends on how multimedia data are transported. As a main task, the mapping at the multimedia support level has to calculate a suitable frame size and frame rate for each media type. For some media such as video, the choice might be obvious. There, each multimedia frame contains a single video image. In case of audio, data are generated in very small samples of either 8 bit, 16 bit or 32 bit, depending on the sampling depth and the number of channels. For reasons of efficiency, several audio samples are packed together into a single audio frame. The size of audio frames depends on the sampling rate, which ranges from 8000 Hz up to 48000 Hz and the amount of time that the sender is allowed to wait until an audio frame is filled. This time is called data acquisition time. It depends on the allowable end-to-end delay which is specified by the application as well as the network delay.

An example of QoS mapping is shown in Table 5.1. The table shows the mapping for a receiver in a video flow.

<table>
<thead>
<tr>
<th>Media Type</th>
<th>$Q_{\text{MMS}}$</th>
<th>$M_{\text{MMS}}(0)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Video</td>
<td>End-to-end delay</td>
<td>deadline = delay \cdot receiver_fraction</td>
</tr>
<tr>
<td></td>
<td>Image length</td>
<td>framesize = length \cdot width \cdot depth</td>
</tr>
<tr>
<td></td>
<td>Image width</td>
<td>period = \frac{1}{framerate}</td>
</tr>
<tr>
<td></td>
<td>Colour depth</td>
<td>ordering = FIFO</td>
</tr>
<tr>
<td></td>
<td>Frame rate</td>
<td></td>
</tr>
</tbody>
</table>

*Table 5.1: Examples of QoS parameter mapping functions*

The QoS mapping functions at the multimedia support layer also calculate the period and the deadline for the flow. These parameters are
layer independent, they are equal for all modules of a flow. For the calculation, the end-to-end delay is used. The end-to-end delay is divided into three parts, namely the processing at the sender, the network transmission delay and the processing at the receiver. Figure 5.19 illustrates the three parts of the end-to-end delay. The delay split factors which are used to split the end-to-end delay into the above mentioned parts are defined once for each flow. The splitting is therefore independent of any particular sender-receiver pair, i.e. the allowed send processing delay is equal for all senders as is the allowed receive processing delay for all receivers. This allows for a separation of the CPU and the end-to-end delay admission test which is needed due to dynamic joins. The splitting factors of a flow are specified by the application\(^1\). Thus, an application can adapt the splitting for a specific situation. For example, if the participants communicate using a local area network, then the factor for the transmission delay can be set to a lower value than if the participants communicate over a satellite link.

---

\(^1\) For applications that do not specify a split factor, a possibly non-optimal default value is being used.
QoS mapping at the transport level depends on the functions provided by the transport module. The mapping functions translate frame rate and frame size to packet rate and packet size at the multicast adaptation level. Two main mechanisms in transport modules make this mapping necessary. Firstly, transport modules may segment frames into several packets. Secondly, transport modules may send intra-flow control data in addition to multimedia frames.

The QoS mapping functions are also used to detect functional inconsistencies among the modules of a flow. A functional inconsistency occurs if a module is not able to provide a requested service, irrespective of the available resources. An example of a possible inconsistency is shown in the following scenario: The multimedia support module requests FIFO packet ordering. The transport module, however, does not implement a sorting mechanism. By inspecting the properties of the multicast adaptation module, the transport module detects that the network does not provide FIFO ordering, either, and thus a functional inconsistency occurs.

Functional inconsistencies occur since MCF uses QoS parameters to describe both functional (e.g. ordering relations) and performance or quality (e.g. frame rate) related service aspects. The separation of parameters into functional and performance related parameters, however, is not always possible. Examples are parameters that describe encoding and compression formats such as JPEG and MPEG encoding for video. At a first glance, these parameters seem to be purely functional. However, the encoding format also affects the quality. JPEG, for instance, uses a lossy compression algorithm which affects the quality of the image. As a result of this considerations and in order to simplify the design, MCF only uses a single type of QoS parameters for both aspects. Besides the fact that function inconsistencies may occur, the mingling of functional and performance related aspects also leads to negative side effects during the QoS re-negotiation. Taking the previous video example, the image quality can not be increased to the maximum level without changing the encoding format. For the highest possible image quality, a loss-less video encoding format has to be used.
Resource Calculation

Resource calculation is done after the QoS parameters have been mapped to each layer. Each module then calculates the resources that it needs using the QoS parameter of its layer. In particular, resource calculation also depends on the number of senders and receivers that may take part in a flow. MCF distinguishes the number of total participants and the number of simultaneously active participants. In an audio conference, for instance, one can assume that at most two participants are talking simultaneously. This means that in this situation, two senders are simultaneously active. CPU resources, for instance, are allocated such that the maximum number of simultaneously active participants can be supported. As with QoS mapping, the details of resource calculation are different for each module. Depending on the resource type, the following calculation methods are used:

- **Multimedia devices**
  Multimedia devices are used by the multimedia support module. Resource calculation for multimedia devices is very simple, each MMS module just states whether a device is used or not.

- **Memory**
  Memory is needed by modules to process incoming data packets and frames. For example, the re-assembly module has to keep buffers for each active senders. Memory is furthermore needed to store the execution state of the module, e.g. to hold local variables. Modules calculate the amount of memory they need based on worst-case assumptions, for example based on the maximum number of participants rather than the average number.

- **CPU**
  The CPU is reserved based on the time a module occupies the CPU during a single period. Each module, therefore, has to calculate the execution time per period. This execution time depends on the endsystem hardware where the module is executed as well as on the QoS the module provides. Since an analytical derivation of the execution time based on the module code is not possible due to its immense complexity, the execution times of the modules are measured and stored in *profile databases*. Each endsystem contains a profile database with measurements for
each module. Each module is analysed concerning the effects of the QoS parameters on the execution time. In the most simple case, the execution time does not depend on any of the QoS parameters at all. An example of such a module is the segmentation module. Irrespective of the QoS parameters, it has to calculate the starting position of the next segment and forward a reference to this position to the multicast adaptation module. Multicast adaptation modules are more complex. The execution time of these modules depends on the packet size, since each packet has to be copied onto the network adapter. However, the execution time depends linearly on the packet size. The profile database for these modules contains a few measurement points for several different packet sizes. Multicast adaptation modules calculate the execution time for the actual packet size using linear interpolation on the stored measurement points, as shown in Figure 5.20. For other modules such as a video module, the execution time depends on several QoS parameters simultaneously, namely on the image size and the colour depth.

- **Bandwidth**
  The calculation of the bandwidth is done at the multicast adaptation layer. The bandwidth is calculated using the maximum packet size and the packet rate. In addition, the per packet overheads introduced by the network has to be considered, too. For example, the multicast adaptation module which interfaces to an
ATM network using AAL5 framing has to take into account the per packet AAL5 framing overhead.

5.3.3 Admission Control and Resource Reservation

Admission control decides whether the demand for resources of a new flow can be satisfied. For each resource type, an admission test is carried out. Admission control only admits flows which pass the admission tests for all resource types. Admission tests for endsystem resource are carried out by MCF. The admission test for bandwidth, being a network resource, is done by the network resource management. The end-to-end delay constitutes a special case, since an admission test is needed although delay is neither endsystem nor network resource.

Endsystem Resources

MCF controls access to endsystem resources by keeping an account for each resource that identifies how much of the resource has been allocated by which flow and how much of the resource is available. The resource account is also used for the decision if a new flow can be admitted or not. Accounting and admission tests are done for the following resource types:

- **Multimedia Devices**
  Endsystem are usually equipped with a single device per media type, for example a video and an audio device. These devices are used exclusively by a single flow. Admission tests for multimedia devices therefore only have to check whether a given device is already used or not. The reservation of multimedia devices is equally simple, the device only has to be marked as used.

- **Memory**
  Memory is allocated in chunks of variable size. Memory admission keeps a record of the amount of allocated and free memory. A request for memory is admitted, if enough free physical memory is available. A flow passes memory admission, if memory for each module has been admitted. The reservation of the actual memory is delegated to the operating system. The operating sys-
tem allocates physical blocks of memory which can not be swapped out to disk. If swapping were enabled, no limit on the execution time of the modules could be given.

- CPU

Admission test for the CPU is tightly coupled with the real-time scheduler. CPU admission is successful, if a valid schedule can be found for all existing flows including the new one. CPU admission uses the scheduling test for the non-periodic rate-monotonic scheduler which is given by Equation 5.1 (see also Section 5.2.4). Each module inside a flow is modelled as a single task. The precedence graph defines the sequence of module calls and also the priorities of the modules. The last module in the graph obtains the lowest priority. In order to test the schedulability of a flow, it is sufficient to test whether the lowest priority module is able to keep its deadline. This is true since all modules in a flow experience the same period and deadline, i.e. the variables $p_i$ and $d_i$ are the same for all modules in a flow. The admission test of lower priority modules differs only in the number of terms that are summed up, i.e. more terms are added to the sum. Thus, the admission test of the lowest priority module is a strengthening of the other admission test. The CPU admission test for a new flow tests whether the lowest priority modules of all flows can be executed within their deadlines. If any flow is no longer schedulable, then the new flow can not be admitted.

$$\max_{1 \leq j \leq n} (e_j) + \sum_{j=i+1}^{n} e_j \cdot \left( \left\lfloor \frac{d_i - e_j}{p_j} \right\rfloor + 1 \right) + e_i \leq d_i$$

(5.1)

Bandwidth

Bandwidth is a network resource, where admission test and reservation are combined in a single operation. The network reserves bandwidth for multicast delivery trees, which connect a single sender with several receivers. The admission test is carried out whenever a new receiver is connected to the tree. Bandwidth allocation involves several endsystems. In MCF, the admission test is controlled by the joining
participant. Depending on whether a new sender or a new receiver joins, MCF uses separate strategies. Bandwidth admission for a sender is successful, if bandwidth to all existing receivers is available. This means that the sender must be able to establish a complete multicast delivery tree to all receivers. The admission test for a new receiver involves all existing senders. There, bandwidth admission is successful, if all senders are able to include the new receiver in their multicast delivery tree. In both cases, the actual bandwidth admission is carried out by network resource managers. The MCF of the joining participant collects the results of these tests and decides whether the admission as a whole succeeds or fails.

**End-to-end Delay**

For the end-to-end delay, an admission test is carried out but no resources are reserved. The end-to-end delay is split into three parts, as shown in Figure 5.19. These parts consist of the processing delay at the sender, the network transmission time and the processing delay at the receiver. A limit on the end-to-end delay is guaranteed, if a limit on all three parts can be guaranteed. Assuming that CPU admission is done before end-to-end delay admission, then the processing delay for both senders and receivers is guaranteed. For the network transmission delay, the network resource managers provide an upper bound on the delay. The admission tests then only as the check whether the sum of the three parts lies within the end-to-end delay bound that was requested by the application.

The admission test for the end-to-end delay involves several end-systems. As with bandwidth admission, the admission test is controlled by the joining participant. A sender that joins tests the end-to-end delay to all its receivers, whereas a receiver checks the end-to-end delay from all existing senders.

**5.3.4 Local QoS Downgrading for Heterogeneous Receivers**

Not all receivers in a flow may be able to support the quality defined by the working point. For continuous media, it is possible to use filters
that reduce the amount of data at the cost of a reduced quality. By reducing the amount of data, fewer resources have to be allocated. Filtering of data always depends on the media and its encoding, see for example [Shacham 92], [Delgrossi et al. 93 B]. In MCF, the encoding of multimedia data is done by the multimedia support module. Figure 5.21 gives an overview on how local QoS downgrading is applied in MCF. The multimedia support modules of the senders classifies multimedia frames using tags. At the receiver side, the multimedia support module instructs the multicast adaptation module to drop packets that carry a certain tag. If the network itself contains filters, then the multicast adaptation module delegates filtering to an intermediate system in the network.

The details of QoS downgrading strongly depend on the encoding scheme used. For video, encoding formats based on wavelet transformations have been proposed. An example is the wavevideo encoding [Dasen et al. 96]. Wavevideo uses a multi resolution representation of individual video images. Each resolution level is encoded into separate data packets. These data packets form a hierarchy, where high resolution data depend on low resolution data. For the decoding, only the lowest resolution data are mandatory. Higher resolution data are used to increase the quality of the decoded image. For the implementation
of local QoS downgrading, the multimedia support module tags the
data packets according to their resolution level. Receivers select a res¬
olution level and instruct filters to remove packets which belong to a
higher resolution level. Thus, the amount of data that has to be proc¬
essed in a receiver is reduced.

Filters directly affect the QoS at the multimedia support level of the
receiver. The QoS is degraded in discrete steps for each configuration
of the filter. In the previous example, the video filter affects the QoS
parameter ‘picture quality’. The rest of the QoS parameters remains
unaffected. Filters are configured during the join operation of the re¬
ceiver. The receiving multimedia module provides a list of filter con¬
figurations which is sorted by decreasing quality. This list of filter con¬
figurations is part of the flow description, such that each application
may use its own notion of quality. For each filter configuration, QoS
mapping, resource calculation and admission tests are carried out until
a configuration is found whose admission test is successful. Of course,
each configuration must lay within the area for downgrading which is
specified by the application.

5.3.5 QoS Re-negotiation

QoS re-negotiation is used to change the working points of a set of
flows. It is triggered by the application in order to change one or more
QoS parameter values. For each flow, the re-negotiation algorithm
finds a working point within the QoS space that can be supported by
all participants. This is in contrast to the unilateral QoS negotiation
which is done during the join operation. There, a participant is allowed
to join only if it can support the current working point. If the join oper¬
ation fails, the joining participant may trigger a QoS re-negotiation to
change the working point. This is possible since the re-negotiation
protocol is transported over a separate control flow. The application
may also use QoS re-negotiation to control how resources are assigned
to flows. The application assigns a priority to each flow. The QoS re¬
negotiation algorithm will first select suitable working points for high
priority flows before negotiating low priority flows. Free resources are
first consumed by flows with high priority.
A crucial point in QoS re-negotiation is to find the best possible working points that can be supported with the available resources. QoS re-negotiation thus requires a backward mapping which translates available resources back into multimedia support layer QoS parameters.

**QoS Backward Mapping**

The design for QoS backward mapping has to deal with the following facts:

- A change in a single vector element \( Q_{MMS}[x] \) may result in changes of several vector elements of \( R_{MMS}, R_{TP}, R_{MCA} \). An example is the transport of audio data. A decrease in the end-to-end delay results in shorter frame sizes as well as a shorter period and deadline for scheduling. The overall overhead increases, since each frame still uses the same network header size, which results in an increase in bandwidth. Furthermore, CPU usage increases as well, since the execution times in the modules per processed byte is higher for smaller packets. Another example occurs with compressed video frames. Lossy video compression such as JPEG reduces the frame size at costs of a reduced image quality. If the compression factor for video transmission is increased, then the frames are compressed to smaller sizes while at the same time the quality of the images is also reduced. The compressed frames may be transported using a smaller bandwidth, while for the compression of the images the execution time is increased.

- The relation between the QoS vector \( Q_{MMS} \) and the resource vectors \( R_{MMS}, R_{TP}, R_{MCA} \) is, in general, non-linear. Consider a video flow that uses a compression algorithm. An increase in the video image size will lead to a logarithmically increase in the memory usage, since most of the compression algorithms work better on larger images. However, CPU usage will experience a polynomial increase. Due to this non-linear relation of QoS and resource vectors, backward mapping can not be solved using linear optimization.
Backward mapping is ambiguous. For a given amount of free resources, no definite QoS vector can be found, since a whole set of QoS vectors \( \overrightarrow{Q_{MMS}} \) exist that result in the same resource vectors \( \overrightarrow{R_{MMS}}, \overrightarrow{R_{TP}}, \overrightarrow{R_{MCA}} \). For example, a video transmission may transport high resolution images at a low frame rate or it may transport low resolution images at a high frame rate resulting in the same resource consumption. A further problem lies in the fact that QoS vectors can not be sorted for 'goodness' or 'strength'. Taking the previous example, it completely depends on the application whether high resolution images at a low frame rate are preferable over low resolution images at a high frame rate. In general, for any two QoS vectors \( \overrightarrow{Q_1} \) and \( \overrightarrow{Q_2} \), the relation that defines whether \( \overrightarrow{Q_1} \) is 'better' than \( \overrightarrow{Q_2} \) completely depends on the application.

In MCF, backward mapping is implemented using forward mapping. Instead of calculating the best possible working point for given resources, the best out of a set of possible candidate working points is calculated. The continuous QoS parameter space is divided into a set of discrete QoS vectors which form candidate working points. MCF decides for each of the candidate working points whether it can be supported with the available resources. For each of the working points, QoS mapping, resource calculation and admission control are carried out. The result is a list of working points that can be supported with the available resources. In order to find the 'best' working point, the application has to specify an ordering relation among the QoS vectors. For this purpose, the application provides a sequence of QoS vectors with the 'best' vector being first. Using this list, MCF is able to select the 'best' QoS vector that is supported by all participants.

**QoS Re-negotiation Protocol**

QoS re-negotiation is used to find a common working point for several flows for all participants. The negotiation is done using a two phase commit protocol. It works as follows:

- The application specifies the flows for which the QoS should be
changed. The flows are sorted by priority.

- The application specifies a sorted list of candidate working points for each flow. The working points are sorted according to their importance to the application.

- All candidate working points are distributed to all participants.

- On each participant, QoS backward mapping is carried out. Backward mapping for high priority flows is done first. The resources which are needed for supporting the working points are reserved. The result of the backward mapping is sent back to the originator.

- The originator collects the results and selects the best working point for each flow that can be supported on all participants.

- If for each flow a working point can be found, it is distributed to all participants, i.e. commit is carried out. Additionally, the flow descriptions which are stored in the GMS are updated.

- If for a flow no working point can be found, then an abort is sent and the resources on each participant are released.

5.4 Summary

MCF offers end-to-end performance guarantees for multipoint-to-multipoint communication services. The design is based on two main parts, the transport subsystem and the control and management subsystem.

The transport subsystem consists of a runtime environment for protocol execution. Protocols form a three layer hierarchy. Each layer consists of a set of modules that implement layer specific protocols. Modules are designed following an object oriented approach, such that modules of the same layer can easily be exchanged. This approach allows for a dynamic composition of protocol stacks and permits that any media may be transported over any supported network.
Protocols are executed using a real-time scheduler. The scheduler guarantees that data are processed inside an endsystem within given deadlines. The scheduler used in MCF is an extended non-pre-emptive rate monotonic scheduler, which provides a simple admission test. The extension of the scheduler consists of a new priority assignment for modules that respects the semantically correct call sequence of the modules.

The control and management subsystem implements MCF’s QoS handling. Applications use QoS parameters to describe their requirements. The application requirements define a QoS parameter space and a working point which is part of the QoS space. This working point defines the service requested by the application. QoS mapping is used to map these request to each layer in the protocol stack. Each layer uses the QoS parameters to calculate the needed resources. Admission tests then decide whether sufficient resource are available to offer the requested service. The control and management subsystem contains admission tests for multimedia devices, memory and CPU. The admission test for bandwidth is delegated to network resource managers, where admission and reservation are done using a single operation. End-to-end delay admission checks whether processing delays and network transmission delays are below a given bound. After successful admission tests, the resources are reserved.

The working point of a set of flows can be changed using QoS re-negotiation. The re-negotiation algorithm takes a list of candidate working points for each flow. The algorithm selects the working points that are preferred by the application and that can be supported by all participants.
A prototype implementation has been done to validate the concepts applied in MCF's design. The implementation is described in the following sections. The first section gives an overview, identifies the components and shortly describes their interactions. The details of the components are described in the subsequent sections. The implementation of the communication protocols is presented by describing the modules that form these protocols. The modules of each layer are presented in Section 6.5, 6.6 and 6.7, respectively. The last section summarizes and critically reviews the implementation.
6.1 Implementation Overview

MCF has been implemented as a user process on top of the Solaris™ operating system, which conforms to the UNIX® System V Release 4 standard [Sun 95 B]. MCF extends the basic communication capabilities of Solaris as described in the last chapter and offers communication services to all applications that want to use it. MCF interacts with multimedia and network devices as well as with applications. Since MCF is not part of the operating system, all interactions are done using mechanisms of the operating system.

![Diagram showing implementation environment](image)

Figure 6.1: Implementation environment

Figure 6.1 shows how MCF is embedded into an endsystem. Applications request services from MCF using an application programming interface (API). The API consists of an API stub at the application that communicates with the API server over UNIX® FIFO streams. Using the API, applications may create, control and destroy multimedia multipoint flows. These flows consist of modules which are executed by the transport subsystem. Modules access devices and network
adapters using interfaces provided by the operating system. Video and audio devices are accessed over the respective device drivers. Data to the frame buffer are written via the direct X interface that circumvents the X11 window server. The ATM network is accessed using FORE’s proprietary API that interfaces to the ATM adaptation layers 3/4 and 5 as well as to FORE’s signalling protocol SPANS [Fore 96]. IP multicasting is implemented in the Solaris kernel. The protocol stack consists of UDP, IP multicasting, IEEE 802.2 and finally the IEEE 802.3 (‘Ethernet’) adapter card. MCF interfaces to this protocol stack using sockets.

6.1.1 Component Overview

The implementation consists of the application programming interface (API), the control and management subsystem, the transport subsystem, and modules. Each of these parts implements a different task. An overview of the components is given below, the details of each component are described in the next sections.

- Application Programming Interface (API)
  Applications request MCF’s communication services over the API. The API consists of two parts, the API stub which is linked to the applications and the API server which is part of the MCF system. Data between these two parts is exchanged using a simple protocol which itself is based on UNIX® STREAMS-based FIFOs. The API stub translates requests from the application into protocol data units (PDUs) of the API control protocol. The API server interprets these PDUs and invokes the corresponding routines of the control and management subsystem.

- Control and Management Subsystem (CMS)
  The control and management subsystem is the central steering component of MCF. The CMS sets up and manages all flows. This task consists of protocol composition for each flow, controlling the QoS mapping and resource calculation in the modules, conducting resource admission and resource reservation as well as triggering protocol execution by the transport subsystem. Furthermore, the CMS implements QoS re-negotiation. For this purpose, the CMS uses a re-negotiation protocol which is imple-
mented in the control flow. The re-negotiation protocol is based on QoS backward mapping, which is also implemented in the CMS.

- **Transport Subsystem (TSS)**
  The transport subsystem executes the protocols which have been composed by the CMS. Protocols are executed by the rate-monotonic real-time scheduler which calls the modules. Besides the real-time scheduler, the TSS offers a timer and buffer management that is used for the implementation of modules.

- **Modules**
  Modules are service elements which implement protocols. Protocols form a stack consisting of multimedia support, transport and multicast adaptation layer. A protocol stack consists of a module for each layer. Modules are implemented using an object-oriented approach, each module offers the same interface. Thus, protocol stacks can easily be composed by instantiating a suitable module for each layer. The protocol function of a module is accessed using module functions, i.e. the module's request and indication function. Additionally, modules also implement control functions such as QoS mapping and resource calculation. These control functions are called by the CMS during the setup of flows and also during the QoS re-negotiation.

Figure 6.2 shows how MCF's components interact for setting up a flow, executing the flow's protocol stack as well as conducting a QoS re-negotiation. As a first step, a request for a service is dispatched by the API server and delegated to the control and management subsystem (1). After the flow has been composed, the CMS activates control functions in the modules for QoS mapping and resource calculation (2). After the completion of these operations, the admission tests are carried out and the resources are reserved. If a flow has been successfully set up, the CMS triggers the scheduling of the flow. The scheduler of the transport subsystem then activates the modules according to their priority. Modules themselves use the timer and buffer management of the transport subsystem (3). For the QoS re-negotiation, the control and management subsystem uses a specialized module in the control flow, the control processor. The control functions of this mod-
ule are used as in the other modules. However, the CMS directly accesses the module functions which implement the two-phase commit re-negotiation protocol (4).

Figure 6.2: Component interactions

6.1.2 MCF Tasks

MCF is implemented as a single UNIX® real-time process. A real-time process belongs to the real-time scheduling class, which is one of three scheduling classes provided by UNIX® SVR4 [Goodheart et al. 93]. Figure 6.3 shows the three classes, the real-time class providing the highest priorities, the system class for system processes with medium priorities and the time-sharing class for interactive user processes with the lowest priorities. Each scheduling class offers several priorities. By assigning the highest real-time priority to MCF, the Solaris™ operating system assures a guaranteed response time.

The protocol functions of the modules form the tasks which are scheduled by the rate-monotonic scheduler. MCF tasks must not be confused with UNIX® processes or threads, they are used inside MCF, only. MCF’s rate-monotonic scheduler is able to execute tasks within their deadlines due to the guaranteed response time of the Solaris™ operating system. Tasks are the only active elements inside MCF. Besides the modules, the API server is modelled as an MCF task, too, such that it can process incoming requests from the applications.
The rate-monotonic scheduler assigns priorities to the modules according to their period and calling precedence (see Section 5.2.4). The period for flows transporting multimedia data is derived from the QoS parameters, whereas the period for control flows and the API server task are defined statically. For control flows, the period is set to $\frac{1}{4}$ of a second, with deadlines such that an end-to-end delay of 200 ms is achieved. Since control protocol stacks also have to pass the admission tests, these numbers control the worst case execution time for the QoS re-negotiation. The QoS re-negotiation is implemented as a two-phase commit protocol, with each phase having a worst-case round-trip time of 400 ms, such that the QoS re-negotiation takes at most 800 ms.

Period and deadline for the API server task are both set to one third of a second. Thus, an application waits at most one third of a second for a reply to arrive.

The priority assignment for a set of typical MCF tasks is shown in Figure 6.4. The example shows an audio and video flow, a control flow and the API server task. The audio flow, consisting of audio, segmentation and ATM module typically has a very short period around 40 ms. The video flow may has a period of ca. 70 ms. The control flow with a period of 250 ms and the API server with a period of $\frac{1}{3}$ of a second are assigned the lowest priorities.
6.2 Application Programming Interface (API)

The API is described in two parts. In the first part, the services and their semantics which are offered by MCF to the application are described. The second part presents the implementation of the API, in particular, how API control data are exchanged between applications and MCF.

6.2.1 API Services

The services offered by MCF allow the application to create, manage and destroy flows. In order to simplify the management of flows, they are bundled together to form so called sessions. All flows in the same session distribute data to the same participants. Flows and sessions are the main objects in the API, the corresponding data structures are called session description and flow description, respectively. The session description is a simple list that contains the names of the flows. A flow description contains the following elements as shown in Figure 6.5:
- **Flow name** The flow name uniquely identifies a flow. It is used in the session description to reference the flows.

- **Protocol definition** The protocol definition defines the protocol stack that is used in the flow. It consists of a module name for each layer.

- **QoS parameters** The application level QoS parameters define the QoS parameter space and the working point of the flow.

- **Limits** Limits on the number of senders and receivers are needed for the resource calculation.

- **Delay split factors** The delay split factors are used to split the end-to-end delay into a delay at the sender, a network transmission delay and a delay at the receiver.

*Figure 6.5: Session and flow data structures*

Applications access MCF’s services through the following functions:

- **Create Flow**
  Create flow is used to create a flow in the GMS space. Each flow first has to be created before it can be used. The application has to provide a valid flow description such that the flow can be cre-
ated. Applications may use flow templates for creating a flow. A flow template defines the module triple as well as the delay split factors and default QoS parameters for typical scenarios such as interactive audio communication or dissemination of video. If the application uses a flow template, it needs to specify the flow name, the maximum number of senders and receivers, only. Additionally, it may override the default QoS parameters.

- **Create Session**
The create session call is used to create a session. The application has to pass a unique session name and the names of the previously created flows that should be included in the session.

- **Join Session**
After a session and its flows have been created, the application may join this session. The application specifies the name of the session that it wants to join. For each flow in this session, it has to specify whether it wants to receive or send data. The result of a join session call is a result vector that returns the results of the admission tests for all flows. Join session succeeds, if all admission tests have been successful.

- **Start Session**
Start session is called to start the scheduler for a previously joined session.

- **Stop Session**
Stop session is the opposite of start session.

- **Leave Session**
When leave session is called, the application leaves a session and all resources are deallocated.

- **QoS Re-negotiate**
QoS re-negotiate is used to trigger a QoS re-negotiation. For each flow whose QoS should be changed, a list of QoS vectors has to be supplied. The first QoS vector in this list has the highest priority. As a result of the QoS re-negotiation, the selected QoS vector for each flow is returned.
6.2.2 API Protocol

The API consists of a stub which is linked to the application and of a server which is part of the MCF. For the communication, a simple request-response protocol is used. Each function offered by the API is mapped to a request and response PDU. The task of the API stub is to generate request functions and send it to the API server. The API server dispatches incoming request PDUs and calls the appropriate functions in the control and management subsystem. The result of these functions are encapsulated in reply PDUs and sent back to the API stub, which forwards the result to the application.

The API protocol runs on top of STREAMS-based FIFOs. In order to avoid demultiplexing for each application, the API server uses a single FIFO for each application. Figure 6.6 gives an overview of the API implementation.

6.3 Transport Subsystem

The transport subsystem provides a runtime environment for the execution of protocols. It consists of a real-time scheduler, a timer management and a buffer management, which are described below.

6.3.1 The Rate-Monotonic Scheduler

The real-time scheduler implemented in the transport subsystem is a non-pre-emptive rate-monotonic scheduler. The main data structure used in the scheduler is the schedule. Theoretically, the schedule is an
infinite list of times at which tasks have to be executed. For practical reasons, the schedule is implemented as an event queue which is constantly being updated. The event queue is sorted chronologically, with the earliest due tasks at the head of the queue. The queue and its elements are shown in Figure 6.7.

![Scheduler's event queue](image)

**Figure 6.7: Scheduler's event queue [Keller 96]**

The variables in the queue elements have the following meanings:

- **time**
  This is the earliest possible time when tasks may be started.

- **action**
  Action is a reference to a function in a module that should be called. The action identifies the task that should be executed, normally a module's request or indication function.

- **instance**
  The instance variable points to a memory block that contains the private variables of the module. For each module in each flow, an instance structure exists. The instance structure corresponds to the protocol control block in TCP implementations.

- **flow**
  Scheduled tasks always belong into the context of a flow. The
scheduler needs a reference to the flow's data structure in order to access period and deadline of the flow.

- **packet**
  Packet is a reference to a buffer that contains a packet or a multimedia frame. Packet is passed as a parameter to the task.

The core of the scheduler is a simple loop that selects events from the event queue according to the rate-monotonic criterion and calls the action function of these events, as shown in the following example pseudo code:

```pseudo
do
  now := get_time();
  if ∃(event with time < now) do
    select event with shortest period;
    call event.action(packet);
    remove event from event queue;
  else
    sleep until ∃(event with time = now)
  endif
loop
```

If no task is ready to run, the scheduler sleeps until a task becomes ready. While the scheduler is sleeping, other UNIX® processes such as system daemons or user processes are executed. The scheduler resumes execution after the sleep call. The sleep call only has an accuracy within the dispatch latency of the UNIX® scheduler, which is about 1 ms, depending on the used hardware. In order to be more accurate, the MCF scheduler reduces the sleep time by the dispatch latency and enters an idle loop until the sleep time is reached, as shown in Figure 6.8. Thus, the accuracy can be increased to at least 10 μs, depending on the used hardware.

Scheduling events are generated in two different ways, depending on whether the tasks which are scheduled belong to a sending flow or to a receiving flow.

For sending flows, scheduling events are inserted into the event queue by the tasks themselves, except for the first task. As mentioned earlier,
6.3. Transport Subsystem

Figure 6.8: Considering the UNIX® dispatch latency [Keller 96] tasks are modules that are being executed. The first task is inserted by the control and management system, it always executes the request function of the multimedia support module. The module will be called and after having generated a packet, it inserts a call to the transport module, which itself inserts a call to the multicast adaptation module. The multicast adaptation module calls the transport module’s indication function, which calls the multicast adaptation module again, until a whole multimedia frame has been processed. Then, the transport module re-schedules the multimedia support module for the next period. An example of this process illustrating the segmentation of audio frames for the transportation over ATM is shown in Figure 6.9.

Figure 6.9: Scheduling of modules
In receiving flows, the multicast adaptation module needs to be called whenever a packet is ready to be processed. Due to network delay jitter, incoming packets may not arrive strictly periodically. The transport subsystem contains a watchdog that checks whether packets arrive from the network. Each multicast adaptation module registers itself by the watchdog. For each packet that arrives, the watchdog generates a scheduling event such that the multicast adaptation module will be called eventually. The watchdog itself is activated by the scheduler between each context switch. This is feasible since the execution time for the watchdog is very small, below 2 µs.

### 6.3.2 Timer Management

The timer management provides one-shot timers for modules such as error control modules. By coupling timer management and scheduling, the implementation of timers comes almost for free. A module that sets a one-shot timer simply inserts a scheduling event into the event queue. It uses the same mechanism as when scheduling a normal task. As normal tasks, one-shot timers inherit the scheduling priority of the calling module. A one-shot timer can be cancelled by removing the scheduling event from the event queue again. The primitives for inserting and deleting scheduling events are as follows:

```c
id = insertModuleCall (activation_time, packet, task, instance, flow);
removeModuleCall (id);
```

### 6.3.3 Buffer Management

Memory needed for packets and multimedia frames is reserved before the protocol is started, as it is done with every resource. During protocol execution, no additional memory is allocated. The buffer manage-
6.3. Transport Subsystem

Management in MCF uses a simple recycling strategy for buffers. Buffers are kept in one of two lists. A free list contains the buffers that are currently not used. Initially, all buffers are in the free list. The used list contains buffers that are occupied. A module that wants to use a buffer requests a free buffer from the buffer management. The buffer management returns a reference to a free buffer and moves the buffer from the free list to the used list. Modules work on buffers using references, only. Reference are also used to pass buffers from one module to another. Buffers are freed by discarding their references. Several references may point to the same buffer. As soon as the last reference is discarded, the buffer is removed from the used list and placed in the free list again. Figure 6.10 shows a typical situation in a sender. The multimedia support module allocates a buffer for each frame that has to be sent. The buffers are passed by reference to the transport module. The transport module may creates additional references to buffers. For example, the multicast error control module has to retransmit lost packets and therefore has to keep references to sent buffers. In a last step, a reference of the buffer is forwarded to the multicast adaptation module. The multicast adaptation module sends the buffers over the network and discards the reference.

![Buffer management diagram](image)

**Figure 6.10: Buffer management**

The data structure used for buffers is optimized for protocol processing. MCF uses the same approach as the Da CaPo framework [Gotti 94]. Buffers consists of a header part and a data part. Reading and
writing of buffers to/from the network or the multimedia devices is done using scather/gather operations, where header part and data part are processed in a single operations. Additionally, the buffer management offers operation for growing and shrinking header as well as data parts.

### 6.4 Control and Management Subsystem

The control and management subsystem implements functions for steering flows and sessions. These functions are accessible through the API, as described in Section 6.2.1. The functions for creating sessions and flows are simple data movements and not further described here. Join session and QoS re-negotiate make up the largest part of the functionality provided by CMS.

#### 6.4.1 Join Session

Join session consists of protocol composition, QoS mapping, resource calculation and resource admission. These operations are needed to create and set up the flow data structure. Internally, a flow is represented through the protocol stack that it executes, QoS parameters, and the resources that it occupies, as shown in Figure 6.11. The protocol stack is given through a triple of modules, one for each layer.

![Figure 6.11: Flow data structure](image)

**Figure 6.11: Flow data structure**
The implementation of modules consists of the executable code which is stored in a library and of a module data structure which is used to access the library code. The library may be extended with additional modules at any time. The module data structure is shown in Figure 6.12. It consists of administrative information such as the module name, of a set of references to module functions which are called by the scheduler and of a set of references to control functions which are used by the CMS. These control functions are described in the following subsections.

**Figure 6.12: Module data structure**

**Protocol Composition**

The protocol composition is the first step of the join session operation. The protocol composition allocates a flow data structure and three module data structures. The module data structures are initialized with the modules that are specified in the flow description by the application. For this task, CMS retrieves the corresponding references in the library which are then stored in the module data structures.
QoS Mapping

The QoS mapping translates QoS parameters which are specified by the application to transport level and multicast adaptation level QoS parameters. The translation from the application level to the transport level is done by the multimedia support module, the translation from transport level to the multicast adaptation level is done by the transport module. The CMS calls the QoS mapping functions of both of these modules and stores the result in the flow data structure. The autoconfiguration of the modules is also done during the QoS mapping, modules use the properties function of the module at the next lower layer to configure themselves, as described in Section 5.2.2 on page 79.

Resource Calculation

The CMS delegates resource calculation to the modules in the same way as QoS parameter mapping. Modules calculate their resource usage for multimedia devices, memory, CPU and bandwidth, as described in Section 5.3.2 on page 102. The calculation of multimedia devices, memory and bandwidth is based on QoS parameters only, whereas the calculation of the execution time for a module needs information that is stored in the profile database. The profile database contains measurements of module functions for each module on each endsystem, as the example in Table 6.1 shows. The execution time of module functions often depends on QoS parameters such as the packet size. Therefore, the profile database contains measurements for several combinations of QoS parameters. These combinations are module dependent and are encoded in a so called quantifier. In the example table, the quantifier for the multicast IP module is the packet size in bytes, whereas the segmentation module does not quantify its measurements. The modules' resource calculation functions calculate the quantifier for the given QoS parameters and retrieve the corresponding execution time. For some combinations of QoS parameters, no quantifier is stored in the module database. In these cases, modules retrieve two measurements and interpolate a measurement value. In most cases, the
execution time only depends on the packet size, in which case a simple linear interpolation can be used.

<table>
<thead>
<tr>
<th>Quantifier</th>
<th>Module Function</th>
<th>Module Name</th>
<th>End-system</th>
<th>Execution time</th>
</tr>
</thead>
<tbody>
<tr>
<td>320</td>
<td>Request Main Sender</td>
<td>Multicast IP</td>
<td>kom25</td>
<td>320 µs</td>
</tr>
<tr>
<td>1764</td>
<td>Request Main Sender</td>
<td>Multicast IP</td>
<td>kom25</td>
<td>440 µs</td>
</tr>
<tr>
<td>320</td>
<td>Request Back Sender</td>
<td>Multicast IP</td>
<td>kom25</td>
<td>190 µs</td>
</tr>
<tr>
<td>1764</td>
<td>Request Back Sender</td>
<td>Multicast IP</td>
<td>kom25</td>
<td>260 µs</td>
</tr>
<tr>
<td>0</td>
<td>Request Main Sender</td>
<td>Segmentation</td>
<td>kom25</td>
<td>10 µs</td>
</tr>
</tbody>
</table>

Table 6.1: Example profile database entries

Resource Admission

CMS implements admission tests for memory, multimedia devices and CPU, whereas admission tests for bandwidth is delegated to the network resource manager and carried out during the resource reservation. For each endsystem resource, an account is maintained that identifies how much of the resource has been allocated.

Memory admission is simplified through the fact that MCF uses a single physical memory block for all flows. The accounting consists of a record of free and allocated memory. Memory admission is reduced to comparing whether enough free memory is available for a flow.

Multimedia devices may be used by a single flow only. The accounting information consists of a list that states for each device whether it is allocated or not. Admission is successful, if a requested device is not allocated.

The CPU admission test is done by evaluating the scheduler’s admission test formula. This formula tests whether a flow can be scheduled within its given deadline (see Section 5.2.4 on page 90 and Section 5.3.3 on page 104). A flow can be admitted, if all existing flows can be scheduled within their deadlines. For the CPU accounting, the CMS keeps a list containing all flows that have to be scheduled, the so called CPU admission test list. The CPU admission test distinguishes receiving flows and sending flows. A sending flow is inserted exactly once
into the admission test list. A receiving flow, however, has to process packets from all active senders during a single period. Therefore, receiving flows are inserted multiple times into the admission test lists. The number of insertions is the maximum number of active senders which is specified in the flow description.

The end-to-end delay is composed of a delay at the sender, a network transmission delay and a delay at the receiver, as shown in Figure 5.19 on page 100. Since both the ATM and IP network do not provide information about the delay, a worst case transmission delay based on measurements is used. This transmission delay is taken into account for the calculation of the processing deadline for sender and receiver, i.e. it is subtracted from the end-to-end delay. The admission test for the end-to-end delay then succeeds if the send delay and receive delay can be assured, which is the case if the scheduling test succeeds.

**Resource Reservation**

The resource reservation uses the results of the resource calculation to reserve resources. An exception is the reservation of the CPU, which is reserved during the CPU admission test by including each flow in the CPU admission test list, as described in the last section. The admission test then guarantees that the CPU does not become overloaded.

Modules are responsible for reserving memory and multimedia devices. Memory is reserved by using the operating systems memory allocation function. Each module reserves the amount of memory that it needs for holding its private data, the so called instance structure. The buffer management reserves memory for holding multimedia frames and packets. The number of needed buffers has been calculated during the resource calculation.

The reservation of bandwidth is delegated to the network resource manager. Since the details of this operation depend on the type of network used, the delegation is done by the multicast adaptation module. Network resource managers carry out an admission test before the
bandwidth is being reserved. The reservation fails if not enough bandwidth is available.

6.4.2 QoS Re-negotiation

The implementation of the QoS re-negotiation consists of a re-negotiation protocol that carries out the actual re-negotiation, of QoS backward mapping which is used to identify the QoS parameters that can be supported on an endsystem and of QoS switching, which reconfigures the flow according to the selected QoS parameters.

QoS Re-negotiation Protocol

The CMS instantiates a control protocol stack per session. If an application specifies that no QoS re-negotiation should be carried out in a session, then no control protocol stack is set up in order to save resources. An instantiated control protocol stack (i.e. a control flow) is shown in Figure 6.13. Unlike other flows which are unidirectional only, control flows are bi-directional. In the control processor module, the QoS re-negotiation protocol is implemented. It is the only module which is instantiated as receiver and sender simultaneously. Reliable data transfer is provided by the multicast error control module, an instantiation of this module is needed for receiving as well as for sending. The error control itself is based on multicast IP, although any multicast adaptation module could be used.

![Figure 6.13: Control protocol stack](image-url)
The QoS re-negotiation protocol is a two-phase commit protocol based on simple reply/request operations. The protocol consists of two parts. The first part allows each participant to identify the presence of all other participants. This information is then used by the second part of the protocol, which carries out the actual re-negotiation.

- **register request**
  The register request is used to register a participant by its partners of the same session. As soon as a participant starts the control processor, a register request is multicast to all other participants of the same flow.

- **register reply**
  Upon receiving a register request, participants send a register reply containing their own identifier back to the sending participant. Thus, the sending participant eventually learns about all others by receiving register replies.

- **unregister request**
  Unregister request is sent if the control processor is stopped. It is used to inform the partner participants that a participant left the session.

- **map request**
  A participant that initiates the QoS re-negotiation distributes a map request. The map request contains lists of QoS vectors which are used as input for QoS backward mapping.

- **map reply**
  The map reply contains the results of the QoS backward mapping. For each combination of QoS vectors, map reply states whether the admission test has been successful or not. A map reply is sent by each participant upon receiving a map request.

- **commit**
  The control processor of the participant that initiated the QoS re-negotiation collects the map replies from all registered participants and carries out an evaluation. The evaluation tries to find the 'best' QoS vector that can be supported by all participants. If such a vector exists, then a commit is sent to all participants containing this vector.
6.4. Control and Management Subsystem

- **rollback**
  If no QoS vector that is supported by all participants can be found, then the initiating participant sends a rollback.

**QoS Backward Mapping**

Backward mapping is used to find the best QoS that can be supported by an endsystem using the available resources. The algorithm used for backward mapping is based on forward mapping with discrete QoS vectors, as described in Section 5.3.5 on page 109. The discretisation of the QoS space is done by the application. The application has to specify a set of QoS vectors which are distributed using the map request and used as input for QoS backward mapping. QoS backward mapping finds the 'best' possible combination of QoS vectors for all flows, where 'goodness' is defined by the application. For each flow whose QoS should be changed, the application specifies a list of QoS vectors. Thus, the application has to specify a list of lists of QoS vectors. The lists are arranged such that the list for the most important flow is first. Likewise, the QoS vectors in a flows’ list are also ordered such that the 'best' QoS vector is first. Figure 6.14 gives a schematic example where the application specified QoS vectors for three flows, namely \( Q_1,1, Q_1,2 \) and \( Q_1,3 \) for the first flow, four vectors for the second flow (\( Q_2,1 \) up to \( Q_2,4 \)) and two for the third flow (\( Q_3,1 \) and \( Q_3,2 \)). A valid combination consists of one QoS vector for each flow, for example (\( Q_1,1, Q_2,3, Q_3,1 \)). By using the above mentioned ordering relation, combination of QoS vectors become comparable. Combination \( Q_1,a, Q_2,b, Q_3,c \) is 'better' than combination \( Q_1,x, Q_2,y, Q_3,z \), if the following equation holds:

\[
(a < x) \lor ((a = x) \land (b < y)) \lor ((a = x) \land (b = y) \land (c < z))
\]  

(6.1)

For all combinations, QoS backward mapping decides whether they can be supported by the available resources. In the above example, \( 3 \cdot 4 \cdot 2 = 24 \) combinations exist. In this case, the result of QoS back-
ward mapping is a list with 24 boolean values. These values are sorted according to the above mentioned ordering relation. Thus, the first value that is TRUE automatically identifies the best combination that is possible with the available resources.

Figure 6.14: QoS parameter specification for backward mapping

In order to decide whether a combination of QoS vectors can be supported with the available resources, QoS mapping, resource calculation and admission tests have to be carried out. The available resources consist of the resources that are allocated in the existing session in addition to the free resources. Therefore, the admission tests have to consider the resources that are already allocated by the session. This leads to the following steps that are carried out by QoS backward mapping:

- The accounting information needed for the admission tests is modified such that the resources of the original session are no longer considered for the admission tests. This is done, for example, by temporarily removing the original flows from the CPU admission test list. However, resources are not deallocated since the flows of the original flows are running during the QoS backward mapping is taking place.

- A temporary session is created that contains copies of the flows of the original session.

- For each combination of QoS vectors, QoS mapping, resource calculation and admission tests are carried out. The result of the admission test is stored in a map reply PDU.
• The temporary session with its flows is freed.
• The original accounting information is recovered.
• The map reply PDU is sent back to the initiator of the QoS re-negotiation.

The algorithm described above does not reserve any endsystem resources. In order to avoid a race condition for endsystem resources, the QoS backward mapping blocks all join and other QoS re-negotiation operations until the QoS re-negotiation is finished, i.e. until a commit or rollback arrives. Bandwidth, however, has to be handled separately. Since bandwidth admission test and bandwidth reservation is integrated into a single operation of the network resource managers, it is not possible to carry out an admission test without actually reserving bandwidth. Bandwidth is allocated as soon as a combination of QoS vector requires more bandwidth than what is already allocated in the original session. The allocated bandwidth is not freed after backward mapping is finished. It remains allocated until a commit or rollback PDU arrives. This leads to a temporary overallocation of bandwidth, since the bandwidth in the original flows also remains allocated.

QoS Switching

After a successful QoS re-negotiation, the new QoS must be instantiated. This operation is also called QoS switching. Its main purpose is to release or allocate additional resources as required by the new QoS. QoS switching is triggered by an arriving commit PDU that carries the new QoS vectors. The first step of QoS switching is to carry out QoS mapping and resource calculation for the new QoS. Then, the resource reservations are changed as follows:

• CPU The CPU admission test lists are updated with the results of the resource calculation.
• Memory The buffer management frees all allocated buffers and allocates new buffers as calculated by the resource calculation.
- **Devices**: Multimedia devices have to be reconfigured. For instance, if a new sampling rate has been chosen, then the audio device has to be reconfigured. The reconfiguration is carried out by the multimedia support modules.

- **Bandwidth**: Bandwidth is changed by the multicast adaptation modules. If the needed bandwidth is increased, then the multicast adaptation modules free the allocated bandwidth and switch to the bandwidth which has been allocated during the QoS backward mapping. Otherwise, the bandwidth which has been allocated during the backward mapping is freed.

### 6.5 Multimedia Support Modules

Multimedia support modules interface to the multimedia devices of the endsystems such as the audio device, the video capture device or the framebuffer. Multimedia data may also are provided by applications, in which case these modules interact with applications instead of devices. The functions implemented by multimedia support modules consist of control and module functions that are distinguished by control and module functions of other layers in the following way:

- The QoS mapping function parses the application level QoS parameters before they are mapped.

- Period and deadline of the flow are also calculated by the QoS mapping function using the end-to-end delay specified by the application.

- On the receiving side, the multimedia support module implements a smoothing buffer that corrects delay jitter effects.

The next section describes the implementation of MCF's audio modules. Section 6.5.2 then explains how delay jitter effects are avoided in MCF. The described solution may be applied in any multimedia support module.
6.5.1 Audio Modules

MCF's prototype implementation contains three audio modules which differ in the supported devices. The audio modules fetch audio samples either from microphone, line-in device or an audio file. The samples are output to the audio device which plays them on a loudspeaker. The audio module can be configured using the QoS parameters shown in Table 6.2. The audiofile QoS parameter is only supported if the samples are read from an audio file.

<table>
<thead>
<tr>
<th>QoS parameter name</th>
<th>QoS parameter value set</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>samplingrate</td>
<td>8000, 9600, 11025, 16000, 18900, 22050, 32000, 37800, 44100, 48000</td>
<td>Sampling rate in Hz, only listed rates are supported by the audio device.</td>
</tr>
<tr>
<td>precision</td>
<td>8 bit or 16 bit</td>
<td>Sampling size per channel</td>
</tr>
<tr>
<td>encoding</td>
<td>μ-law, A-law or linear.</td>
<td>μ-law or A-law for 8 bit precision, linear for 16 bit.</td>
</tr>
<tr>
<td>channels</td>
<td>1 or 2</td>
<td>1 channel = mono, 2 channels = stereo.</td>
</tr>
<tr>
<td>delay</td>
<td>larger than 40 ms</td>
<td>end-to-end delay, useful range from 60 ms to 300 ms</td>
</tr>
<tr>
<td>audiofile</td>
<td>filename</td>
<td>The filename of the audiofile from which the samples are taken.</td>
</tr>
</tbody>
</table>

Table 6.2: Audio application level QoS parameters

Audio QoS Mapping

The QoS mapping function parses the application level QoS parameters and maps them to transport level QoS parameters. The QoS mapping also calculates the flow's period, which directly depends on the end-to-end delay. The QoS mapping functions are based on the way the protocol functions are modelled: During each period, the audio module on the sending side produces an audio frame containing audio samples. The amount of time that is used until enough samples are available at the audio device is called *data acquisition time*. The data
acquisition time equals one period. After an audio frame has been generated, it is processed by the sender, sent over the network and then processed by the receiver. The receiver buffers audio frames in order to smooth out jitter effects, as described in Section 6.5.2. The end-to-end delay in the case of audio consists of the parts shown in Figure 6.15.

![Figure 6.15: End-to-end delay for audio flows](image)

The QoS mapping functions are shown in Table 6.3. The result of the QoS mapping are transport level QoS parameters (frame size, frame rate, and frame ordering) as well as the flow’s period and deadline.

**Queuing and Mixing**

A receiving audio module may receive audio frames from one or more senders. Since the audio device may only output a single frame at a time, frames of different senders have to be mixed together. Additionally, audio frames have to be queued in order to smooth out delay jitter effects, as described in the next section. The implementations of queuing and mixing are tightly coupled. Audio frames that are received by the audio flow will eventually be processed by the audio receive main function. The receive main function places audio frames into queues according to their sender, there is a single queue for each sender. The receive main function does not output frames to the audio device. This is done by a task called *mixer*. The mixer examines the audio frames in the queues. Those frames that have been delayed until the playback point is reached are mixed\(^1\) together and sent to the audio device. The mixer is implemented as an MCF task and activated periodically. Fig-

---

1. Currently, mixing is implemented for 16-bit linear audio. Samples are mixed together by building the sum modulo the size of the maximum amplitude.
6.5. Multimedia Support Modules

### Table 6.3: Audio QoS mapping functions

<table>
<thead>
<tr>
<th>Mapping function</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>send_delay = delay \cdot \text{sender_fraction}</code></td>
<td>The delay at the sender is a fraction of the end-to-end delay. The size of this fraction is part of the flow description.</td>
</tr>
<tr>
<td><code>acquisition = send_delay \cdot \text{acq_fraction}</code></td>
<td>The acquisition delay itself is a fraction of the sender's delay. The acquisition delay is 60% of the sender's delay, but at least 20 ms.</td>
</tr>
<tr>
<td><code>period = acquisition</code></td>
<td>The period is necessarily the same as the acquisition delay.</td>
</tr>
<tr>
<td><code>deadline = send_delay - acquisition</code></td>
<td>The deadline is calculated from the delay at the sender less the acquisition delay.</td>
</tr>
<tr>
<td><code>framesize = rate \cdot depth \cdot channels \cdot \text{period}</code></td>
<td>The frame size is the amount of data that is produced during a single period. It depends on the sampling rate, the sampling depth and the number of channels.</td>
</tr>
<tr>
<td><code>framerate = \frac{1}{\text{period}}</code></td>
<td>During each period, a single audio frame is produced.</td>
</tr>
<tr>
<td><code>ordering = FIFO</code></td>
<td>The audio frames require FIFO ordering.</td>
</tr>
</tbody>
</table>

Figure 6.16 shows how the involved tasks inter-operate in order to provide queuing and mixing.

#### 6.5.2 Calculation of the Playback Point

The end-to-end delay that a packet experiences is divided into the parts shown in Figure 6.15. The real-time scheduler of MCF guarantees an upper bound on the processing delay at the sender and the receiver. The network also guarantees an upper bound on the delay. However, packets may experience a varying end-to-end delay that is smaller than the sum of the bounds. This variation is called delay jitter.
Without taking jitter into account, the playback of multimedia frames, in particular audio frames, can become disrupted. This happens if a receiver runs out of data, the receiver is then said to “starve”. Starvation due to jitter can be eliminated by delaying frames until the guaranteed maximum end-to-end delay is reached, as argued in [Partridge 91] and proved in [Anderson 93]. The problem is to determine the exact time when the maximum end-to-end delay, the so called playback point, is reached. Solutions have been proposed where sender and receiver have synchronized clocks [Partridge 91], [Ferrari 91] or where the delay jitter is approximated by an iteration over the arrival time variation of several packets [Schulzrinne et. al 96].

The solution implemented in MCF is to calculates an estimation of the playback point without the use of synchronized clocks, as proposed by [Partridge 91]. Table 6.4 shows the variables needed in the calculation. The first four values are delay bounds provided by the real-time scheduler and by the network resource manager. The exact playback point could be calculated if the actual end-to-end delay was known. Then the packet had to be delayed by the maximum end-to-end delay less the actual end-to-end delay. The difference of the guaranteed delay to the actual delay is called workahead [Anderson 93]. The problem is that the workahead in the network can not easily be measured without synchronized clocks. Therefore, MCF’s implementation uses an estimation of the network workahead.

By timestamping each packet, the actual processing delay is measured. The processing workahead at the sender and the receiver is then
calculated from this measurements, as shown by the bold printed entries in Table 6.4. For the network workahead, a worst-case assumption can be made by taking the network delay bound and subtracting the minimum network delay. Packets are delayed at the receiver by the measured processing workahead plus the bound for the network workahead.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$B_{SP}$</td>
<td>Bound on the processing time at the sender</td>
</tr>
<tr>
<td>$B_{ND}$</td>
<td>Bound on the network delay</td>
</tr>
<tr>
<td>$B_{RP}$</td>
<td>Bound on the processing time at the receiver</td>
</tr>
<tr>
<td>$N_{DMIN}$</td>
<td>Minimum network delay (if unknown: 0)</td>
</tr>
<tr>
<td>$A_{SP}$</td>
<td>Actual processing time at the sender (measured)</td>
</tr>
<tr>
<td>$A_{ND}$</td>
<td>Actual network delay (unknown)</td>
</tr>
<tr>
<td>$A_{RP}$</td>
<td>Actual processing time at the receiver (measured)</td>
</tr>
<tr>
<td>$W_s$</td>
<td>Workahead at the sender: $W_s = B_{SP} - A_{SP}$</td>
</tr>
<tr>
<td>$W_r$</td>
<td>Workahead at the receiver: $W_r = B_{RP} - A_{RP}$</td>
</tr>
<tr>
<td>$W_{s+r}$</td>
<td>Sum of processing workahead: $W_{s+r} = W_s + W_r$</td>
</tr>
<tr>
<td>$B_{NW}$</td>
<td>Bound for network workahead: $B_{NW} = B_{ND} - N_{DMIN}$</td>
</tr>
<tr>
<td>$E_{GEDD}$</td>
<td>Guaranteed end-to-end delay: $E_{GEDD} = B_{SP} + B_{ND} + B_{RP} + B_{NW}$</td>
</tr>
<tr>
<td>$E_{EEDD}$</td>
<td>Experienced end-to-end delay: $E_{EEDD} = A_{SP} + A_{ND} + A_{RP} + W_{s+r} + B_{NW}$</td>
</tr>
</tbody>
</table>

Table 6.4: Components of playback point calculation

Figure 6.17 gives a graphical representation of the computation. It uses the variables introduced in Table 6.4. The gray, bold arrows running from top left to bottom right show the delay guarantees which are given by the scheduler and the network. Actual frames which travel from sender to receivers usually experience a shorter delay. The black, finer arrows which run also from top left to bottom right show the delay experienced by an example frame. The actual delay is a priori unknown. However, the processing delay at both sending side and receiving side is measured. Using these measurements, the processing workahead is calculated. The first frame that arrives at a receiver is delayed by the processing workahead and the bound for the network workahead, before it is output to the multimedia device. By doing so, it is guaranteed that the next multimedia frame arrives before the previous one is completely played back. The first frame therefore deter-
mines the experienced end-to-end delay. All following frames will be delayed up to this delay. The guaranteed end-to-end delay as requested by the application is larger than the experienced end-to-end delay.

Figure 6.17: Calculation of the playback point

6.6 Transport Modules

On the transport layer, two protocol functions are implemented. For multimedia flows, a segmentation and reassembly module is provided. For control flows, an error control module offers reliable data transfer. Both modules are described in the following sections.

6.6.1 Segmentation and Reassembly

The segmentation and reassembly module is used as the standard transport module for multimedia flows. The main function can be summa-
rized as follows: If frames forwarded by the multimedia support module are larger than the maximum transfer unit of the multicast adaptation module, then they have to be segmented. Otherwise, frames are directly forwarded to the multicast adaptation module.

Segmentation
Segmentation is done at the sender's side. A multimedia frame is segmented into fragments the size of the maximum transfer unit. The segmentation itself is implemented using references that point into the original frame. Thus, no copy operations are needed. Each fragment is equipped with a header that contains the following elements:

- **Fragment**: Boolean value that tags fragments in contrast to whole multimedia frames.
- **Original length**: Length of the original frame in bytes.
- **Sequence number**: Sequence number that identifies the original frame. Fragments of the same frame carry the same sequence number.
- **Fragment number**: Position of the fragment in the original frame, e.g. first fragment, second fragment etc.

Reassembly
The reassembly part is implemented using state machines. For each sender, a separate state machine is used. The number of states that is needed depends on the number of fragments per frame. Figure 6.18 shows a state machine that is used to reassemble a frame consisting of n fragments. Transitions are done based on the packet's fragment and sequence number. The first fragment carries the sequence number that identifies the original frame. If fragments of the same frame arrive in the correct order, then the frame eventually will be accepted. A fragment in the wrong order resets the state machine and received fragments are discarded.
6.6.2 Multicast Error Control

The multicast error control module implements a general purpose receive-initiated error control protocol. The protocol is described in detail in [Bauer et al. 96] and [Nigg 96]. It provides the following functionalities:

- Out of order delivery of packets at a receiver side is corrected by re-ordering incoming packets.
- Duplicated packets are identified and discarded.
- Receivers detect lost packets by gaps in the sequence numbers of arriving packets in combination with time-outs. Lost packets are requested from the sender using retransmission requests.
- If no payload data are there to be distributed, senders distribute status packets containing the last valid sequence number. This status packet allows receivers to detect packet losses even if no payload data is being distributed. Status or keep-alive packets
are sent in increasing intervals in order to reduce bandwidth consumption.

Receivers detect lost packets by gaps in the sequence numbers. However, a gap indicates not only lost packets, but also delayed or out-of-order packets. Therefore, receivers send retransmission requests only after a time-out in case the presumed lost packet arrives later. The retransmission request time-out is set to the estimated average round-trip time. Retransmission requests (RRQs) for several packets are grouped together. Instead of sending a single RRQ for each lost packet, a single retransmission request is sent that contains the sequence numbers of all lost packets. For each RRQ that is sent, a time-out is started. If the requested packets arrive, the time-out is cancelled. Otherwise, another retransmission is sent. The duration of this time-out also depends on the average round-trip time.

Senders group retransmission requests, too. If a packet gets lost, its retransmission will likely be requested by several receivers. Therefore, the sender will receive several RRQs for the same packet. Senders retransmit a packet as soon as the first request arrives. Additional retransmission requests are ignored as long as they arrive no later than the average round-trip time.

Senders calculate an average round-trip time (RTT) based on the RTTs to all receivers. The accuracy of the calculated RTT depends on the variance of the RTTs to single receivers. A large variance does not affect the functionality of the protocol, only the performance. An average RTT which is too small for some receivers leads to multiple retries being sent by these receivers. On the other hand, an RTT which is too large for some receivers causes them to wait too long before they send the first RRQ.

The round-trip time is calculated using incoming retransmission requests, as shown in Figure 6.19. If packet n is lost, packet n+1 will start a retransmission request timer on the receiver. This timer triggers the retransmission request for packet n. The retransmission request contains the time-out value \( t_{out} \) and a reference to packet n+1 that caused the request. Upon receiving a retransmission request, the send-
er is able to calculate the time that passed since packet \( n+1 \) has been sent. Subtracting the time-out value \( t_{out} \) of the receiver results in the round-trip time. Due to the variations in the experienced round-trip times, an adaptation of an initial RTT value is used. The adaptation uses the same formula as TCP's back-off mechanism. The new average RTT is calculated using the old RTT and the measured RTT:

\[
RTT = (a \times \text{old}_\text{RTT}) + (1 - a) \times \text{measured}_\text{RTT};
\]

with \( 0 < a < 1 \)

The convergence of the above formula depends on the value for \( a \). A small value leads to a very fast convergence, which means that the RTT changes very often. The current implementation uses a value of 0.9 for \( a \). This leads to a long convergence time, but also to a stable behaviour.

Figure 6.19: Round-trip time calculation [Bauer et al. 96]

6.7 Multicast Adaptation Modules

Modules on the multicast adaptation level interface to multicast capable networks. The current implementation supports IP (IPv4) and ATM networks, although IP does not provide any bandwidth or delay guarantees. The multicast IP module uses the socket interface to access UDP and IP multicasting which are part of the operating system. The module can be viewed as a simple interface wrapper. It does not
provide more functionality than the unreliable multipoint-to-multipoint multicasting offered by IP multicasting. The ATM module, on the other hand, has to enhance ATM’s multicast capabilities as described in the next paragraphs.

FORE’s proprietary ATM signalling protocol SPANS provides unidirectional point-to-multipoint multicasting with bandwidth guarantees [Fore 96]. Multicasting is strictly sender oriented, multicast associations are established by senders, only. The implemented ATM module enhances this basic service and provides multipoint-to-multipoint multicasting with bandwidth guarantees. Furthermore, it allows dynamic joins of receivers as well as senders.

![Diagram](image-url)

*Figure 6.20: ATM multicast conversion*
For each multipoint-to-multipoint association, the ATM module maps the topology to several point-to-multipoint connections. For the administration of the point-to-multipoint connections, an ATM multicast server has been implemented (see Figure 6.20). This multicast server runs as separate UNIX® process. It distributes control information and manages the multipoint associations.

The mapping of the multipoint-to-multipoint topology is done by building a point-to-multipoint connection rooted at each sender. The critical operation is the join operation, which is different for senders and receivers. Whenever a sender joins, the point-to-multipoint tree has to be established. Therefore, the endpoint addresses of every receiver have to be transferred to the new sender, which builds the multicast tree. If a receiver joins, this receiver's address has to be sent to all existing senders, which in turn add this address to the multicast tree. For this purpose, a simple control protocol has been implemented. It is used by senders and receivers to access the services provided by the multicast server. It consists of the following primitives:

- **Sender_Join(c)** A sender registers itself at the multicast server for association c and requests the ATM endpoint addresses of all receivers of this association.

- **Join_Reply(c, a1, a2, ...)** The multicast server replies with the ATM endpoint addresses of all receivers of association c.

- **Recv_Join(c, ax)** A receiver registers itself at the multicast server for association c. The receiver supplies its own ATM endpoint address which allows senders to include this receiver in their multicast delivery trees.

- **Join_Indication(c, ax)** The multicast server notifies all registered senders of association c that a new receiver joined. The senders include the new sender in their point-to-multipoint connections.
6.8. Critical Assessment and Summary

There are no primitives for leaving a multicast association. This is due to the fact that ATM signalling already provides the needed functionality. If a sender or a receiver leaves a multicast association, it will be indicated by the ATM signalling to the peers. Similarly, closing a control connection will be indicated to the multicast server.

6.8 Critical Assessment and Summary

The described prototype implementation has been realized using SUN workstations that run Solaris™ and are interconnected using an IEEE 802.3 and ATM network. The implementation provides efficient transport services for multimedia, multipoint applications, as will be shown in the next chapter. However, the implementation also has some imperfections which are described in the following sections. The last section shortly summarizes the capabilities of the implementation.

6.8.1 Shortcomings of the ATM Implementation

The ATM module gains access to the local ATM network by means of FORE’s proprietary API, which itself is based on ATM adaptation layer 5 and FORE’s signalling protocol SPANS [Fore 96]. As a network adapter, FORE’s SBA-200 adapter card is used. The API is a socket-like interface that offers a single service either over AAL5 or AAL3/4. The service may be configured during the connection setup using parameters for peak bandwidth, mean bandwidth and mean burst size. As such, the service very much resembles the real-time Variable Bit Rate service (rt-VBR) specified by ATM Forum’s traffic management specification [Forum 96]. However, unlike rt-VBR, FORE’s API neither allows to specify a delay bound nor does it indicate a delay bound for established connections. Therefore, the ATM module estimates a delay bound which is based on off-line measurements. It is clear that this delay bound may be too large or too short, depending on the environment.

Another imperfection lies in the fact that for established connections, no parameter re-negotiation can be done. Therefore, the straight forward solution is to close and re-open a connection with changed pa-
rameters. However, in this situation the following sequence of events may occur, leading to an obvious problem:

- A connection is closed because the bandwidth should be increased.
- The new connection setup is turned down since the additional bandwidth is not available.
- The 'old' connection also can not be re-opened since the previously allocated bandwidth has been re-used by the network.

The solution applied in MCF is to first establish the new connection before closing the old one. However, this solution has the drawback that resources are temporarily over-allocated and that due to this over-allocation a reconfiguration may fail when it shouldn't. This is the case if the new connection can not be established since a part of the needed bandwidth still is allocated by the old connection.

At the time of writing, FORE is designing a new API which overcomes the first of the above mentioned drawbacks. This new interface supposedly complies to the X/Open XTI standard and offers an rt-VBR conform API.

### 6.8.2 GMS Integration

For the group and management system described by [Wilde 97], a prototype implementation exists. This prototype consists of both user agents and system agents. In MCF's design, flow and session descriptions are stored in the system agents where they are accessed over the GMS user agent. However, MCF's current implementation does not include a GMS user agent. The reason is that the existing user agent has been implemented using UNIX® threads. If called by MCF, these threads are competing for the CPU with the real-time scheduler and thus prevent that tasks can be executed within their deadlines. The problem occurs with any library that uses multiple threads. A solution of this problem is to redesign the libraries and replace the UNIX® threads with MCF tasks.
The current implementation of MCF runs in a local environment only. In this environment, all workstations share a common file system. Flow and session descriptions are stored on the shared file system and thus don’t have to be distributed by the GMS.

6.8.3 Support for Heterogeneous Receivers

The prototype implementation does not include support for heterogeneous receivers.

6.8.4 Summary

MCF's prototype implementation offers multipoint communication services with performance guarantees. Guarantees are given by reserving resources in the endsystem and the network. In order to provide end-to-end guarantees, the prototype uses real-time capabilities of SVR4 UNIX® and the local ATM network.
In the previous chapters, the design and the implementation of MCF have been presented. In order to evaluate the usability of MCF’s approach, several points have to be investigated. First, the usability as a communication framework is regarded. Section 7.1 identifies the applications which benefit most from MCF. Additionally, a simple application is presented that shows how MCF may be applied. The last part of Section 7.1 shows how a user perceives the quality of MCF’s services. In section two, the performance of the implementation is evaluated. For the transport subsystem, scheduling overhead and protocol execution times are presented. This evaluation shows the benefits of real-time scheduling in combination with dynamic protocol composition. In the control and management subsystem, performance meas-
urements of the QoS re-negotiation are shown. In the following section, scalability in the number of receiver and limits thereof are presented. The last section summarized the results of the evaluation.

7.1 Usability as a Communication Framework

This section evaluates the usability of MCF as a communication framework. For the usability evaluation, three criteria are applied. First, classes of applications which benefit from using MCF are identified. Clearly, a communication framework should be able to support a large number of different applications. The second criteria evaluates how applications access the services provided by MCF. Access to the communication services should be user friendly and simple. A sample audio conference application is presented that shows how the API is being used. In the last subsection, the quality of MCF's services is evaluated. In particular, the end-to-end performance guarantees are critically reviewed.

7.1.1 Supported Application Classes

MCF has been developed to support multipoint multimedia applications. It offers multipoint communication services with end-to-end performance guarantees. Target applications are groupware applications that distribute continuous media among groups of users. Examples are conference applications that use audio and video to create virtual meeting places and applications that disseminate audio and video to a possibly large number of receivers. The main requirements of these applications are performance guarantees and a multipoint communication service. Performance guarantees are provided by using a real-time scheduler for periodic tasks that guarantees a worst-case execution time for all modules. In particular, the guarantees include the presentation of multimedia data to the user, since this presentation is done by the multimedia support modules. The multipoint communication service offered by MCF provides multipoint-to-multipoint associations where participants are allowed to dynamically join and leave associations without disturbing other participants.
MCF, however, may also be used for transporting non-continuous media data among a group of participants. Data are then fetched from the application instead of a multimedia device. For these data, MCF also provides performance guarantees, i.e. a minimum throughput as well as a maximum end-to-end delay. Examples of applications that need performance guarantees for non-continuous media data are applications that collect real-time data measured by some sensors or applications that control and steer a set of machines.

Of course, MCF may also be used by applications that do not need strict performance guarantees. In particular, MCF might be used by applications that need a reliable multipoint-to-multipoint service, as it is used for the QoS re-negotiation protocol. Examples are session management applications in conferencing systems or shared whiteboard applications. Although these applications do not need performance guarantees, they still need to specify QoS requirements which will lead to an exclusive reservation of endsystem and network resources.

MCF may also be used to provide point-to-point communication services with performance guarantees, since a point-to-point association is a special case of a multipoint-to-multipoint association. This special situation will not lead to a waste of resources, since MCF takes into account the maximum number of senders and receivers for the resource reservation.

Applications that neither need multipoint communication services nor performance guarantees do not benefit from MCF. These applications better use existing communication services as provided by TCP or UDP.

7.1.2 Audio Conference Application

The goal of the audio conferences application is to allow several users to speak and listen to each other simultaneously, as shown in Figure 7.1. Each participant uses a workstation which is equipped with a loudspeaker and a microphone. As in a telephone conference, everybody can hear what everybody else is saying. Audio data is exchanged
using a single audio flow. Microphones are used as sources, loudspeakers as sinks of multimedia data.

![Audio conference application](image)

**Figure 7.1: Audio conference application**

The implementation of the application consists of two parts. A first part is used to create session and flow data structures and store their descriptions in the GMS\(^1\). This first part needs to be executed only once, typically it is used by the initiator of the conference. The details of the create operation is shown in the following code fractions. First, the data structure for the session is created. The session contains the same audio flow once for sending and once for receiving:

```plaintext
GmsSession session := {
    "audioconference", /* name of the session */
    "audioflow",    /* flow used for sending */
    "audioflow",    /* flow used for receiving */
};
```

The audio flow is defined in the ensuing piece of code. It selects the modules for each layer. On the multimedia layer, an audio module that reads data from the microphone is chosen, on the transport layer, the segmentation and reassembly module and on the multicast adaptation layer the ATM module are selected. Furthermore, limits on the number of senders and receivers are given as well as delay split factors which are used to split the end-to-end delay into three parts. In this example, the send delay is 60% of the end-to-end delay, the network transmission is 15% and the remaining 25% is the delay at the receiver. As a

---

\(^1\) The current implementation emulates the GMS functionality by distributing flow and session descriptions to all participants using a distributed file system.
last element, the group address is specified. The group address is an integer value for both ATM and multicast IP module. In case of IP, the value is directly used as IP multicast address, whereas for ATM any value may be used.

```
GmsFlow audioflow := {
    "audioflow", /* name of the flow. */
    "AudioMic", /* audio module */
    "Segmenter", /* segmentation and reassembly */
    "AtmAAL5", /* ATM multicasting over AAL5*/
    5, /* max nr. of senders */
    5, /* max nr. of active senders */
    5, /* max nr. of receivers */
    5, /* max nr. of active receivers */
    0.6, /* delay split factors. */
    0.15,
    12345; /* group address identifier */
};
```

The creating application also defines the QoS parameter space of the flow. Each QoS parameters consists of a name and of a value part. The value part is composed of a type which is either an integer range, a sorted list or a character string. Furthermore, the value part consists of a default value that denotes the current working point as well as of the strongest and weakest value that are allowed. As an example, the QoS parameters for the sampling rate and the number of channels are given. The values in the sampling rate parameter are specified as list; it contains all the rates which are supported by the audio device. The number of channels is given as an integer range.

```
qos[0].name := "samplingrate";
qos[0].type := List;
qos[0].val.value := 9; /* index in list */
qos[0].val.weakest := 1; /* index in list */
qos[0].val.strongest := 10; /* index in list */
qos[0].val.elements := {8000,9600,11025, 16000, 18900, 22050, 32000,37800, 44100,48000};
qos[1].name := "channels";
qos[1].type := Int;
qos[1].val.value := 2;
qos[1].val.weakest := 1;
qos[1].val.strongest := 2;
```
The session and the two corresponding flows are created using two primitives of the API, as shown below:

```c
api_CreateSession(session);
api_CreateFlow(audioflow,qos);
```

After session and flow have been created, participants may join the session in order to take part in the audio conference. In order to join, the application only has to provide the name of the session as well as the role of the flows. The role identifies whether a flow is being used for sending or receiving. In the conference example, the audio flow is used once for sending and once for receiving. Furthermore, the application has to state whether a control flow for QoS re-negotiation should be created. Without a control flow, no QoS re-negotiation can be done. The result of the join session operation is a positive session identifier if the call succeeded. Otherwise, join session failed. The reason for the failure is passed in the `perflowres[]` array, which indicates the results of all admission tests for all flows. After a successful join operation, the session may be started using the session id that has been provided by the join session function.

```c
role[] := {sender, receiver};
ctrlflow := TRUE;
sid := api_JoinSession("audioconference", role,
                        ctrlflow, perflowres[]);
api_StartSession(sid);
```

After the session has been started, data is being transported autonomously by MCF. The application is not blocked by the start session call and may proceed with any other task.

### 7.1.3 Perceived Quality of MCF’s Services

This subsection evaluates the perceived quality of the communication services that are provided by MCF. As an example, the previous audio conference application is being used where the quality of the audio flow is being investigated. The perceived quality depends on whether end-to-end guarantees can be provided by MCF.
A user perceives a satisfactory quality, if end-to-end performance guarantees can be provided. Taking the previous application as an example, end-to-end spans the path from the microphone of a sender to the loudspeakers of the receivers. If data is produced according to a linear bounded arrival process (LBAP), then it is sufficient to provide throughput and delay guarantees on each component that processes multimedia data on the end-to-end path [Anderson 93]. A LBAP produces messages with a maximum size, a maximum rate and a maximum burst size. Digital audio data are sampled at a fixed rate with fixed samples sizes, e.g. 16 bit samples at 44100 Hz and thus conform to the LBAP definition. Video data also conforms to a LBAP.

The components which are used in the end-to-end path of an MCF audio flow are shown in Figure 7.2 and described below:

1. **Audio device sender**
   The audio device of the sender samples the analog audio data of the microphone. The digital data are transported to MCF using a device driver. The audio device is exclusively used by MCF's audio flow. Limits on throughput and delay are guaranteed, since the device driver is operated within the context of the MCF process, which itself is being executed with real-time priority.

   The workstations which have been used for testing are equipped with audio devices that do not provide high-end analog-digital conversion. The digital data contains background noise; its quality is less than HiFi, even if sampled at 44100 Hz and 16 bit sample size.
(2) **MCF sender**

The audio samples are processed by modules inside MCF. The processing is controlled by the real-time scheduler, such that all of the samples are processed within their deadlines. The behaviour of the scheduler has been verified using the following simple test: While transmitting the contents of an audio compact disc, the scheduler has been instructed to log each missed deadline. At the same time, several CPU intensive processes have been started on the endsystem. However, the scheduler, using the real-time priorities provided by Solaris\textsuperscript{TM} did not miss its deadlines.

(3) **End-to-end multicasting service at the sender**

MCF forwards the audio samples using the end-to-end multicast transport service provided by the operating system. End-to-end multicast transport services are provided by both ATM and UDP/IP multicasting protocols. However, these protocols also forward data which origin from other processes that run on the same endsystem. In order to guarantee throughput and delay, a packet scheduler that schedules the data according to their priorities is needed. Currently, such a scheduler is only provided for ATM in the form of an ATM cell scheduler but not for IP. Only if the current version of IP is enhanced with a packet scheduler such as described in the integrated services architecture for the internet, throughput and delay guarantees can be given.

(4) **Data transport in the network**

The network is used to transport the audio samples to the receiving endsystems. MCF instructs the network resource manager to reserve the needed bandwidth and to guarantee limits on the delay. The ATM network transport the audio samples with the needed guarantees, whereas the current version of IP transports data with ‘best effort’.

(5) **End-to-end multicasting service at the receiver**

The end-to-end multicast transport service at the receiver forwards the incoming audio samples to the receiving MCF system. Similar to the sending side, data originating from different associations have to be processed according to their priorities.
In particular, a priority based buffer space management is needed in order to provide performance guarantee. Priority based processing is provided by ATM, but not by the current implementation of IP.

(6) MCF receiver
The receiving MCF system processes data also using the real-time scheduler and thus provides performance guarantees analogous to the sending MCF system.

(7) Audio device receiver
The audio device is accesses exclusively by MCF's audio flow. It converts the digital audio samples into analog signals which are then played on the loudspeaker. Throughput and delay are guaranteed, since the device driver operates in the context of MCF, which runs with real-time priorities.

On top of ATM, MCF is capable of transporting the audio data continuously from the sender to the receivers. A user of an audio flow perceives a smooth and steady playback of the audio data without interruptions. Due to the analog-digital conversion, background noise is introduced, such that audio data which are sampled at 44100 Hz with 16 bit sample sizes are played back with the quality of a radio receiver.

7.2 Performance Measurements

The performance of the implementation allows to draw conclusions about MCF's design. An adequate performance proves that the design can be implemented efficiently.

Performance measurements have been carried out for the scheduling overhead, the protocol execution times and for the QoS re-negotiation, as described in the following sections. The measurements have been conducted using the audio and control flow of the aforementioned audio conference application with different QoS settings. Three different types of endsystems have been used to measure the performance, namely Sparc10™ workstations clocked with 40 Mhz, Sparc20™
workstations with 60 Mhz and UltraSparc™ machines clocked at 167 Mhz. Thus, the performance on slow, medium and very fast workstations can be shown.

7.2.1 Scheduling Overhead

The scheduling overhead is measured in order to evaluate the suitability of the rate-monotonic scheduler for protocol execution. Scheduling overhead is understood as the time that is used for MCF tasks switches as well as for the administration of the schedule, i.e. inserting and removing scheduling events, compared to the protocol execution time. The scheduling overhead directly depends on the average execution time per task. Table 7.1 shows measurements for different flows which have been carried out on a UltraSparc™ 167MHz workstation. Flow A is an audio flow that is used to receive telephone quality audio from a single sender. The low audio quality results in very small packet sizes of 336 bytes and thus a very low protocol execution time per period. The scheduling overhead of the receiving flow is quite large, about 15%. Flows B and C are receiving and sending flows for CD quality audio samples. Due to a larger packet size, the scheduling overhead is smaller than in the first flow. Flow D is an audio flow that uses the multicast error control module as transport module. An artificial packet loss rate of 20% has been used to stimulate frequent retransmission and thus increase the protocol execution time, resulting in a scheduling overhead of about 6.5 percent. Flow E is an audio flow that receives CD quality audio samples from 4 senders simultaneously. Since the flow has to carry out audio mixing, the protocol execution time is larger than for the other flows, resulting in a scheduling overhead of about 3.5%. The last flow shows an audio flow for receiving which has been executed on a Sparc10™ 40MHz workstation. As it can be seen, the scheduling overhead is very low.

The large differences in the scheduling overhead are caused by the handling of the UNIX dispatch latency in MCF’s scheduler. The sleep call which is used by the scheduler only has an accuracy equal to the dispatch latency of the underlying UNIX system. The scheduler therefore reduces the sleep time by a worst-case assumption of the dispatch latency and enters an idle loop until the effective sleep time has arrived
7.2. Performance Measurements

<table>
<thead>
<tr>
<th>Flow</th>
<th>Mean execution time per task</th>
<th>Absolute overhead in μs per task switch</th>
<th>Relative overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>44 μs</td>
<td>6.4 μs</td>
<td>14.3 %</td>
</tr>
<tr>
<td>B</td>
<td>52 μs</td>
<td>6.3 μs</td>
<td>12.0 %</td>
</tr>
<tr>
<td>C</td>
<td>61 μs</td>
<td>6.3 μs</td>
<td>10.3 %</td>
</tr>
<tr>
<td>D</td>
<td>97 μs</td>
<td>6.3 μs</td>
<td>6.5 %</td>
</tr>
<tr>
<td>E</td>
<td>182 μs</td>
<td>6.3 μs</td>
<td>3.5 %</td>
</tr>
<tr>
<td>F</td>
<td>410 μs</td>
<td>5.6 μs</td>
<td>1.4 %</td>
</tr>
</tbody>
</table>

Table 7.1: Scheduling overhead

(see Section 6.3.1 on page 122). Most of the CPU time in the scheduler are consumed by this idle loop. For both UltraSparc™ and Sparc10™, the same worst-case value for the dispatch latency is used. This results in a larger relative scheduling overhead on very fast machines since the protocol execution times are small on those machines (see Figure 7.3). However, on heavily loaded machines, fewer sleep calls are needed and thus the scheduling overhead is reduced.

![Diagram](image)

Figure 7.3: Dispatch latency and scheduling overhead

MCF’s scheduling overhead is comparable or smaller than that of other systems. In a typical UNIX system, the overhead is about 10% but may be higher than 20% on a heavily loaded system [McKusick et al. 84]. MCF’s scheduling overhead is acceptable. By fine tuning the dispatch latency on each platform, the overhead could be reduced to be-
low 10% in the worst case. However, it must be noted that the UNIX scheduler provides pre-emptive scheduling, which induces a higher overhead.

### 7.2.2 Protocol Execution Times

The execution time of protocols shows whether dynamically composed protocols are suited for multimedia data transport. In a complex multimedia application such as a conferencing system, data transport is only a part of the whole application and therefore must not consume all of the available CPU. In Table 7.2, the performance of audio flows in different configuration is shown. Flows A, B, and C have been measured on an UltraSparc™, flow D on a Sparc10™. Flow A sends CD quality audio using a segmentation and re-assembly module over ATM multicasting. The execution time of the modules is very small, resulting in a relative CPU load of only 0.5%. Flow B is the corresponding audio receiver which results in about the same load of 0.6%. Flow C is an audio receiver that uses the multicast error control module on top of IP multicasting. An artificial packet loss rate of 20% is being used in order to trigger packet retransmissions. As it can be seen, the error control module has a very low execution time, since it operates with packet references, only. Due to retransmission requests which have to be sent and status packets which are received, the multicast IP module contributes significantly to the overall protocol execution time, which is about 1.4% on an UltraSparc™. Flow D has the same configuration as flow C, but it has been executed on a Sparc10™ workstation, where the protocol requires 5.1% of the CPU.

<table>
<thead>
<tr>
<th>Flow</th>
<th>Audio module</th>
<th>Transport module</th>
<th>Multicast module</th>
<th>Period</th>
<th>Relative CPU load</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>112 μs</td>
<td>5 μs</td>
<td>91 μs</td>
<td>40000 μs</td>
<td>0.5%</td>
</tr>
<tr>
<td>B</td>
<td>161 μs</td>
<td>3 μs</td>
<td>82 μs</td>
<td>40000 μs</td>
<td>0.6%</td>
</tr>
<tr>
<td>C</td>
<td>161 μs</td>
<td>7 μs</td>
<td>381 μs</td>
<td>40000 μs</td>
<td>1.4%</td>
</tr>
<tr>
<td>D</td>
<td>1187 μs</td>
<td>8 μs</td>
<td>832 μs</td>
<td>40000 μs</td>
<td>5.1%</td>
</tr>
</tbody>
</table>

*Table 7.2: Protocol execution time*
The CPU load induced by MCF's protocols is very small. This means that abundant CPU time is available for application processes. Furthermore, the measurements also show that MCF's dynamic composition of protocols provides an acceptable performance.

### 7.2.3 QoS Re-negotiation Performance

For the performance of the QoS re-negotiation, two values are important. These two values are the execution time of the QoS re-negotiation on a single machine and the time that the application has to wait until QoS re-negotiation is finished. In the following, measurements are presented that have been carried out on a Sparc20™ workstation, a moderately fast workstation which uses a Supersparc TMS390Z55 processor clocked at 60 MHz.

**QoS Re-negotiation Execution Time**

QoS re-negotiation is carried out in two steps. In a first step, each participant is requested to perform QoS backward mapping. In a second step, the participants execute QoS switching. The CPU time which is needed for those two operations is no longer available for the processing of flows.

The amount of time that is needed to perform backward mapping depends on the number of QoS vectors which are provided by the application. Figure 7.4 shows the execution times for QoS backward mapping of an audio flow. The backward mapping was carried out with two, four and eight QoS vectors, where each vector contained five QoS parameters. The execution time lies between 5.5 ms and 7 ms on a Sparc20™ workstation. This is more than ten times smaller than the average period of a flow and therefore small enough to not disturb the processing of flows. Furthermore, the duration increases linearly with the number of vectors, as could be expected. The number of vectors which is needed depends on the application. For practical reasons, however, applications are likely to use less than 5 vectors for the QoS re-negotiation.
Figure 7.4: QoS backward mapping

The QoS switch operation is used to put the selected QoS vector into effect. QoS switching first carries out admissions test in order to check whether sufficient resources are available. Then, each module is requested to release unnecessary or reserve additional resources as required by the new QoS vector. In particular, the multimedia module is requested to reconfigure the multimedia device. Figure 7.5 shows the distribution of the execution time of the QoS switch function. QoS switching is completed within 2 ms. As can be seen, most of the time is needed for the reconfiguration of the audio device, e.g. configuring another sampling rate or sampling size. The reconfiguration of the modules themselves takes considerably less time. The segmentation module, for instance, completes its reconfiguration within 1 μs and therefore does not appear in the diagram.

QoS re-configuration is composed of QoS backward mapping and QoS switching. The total execution time is about 10 ms in two separate tasks. These tasks are executed within the control flow, which has a period of 250 ms. In addition to the 10 ms CPU for the QoS re-negotiation, the protocol overhead of the control flow must be taken into account, too. As shown in Table 7.2, the multicast error control and the IP multicasting module require less than 1 ms execution time for a sin-
7.2. Performance Measurements

Audio device reconfiguration
Admission tests
ATM reconfiguration
Resource calculation
QoS mapping

(Seeder reconfiguration not shown)

QoS switch execution time: 1908 µs

Figure 7.5: QoS switching execution time distribution

gle protocol step. For the QoS re-negotiation, being a two-phase commit protocol, two protocol steps are required. Therefore, the overhead introduced by the QoS re-negotiation is about 12 ms of 250 ms, which is about 5% of the total CPU load.

Perceived QoS Re-negotiation Duration

The time an application has to wait until a QoS re-negotiation is finished is considerably larger than the execution time of the involved tasks. Delay is introduced by two factors. Firstly, the QoS re-negotiation tasks have a low priority, i.e. a long period. Therefore, it may take some time until the scheduler selects these tasks. Secondly, the network delay has to be taken into account. Since the QoS re-negotiation protocol is a two-phase commit protocol, two round-trip times are needed.

Figure 7.6 shows a number of measurements which have been done in various configurations. As a testbed, a number of workstations connected to both a local IEEE 802.3 and an ATM network have been used. The control flow ran on top of the IEEE 802.3 network, whereas the multimedia flow used the ATM network. The measurements show the time that elapses on the workstation that triggered the QoS re-negotiation until the negotiation is finished, i.e. the new vector has been instantiated. The first series has been done using a Sparc20™ and an UltraSparc™ workstation. The execution time spans from 25 ms to
50 ms, depending linearly on the number of vectors that are negotiated. In a second test, four workstations have been used. In particular, a relatively slow Sparc10™ has been used. In this set, the time needed for the QoS re-negotiation reaches from 40 ms up to 130 ms. Consequently, the single slow machine leads to an increased negotiation time. The last two series have been carried out with six and eight participants. In these tests, UltraSparc™, Sparc20™ and a single Sparc10™ have been used. Since the additional machines are all faster than the Sparc10™, the negotiation time is not increased significantly, it lies between 50 ms and 150 ms, depending on the number of QoS vectors.

![Figure 7.6: Perceived QoS re-negotiation time](image)

The results show that the QoS re-negotiation can be done in about a tenth of a second, which is almost instantaneous from a user's perspective. Although this time seems to be long compared with protocol execution times, it must be noted that multimedia data streams are not interrupted during QoS re-negotiation, but only during QoS switching, which is completed in about 2 ms.
7.3 Scalability

In the scalability evaluation, MCF is evaluated with respect to the number of participants that it can support. The scalability contemplations is done in three parts. First, the number of receivers and second the number of senders that may be supported is evaluated. As a third point, the QoS re-negotiation protocol and its effect on scalability is investigated.

7.3.1 Scalability in the Number of Receivers

For the evaluation of the number of receivers, it is assumed that data are disseminated by a single sender to a number of receivers. Furthermore, it is assumed that on the transport level the segmentation and re-assembly module is used and that no QoS re-negotiation protocol is instantiated. In this situation, scalability is affected by the following factors:

- **Join**
  For the join operation, receivers request session and flow descriptions from the GMS. Scalability for this operation is only affected by the scalability of the GMS. As it is argued in [Wilde 97], GMS is as scalable as the world-wide domain name system.

- **Admission tests and resource reservation**
  Whereas the management of endsystem resources is a local matter and therefore does not affect scalability, network resources have to be considered carefully. For these resources, admission test and reservation is integrated into a single operation. In the given scenario, each joining receiver has to check whether sufficient bandwidth from the sender with the required delay is available. The scalability of these operation depends on the network and its resource manager. In MCF's prototype implementation for ATM, a multicast server is the limiting factor for scalability. The current implementation supports 128 receivers. The IP multicasting implementation does not reserve any network resourc-
es, scalability is therefore not limited by network resource management.

- **Endsystem resources**
  The endsystem resources required by each receiver are independent of other receivers and therefore do not pose any scalability limitations. Likewise, the sender’s resources are also independent of the number of receivers. Therefore, scalability is not limited by endsystem resource.

- **Protocol execution**
  The protocol stack used in the given scenario is unidirectional only, data are transported from sender to receivers. At the sender, the multimedia support and the transport layer are not affected by the number of users. The multicast adaptation module uses the native multicast capabilities of the underlying networks, its scalability therefore depends on the scalability of the network’s multicast implementation. IP multicasting has proved to be scalable to hundreds of users during IETF audiostream sessions. In the case of ATM, switch manufactures give scalability figures in the range of several hundred receivers.

### 7.3.2 Scalability in the Number of Senders

In order to estimate the number of senders that can be supported, it is assumed that a single receiver gets data from several senders simultaneously. In this scenario, the receiver has to process data from all senders. Scalability is limited in the first place by the receiver’s resources. For each active sender, the receiver allocates CPU, since during a single period packets of all existing senders have to be processed. Furthermore, modules may reserve memory depending on the number of senders. The segmentation and reassembly and the error control modules have to keep a state machine for each sender. Error control modules additionally have to keep a buffer per sender that is needed for re-ordering incoming packets.

As an example, the number of senders that may be supported simultaneously in an audio flow is limited to less than 20 on a Sparc10™ workstation and less for flows requiring more CPU time.
The number of active senders that can be supported is far smaller than the number of receivers. This limitation affects mainly applications that implement symmetric communication such as the audio conference application presented in Section 7.1.2. Such applications, however, are not suited for a large number of participants, anyway. For practical reasons, a bi-directional audio communication may only be conducted if the number of simultaneous speakers is less than two or maybe three. An audio conference application with a large number of participants, but only a small number of simultaneous speakers can still be implemented using MCF, since most of the resources are only allocated for active users. However, no QoS re-negotiation can be done for applications with a large number of participants, as the next section shows.

### 7.3.3 QoS Re-negotiation and Scalability

For the QoS re-negotiation protocol, each participant of a session is a sender and a receiver at the same time. Scalability in the QoS re-negotiation protocol is limited, since each participant that may trigger a QoS re-negotiation has to reserve resources such that incoming control messages from all other participants may be processed. Scalability in the number of participants is therefore reduced to at most 30 participants when using Sparc10™ workstations.

The question is whether QoS re-negotiation can be done more effectively such that scalability can be improved. Multilateral QoS re-negotiation always consists of a proposal for a change in the QoS that is distributed to all participants. Each participant has to reply to this proposal. Scalability now is limited due to the fact that all of the proposals have to be collected and evaluated centrally. In order to increase the number of participants, the following methods may be applied:

- **Sequential processing**
  Participants send their reply sequentially such that only a small number of replies has to be processed simultaneously. Thus, the amount of resources needed is decreased, the approach is more scalable. However, the time that is required for a QoS re-negotiation is proportional to the number of participants. Furthermore,
this approach requires that the participants coordinate the sending of replies, which also requires communication among the participants.

- **Hierarchical processing**
  In the hierarchical approach, participants are arranged in a tree-like structure. The participant that triggers the QoS re-negotiation is at the root of the tree. The re-negotiation request travels down to the leaves of the tree. Replies are sent from the leaves to the root of the tree. Nodes inside the tree collect all the replies of their children, preprocess them and forward them towards the root. This solution increases scalability, since each node only has to provide resources for its direct children. A minor disadvantage is that the time that is needed for a QoS re-negotiation increases logarithmically with the number of participants. However, the main problem of the approach is that for each participant that wants to carry out a QoS re-negotiation, the tree structure has to be established. The coordination effort and time that is required to build such a tree easily surpasses the one required to carry out the actual QoS re-negotiation.

A solution to this scalability problem is to change the type of QoS re-negotiation. For applications that require a form of QoS negotiation and at the same time have to be scalable, an unilateral QoS re-negotiation can be used. In this form of QoS re-negotiation, a new QoS vector is selected and distributed to all participants. Participants try to switch to the new vector. If they fail, they have to leave the session. Although this is no longer a true re-negotiation, it allows for changing the QoS in a session by being scalable at the same time.

### 7.4 Critical Assessment

The evaluation investigated the usability of MCF as a supportive tool for application development. It proved that applications requiring multimedia multipoint communications can easily be build using the available toolkit. Furthermore, the evaluation also showed that MCF indeed provides true end-to-end performance guarantees.
So far, MCF has not been used for building a large application such as a complex conference system consisting of audio, video and application sharing components. Such an application might reveal weak points in the system and may also lead to the development of more advance modules.

The performance measurements have been conducted in a local environment and demonstrated that an efficient implementation of the design can be done. A quantitative estimation revealed that MCF scales to about 100 receivers which is sufficient for most applications, although the scalability could be better. Scalability in the number of senders is limited. In particular, QoS re-negotiation limits scalability to about 30 participants. By using an alternative, less capable algorithm, scalability could be improved.
Chapter 8

Conclusions and Outlook

The design and implementation of a multimedia, multipoint communication framework has been presented in this thesis. Its fundamental concepts are multipoint communication and quality of service principles which have been presented in Chapter 2 and 3, respectively. The analysis of related work in Chapter 4 revealed that no framework exists that fulfils the requirements which have been identified in Chapter 1. In the following section, the claims put forward in Section 1.2 are verified by assessing design, implementation and evaluation of the multipoint communication framework.
8.1 Review of Claims

In this chapter, the claims which have been identified in the first chapter are revisited. Each claim is verified with respect to MCF's design and implementation.

Claim 1: Multipoint-to-Multipoint Communication
*MCF offers multipoint-to-multipoint communication, where several senders distribute data to groups of several receivers.*

MCF's architecture distinguishes a transport and a multicast adaptation layer. Multipoint-to-multipoint communication services are embedded in the multicast adaptation layer, which are then used on the transport layer by any transport module. The details of multicast adaptation modules depend on the underlying network. The implementation showed that IP multicasting as well as ATM multicasting can be used to build an efficient multipoint service. In case of ATM, the basic point-to-multipoint service has been extended to a multipoint-to-multipoint service. The architecture also allows to build multicast services on top of point-to-point networks. However, such a service would not be scalable, since for each sender/receiver pair, a single association would be necessary.

Claim 2: Multipoint Dynamics
*The design supports multipoint dynamics, i.e. dynamic join and leave of participants during the communication.*

The dynamic join capabilities are made possible by several architectural measures. First, session and flow descriptions which are needed to join are stored and distributed separately from multimedia data. The group and session management system is used to hold this control information, where it can be accessed by any participant that wishes to join a session. As a second point, the join operation of a new participant does not affect the resource management of existing participants. In particular, the end-to-end delay is split into a delay at the sender, a network transmission delay and a delay at the receiver. Thus, newcomers only have to consider local and network resources, but not resources of other participants. As a third point, the control flow which is needed for QoS re-negotiation dynamical-
ly adapts to changes in the multicast group. Furthermore, as the control protocol is fully symmetric, each participant may trigger a QoS re-negotiation.

Claim 3: End-to-End Performance Guarantees

*MCF provides performance guarantees for multimedia data. This means, for example, that a constant frame rate for video transmission can be maintained.*

Performance guarantees are provided by reserving endsystem and network resources. Endsystem resources such as CPU, memory and multimedia devices are managed by MCF itself, whereas admission tests and reservations for bandwidth and network transmission delay are delegated to network resource managers.

The admission test and resource reservation for the CPU is done using a real-time scheduler that works on periodic task sets. This scheduler guarantees that flows are executed within their deadlines. Flows reserve memory and multimedia devices only after a successful admission test. Since all resources are reserved prior to being used, performance guarantees can be provided.

Claim 4: Scalability

*MCF supports applications that disseminate data to a large number of receivers.*

Measurements have shown that protocols in MCF are executed very efficiently. By using the distributed GMS system, a large number of receivers may be supported. However, scalability in the number of receivers is limited to about 100 in the case of ATM. Limitations are induced by the centralized architecture that is needed in order to provide multipoint-to-multipoint multicasting over ATM. In order to improve this weakness, alternative designs must be investigated. In particular, the use of advanced ATM signalling protocols such as UNI 4.0 has to be considered.

Claim 5: Transparency

*Applications specify their requirements in a transparent way and do not have to know any mechanisms or algorithms used inside the framework.*
Applications use the API to access MCF's services. Two API primitives have to be distinguished. For the create operation, which has to be carried out once for each session, the application has to specify session and flow descriptions. These descriptions contain media specific QoS parameters, limits on the number of participants as well as delay split factors which specify how much of the end-to-end delay may be used for sender, network transmission and receiver. The specification of delay split factors clearly opposes the transparency requirement. Delay split factors are configurable by the application to allow maximum flexibility, such that the application may set different factors depending on the network topology. Applications that require full transparency may use pre-defined templates to create their flow descriptions. The QoS parameters only depend on the media type that is transported. MCF uses QoS parameter mapping and resource calculation to provide a complete transparency to the application.

The second primitive is the join operation which is carried out by each participant of a session. The join operation only asks for a session name, MCF's services are completely hidden behind the API.

Transparency of network multicast capabilities are hidden from the application by the protocol layering. Since each multicast adaptation module offers the same multicast capabilities to the transport layer, a general abstraction of a multipoint communication service is created.

Claim 6: Adaptable Media Quality

*Applications may change the quality of the real-time media during the communication without having to stop and restart the communication.*

The quality of the transported media may be changed by each participant using QoS re-negotiation. For the QoS re-negotiation, the application has to specify a list of QoS vectors which define the desired quality. The QoS vectors are sorted by the application's preference. The current QoS re-negotiation algorithm finds the preferred quality that can be supported by all participants, although alternative algorithms could be used as well.
QoS re-negotiation and transport services are executed simultaneously. A short interruption of flows occurs only when they are reconfigured during the QoS switch operation.

Claim 7: Heterogeneity

The framework allows for heterogeneous receivers. Receivers that are unable to process the offered amount of data may reduce the volume, albeit at the expense of a degraded quality. Likewise, some receivers may tolerate a larger delay than specified by the application.

Support for heterogeneous receiver is included in MCF's design. The create operation allows for the definition of a so-called downgrading area in the QoS space that defines to which extend the quality might be downgraded. Downgrading of the quality is done by either configuring filters which are placed in the network's intermediate system or by filtering incoming packets in the multicast adaptation modules. The prototype implementation, however, does not support heterogeneous receivers.

Claim 8: Flexibility and Extensibility

Applications may distribute any real-time media over any network that is supported by MCF. The composition of media type and network is done at runtime, such that applications don't have to be adapted. Furthermore, MCF can be extended to support new media as well as new networks without affecting existing applications.

Flexibility is provided by composing protocol stacks at runtime. Protocol composition allows applications to use any combination of multimedia, transport and multicast adaptation modules. The protocol definition is part of the flow description which is provided by the application during the create operation. By exchanging a single parameter in the flow description, the used network type can be changed. The same approach also offers extensibility. New media and network types may be supported by implementing and integrating additional modules into MCF. Existing applications that do not require the new functionality don't have to be changed.

As the above list shows, MCF's design fulfills the claims which have been identified in the first chapter. However, not all of the design
points has been realised in the prototype implementation. In particular, support for heterogeneous receivers is not part of the current implementation. Nevertheless, the prototype implementation shows that all of the other claims can be implemented efficiently.

8.2 Suggestions for Future Work

In this section, suggestions for further work for MCF and for general research topics are given.

8.2.1 Improvement of MCF

MCF's implementation provides a suitable base for testing design extensions and conducting other experiments. The implementation may be developed further in the following points:

- **Support for heterogeneous receivers**
  The implementation should be extended to include support for heterogeneous receivers. For this purpose, filters have to be implemented in the multicast adaptation modules. Furthermore, the join operation has to be extended such that filters are configured according to results of the admission tests.

- **Alternative QoS re-negotiation algorithms**
  The QoS re-negotiation algorithm implements a multilateral negotiation among all participants. The QoS vector is selected by choosing the most appropriate vector that can be supported by all participants. Alternative algorithms may need other mechanism for selecting the resulting QoS vector. A possible algorithm might select the most appropriate QoS vector that is supported by at least 80% of the participants. Additionally, the basic re-negotiation scheme might has to be changed such that QoS re-negotiation becomes more scalable.
  An implementation of additional QoS re-negotiation algorithms should be done such that the application can choose which of the algorithms it wants to use. This is easily possible by using the dynamic protocol composition. The current algorithm is already
8.2. Suggestions for Future Work

implemented as a module which is dynamically instantiated for each session that requires QoS re-negotiation.

- **Synchronisation of multimedia streams**
  MCF provides guarantees on the maximum end-to-end delay. Thus, a coarse synchronisation of streams originating from a single sender can be done by specifying the same end-to-end delay for all these streams. However, some applications require a finer control for the synchronisation or they need to synchronize multimedia streams of several different senders. For these applications, synchronisation has to be included in MCF. The main challenge in designing synchronisation mechanisms for MCF lies in the integration of these mechanisms with the existing real-time scheduler.

- **Automatic generation of profile database entries**
  The profile database contains measurements of execution times for each protocol function in each module for each machine. Each time MCF is executed on a new machine or each time the hardware configuration of a machine changes, the profile database needs to be updated. It is desirable that these measurements are conducted automatically, since it takes a considerable amount of time if done manually.

- **Integration of an IP integrated service architecture**
  Currently, several prototype implementations of the integrated service architecture for the internet are being developed. These implementations consist of the RSVP signalling protocol which interfaces to a packet scheduler which implements class-based queuing. MCF needs to be extended by an interface to the integrated service architecture, such that IP multicasting can be extended with performance guarantees. Furthermore, MCF can then be used to compare the performance guarantees of ATM and the integrated service architecture.

8.2.2 Future Research Topics

MCF is based on the assumption that both network and endsystem are able to provide performance guarantees. MCF showed that modern operating systems such as UNIX® SVR4 provide the needed capabili-
ties. As a network technology, ATM has been used which offers bandwidth and delay guarantees, although the used equipment does not provide any information about actual end-to-end delay. It can be assumed that in the near future, real-time capable operating systems will be used on almost any workstation. However, only very few of these workstations will be connected to a network that provides bandwidth and delay guarantees. Although a few years ago it was argued that 'ATM to the desktop' will be ubiquitous, the current situation proves the contrary and is very likely not to change in the near future. Alternative technologies, mainly the integrated services architecture for the internet, have been hot research topics for the last few years. However, results in this area also show that an integrated services network is more complicated than originally assumed. Protocols and mechanisms are still being designed, standards are not in sight. Therefore, an integrated services network will not be operational for the next few years.

In this situation, the assumption that networks provide bandwidth and delay guarantees has to be dropped. Earlier research in the area of multimedia communication over best effort networks resulted in so called adaptive applications. These applications constantly adapt themselves to changes in communication services. As a further step, adaptive applications could be built that adapt to variations in network performance but rely on the guaranteed performance of the endsystem. By providing processing guarantees to adaptive applications, the adaptation on network changes may occur with the needed priorities. In particular, the network monitor that is needed to constantly observe the network's performance can be executed as a real-time task and therefore is capable to provide more accurate information than what's possible with a non-real-time monitor.

An architecture implementing real-time adaptive applications could use the same mechanism for providing endsystem guarantees as MCF. Endsystem resource management, including QoS mapping, resource calculation and resource reservation could be applied as well as the transport subsystem with its real-time scheduler and the profile database.
References


References


G. Coulson, A. Campbell, P. Robin, G. Blair, M. Papathomas, D. Hutchison: "The Design of a QoS Controlled ATM Based Communications System in Chorus", IEEE Journal on Selected Areas in Communications, Special Issue on ATM Local Area Networks, 1994


A. Danthine, O. Bonaventure: "From Best Effort to Enhanced QoS", Public Deliverable of RACE II Project CIO, R2060/ULg/ CIO/DS/P/004/b1, Liège, 1993


S. Deering: "Host Extensions for IP Multicasting", RFC 1112, August 1989
References


References


References


[Hadzilacos et al. 94] V. Hadzilacos, S. Toueg: “A Modular Approach to Fault-Tolerant Broadcasts and Related Problems”, Technical Report TR94-1422, Dept. of Computer Science, Cornell University. (The first part of this report can be found in [Mullender 93])


J. Moy: "Multicast Extensions to OSPF", RFC 1584, March, 1994


ITU-TS: "Broadband Integrated Services Digital Network (B-ISDN) - Digital subscriber signalling system no. 2 (DSS 2) - User-network interface (UNI) - Layer 3 specification for basic call/connection control", ITU-T Recommendation Q.2931, February 1995


C. Partridge: "Isochronous Applications Do Not Require Jitter-Controlled Networks", RFC 1257, September 1991
References


References


[Sun 95 B] Sun Microsystems: "Solaris 2.5 Introduction", Solaris 2.5 User Answer Book, Chapter 1, 1995


References


Acknowledgements

I would like to express my sincere gratitude to my examiner, Prof. Dr. B. Plattner, who introduced me to the world of academic research and also provided an excellent research facility for the work on MCF. I also like to thank my co-examiner, Prof. Dr. T. Plagemann, for many fruitful discussion. Together with Prof. Plattner, he taught me to discern sense from non-sense and kept me on the right track.

Many thanks also to my colleagues at the Computer Engineering and Networks Laboratory: Dr. B. Stiller and Dr. T. Walter for providing all the help in technical and non-technical matters; E. Wilde, G. Caronni, Hasan, Dr. T. Gutekunst, Dr. H. Lubich, M. Vogt, M. Brunner, C. Conrad, G. Fankhauser, C. Class, M. Dasen, G. Sidler for sharing the ups and downs of a life as a research assistant. Many thanks to the people who read and commented on the manuscript: Dr. B. Stiller, G. Fankhauser and C. Class.

Not to forget all the students who enjoyed my supervision and contributed significantly to the implementation of MCF: R. Keller, G. Nigg, F. Iten, M. Büechi and M. Schlapbach.

Also many thanks to my parents and parents-in-law who supported me during my work on this thesis.

The biggest thanks go to my wife Bettina Bauer, who patiently accepted that “some” people’s thesis take considerably longer then “other” people’s thesis. Without her encouragement and support, this work would not have been possible.
Curriculum Vitae

I was born in Oberbüren, Switzerland on February 13, 1968. I attended primary school and high school and graduated in September 1987 with a language/science diploma (Matura, Typ C). After the compulsory military service, I started my studies in computer science at ETH Zürich in October 1988. In March 1993, I received the diploma degree (Dipl. Informatik Ing. ETH) from ETH Zürich.

In my diploma thesis, which I carried out at the Computer Engineering and Networks Laboratory, I have implemented parts of a distributed editor system. After having finished my diploma, I became a research assistant at the Computer Engineering and Networks Laboratory. During the first two years, I have been member of the JVTOS project team, where an application sharing tool based on X windows has been developed. In 1995, I started my work on the multipoint communication framework.