INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

ProQuest Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

Université d'Ottawa · University of Ottawa

Quality of Service Management for Multicast Applications Using MPEG-4/DMIF

By

Zhen Yang

A thesis submitted to the
Faculty of Graduate and Postdoctoral Studies
In partial fulfillment of the requirements for the degree of

Master of Computer Science

In

Computer Science

Ottawa-Carleton Institute for Computer Science
School of Information Technology and Engineering

Faculty of Engineering
University of Ottawa

October, 2000

Abstract

The main objective of this thesis is to introduce an approach of quality of service management for multicast multimedia applications based on the Delivery Multimedia Integration Framework (DMIF) of MPEG-4 as a session management protocol. We consider that the distributed multicasting applications may involve a large number of users, therefore, a single quality of service level may not be appropriate for all participants. It is necessary to distribute part of the QoS management process and allow each user process to make certain QoS decisions based on its local context. This report gives an overview of some background knowledge, presents the design of a teleteaching system that uses our paradigm for QoS management, explains how DMIF can be adapted as a session protocol for such an application, and finally, provides the implementation details of the system.

Acknowledgements

I would like to thank some persons for their contributions to this thesis.

First, I would like to express my sincere appreciation to my supervisor, Dr. Gregor v. Bochmann, for giving me this opportunity to participate in the "Quality of Service Monitoring and End-User Control" research project. I really enjoyed working under his supervision. His work ethics, rich knowledge, encouragement, and patience throughout my research were the most important factors that helped me finished my M. Sc. program.

Many thanks to Mr. Vasilios Darlagiannis (student at the MCRLab) for the friendly discussions and the exchange of knowledge, which helped me a lot in understanding the DMIF concepts. I really owe many thanks to him since his implementation of the DMIF DNI layer is used in my implementation.

I am also deeply grateful to Dr. Nicolas. D. Georganas (Director of the MCRLab) and Mr. Francois Malric (Manager of the MCRLab) for providing me a comfortable environment and adequate research facilities in the early time of my research.

I would like to thank my colleagues at the "Distributed Systems" lab, especially Khalil El-Khatib, Qing Zhu, Tiejun Tang and Xiaoqing He, for their valuable comments, interesting discussions, supportiveness, and kindness. I really appreciate all of their help.

I would also like to acknowledge the support of CITO (Communications and Information Technology Ontario) and Nortel-Networks for the funds for this project.

Table of Contents

ABSTRACT I

ACKNOWLEDGEMENTS II

TABLE OF CONTENTS III

LIST OF ACRONYMS V

LIST OF FIGURES VI

LIST OF TABLES VII

CHAPTER 1: INTRODUCTION 1

1.1 Background 1
1.2 Main Contributions 2
1.3 Thesis Outline 3
1.4 Publication Arising from the Research 4

CHAPTER 2: MULTIMEDIA APPLICATIONS OVER INTERNET 5

2.1 Internet Protocols for Multimedia Applications 5
2.1.1 Internet Protocols - UDP/TCP/IP 5
2.1.2. IP Multicasting and MBone 9
2.1.2.1 IP Multicasting Overview 9
2.1.2.2 Group Addressing and TTL 12
2.1.2.3 MBone 14
2.1.3. Multicasting Protocols 18
2.1.3.1 Group Management Protocol 19
2.1.3.2 Multicast Routing Protocols 21
2.1.4 Higher Level Protocols 24
2.2 Different Voice / Video Encoding 26
2.2.1 Traditional Voice / Video Encoding 26
2.2.2 MPEG-4 29

CHAPTER 3: QUALITY OF SERVICE MANAGEMENT 32

3.1 Network QoS 32
3.1.1 QoS Definition 32
3.1.2 Related Works 34
3.2 Adaptive Applications 35
3.2.1 Adaptation Principles 35
3.2.2 Related Works 36

CHAPTER 4: SYSTEM REQUIREMENTS 40

4.1 Video Conferencing / Tele-teaching Applications 40
4.2 Requirement Specification 42

CHAPTER 5: PROVIDING QOS ALTERNATIVES FOR MULTICAST APPLICATIONS **45**

5.1 General Assumptions 45
5.2 QoS Alternatives 47
5.3. General System Architecture 47
5.4 A Typical Interactive Scenario 51

CHAPTER 6: PROTOCOL ALTERNATIVES **54**

6.1 Data Transport Protocols 54
6.1.1 TCP/IP vs. UDP/IP 54
6.1.2 RTP and RTCP 55
6.1.3 Protocol Choice and Overhead Analysis 56
6.2 Session Protocols 57
6.2.1 SAP/SDP 57
6.2.2 SIP 58
6.2.3 DMIF 59
6.2.4 Protocol Choice and Overhead Analysis 63

CHAPTER 7: USING DMIF FOR SESSION MANAGEMENT IN TELE-TEACHING APPLICATION **66**

7.1 A Typical Interactive Scenario 66
7.2 Changes to DMIF 71

CHAPTER 8: PROTOTYPE IMPLEMENTATION **74**

8.1 Language and Environment Choices 74
8.2 Key Design Issues 76
8.2.1 Global Program Structure 76
8.2.2 QoS Manager and QoS Agent Implementation 78
8.2.3 Format of QoS Alternatives 79
8.2.4 Format of DAI Primitive Parameters 79
8.2.5 User Profile Manager APIs 80
8.2.6 User Interface 81
8.2.7 QoS Adaptation Algorithm 88
8.3 Tests and Results 92

CHAPTER 9: CONCLUSION AND FUTURE WORK **95**

REFERENCES **99**

APPENDIX A: MULTICAST ROUTING ALGORITHMS **105**

APPENDIX B: FORMAT OF DAI PRIMITIVE PARAMETERS **109**

APPENDIX C: USER PROFILE MANAGEMENT API **115**

List of Acronyms

AVO	Audio/Visual Object
CBT	Core Based Tree
DAI	DMIF Application Interface
DCDT	Data Consumer DMIF Terminal
DMIF	the Delivery Multimedia Integration Framework
DNI	DMIF Network Interface
DPDT	Data Producer DMIF Terminal
DVMRP	Distance Vector Multicast Routing Protocol
ES	Elementary Streams
IANA	Internet Assigned Numbers Authority
IETF	Internet Engineering Task Force
IGMP	Internet Group Management Protocol
IP	Internet Protocol
ITU	The International Telecommunication Union
JMF	Java Media Framework
MBone	Multicast Backbone
MOSPF	Multicast Extension to OSPF
MPEG	Moving Picture Experts Group
MRouter	Multicast Router
PIM	Protocol Independent Multicast
PIM-DM	PIM-Dense Mode
PIM-SM	PIM-Sparse Mode
QoS	Quality of Service
RMI	Remote Method Invocation
RPB	Reverse Path Broadcasting
RPM	Reverse Path Multicasting
RSVP	Resource Reservation Protocol
RTCP	Real-time Transport Control Protocol
RTP	Real-time Transport Protocol
RTSP	Real-time Stream Protocol
SAP	Session Announce Protocol
SDP	Session Description Protocol
SIP	Session Initiation Protocol
TCP	Transmission Control Protocol
TRPB	Truncated Reverse Path Broadcasting
TTL	Time-To-Live
UDP	User Datagram Protocol

List of Figures

Figure 2.1 Example of Internet Layer Operation 6
Figure 2.2 Example of Unicast Transmission 9
Figure 2.3 Example of Multicast Transmission 10
Figure 2.4 IPv4 Class D Address (Multicast Address) 11
Figure 2.5 IPv6 Multicast Address 12
Figure 2.6. MBone Architecture 14
Figure 2.7 SDR Main User Interface 15
Figure 2.8 VAT User Interface 16
Figure 2.9 VIC User Interface 17
Figure 2.10 Example of IGMP 19
Figure 2.11 Example of DVMRP 21
Figure 2.12 General MPEG-4 Architecture 30
Figure 5.1 Overall System Architecture 44
Figure 5.2 General System Architecture 47
Figure 5.3 Example Interactive Scenario 52
Figure 6.1 Role of DMIF 59
Figure 6.2 DMIF Session Creation 59
Figure 6.3 Example Scenario of a Session Initiation 62
Figure 7.1 Multicast Session Interaction Scenario including DMIF (Part 1) 69
Figure 7.1 Multicast Session Interaction Scenario including DMIF (Part 2) 70
Figure 8.1 Sender Side Class Diagram 75
Figure 8.2 Receiver Side Class Diagram 76
Figure 8.3 QoS Alternatives Class Diagram 77
Figure 8.4 User Profile Manager Program Module Class Diagram 78
Figure 8.5 Sender Side Main User Interface 80
Figure 8.6 Video Variants Create and Edit 81
Figure 8.7 Channel Request by Receiver 82
Figure 8.8 Receiver Side Main User Interface 82
Figure 8.9 Create a New User 82
Figure 8.10 Profile Edit 84
Figure 8.11 QoS Management 85
Figure 8.12 Session Initiation 85
Figure 8.13 Monitoring Window 86
Figure A.1 Example of Spanning Tree Routing Algorithm 102
Figure A.2 Example of Reverse Path Broadcasting (RPB) Routing Algorithm 103
Figure A.3 Example of Reverse Path Multicasting (RPM) Routing Algorithm 104

List of Tables

Table 2.1 Summary of Audio Coding Standards..... 26
Table 2.2 Summary of Video Coding Standards..... 28
Table 5.1 QoS Table for Logical Multimedia Streams.....46
Table 8.1 Two Different QoS Metrics..... 88
Table 8.2 video variants for testing..... 89
Table 8.3 Testing Results..... 91
Table B.1 QoS Mode..... 110

Chapter 1: Introduction

1.1 Background

With the fast development of multimedia and network techniques, transmitting video and audio across the networks is attracting many applications. A large number of commercial and research organizations are involved in this area in various ways. The typical applications include video conferencing, tele-teaching, network telephony, and so on. These commercial products and research projects are diverse in technological infrastructure, such as networks employed, protocols used, and compression standards supported. In this very competitive field, good quality becomes an important factor. Quality of Service (QoS) issues are getting more concerns. Video and audio applications normally have a large amount of data, which may cause a high consumption of the bandwidth. Many products provide QoS guarantees based on ATM or ISDN connects because of the nature of this type of connections. The traditional best-effort network, e.g. the Internet, cannot guarantee satisfying video and audio quality since it was not designed to support this type of applications.

The IP multicast concept emerged about eleven years ago. It provides a bandwidth-efficient way to transmit video and audio data to multiple parties over the current Internet. More and more Internet video and audio applications have been built based on IP multicasting and research efforts have been made to integrate QoS management into the multicast applications. However, when a large number of users are involved, QoS management becomes more complex because of the heterogeneous nature of the Internet. The complexity is caused by the conflicts among the receivers. A single quality of service

level may not be appropriate for all the participants because of the different link capacities. Some users may participate with a limited workstation that is not able to provide the quality that most of the other participants adopted. Some users may prefer to pay a higher price to obtain a maximum quality, while others may prefer a lower quality to keep the cost down. The source providing a single high-quality stream may cause the low-bandwidth users to suffer high packet loss, and a single low-quality stream may bring complains from the high-bandwidth users. There are many open research issues in this area and one aims at building a simple and easily-implemented QoS architecture. QoS adaptation schemes are often discussed for the case that the quality level is changing, aiming at adapting the application to the available bandwidth. However, the number of applications in this area is small. We also noticed that in this category, few applications take the user's satisfaction into account. Most of them are from the network point of view. Implementing QoS adaptation from the user's point of view is an important aspect of our project.

1.2 Main Contributions

A QoS adaptation approach in the multicast environment was proposed by Stefan Fischer and other scholars [51]. In their paper, they described an architecture where the source provides QoS alternatives concerning frame rate, video color, and video resolution, and the adaptation is performed according to the available bandwidth combined with the users' preferences, which can be defined through a user interface. The session is controlled by some so-called QoS Agents. Our research goal is to prove the feasibility of this approach. We adopted a system design, in which video, voice and session control messages are transmitted over separate channels. Therefore, the most important design

concern was to choose an appropriate session control protocol. Through the study of several session protocol alternatives, we found that the DMIF (Delivery Multimedia Integration Framework), which has been designed as a tool to support the MPEG-4 system features, was the best candidate for our system. Although the DMIF was intended to be used with MPEG-4 systems, it includes the definition of a generic session level protocol to fulfill the requirements of multimedia applications. We will explain in detail how DMIF has been adapted into our system.

A simple case study, tele-teaching application, was developed. The system build-up process went through the usual phases of specification, design, implementation and test. The implementation and test results met our goals.

We believe that this research is valuable since (a) it has demonstrated that Fischer's QoS adaptation approach is feasible, (b) it provided a general architecture for supporting QoS adaptation in a multicast environment, which can be employed by other similar types of video or audio applications, and (c) it provided a DMIF Application Interface (DAI) based on a DMIF protocol implementation provided by Mr. Darlagiannis.

A similar approach was used in [60] for transmitting multimedia documents over an MBone session. The difference between this system and our system is that this system supports the remote retrieval of pre-recorded streams stored in a multimedia database, while our system supports the real-time video transmission.

1.3 Thesis Outline

The thesis is organized in the following way. Chapter 2 gives an introduction of some background knowledge. The first section introduces some protocols related to multimedia delivery over the Internet. We will mention first the Internet protocol set TCP/UDP/IP,

then IP multicasting concepts and protocols, and finally some higher level protocols that are often used to provide feedback on multimedia delivery characteristics. The reason of introducing multicast concepts is that our work is based on the Multicast Backbone (MBone), which is a test bed for Internet IP multicasting research. The second section is a short description of various video and audio coding schemes. We will discuss the Quality of Service (QoS) principles and adaptation mechanisms in Chapter 3. Also some related work are mentioned. Chapter 4 begins with a survey of current video conferencing applications, followed by a discussion of what kind of system we wanted to build, its characteristics, and in which way it is different from other systems. General system architecture and a typical interactive scenario are described in Chapter 5. To implement this system, we need to choose appropriate protocols for video delivery and session control. We will list several protocol alternatives in Chapter 6, and analyze the advantages and disadvantages of using those protocols, including their overhead. DMIF was our final choice for session control. Chapter 7 explains how we adapted DMIF into our system. The system design and test results are given in Chapter 8. We describe some key design issues and implementation choices. Some other design details are given in the Appendixes. In the final chapter, "Conclusion and Future Work", we summarize the contribution of the work, illustrate other type of applications to which this scheme can apply, and finish this thesis with a discussion of future work.

1.4 Publication Arising from the Research

G. v. Bochmann, Z. Yang, "Quality of Service Management for Tele-teaching Applications Using the MPEG-4/DMIF", International Workshop on Interactive

Distributed Multimedia Systems and Telecommunication Services, Toulouse, France, Oct 12-15, 1999, Proceedings, pages 133-145.

Chapter 2: Multimedia Applications over Internet

2.1 Internet Protocols for Multimedia Applications

2.1.1 Internet Protocols – UDP/TCP/IP

Data communication networks were developed to allow people share computer resources from different places. As computers have spread among organizations and families, the early networks, where a user of one network was not able to access other networks, became inadequate for information sharing among businesses or individuals. Therefore, the concept of internetworking emerged, and also technologies and standards were developed to allow users to communicate with each other from different networks. In the early 1970s, several groups around the world started to address this issue. The International Telecommunication Union – Telecommunications Standardization Sector (ITU-T), the International Standards Organization (ISO), and some designers from the Advanced Research Projects Agency (ARPANET) were the pioneers in this field. With the initial efforts of several ARPANET designers, some early protocols and network control programs were developed, and later, the Transmission Control Protocol / Internet Protocol (TCP/IP). Today, TCP/IP has become the standard suite of data communication protocols on the Internet.

Network software and hardware require a wide range of functions to support the communication activities. To solve this problem, the protocol functions are usually structured in a layered architecture. Different from the ISO 7-layer architecture, the Internet is based on 5 layers, from top-down: application layer, transport layer, network layer, data link and physical layer.

Here is an example of the networking operation. Suppose Host A wants to send information to Host B. The data are going downwards as arrow 'a' shows in Figure 2.1. To perform corresponding functions, each layer adds headers to / encapsulates the data from the upper layer, until the data reach the physical layer, which is responsible for launching the data into the network. The data go to the router and are passed to the IP layer in the router, where, based on the address provided by Host A, the routing functions are working to find the route to Host B. Once the data arrive at the Host B, the headers are stripped off / decapsulated at the appropriate layer, as arrow 'b' shows in Figure 2.1, and finally the Host B gets the proper information.

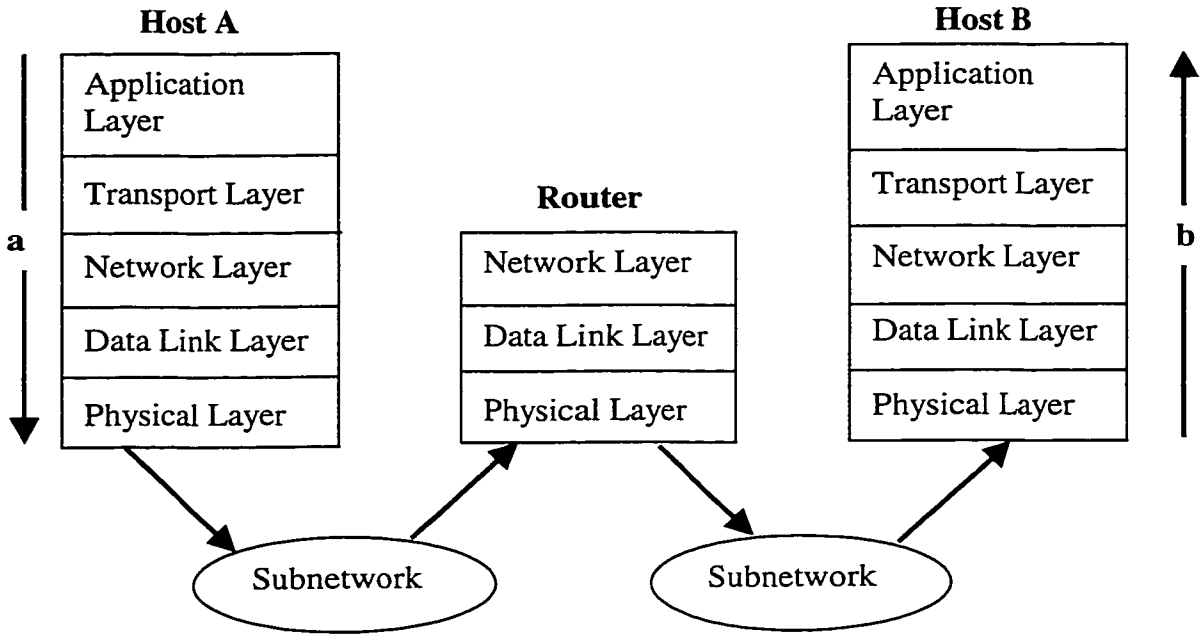

Figure 2.1 Example of Internet Layer Operation [2].

- **Internet Protocol (IP)**

IP [1] is the internetworking protocol. It allows the exchange of traffic between two host computers, hiding the underlying network from the user. IP provides a connectionless service, which means *"No logical connection between the user and the network is*

established prior to data transmission. The data units are transmitted as independent units." [2]. Because of this feature, IP is robust, however unreliable. An IP packet can be lost, duplicated or arrive out of order. IP was not designed to deal with these problems. It does not provide error recovery or flow control. These functions can be provided by an upper layer (transport layer) connection-oriented protocol, e.g. TCP.

An Internet IPv4 address has 32 bits divided into 2 segments: network address, which identifies the sub-network where the destination host is located, and the host machine address. Three classes of IPv4 addresses, Class A, B, and C, were designed to fit in with the needs of allocating IP addresses to hosts whose networks have different sizes. Another class of IP address, Class D, was defined later for multicasting.

- **Transmission Control Protocol (TCP)**

TCP [3] was designed to run above IP, providing reliable data transmission with flow control. TCP is a connection-oriented protocol, which means "*A user and network set up a logical connection before transfer of data occurs. Usually, some type of relationship is maintained between the successive data units being transferred through the user / network connection.*" [2]. TCP uses sequence numbers and checksum facilities to ensure that a segment of data is not damaged during the transmission. TCP also allows retransmission by sending acknowledgement message back to the sender. When the segment is received correctly, a positive acknowledgement (ACK) is returned to the sender, otherwise, a negative acknowledgement (NACK) is returned; in this case, the sender would retransmit the data. In addition, TCP also uses the sequence numbers to deliver the segments in order even if the segments arrive over the network out of order. TCP also checks for the duplication. Another useful feature provided by TCP is flow

control. It is based on the "sliding-window" technique. A window size value is assigned to the transmitter. The transmitter is only allowed to transmit a specified number of bytes within this window. On receiving of the correct ACKs, the window slides forward. The transmitter must stop the transmission when the window is closed. Another point to mention is the port number. Each application process needs to identity itself by a port number, which is used to identity which application program should receive the incoming traffic. Since the port number allows several programs to communicate concurrently, it can be used to support multiplexing capabilities.

- **User Datagram Protocol (UDP)**

UDP [4] is a connectionless transport protocol. UDP is running faster than TCP because it has no reliability, flow-control or error-recovery features. It provides port numbers for distinguishing different information flows. This protocol assumes that the underlying protocol is IP.

2.1.2 IP Multicasting and MBone

2.1.2.1 IP Multicasting Overview

In recent years, demands for interactive multimedia applications were rapidly increasing. In these applications, the video and audio data are captured, compressed and transmitted to usually a group of receiver stations. Although lots of developments in compression techniques have emerged, the multimedia applications still need a lot of bandwidth.

In the traditional setting, Internet applications employ point-to-point (unicast) communications. In the case that there is a group of receivers, the packet of media

information is replicated to the same number of receiver hosts and is forwarded until it finally reaches the receiver stations. An example is given in Figure 2.2.

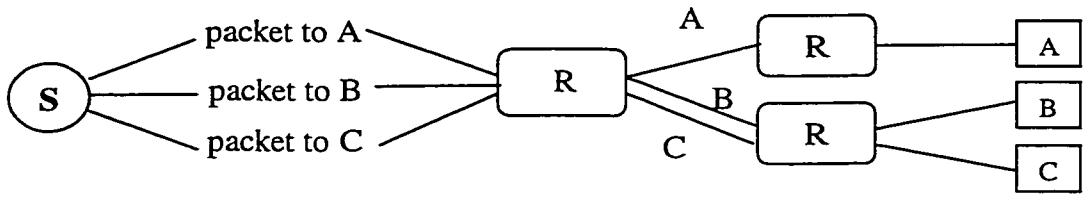

Figure 2.2 Example of Unicast Transmission

If the sender host "S" is sending a packet to host "A", "B" and "C", then it has to replicate the packet in three copies, and transmit them independently to the three receiver hosts. When one-to-many or many-to-many communication involving a large number of hosts is considered, this transmission style is quite bandwidth consuming and has poor scalability over a wide area network.

The concept of IP Multicasting arose years ago. It is an extension to the standard IP protocol. Steve Deering described IP Multicasting in his paper "Host Extension for IP Multicasting" [5]: *"The transmission of an IP datagram to a host group, involves a set of zero or more hosts identified by a single IP destination address. A multicast datagram is delivered to all members of its destination host group with the same best-efforts reliability as regular unicast IP datagrams. The membership of a host group is dynamic, that is, hosts may join and leave groups at any time. There is no restriction on the location or number of members in a host group. A host may be a member of more than one group at a time."*

IP Multicasting is an important advance in IP networking. It provides a scalable delivery mechanism that efficiently supports one-to-many or many-to-many transmission by enabling the source host to send a single copy of the information to multiple receivers. IP

Multicasting involves groups of recipients on a multicast channel. A receiver needs to explicitly join a group to receive the traffic that is only forwarded to the group members who want to receive the information.

Consider the same example in Figure 2.3 again. The transmitting host "S" does not have to duplicate the packet for each intended receiver any more. Instead, only one copy is sent out. The sender host does not have to maintain the information of the location and number of the receivers. It is the receiver's responsibility to register itself as a member of the multicast group through an appropriate protocol. The packets are forwarded by so-called Multicast Routers (MRouters), which are a special kind of routers that support IP Multicasting. A receiver node joins a group by registering with its directly connected MRouter. The data messages are only duplicated as needed by the intermediate multicast routers.

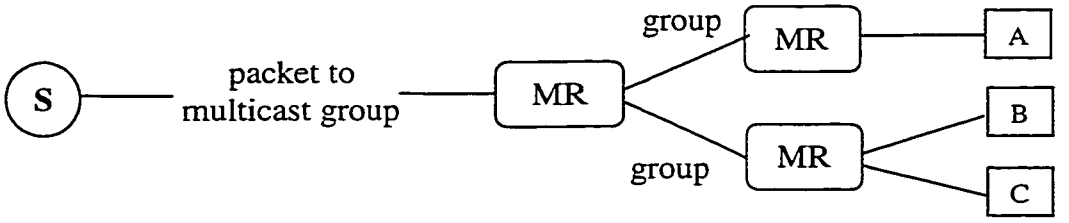

Figure 2.3 Example of Multicast Transmission

IP Multicasting significantly reduces network load thus provides a scalable solution for the Internet multi-party communication. It is far more efficient than point-to-point (unicast) communication in the sense that it allows the source host sending a single copy of a message to a group of recipients, instead of one copy for each recipient, where the available bandwidth of the sender limits the number of the receivers. It is also more efficient than another transmission style – broadcasting, in which case one copy of a

message is sent to all nodes on the network. Since many receiver nodes may not want to receive the message, broadcasting may consume unnecessary bandwidth.

To send the information to a group of receivers, a special kind of IP address is needed to describe a group of host. Also, IP multicast uses the Time-To-Live (TTL) field in the IP header to limit the packets' propagation range. We will introduce the group IP address mechanism and the TTL usage in the following section. Furthermore, MBone (Multicast backBone), a multicasting test bed that virtually layers on top of the normal Internet IP protocol will be introduced in Section 2.1.2.3.

2.1.2.2 Group Addressing and TTL

- **Group Addressing**

In IP multicast, a group of receivers is identified by a multicast group ID. When a sender node wants to send messages to multiple receivers, it sends to a group ID, which specifies the destination group. If a host wants to receive the information sent to a group, it needs to explicitly join that group. The group ID is based on a set of IP addresses, called "class D IP address" in IPv4, and "multicast IP address" in IPv6.

The 32-bit addresses of IPv4 are expressed in standard "dotted-decimal" notation. A class D IP address has the first 4 bits set to "1110", followed by a 28 bit multicast group ID, ranging from 224.0.0.0 to 239.255.255.255.

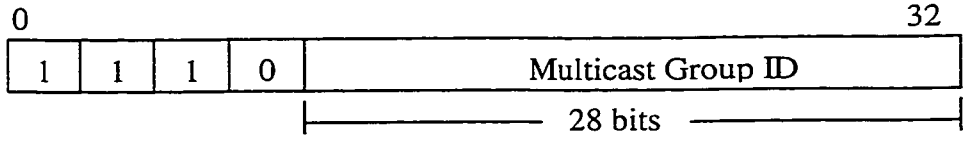

Figure 2.4 IPv4 Class D Address (Multicast Address)

However, not all of the addresses are available for the users. Some addresses are reserved by the Internet Assigned Numbers Authority (IANA). For instance, 224.0.0.0 cannot be

assigned to any group. 224.0.0.1 – 224.0.0.255 are reserved for routing protocols or other low-level protocols. The reserved numbers are defined in [6].

IPv6 [7] increases the IP address size from 32 bits to 128 bits, in order to support a large addressing hierarchy. There are 3 types of addresses, unicast, anycast and multicast [8]. IPv6 multicast addresses have their high-order 8 bits set to "11111111", followed by 4-bit flags, 4-bit scope and 112-bit group ID.

8 bits	4 bits	4 bits	112 bits
1 1 1 1 1 1 1 1	flags	scope	group ID

Figure 2.5 IPv6 Multicast Address

The "flag" has the format "000T", where "T" is a flag to differentiate the reserved addresses and the user available multicast address. "T=0" refers to reserved addresses, and "T=1" refers to user available multicast addresses.

The "scope" field limits the scope of multicast message forwarding. IPv4 does not have this scheme. Instead, it uses the TTL field to fullfil the same function.

The already assigned multicast addresses of IPv6 are defined in [9].

- **TTL**

Each IPv4 multicast packet uses the Time-To-Live (TTL) field in the IP header as a scope limit parameter. It provides a mechanism to control the multicast packet propagation within an expected range. The TTL value is specified by the sender application. Each time a router forwards a packet, it decrements the TTL value by 1, until the TTL expires (=0). A packet with an expired TTL will be dropped without any error notification to the sender. The usage of TTL provides a convenient way to prevent information from being unnecessarily transmitted to regions that are beyond the subnet

that contains all group members. There is a set of recommended TTL values [10] defined according to the MBone infrastructure. Multicast packets sender can consult with these values to specify the transmission scope.

2.1.2.3 MBone

MBone [11, 12] stands for "Multicast backBone". It was established in early 1992 with an attempt to transmit video and audio from the Internet Engineering Task Force (IETF) meetings. It was setup as a test bed in the Internet to support the research of multicast applications. MBone uses special routers, as we mentioned before, MRouters, to distinguish unicast packets and multicast packets and to forward multicast packets by implementing multicast routing algorithms. Moreover, it uses a so-called "IP-tunneling" scheme to operate over the standard Internet IP unicast architecture.

- **IP Tunneling**

MBone is a virtual network layered on top of the unicast IP service. It consists of MRouters, which run the MRouted multicast routing daemon to support the routing of multicast traffic. Since not all routers support IP multicasting and not all multicast routers are connected with each other, a mechanism has to be employed to support the delivery of multicast packets through unicast routers. This mechanism is call "IP Tunneling". A picture from [10] gives an overall architecture of MBone IP tunnels (Figure 2.6).

Each tunnel is configured with a metric and a threshold. The metric is the cost associated with sending a packet on the tunnel. The threshold is the minimum TTL that is required to send a packet over the tunnel. It is used to limit the transmit scope for multicast packets. Whenever a MRouter forwards a multicast packet over a particular tunnel, it checks whether the TTL value of this packet is greater than the threshold of the tunnel.

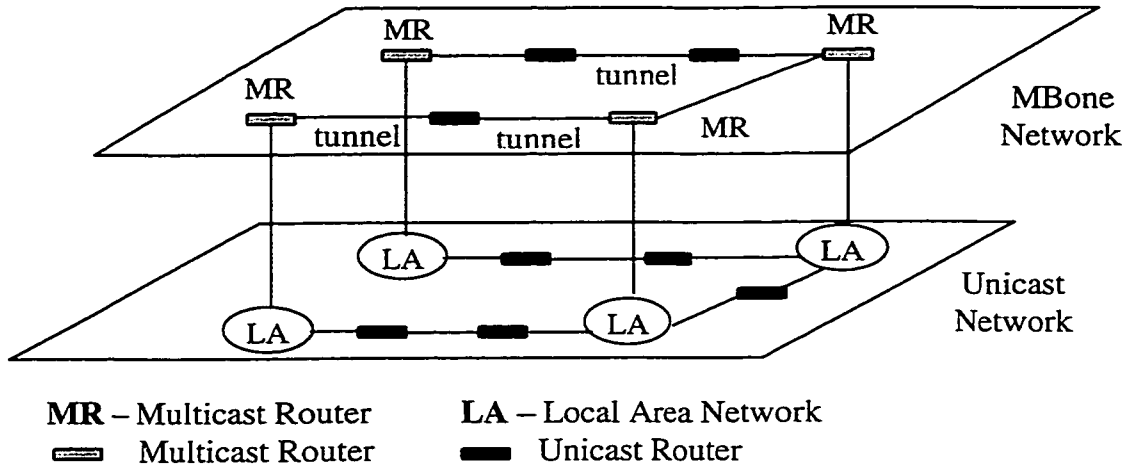

Figure 2.6. MBone Architecture

The MRouter will only forward the packets with greater TTL. Otherwise, the packets will be discarded.

- **MBone Applications**

A number of MBone applications are available on the Internet nowadays. They are categorized into several groups according to the functionality, including video / audio utilities, session announcement utilities, debugging tools, etc. This section will introduce three typical applications: the session tool – SDR, video tool – VIC, and audio tool – VAT. Archives of other software can be found at [13]. We use VIC in our project as a video transmission tool. We prepared a relatively complete introduction of MBone applications in [59].

SDR

SDR is a Session Directory that was designed for collecting MBone multimedia

conference information, and also for announcing and scheduling multimedia conferences on the MBone.

SDR makes use of two session protocols: SDP (Session Description Protocol) [14] and SAP (Session Announce Protocol - Internet draft, working in progress). Through these two protocols, all session related messages are periodically broadcast over the MBone. SDR captures those messages and shows them on the screen, providing an easy way to receive multicast sessions. A user can also create his own advertisement sessions and broadcast on MBone.

A session may contain several single media sessions, such as a video session, an audio session, etc. A user may receive them independently by selecting a proper tool according to the media type used. For instance, a user may use VIC for video, VAT for audio, WBD for whiteboard, etc. These applications can be easily plugged in and automatically launched by SDR.

The main user interface, which shows a list of sessions, is given in Figure 2.7.

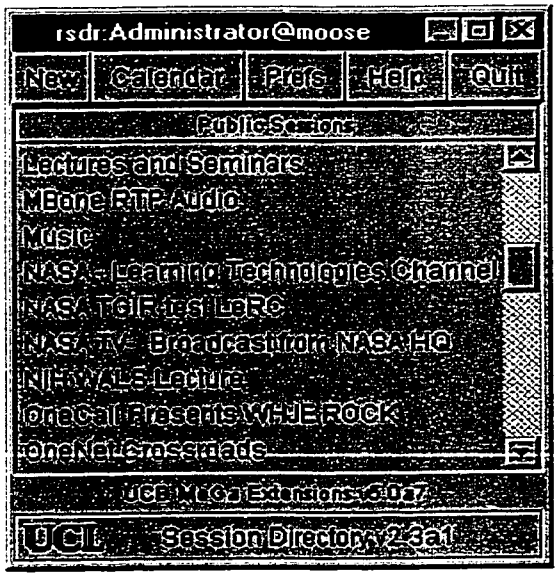

Figure 2.7 SDR Main User Interface

VAT

VAT, which stands for Visual Audio Tool, is the original MBone audio tool developed by Van Jacobson of the Lawrence Berkeley National Laboratory (LBL). It allows the user to conduct point-to-point or multi-point audio conferences over the Internet.

VAT is used to send and receive audio in several coding schemes. It is based on a real time transport protocol, RTP, which is entirely implemented within VAT.

Through a simple user interface, multiple users can talk with each other. By clicking "listen" or / and "talk" button, a user can select receiving and / or sending audio messages. A pair of vertical sliders allows the user to adjust the microphone and speaker's volume. VAT has the following user interface:

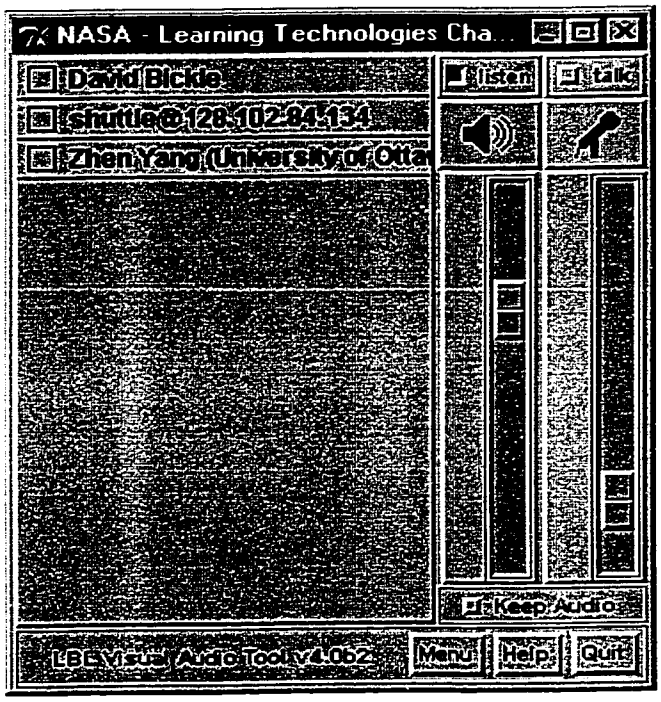

Figure 2.8 VAT User Interface

VIC

VIC is a tool for video conferencing over the Internet. It allows users to receive and

transmit video streams from their desktops. This application was also developed by Van Jacobson and Steven McCanne of the Lawrence Berkeley National Laboratory (LBL) [15]. VIC can be run point to point but it was primarily intended as a multiparty conferencing application.

The VIC supports a number of encoding schemes for M-JPEG, H.261, H263, nv and cellb. Like VAT, VIC is based on the RTP. The VIC interface allows easy access to basic functions like receiving and transmitting video. When receiving video, VIC shows a thumbnail sketch of each transmitting source with information about the transmitter to the right of the thumbnail. The user can select to see one or more of these videos. When transmitting video, the user can control the maximum bandwidth and the maximum number of frames per second by selecting the appropriate options from the "Menu" button of the VIC window.

VIC offers the following receiving and transmitting user interfaces:

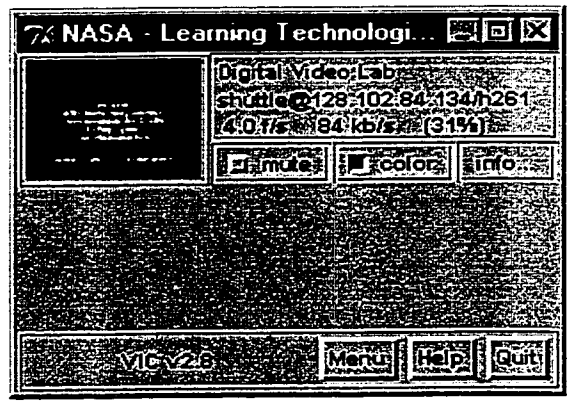

Figure 2.9 VIC User Interface

2.1.3 Multicast Protocols

Multicast is different from unicast in several aspects. First of all, in real world multicast applications, a receiver can join or leave a multicast group at any time. As we mentioned

before, an appropriate protocol is needed to implement the group management function. It allows the receiver nodes to join or leave a group and lets the intermediate router know when to forward the messages to its directly connected local network.

Secondly, the delivery of multicast messages is different from that of unicast messages. Multicast routing protocols are needed for MRouters to keep track of each other and determine proper ways to forward information among neighboring routers.

Finally, a number of higher level protocols have been developed to enhance the Internet architecture to support typical multicast applications – such as multimedia conferencing.

2.1.3.1 Group Management Protocol

Multicast is a receiver-based concept. Hosts need to join a multicast group to listen to the messages that are sent to the group. To join or leave a group, a host sends messages to inform the multicast router on its sub-network. If there is no such a router on the sub-network, a host is not able to receive the multicast packets. In this way, the multicast routers know the group membership on their networks, so that they can determine whether or not to forward multicast packets. A multicast router forwards multicast packets of a group to its network only when there are hosts registered as members of that group. The Internet Group Management Protocol (IGMP) [5] is developed for this purpose.

IGMP is used by receive nodes to join or leave a multicast group, and by multicast routers to learn the existence of group members on their directly connected sub-networks. It accomplishes this purpose by sending IGMP queries from multicast routers and having hosts report their group memberships back to the multicast routers. Up to date, IGMP has three versions. The later versions improve on their previous versions in different aspects.

IGMP (version 1)

A MRouter periodically transmits a *Host Membership Query* to the all-hosts group (224.0.0.1) to determine whether there are multicast members on its directly connected network. Since the *Query* message should not go beyond the local network, it has IP TTL equal to 1. When a host receives a *Query* message, it needs to respond with a *Host Membership Report* for each host group to which it belongs. In our example, host 1 reports its membership of group 1, host 2 of group 1 and 2, and host 3 of group 1, so that the multicast router knows that messages sent to group 1 and 2 need to be forwarded to this sub-network. Rather than sending the *Report* immediately, a host waits for a randomly chosen delay. If in this period, another *Report* of this group is heard, the host resets the timer. This procedure is to prevent the flooding of *Reports* on a network.

Since a host can join or leave a group at any time, a MRouter needs to periodically transmit the *Query* to update its group membership information. If a MRouter does not receive a *Report* for a particular group after a certain number of *Queries*, it removes the group from its list. When a host first joins a group, it does not wait for a *Query* to respond. Instead, it immediately transmits a *Report* to the multicast router to claim the existence of a group member.

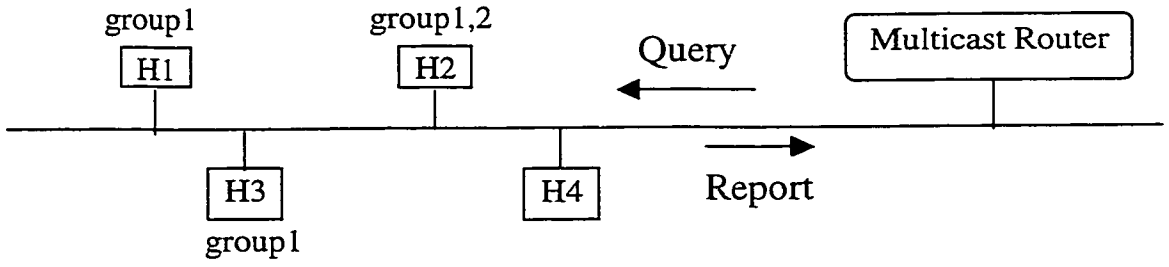

Figure 2.10 Example of IGMP

- **IGMP (version 2)**

When there are more than one MRouter attached to a local network, one MRouter is elected as a multicast querier. The election of the querier in IGMPv1 is determined by the multicast routing protocol. Different protocols may use different methods to decide the querier. To avoid the uncertainty problem, IGMPv2 [16] stipulates that the querier has to be the router with the lowest IP address on the local network.

To reduce the traffic, IGMPv2 defines a *Group Specific Query* message, which allows a multicast router to query a particular group. When a host wants to leave a group, for example, host 2 wants to leave group 2, another new introduced type of message *Leave Group* message is sent to the multicast router. The router will send *a Group Specific Query* regarding group 2. If no reports have been received in a period of time, group 2 will be eliminated from the router's list.

- **IGMP (version 3)**

A multicast group may have several sources. Further refinements have been made in IGMPv3 (Internet draft, working in progress) to help to conserve the bandwidth by adding *Group Source Report* and *Group Source Leave Report*, which make it possible to identify a specific source of a multicast group. IGMPv1 and IGMPv2 do not have this mechanism. Even if a host wants to receive messages from only one source of a multicast group, the messages from all sources have to be forwarded onto the sub-network.

2.1.3.2. Multicast Routing Protocols

IGMP only provides the final step in multicast packet delivery. A multicast routing protocol is needed for the multicast routers to construct multicast delivery trees and forward multicast packets between neighboring routers or across the inter-network.

Several routing algorithms have been proposed to build the multicast trees [17, 18]. They can be used to implement multicast routing protocols. In this section, we introduce some currently used multicast routing protocols. The description of multicast routing algorithms is given in Appendix A.

- **Multicast Routing Protocols**

Distance Vector Multicast Routing Protocol (DVMRP)

DVMRP was originally defined in [19]. It was based on the Truncated Reverse Path Broadcasting (TRPB) algorithm, and enhanced to the Reverse Path Multicasting (RPM) algorithm later on. The latest standard of this protocol is developed by the IETF Inter-Domain Multicast Routing (IDMR) working group. In DVMRP, multicast packets are forwarded on a multicast delivery tree, which is constructed by pruning a broadcast delivery tree. The pruning is done by Mrouters sending a so-called *prune* message back towards the source, until the source is reached. In addition, DVMRP uses a *graft* message to allow a *prune* message to be canceled.

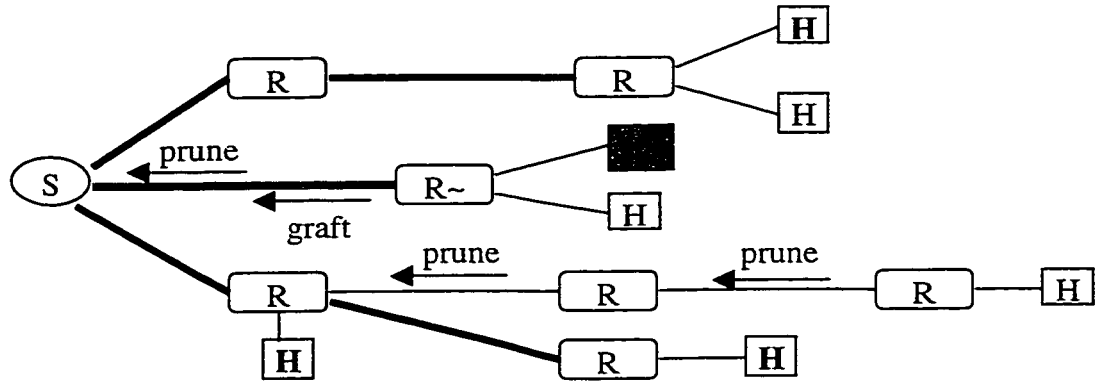

Figure 2.11 Example of DVMRP

Figure 2.11 shows how the multicast delivery tree is constructed. Assume that the hosts, which are expressed in bold, want to join a group. The routers, which do not have group members send the *prune* one hop back towards the source. Host H~ joins the group

shortly after the router R~ sent out a *prune* message. Then the R~ should immediately send a *graft* to the upstream router, in this case the source, to cancel the previous prune. DVMRP is used by the majority of MBone routers. Others are using MOSPF or PIM protocols.

An implementation of the DVMRP is called *MRouted* [20], which is a MBone routing utility used by MBone routers to support IP multicasting. *MRouted* supports the IP tunneling mechanism. IP multicast packets are encapsulated for transmission through tunnels, so that for the intermediate non-multicast-capable routers and sub-networks, they look like normal unicast packets. The encapsulation is added on at the entry of a tunnel, and stripped off at the exits from a tunnel. By default, the packets are encapsulated using the IP-in-IP protocol (IP protocol number 4).

Multicast Extension to OSPF (MOSPF)

MOSPF is an IP multicast extension of OSPF, which is a link-state protocol defined in [21]. MOSPF added a new "*Group-Membership-LSA* (Link State Advertisement)". A multicast delivery tree is constructed using Dijkstra's algorithm based on the OSPF link state information, then the non-group links are pruned by conducting the group membership information. MOSPF supports hierarchical routing. Hosts are divided into "Autonomous Systems", and further partitioned into areas. There are three types of routing: intra-area, inter-area, and inter-AS routing. Since this is not the focal point of this thesis, we do not discuss the details here. MOSPF is specified in [22].

Protocol-Independent Multicast (PIM)

PIM is being developed by the IETF Inter-Domain Multicast Routing working group. They consider that the DVMRP and MOSPF cannot work efficiently where group

members are sparsely distributed and bandwidth is limited, because DVMRP needs to refresh the multicast tree by periodically flooding the network with a packet, and MOSPF sends group membership information over all the links. Therefore, there are two protocols in PIM: PIM-Dense Mode (PIM-DM) and PIM-Sparse Mode (PIM-SM).

PIM-DM applies in a situation where group members are relatively dense and bandwidth is plentiful. In this case, PIM-DM is very similar to DVMRP. One difference is that PIM-DM is independent from mechanisms employed by the unicast routing protocols; it is a "protocol independent protocol". The other difference is that PIM-DM simply forwards all multicast traffic on all downstream interfaces until explicit prune messages are received, while DVMRP forwards traffic along the multicast tree.

PIM-SM [23] is efficient in the situation where group members are distributed across many regions and bandwidth is not widely available. The main difference from PIM-DM is that to join a sparse mode multicast tree, routers are required to transmit explicit join messages.

2.1.4 Higher-Level Protocols

This section gives an overview of several typical higher level protocols used with IP multicasting [24]. These protocols have been developed to enhance the Internet support for real-time multimedia applications. They are Real-time Transport Protocol (RTP), Real-time Transport Control Protocol (RTCP), Real-time Stream Protocol (RTSP), and ReSource reserVation Protocol (RSVP). Our project is developed using RTP and RTCP.

- **RTP**

RTP provides end-to-end delivery services to support real-time data transmission. It typically runs on top of the transport layer protocol, UDP, and works together with it to

carry the real-time data across the network. It was proposed by the IETF Audio and Video Transport working group. The original version was defined in [25], and followed by some Internet drafts with some further improvements. We will introduce RTP in more detail in Section 6.1.2.

- **RTCP**

RTCP works together with RTP to provide feedback information about RTP streams. Each participant in a session periodically transmits RTCP control packets to all the other participants. There are 5 different types of RTCP packets to carry various control messages, including Sender Report (SR), Receiver Report (RR), Source DEScription (SDES), BYE, and APPlication specific information (APP). We will discuss in Section 6.1.2 the functions that RTCP provides and how they can benefit our project implementation.

- **RTSP**

RTSP is an application-level protocol, which was designed to work on top of RTP, to control the delivery of data with real-time characteristics. RTSP provides a framework for streaming multimedia data in one-to-many applications. Sources of data may include both live data feeds and stored clips. This protocol is intended to control multiple data delivery sessions. It provides a way to choose delivery channels such as UDP, multicast UDP and TCP, and also provides a means to choose the delivery mechanisms based upon RTP. Detail information can be found in [26, 27].

- **RSVP**

RSVP was developed to provide end-to-end quality of service guarantees for data streams between senders and receivers from the network. It enhances the current Internet

architecture with the support for quality of service. RSVP operates over IPv4 or IPv6, serving as a control protocol through the transmission of QoS parameters. It was also designed to route RSVP messages along all the nodes in the reserved path to establish and maintain state information to provide the requested service. RSVP was defined by the IETF Networking working group in [28].

2.2 Voice / Video Encoding

2.2.1 Traditional Voice / Video Encoding

Multimedia applications may contain different types of media, such as video, audio, image, text. Normally, this kind of applications has a large amount of data, which causes high cost of bandwidth in the network. The multimedia information has to be compressed and converted into binary bits of data at the source node (encoded), then delivered over the network, and finally resumed (decoded) by the destination node. Some technologies and standards have been developed to support the encoding and decoding of multimedia information. ITU and IETF are at the forefront of standards development for communications and conferencing products. In this section, we introduce some video and audio coding standards.

- **Voice Coding Standards**

The changes in air pressure create audio waveform that reaches the human eardrum. We can hear sound if the frequency of air disturbance is between 20 to 20,000 Hz. To send the audio across the network, the audio waveform has to be converted into binary bits, then encapsulated by a transport protocol. The conversion contains several phases: analog to digital, sampling, quantization, and compression. PCM (Pulse Code Modulation) is the

most popular used encoding method. ITU has standardized some audio codecs, which are different in bandwidth and transmission rate. The summary is given in the following table [29].

Name	Audio Frequency	Transmission Bit Rate	Codec Algorithm
G.711	3.4 – 300 KHz	56K, 64 K bps	PCM
G.728	3.4 – 300 KHz	16K bps	LD-CELP
G.722	7 – 50 KHz	48K, 56K, 64K bps	ADPCM
G.723.1	3.4 – 300 KHz	5.3K, 6.3K bps	ACELP, MP-MLQ

Table 2.1 Summary of Audio Coding Standards

- **Video Coding Standards**

An image in the human eye remains for several milliseconds. If a sequence of individual images is flashed quickly, the human eyes consider it as a motion picture. The video system takes advantage of this principle. Therefore, video contains a huge amount of data. Compression is included in the encoding processes, which requires high computational power. The most important video coding standards are H.261, H.263, MJPEG, MPEG1, MPEG2, and MPEG4. A brief description of each standard is given below.

H.261

H.261 is designed for low data rates and relatively low motion. It includes a mechanism that optimizes bandwidth usage by trading picture quality against motion. A quickly moving picture has a relatively low quality compared with a static picture. H.261 is optimized for low data rates, but generally, it cannot provide a good quality.

H.263

H.263 is an advancement of the H.261 standard. It was designed for producing better quality with low bit-rate communication. The coding algorithm of H.263 is similar to that of H.261, with some refinements to improve the performance and error recovery.

M-JPEG

JPEG was designed for compressing either full-color or gray-scale images of real world scenes. JPEG can achieve 10:1 to 20:1 compression without visible loss, and 30:1 to 50:1 with small or moderate defects. 100:1 can be achieved if only low quality is required. M-JPEG stands for Motion-JPEG. Various vendors applied JPEG to individual frames of a sequence of video, and they call the result M-JPEG. Since there is no standard for sequencing the individual video frames, there is an incompatibility problem among different vendors' products. It is popularly used in digital studios because each video frame can be accessed directly.

MPEG-1

MPEG-1 coding targets a bandwidth of 1-1.5 Mbps. It can provide excellent audio and video quality and can be used in many environments. However, to achieve this quality, MPEG-1 requires fairly powerful playback hardware, such as sufficient CPU power for basic playback and high-quality video card for playback acceleration. Moreover, the interdependency of P (predicted) and B (bi-directionally predicted) frames makes the video quality highly affected by packet loss.

MPEG-2

MPEG-2 was designed to broadcast high quality digitally encoded video and audio. It provides excellent quality for TV-broadcast applications. MPEG-2 enhances MPEG-1 by

including support for higher resolution video and increased audio capabilities. The target bit-rate is 4-15 Mbps. Compare to MPEG-1, MPEG-2 requires more complicated hardware to encode and decode.

MPEG-4

The initial goal of MPEG-4 was to provide a compression scheme suitable for video conferencing, with very low data rates – less than 64 Kbps. MPEG-4 is based on AVOs (audio / visual objects), which can be multiplexed for transmission over heterogeneous networks. The detailed information will be given in the next section.

A summary of video resolution and transmission rates for these video coding schemes [29] is given in the following table.

Name	Video Resolution	Transmission Bit Rate
H.261	CIF (Common Intermediate Format) – 352*288, QCIF (Quarter CIF) – 176*144	64K – 2M bps
H.263	CIF (Common Intermediate Format) – 352*288, QCIF (Quarter CIF) – 176*144, SQCIF (Sub QCIF) – 128 * 96, 4CIF (4 * CIF) – 704 * 576, 16CIF (16 * CIF) – 1408 * 1152	15K – 20K bps
MPEG-1	352*240	1.2 M bps
MPEG-2	352*240 (low), 720*480 (main), 1920*1080 (high), 1440*1152 (high 1440)	4M – 9M bps (can be up to 80Mbps)

Table 2.2 Summary of Video Coding Standards

2.2.2 MPEG-4

MPEG-4 [30] is a new ISO standard on multimedia stream coding that goes beyond the previous MPEG-1 and MPEG-2 standards by providing adaptation to low-data-rate transmission and support for multiple video and audio streams, which can be useful for teleconferencing applications and applications involving virtual environments. In contrast

to traditional coding standards, MPEG-4 allows the definition of video streams which represents objects with arbitrary contours, not only the rectangular screens of TV or film presentations.

MPEG-4 provides a standardized way to:

- Describe units of visual or audiovisual content in the form of "objects", called "Audio/Visual Objects" (AVOs), which form the audiovisual scenes. The AVOs are organized in a hierarchical manner. Basic units at the leaf level are called primitive AVOs, such as a picture of a car, the voice associated with the car, etc.
- Compose AVOs to create compound audiovisual objects, which form the final scenes. A scene is composed of individual objects, and compound AVOs that groups AVOs together, such that AVOs are organized in a hierarchical way.
- Multiplex and synchronize the data associated with AVOs, so that they can provide an appropriate QoS when transmitted over the network. AVO data is conveyed in one or more Elementary Streams (ES), which carries the requested QoS, such as bit error rate, maximum bit rate, and also necessary parameters like stream type information, encoding timing information, etc.
- Interact with AVOs. MPEG-4 provides standards to compose a scene, such that an AVO can be placed anywhere, a user can change his viewing or listening point to anywhere in the scene, etc. A user can drag an object to another position in a scene, trigger a series of events including starting or stopping a video stream, change the view point, etc. The flexibility of the operation with AVOs allows the composition of an arbitrarily shaped screen.

A general MPEG-4 architecture contains 3 layers as shown in the following figure:

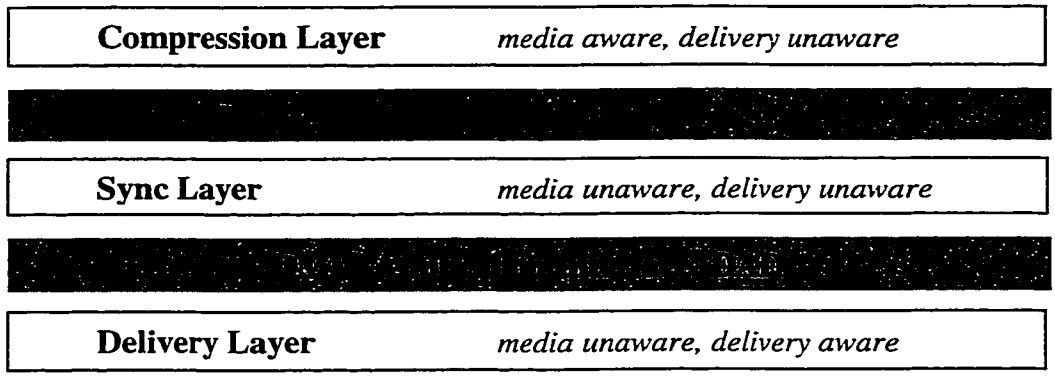

Figure 2.12 General MPEG-4 Architecture

The function of the Compression Layer is to encode and decode media into Elementary Streams (ES); the Sync. layer keeps the synchronization and hierarchical relations of the ESs; and the Delivery Layer provides transparent access to MPEG-4 contents irrespectively of the transport network technology. The boundary between the Sync layer and the Delivery Layer is the DMIF Application Interface. We will introduce DMIF in detail in Section 6.2.3.

Chapter 3: Quality of Service Management

3.1 Network QoS

3.1.1 QoS Definition

Quality of Service has a number of different definitions in different research fields. In general, it is a term that is very often used in computer networks. One of the definitions published in a web technology dictionary [31] is: *On the Internet and in other networks, QoS (Quality of Service) is the idea that transmission rates, error rates, and other characteristics can be measured, improved, and, to some extent, guaranteed in advance. QoS is of particular concern for the continuous transmission of high-bandwidth video and multimedia information. Transmitting this kind of content dependably is difficult in public networks using ordinary "best effort" protocols.*

The following parameters [32, 33] are normally used to evaluate the QoS of a system component:

- **Transit Delay**: A packet is sent from one computer to another. The time is different between the moment when the packet is sent out at an output port and the moment when this packet is received at an input port. Transit delay is used to express the time gap.
- **Transit Delay Jitter**: For the same stream, different data units may have different delays. Transit delay jitter indicates the variation of the transit delay.
- **Loss Rate**: Data may be lost during the transmission. Loss rate is the fraction of data loss.

In the context of multimedia applications, another parameter, throughput, can be used to express the communication requirements for the multimedia stream.

In the real world, networks are interconnected. The result of end-to-end service can be considered to be the concatenation of the individual network services [32]. *The QoS parameters P^{ee} of the end-to-end service may be calculated from the QoS parameters P^i of the i-th network (i-1,..., n).*

- *$Available\_Throughput^{ee}$ = minimum (for all i = 1,..., n) of $Available\_Throughput^i$.*
- *$Delay^{ee}$ = sum (for all i = 1,..., n) of $Delay^i$.*
- *$Jitter^{ee}$ = sum (for all i=1,..., n) of $Jitter^i$ [assuming that the jitter is defined as the difference between maximum and minimum delay].*
- *$Jitter^{ee}$ = square-root of sum (for all i = 1,..., n) of square of $Jitter^i$ [assuming that the jitter is defined as the average deviation of the delay from the average delay, and the delay is assumed to have a normal distribution].*
- *$Log\ (1 - Lossrate^{ee})$ = sum (for all i= 1,..., n) of $log(1 - Lossrate^i)$.*

While the QoS provided by the networks can be characterized by the above parameters, the delivery guarantee can be categorized into the following levels:

- **Best-effort**: network performs as well as it can. No consideration of user's requirements. The Internet today is in this mode.
- **Target-objectives**: network knows the user's requirements. It performs as well as it can to meet the requirements, but with no guarantee.
- **Statistical-guarantee**: deterministic guarantee for a certain fraction of the traffic, e.g. delay is less than 0.1 second for 95% of packets.

- **Deterministic-guarantee:** network service is always equal or better than the specified QoS requirements.

3.1.2 Related Work

Over the past years, there has been a large amount of research focussed on QoS management issues. The topics range from end-to-end QoS specification and adaptive QoS architecture, to QoS management agents, etc. The research covered both architecture and implementation issues of QoS management functions, such as QoS negotiation, QoS re-negotiation, QoS adaptation, QoS mapping, resource reservation, and QoS monitoring. We summarize several of these aspects in the following:

Local Components: The research in this field puts effort in providing QoS in servers and end-systems, such as CPU scheduling, memory management, and disk scheduling mechanisms [34, 35].

Advanced Reservation: For some applications, immediate reservation may not be suitable. The reservation needs to be set in advance. [36] describes a QoS negotiation approach with future reservations that de-couples service starting time and service request time. It allows to compute the QoS that can be supported for the time the service request is made, and at certain later time carefully chosen. If the requested QoS cannot be supported for the time the service request is made, the proposed approach allows to compute the earliest time, when the user can start the service with the desired QoS.

Traffic Management: Networks are operated in a store and forward paradigm, where the queuing strategies play an important role. Several queuing schemes are mainly used today: FIFO (First In First Out) queuing, Priority queuing, Class-Based queuing, and Weighted-Faired queuing. Each of them has advantages and disadvantages in the context

of QoS control. In addition, traffic shaping can be operated with non-FIFO queuing disciplines, to control what data are transmitted to the network and the transmission rate. Two methods for traffic shaping exist: Leaky-Bucket and Token-Bucket methods [37].

QoS Based Routing: Attempts in this field have been made towards providing routing algorithms that take QoS requirements into account. The state information is exchanged, so that a router can find a suitable route that satisfies the QoS constraints [38].

Pricing: Pricing can be an additional parameter of QoS negotiation. The issues are: who pays for the service, how is it indicated, what benefits can the receivers get from paying higher or lower prices, the fairness among all users, etc. [39]

We have introduced the QoS definition and some related issues in this section. An important branch of the QoS management named adaptation, will be discussed in the next section.

3.2 Adaptive Applications

3.2.1 Adaptation Principles

Adaptation is described in [40] as: *Adaptive methods, using scaling and filtering, may be seen as an alternative to reservation-based QoS provisioning, e.g., if reservations are not supported by the used networks. These techniques adapt the generated workload to the available resources by changing the characteristics of the transmitted data stream, e.g., lowering the frame rate of a video stream, and, thus, allow a smooth decrease in quality.*

Scaling means to reduce the load when the current service cannot be supported. To apply scaling at the receiver node, the network load and the end-system performance has to be monitored, based on the QoS parameters we introduced earlier, so that when a change appears that the service cannot be kept on the current level, an appropriate action can be

performed to reduce the load. For example, the reduction can be done by informing the sender node to slow down the transmission. The receiver node can achieve this goal either by sending an explicit message back to the sender, or through some other feedback mechanism. Different applications may use different scaling mechanisms.

Filtering is considered mostly in distributed applications that involve a large number of participants. Such kind of applications often possesses heterogeneous properties, which do not allow the implementation of scaling mechanism. One example is a multicast application, which involves many receivers who may have different system configurations, hence, may require different amounts of load from the sender. The sender cannot slow down the transmission because of a single receiver's request, while other receivers still request the higher load. A filter mechanism can be used to support such cases. It is often applied at an intermediate node, which can change the transmission load to adapt to different requests. The change can be done by re-encoding or discarding parts of the data. Removing parts of the data requires that the data is encoded into a hierarchy of layers, such as with MPEG-2 [41] or MPEG-4. An implementation of such filtering is described in [42]. The receiver starts with the base information, adding up enhanced layers until all the layers are reached, or a change happens, and the current parameter cannot be maintained.

3.2.2 Related Work

Adaptive applications are applications that can cope with wide and unexpected QoS variants and maintain the performance in an acceptable way. The issue is "*How to adapt multimedia application dynamically and continuously to their environment to make them deliver the best possible service under any given set of conditions*"[43].

A number of applications are developed along this line. They have been trying to deploy adaptation from different points of view, and to deal with different circumstances.

[43] gives an overview of adaptation algorithms and applications. Some examples of adaptivity are introduced. They are categorized into three main aspects: user centered – to emphasis on the user interface and interaction; system centered – to maintain a system parameter at a constant level; and mixed – to observe parameters from both the system and the users.

P. Moghe and A. Kalavade discussed in their paper [44] that recent work on QoS issues have two distinct trends: networking aspects and operating system aspects, including two kinds of QoS measures: end-to-end QoS measures, such as delay, packet loss, delay jitter, and terminal QoS measures, such as application processing delay. In their research, they tried to find the interaction between these two approaches. They considered a terminal that uses end-to-end QoS feedback to support adaptive applications according to their adaptation algorithm. The question was what the effect can adaptation have on the terminal processing delay. A theoretical framework was used to quantify the result. The goal of this research was to find a maximum adaptation level at which the processing delay is acceptable, and to use a terminal QoS measure to tune the adaptation algorithm. At the time this paper was published, the experiments were still in process.

[45] presented a scheme to adapt the transmission rate to the level of network congestion in the context of multimedia applications. The research was based on RTP and RTCP. The authors of this paper introduced a new algorithm, which can be used by the source node to control the transmission rate according to the feedback information. The results

were obtained from simulations. They considered that this scheme was efficient in utilizing the network resources and decreasing the packet loss rate.

In the CITR (Canadian Institute for Telecommunications Research) project "Quality of Service negotiation and adaptation", solutions were developed for applications involving access to remote multimedia databases. [32] discussed a few principles that can be used in QoS management. Three possibilities of QoS adaptations were introduced, in which an application can adapt to reduced network performance, related to throughput, loss rate, delay, and delay jitter. The first possibility is – when congestion occurs, the network may refuse new connection request – to delay the application and try later, or to degrade the QoS parameters in a session. The second way is to adapt to limited quality of communication service through alternative configurations, such as switch to another server or use another network. In the context of multicasting, another multicast tree may be selected. The last possibility is to use an alternative document structure, such as to present some text instead of some corresponding video stream.

[33] proposed an adaptive approach that allows to recover automatically from QoS violations. Three schemes were suggested. *Component Reconfiguration Scheme*: to replace the overload component by another component that is able to support the initially agreed QoS level and with the same functionality. *Resource Reconfiguration Scheme*: in response to QoS violation, to change the amount of resources reserved by the components, such that the end-to-end requirements can still be met. *Delay Recovery Scheme*: to request a higher level QoS from the other system components when a given component experiences a delay violation.

QoS adaptation in a multicast application is different in the sense that this kind of application usually involves many participants that may have a variety of QoS requirements. From this perspective, the end-to-end QoS adaptation algorithms are not applicable anymore. We will discuss this problem in the next chapters.

Chapter 4: System Requirements

4.1 Video Conferencing Applications

Video conferencing is a rapidly growing and changing field. Tele-teaching, which usually consists of one teacher site and some students sites, is a special type of video conferencing. There are numerous products available today. Products vary widely in the features, platforms, network techniques, and so on. New products are still being introduced. Since the concept of IP multicasting arose years ago, it has been playing an important role in distributed multimedia applications. In fact, a number of today's products and research applications are based on IP multicasting, for example, a number of video applications on MBone are available at [13]. In the very competitive field of distributed multimedia application, the ability to provide the best possible quality over a range of networks becomes highly desirable. Hence, QoS adaptation is discussed more and more in this field. However, QoS management in the context of multicast applications has only been addressed recently.

The majority of video conferencing applications do not provide QoS adaptation. Some applications with QoS adaptations are built on top of ATM or based on ISDN. The number of applications in the field of QoS over IP is small, and most of them provide end-to-end QoS adaptation. The applications using IP multicasting are fewer.

Among the applications that provide dynamic end-to-end QoS adaptation, the most common technique is dropping frames. Vosaic [46], a research project at the University of Illinois, provides dynamic adaptation by dropping frames within a pre-determined boundary. However, simply frame dropping can cause loss of synchronization.

Alternatively, a layered compression technique is used in many cases. For instance, VXtream [46] has a number of WebTheater products, which use a layered compression scheme to divide the compressed video into multiple streams with different priorities based on the impact on the video quality. The Department of Computer Science and Engineering, Oregon Graduate Institute of Science and Technology constructed an MPEG video and audio player [47] that supports frame dropping to cope with reduced bandwidth. The Application Level Gateway project [48] at the University of California, Berkeley uses the frame dropping algorithm and another technique, called transcoding, to achieve the dynamic adaptation. Video transcoding is the conversion from one encoding format to another. Another application, called SuperNOVA (Negotiated Online Video Access) also uses this technique [49]. However, this is a computationally expensive method.

There are only few applications with QoS adaptation based on IP multicasting. Scholars from the Lawrence Berkeley National Laboratory and the Department of Electrical Engineering, Texas A&M University proposed and implemented a dynamic QoS control scheme for video conferencing in the Internet [50]. This approach also uses the layered frame dropping mechanism. A source transmits layered video streams through multiple channels. This information is available to the receivers, and the receivers can dynamically add or drop layers to reduce the packet loss.

As a summary, we can say that most of the applications provide QoS adaptation by frame dropping. And few applications consider IP multicasting. We put our focus in this relatively new area, and developed our system using another adaptation approach. We will describe the approach and specify the system requirements in the following section.

4.2 Requirement Specification

Simply put, the system we want to build is an adaptive application that suits different QoS requirements of different users based on IP multicasting. Implementing QoS adaptation from the user's point of view is the main driving force of this project. Stefan Fischer, together with Abdelhakim Hafid, Gregor v. Bochmann and Hermann de.Meer, proposed an approach called "Cooperative QoS Management" [51], which meets our requirement to a large degree. Our project was based on this approach.

In their paper, they present an approach, where different levels of qualities are provided in the context of a distributed multimedia application, and so-called application-oriented QoS Agents are distributed throughout the network and in the end-systems, which communicate with each other and manage the QoS obtained by the different users. Distributed multimedia applications may involve many users. Therefore, the negotiation between the sender and all receivers becomes difficult, since the sender would have to keep state information for each receiver. Obviously, there is a scalability problem. The idea of Fischer's approach is not to do such direct negotiation, but to distribute part of the QoS management process to the receivers and allow each receiver to make certain QoS decisions locally based on its local QoS context.

It was assumed that for some mono-media component of a multimedia application, the sender process is able to offer several variants with identical content but different QoS characteristics. For example, a video clip may be available in several variants, having different frame rates, color qualities and/or resolutions. These variants are multicast to the receivers' workstations throughout the network.

Based on this idea, the requirements of our system are defined as following:

- We should have two basic types of modules: a sender module which multicasts several stream variants, and receiver modules that select a particular QoS variant. In each module, a logical entity, called QoS Agent, which is separate from the media data transmission should be used to fulfill the session and QoS management functions.
- The QoS variants should be defined in an appropriate format.
- The QoS Agent that resides in the receiver node should know QoS variants information, such as the number of variants, parameters, and so on. The QoS Agent also should know the user's preference. When the user wants to join a session, the QoS Agent should be able to select the best variant according to the user's preference using an appropriate algorithm. Once a session is established, the QoS Agent has to monitor the overall system. When temporary difficulties arise and the user-specified requirements cannot be met with the selected QoS variant (we say that a QoS violation occurs), the QoS Agent should be able to find an alternative variant to maintain the ongoing session. The QoS Agent should also provide the user the ability to modify the originally specified QoS criteria at any time during a session.
- The session control entity that resides in the sender node, which we call the QoS Manager, should decide the parameters of the variants.
- The user interfaces at the sender side and at the receiver side should be friendly and intuitive. At the sender side, the user should be able to define the parameters of the variants. At the receiver side, the user should be able to define the criteria that the QoS Agent will use to select from the available variants the most suitable one.

- There should be a separate entity that stores the receiver user profiles. The user does not have to define his / her preference every time he / she starts a session.

Having specified the system requirements, we will discuss how the system was constructed in the next chapters.

Chapter 5: Providing QoS Alternatives for Multicast Applications

5.1. General Assumptions

Our goal is to develop multimedia applications that can adapt to changing QoS conditions in the underlying transport service. The adaptation of applications to the different QoS requirements of different users in the context of multicast applications is considered. We consider a typical distributed multimedia application, teleteaching, where most data originates in one computer system at the teacher's side, which we call the sender node, and is multicast to a large number of student systems, the receiver nodes. We note that in such an application, there may also be some real-time data going from a receiver node to the sender and / or to the other receiver nodes. For instance, a student may ask a question which is broadcast to the teacher and all the other participants in the teaching session. However, we ignore this aspect in our discussion.

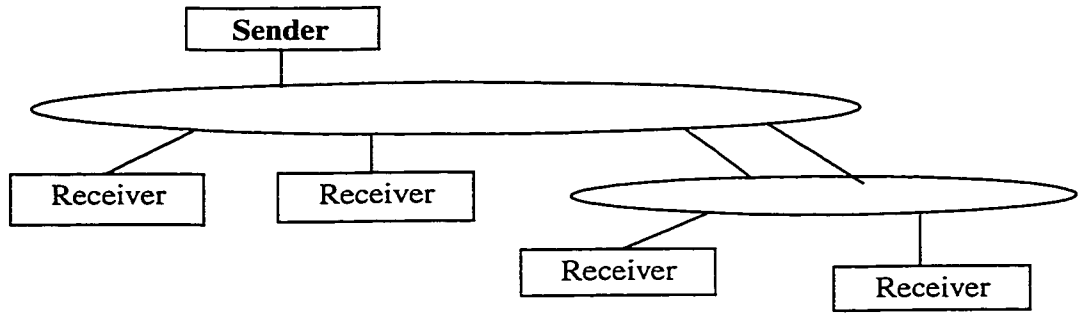

Figure 5.1 Overall System Architecture

Figure 5.1 shows the overall system architecture which we consider for a single instance of our multimedia application. The sender and receiver nodes are connected to one another through a communication network which supports a multicast service. The video

and audio streams originating from the sender are therefore multicast to the different receiver nodes that participate in the application session.

We assume that the users at the different receiver workstations have different quality of service requirements. These differences may be due to the following reasons:

- Different hardware and / or software resources available in the end system. Distributed multimedia applications may involve a range of end-user equipment, from low-end PCs to powerful workstations, which poses new challenges to QoS management.
- Different transmission-level QoS parameters provided by the network. Distributed end-user equipment may be connected to various network technologies, from wireless links to high-speed attachment. The transmission-level QoS parameters are different for the different receivers due to the specific network architecture and interconnection structure.
- Different user-level QoS parameters (that is, user preferences), such as requiring low-cost network service or high reception quality, which may imply higher costs. There may be different user-defined priorities for different aspects of quality, such as frame rate, color, resolution, etc. The priority factors show the user's preferences for a particular aspect of quality.

In order to accommodate these different QoS requirements, we assume that the sender node provides for each *logical multimedia stream* (for instance "video of teacher", "video of demonstration", and "audio of teacher") different *stream variants*, each representing a specific choice of user-level QoS parameters.

5.2 QoS Alternatives

Two logical streams are shown in Table 5.1, to give an example of what the QoS alternatives may look like. Video stream variants are distinguished by different frame rates, colors, resolutions, and coding schemes. Audio stream variants are different for audio qualities and coding schemes. Each variant is associated with a cost value, which gives the price that a user needs to pay in order to get this quality. The active flag indicates whether the stream variant is currently transmitted. The values of active flags are decided by the QoS Manager at the beginning of a session and also during the session when there is a state change of the corresponding channel. The details will be explained in the next section.

Video Stream A			
	Ch1	**...**	**Chn**
Frame Rate	10		30
Color	Grey		Color
Resolution	640*480		640*480
Coding Scheme	h.261		jpeg
Cost	10		20
Active Flag	Yes		No

Audio Stream B			
	Ch1	**...**	**Chn**
Quality	CD		Phone
Coding Scheme	PCM		PCM
Cost	2		1
Active Flag	Yes		No

Table 5.1 QoS Table for Logical Multimedia Streams

5.3 General System Architecture

Based on the system requirements we defined in the previous chapter, we construct a general system architecture for distributed multicast applications with QoS alternatives shown in Figure 5.2. The figure includes one sender node and two receiver nodes. The

receiver nodes communicate with a user profile manager that contains information about the user's QoS preferences and may also be used for user authentification.

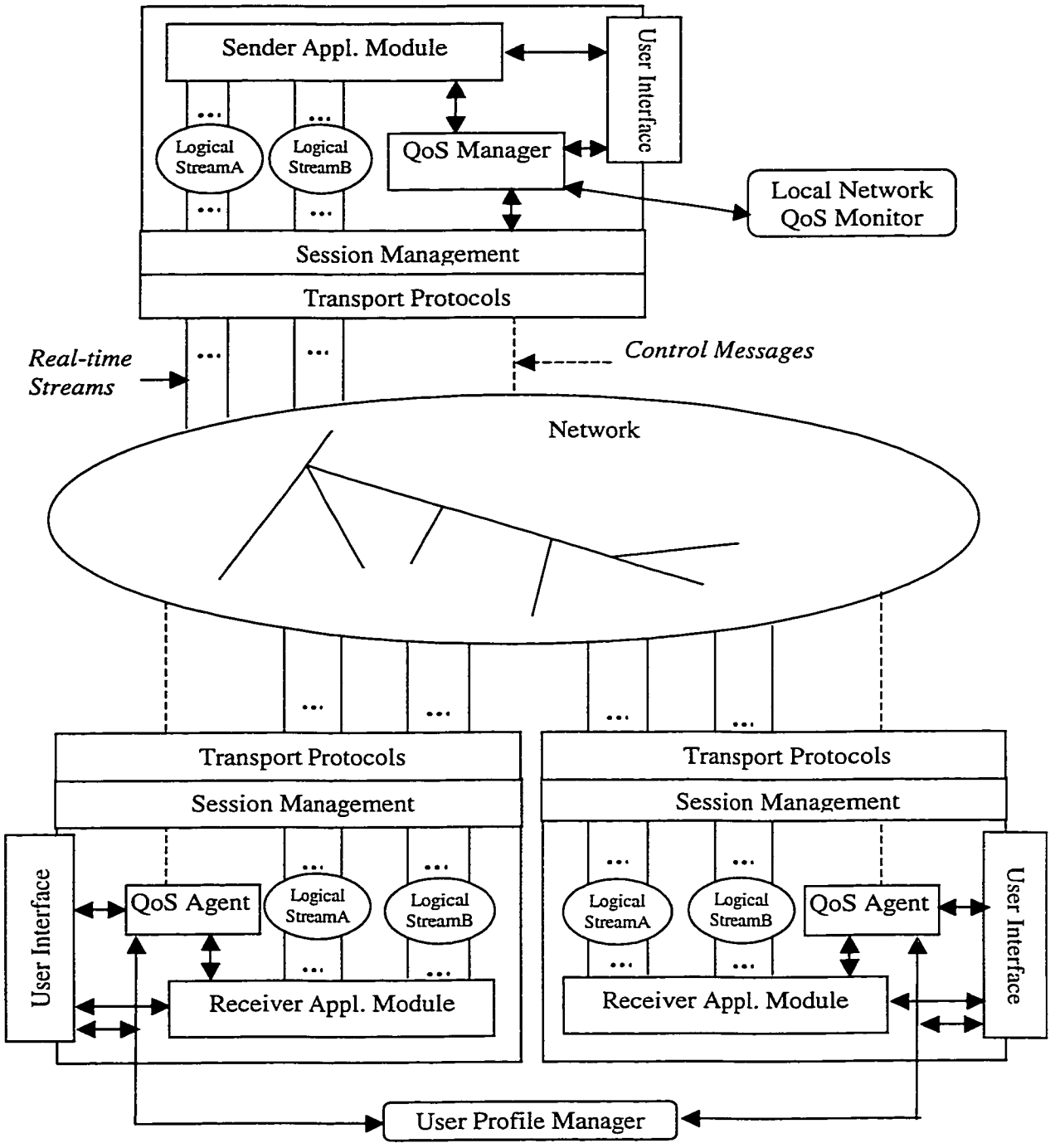

Figure 5.2 General System Architecture

These profiles can be produced offline and stored for different users or applications. The sender node may also communicate with a local network QoS monitor which, in the case of best-effort networks, provides information about the transmission quality that is presently available. We have not implemented such a separate network monitor in our prototype application described in Chapter 8. However, our prototype application monitors the network QoS, using parameters such as loss rate, delay, etc. at the receiver's side in order to detect any QoS violation.

There are two different kinds of channels involved, real-time channels that are shown as solid lines, and a signalling channel shown as a dotted line. Different variants are transmitted over different channels. One session may have several real-time channels for transmission of real-time session video and one signalling channel for exchange of session control messages between QoS Manager and QoS Agents.

The architecture of the sender and that of the receiver nodes are similar. They contain the application module that performs the application-specific functions, including all real-time media stream processing, such as video capturing, coding in several stream variants, transmission, reception, decoding and display, etc.

The transport layer provides a data transmission service with multicasting. In our prototype implementation, we assume that this includes the protocols IP with MBone multicasting, UDP and RTP/RTCP. We will discuss in the next chapter why we chose these protocols, how they benefited our implementation. The session management layer looks after the management of the application session, including the management of the transport channels for the different multimedia streams and the knowledge about the participating users.

The user (teacher) at the sender side determines the list of potential stream variants for each logical multimedia stream. These variants are defined through an appropriate user interface and passed to the QoS Manager in the sender node. However, not all of these variants will actually be transmitted at a given time. A variant will be transmitted if its active flag is true. This active flag is decided by the QoS Manager. The active streams transmit over different channels (multicast trees) all the time during a session. The receiver may choose one of them that best suites his / her preference. The receiver is connected to one channel at one time. For the selection of the potential stream variants and the activation of some of these variants, the QoS Manager may take into account the information about the presently available network transmission quality which can be obtained from the local network QoS monitor and through the monitoring of the active transport channels. It may also take into account specific requests sent by the users participating in the application. A stream variant becomes active and is transmitted if the QoS Manager considers that there are enough users that have requested the stream to be activated. On the other hand, through monitoring the active channels, the QoS Manager may decide to deactivate a channel (stop transmitting a stream variant) when there are not enough users that are connected to that channel. An example of potential stream variants is shown in Table 5.1.

The QoS Agent in the receiver node obtains information about the user QoS preferences either directly through the user interface, or by retrieving the user's QoS profile. Through an appropriate protocol, it can also acquire the potential stream variants information from the QoS Manager in the sender node. And it selects for each logical multimedia stream a specific stream variants which best fits the QoS preferences of the user. In case that the

most suitable stream variant is not active, it sends an activation request to the QoS Manager in the sender node. The QoS Agent is also responsible for monitoring the transport channel. When the current stream variant's quality cannot be maintained, it declares a QoS violation, and selects another suitable variant from the other potential variants. A user should be able to overwrite the automatic choice by selecting one channel manually or he / she may choose to leave the group. It is totally the user's decision. In addition, the user can stop the session at any time, and redefine his profile.

The user interface in the receiver node lets the user define his/her QoS profile, which is passed to the QoS Agent and stored by the User Profile Manager. Through the user interface, the user observes the circumstances of the current session in the context of what potential variants are available, which ones are active, and which one is actually used, etc.

5.4 A Typical Interactive Scenario

An example of a possible interaction scenario of QoS adaption is shown in Figure 5.3.

Point [1]: When a sender node starts a session, the QoS Manager initiates the session by multicasting the *Give_QoS_Info(session_ID, qos_List)* message. It specifies a particular *session_ID*, and encodes the information about the potential stream variants into *qos_List*. This primitive is multicast over the signalling channel.

Point [2]: When a new receiver joins the session, the user defines a profile or selects one profile from the pre-defined profile list first, and informs the QoS Agent to start the session. The QoS Agent sends an *Ask_QoS_Info (session_ID, user_ID)* to the QoS Manager, asking for the session information. The QoS Manager then responds with a *Give_QoS_Info (session_ID, qos_List)* to the Agent, providing information for all

potential stream variants, as shown in Table 3.1. According to the user's profile, the QoS Agent then selects the best stream variant (*port1* in our example). In case that no active stream is acceptable but some inactive streams are acceptable, the QoS Agent will select the best variant from the inactive ones, and go to Point 4. Otherwise, neither active nor inactive stream is acceptable, the QoS Agent will show the user a message, suggesting him / her to change his / her profile.

Point [3]: When a QoS violation is detected by the QoS Agent, it first checks whether one of the other currently active streams is acceptable. If so, it switches to it (*port 2*, in our example).

Point [4]: If some inactive streams are acceptable, the QoS Agent selects the best one from these streams, and asks the QoS Manager to activate this potential stream (*port 4*, in our example) by sending an *Ask_Add_QoS (session_ID, user_ID, qos)*. The QoS Manager determines whether or not to activate this channel. If the requested channel is activated, the QoS Agent simply switches to it. Otherwise, the QoS Agent shows a message to the user, reporting the current circumstance. If none of the streams, whether active or not, satisfies the user's expectations, the QoS Agent should inform the user about this situation and check whether the user wants to change the preferences in his/her profile.

Whenever there is a change in the table of potential streams, the QoS Manager should broadcast this information to all the receivers by sending a *Give_QoS_Info (session_ID, qos_List)* message.

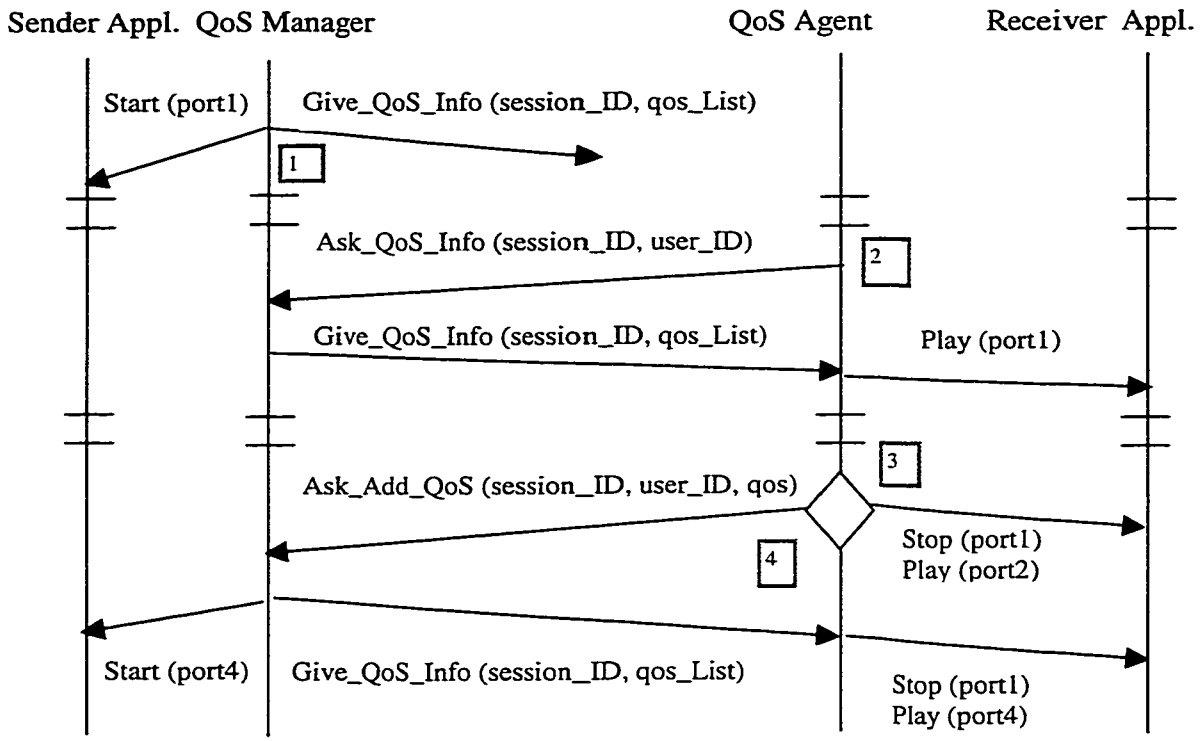

Figure 5.3 Example Interactive Scenario

We have specified the system requirements and the system architecture. Before implementing this system, we have to choose appropriate protocols for the real-time data transmission, and more importantly for the session control operations. In the next chapter, we discuss several protocol alternatives, and analyze our final choices, explaining how these protocols can simplify or benefit our implementation, and what the costs of using them are.

Chapter 6: Protocol Alternatives

6.1 Data Transport Protocols

We have introduced UDP, TCP and IP in Section 2.1.1, and also mentioned RTP/RTCP briefly in Section 2.1.4. For the video transmission part, we decided to use RTP over UDP/IP. In this section, we first describe why UDP/IP was our choice instead of TCP/IP, then illustrate what the benefits of using RTP/RTCP are, and finally discuss the costs, for example overhead problem of using these protocols.

6.1.1 TCP/IP vs. UDP/IP

TCP/IP protocols were designed to provide reliable data transmission with minimum delay. The Internet has been used primarily for this type of traffic. TCP/IP works very well in this context. However, for multimedia application, which comprises a significant amount of traffic and possesses different characteristics, TCP/IP is not quite suitable. Most media playback algorithms can tolerate missing data much better than long delays caused by retransmissions, and they do not require guarantee in sequence delivery. For example, in the case that a small portion of the data is lost in the transmission, if the receiver has to wait for a TCP retransmission, there will be an unacceptable gap in playing the real-time data. In fact, the loss of this small portion of data may not influence the play effect. Moreover, TCP's slow-start congestion control mechanism can interfere with the audio and video natural playing rate. Given the reasons above, multimedia applications generally ignore the complexity of TCP and use instead the simpler UDP. Most of today's multimedia applications are based on UDP/IP.

6.1.2 RTP/RTCP

RTP is intended to support the real-time multimedia transmission because it provides mechanisms that suite the characteristics of this type data. It has become a very popular protocol and is used by the majority of the multimedia applications on the Internet. RTP provides (a) payload type identification to identify the type of data, video, audio, etc. that is being passed to the receivers; (b) a sequence number to determine whether packets have been lost or have arrived out of order; and (c) a timestamp to reconstruct the time the packet was produced by the source. RTP does not provide QoS guarantees, nor does it ensure timely and in-order delivery. However, it can be used in conjunction with RTCP to monitor the QoS.

RTCP provides the following functions:

QoS feedback information: RTCP provides feedback information on the quality of data reception. For instance, the header of a receiver report includes several QoS related items, such as "fraction lost", "cumulative number of packets lost", "inter-arrival jitter", etc. Since RTCP control packets are sent from each participant to all the other participants, it can be easily used by a receiver or a third-party monitor to locate the network problem. This feature has significantly simplified our implementation because this information can be used by the monitoring entity to detect any QoS violation.

RTP source identification: RTCP uses a canonical name (CNAME) to identify each RTP source. CNAME is used to track the participant in a session and can also be used for synchronization purpose, e.g. to synchronize multiple media streams from one participant.

RTCP transmission interval control mechanism: Since RTCP packets have to be sent from each participant from time to time to all participants, to prevent network congestion, a mechanism is needed to control the transmission interval. Otherwise, a session with a large number of participants will occupy a lot of bandwidth. RTCP provides a scheme to scale the transmission interval according to the number of participants. Generally, the more the participants, the longer the interval.

Minimal session control information: This is an optional function of RTCP. In the case that participants frequently join and leave the session, the RTCP allows to convey a minimal amount of control information to all the session participants.

6.1.3 Protocol Choice and Overhead Analysis

Using RTP over UDP/IP does have drawbacks. The basic RTP header is twelve bytes long. This does not include the contributing source identifiers (CSRC) field and header extensions. The combination of RTP/UDP/IP has at least a forty-byte header which seems to be a large overhead. However, given the advantages and necessities of RTP/RTCP and UDP discussed in the previous sections, we came to the conclusion that there are compelling reasons to use RTP/UDP/IP for our project, although it may or may not outweigh the costs of using them. For further improvement, there are techniques to reduce this overhead, as introduced in [52]. We do not further discuss this technique in this thesis.

One good thing is that RTP does have controls of the RTCP traffic. As we discussed in the previous section, periodically transmitting RTCP packets from each participant to all of the other participants may overwhelm the network. RTP provides a mechanism to reducing the traffic to at most 5% of the total session traffic by adjusting the RTCP

packets generating rate. This mechanism helps to reduce the traffic on the network, thereby alleviating the overhead problem to some degree.

6.2 Session Protocols

6.2.1 SAP/SDP

SAP (Session Announcement Protocol) is a protocol that is used to advertise multimedia conferences and to communicate the sufficient conference information necessary for participation. A SAP entity announces a session by periodically multicasting packets to a well-known multicast address. Recipients listen to that address to get the session information. Such sessions are described by SDP (Session Description Protocol). SDP is used to describe the session specific information, such as the media type, attributes, session sender's information, and so on. Both SAP and SDP are text-based protocols. A SAP message includes a header and a text payload, which may be a SDP message. SAPv0 requires that the payload is SDP, while SAPv1 allows non-SDP payload.

SAP, together with SDP, is an efficient tool in terms of conveying session information between the sender and receivers. One good reason of using SAP/SDP as the session protocol in our system is that SDP defines a field "i", which is intended for describing the media specific attributes. It is claimed that it is most likely to be useful when a single session contains more than one distinct media streams of the same media type. This feature makes it perfectly suitable for our need of conveying information on different channels to the receivers. However, there is a significant problem with SAP/SDP in our context.

SAP is sort of a "one-way" protocol. Session initiators (senders) use it to spread over the session information to the session participants (receivers). One of the important

requirements of our system is to allow the receiver to request the sender to add a new channel into the session. This operation can not be done by SAP/SDP.

6.2.2 SIP

SIP (Session Initiation Protocol) was originally a research product of the IETF Multiparty Multimedia Session Control (MMUSIC) group. It was submitted as Internet-Draft in July, 1997 and assigned as RFC [53] in March, 1999. As the number of people that are interested in this protocol has increased rapidly, the IETF has recently created a new working group, called SIP group. In the early 1990's, SAP and SDP were used to establish MBone sessions. The initial intent of SIP was to create a way to invite new users into a conference. SIP is a client-server protocol. A client is an entity that originates messages, called requests, and a server is used to respond to or forward the messages. Logical SIP entities include the user agents and network servers. A user agent may contain a User Agent Client to initiate SIP requests and / or a User Agent Server to return SIP requests. A User Agent Server needs to register with a SIP server at start up time, so that it can be localized afterwards. A SIP network server may be one of three types: proxy, redirect and registrar server. The proxy server decides the next hop and forwards the requests; the redirect server sends address of the next hop back to the client; and the registrar server is usually co-located with the proxy or redirect server to offer the location services. An invitation to a session starts with a client issuing an INVITE request. The request may be processed by either a proxy server or a redirect server. A proxy server receives a request and forwards it to a next hop server, which may be another proxy server or the destination client. If the next hop is another proxy server, this procedure goes on until the request reaches the destination. A redirect server receives a request and

returns one or more next hop server addresses to the client so that the client can contact them directly. However, the usage of proxy servers or redirect servers is not mandatory. In the case that a client knows the destination IP address, it can send the INVITE request directly to the other end. However, SIP does not prescribe how a conference is managed. SIP is also a text-based protocol, which makes it relatively easy to implement. Furthermore, SIP is also a good protocol in terms of flexibility, extensibility, and scalability. It is gaining more and more acceptance by vendors and in fact, it is appearing in many products today. All the above advantages make SIP a very strong competitor with other session protocols. However, from the perspective of our system requirements, SIP was not the best choice since the QoS issues were totally outside the scope of SIP. Extensions to SIP are being introduced to make it useful for more specialized applications or to satisfy additional requirements. The QoS extension is one of them. However, at the time this project was developed, this extension did not exist.
We will see in the following section that DMIF's QoS model significantly simplifies our design.

6.2.3 DMIF

DMIF is the abbreviation for Delivery Multimedia Integration Framework, which is included in the suite of MPEG-4 standard specification. (ISO/IEC 14496-6). It is functionally located between the MPEG-4 application and the transport service, providing functionality that is needed to establish sessions and transmission channels between an application at one DMIF terminal and an application at another DMIF terminal over the network. The main purpose of DMIF is to define a session level protocol for the management of real time, QoS sensitive streams, to hide the delivery

technology details from the DMIF user, and to ensure interoperability between end-systems in the control plane.

Figure 6.1 Role of DMIF

An interface, called DMIF Application Interface (DAI), is defined in order to hide the delivery technology details from the applications. Also, by using media related QoS metrics at the DAI interface, applications are able to express their needs for QoS without the knowledge of the delivery technology. It satisfies three major technologies: broadcast technology, local disk technology and interactive network technology. Applications interact through the DAI without concerns of which technology is used. In addition, in case of interactive operations across a network, it ensures interoperability between end systems through a common DMIF protocol and network interface (DNI), which is mapped into the corresponding native network signaling messages.

Figure 6.2 provides a high level example of how DMIF creates a session. The arrows represent the following actions:

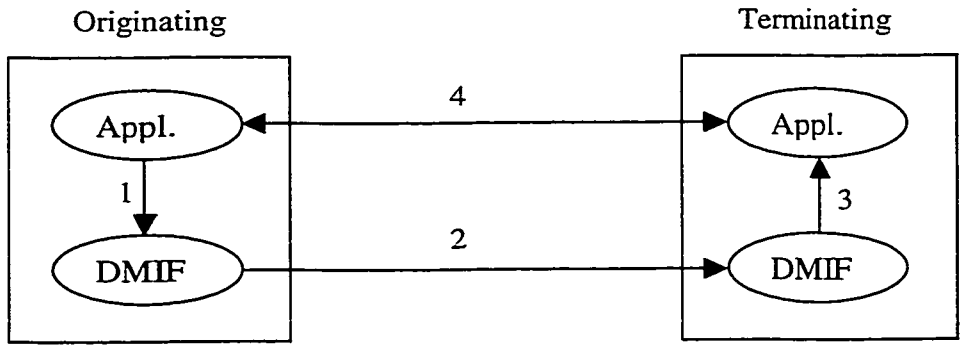

Figure 6.2 DMIF Session Creation

<1> To create a service session, the originating application requests a service activation

through DAI to its local DMIF.

<2> The originating DMIF established a network session with the target DMIF.

<3> The target DMIF forwards a service activation request to its application.

<4> The target application creates channels that carry actual data.

The basic DMIF concepts are defined in MPEG-4 version 1. Version 2, now being developed, specifies extensions for multicast scenarios [54]. Our project is based on these extensions. A DMIF terminal in a multicast session can be either a Data Producer DMIF Terminal (DPDT) – an information source, or/and a Data Consumer DMIF Terminal (DCDT) – an information receiver. A DMIF multicast session consists of a DMIF multicast signaling channel (C-plane / Control-plane) to distribute the state information of the session, and one or more multicast transport channels (U-plane / User-plane) to deliver the multimedia data. We can use the DMIF C-plane in our project since it meets our need for message exchange between two session control entities.

The DAI provides the *DA_ServiceAttach* and *DA_ServiceDetach* primitives for an application to create a new session (to attach itself to a given session), or to terminate a session (to detach itself from the session), and the *DA_ChannelAdd* and *DA_ChannelDelete* primitives for adding or deleting a transport channel for the session. The *DA_ChannelMonitor* primitive allows an application to inform the local DMIF entity to start monitoring a certain channel. The *DA_EventHandle* primitive is used by a DMIF entity to report a QoS violation to the application. In addition, it introduces a *DA_UserCommand* primitive that provides a means for transmitting control information

between the applications. Similar primitives exist at the network interface (DNI). Since in a multicast scenario, there may be more than one source involved in a session, the DNI includes in addition so-called *SyncSource* and *SyncState* primitives, which allow the receivers to identify different senders in a session and the senders to distribute the information about the multimedia streams to all receivers.

The following points are some details on the establishment of a multicast session between data producer and consumer DMIF terminals:

- A DMIF multicast session is identified by a DMIF-URL. A DMIF-URL is used to identify the location of a remote DMIF instance. In the multicast scenario, this URL is extended with the role of the DMIF terminal in the multicast session (DPDT / DCDT). So that the local DMIF layer is capable of recognizing the role of the application (sender or receiver) in the multicast session.
- The session's signaling channel address (IP address and port number) is available to the interested DMIF terminals by means of a session directory or through e-mail, etc.
- Each DPDT must explicitly join and leave the DMIF multicast session by sending messages over the signaling channel.
- When a DCDT joins a session, in order to reduce the number of signaling messages, it should listen on the signaling channel to collect the state information from all DPDTs participating in the session. If the DCDT does not acquire the information within a given time period, it should ask the DPDTs for their state information.

Figure 6.3 illustrates the scenarios of a session initiation.

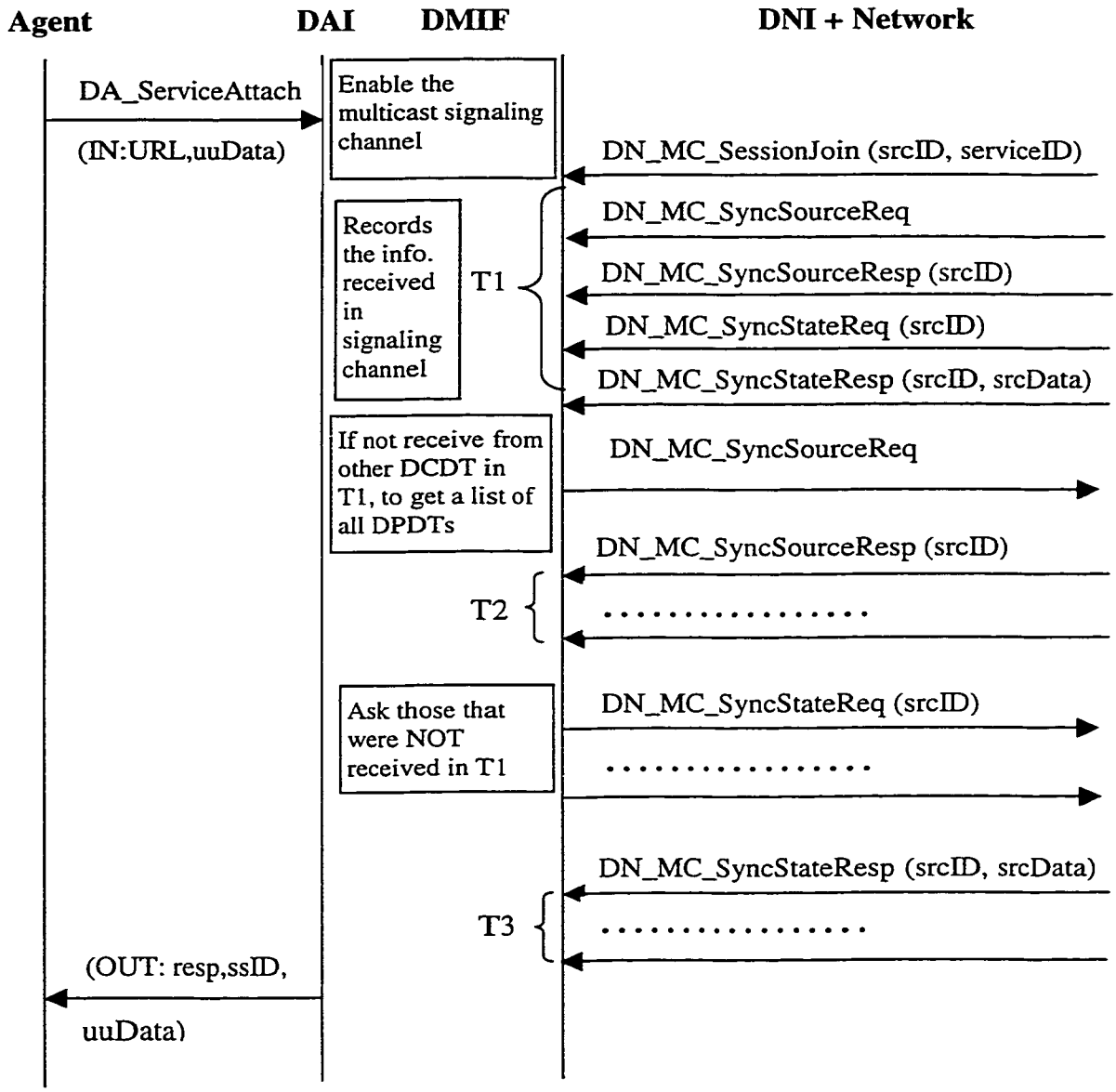

Figure 6.3 Example Scenario of a Session Initiation

6.2.4 Protocol Choice and Overhead Analysis

In the previous sections, we provided short introductions to three session protocol alternatives: SAP/SDP, SIP, and DMIF. SAP/SDP solution has a significant drawback

that does not meet our requirements. SIP is a strong competitor, but compared with DMIF, it is not the best candidate for our system.

As a final decision, we came to DMIF. DMIF, as a session control scheme in our prototype, has the following advantages:

- The QoS model of DMIF significantly simplifies our implementation. Assume that we could use either SAP/SDP or SIP solution, we still would have to extend the protocol with QoS parameters, because the QoS issues are out of the scope of those protocols. Whereas, DMIF (version 2) provides two primitives, *DA_ChannelMonitor* and *DA_EventHandle*, for the purpose of monitoring a certain channel and reporting the violation, which fully meet our requirements.
- DMIF is a signaling protocol, which separates the user data from the control messages by using two different kinds of channel types: the session data channel and the control message channel. In this way, the exchange of control messages does not interfere with the media content. This is a common advantage of signaling protocols, including SAP and SIP.
- The concept that one session may contain several channels perfectly meets our requirements.
- A parameter uuData (user data), is defined in most of the primitives. This parameter can be used to handle the transmission of proprietary data in our system, hence gives the implementation the maximum flexibility.

Having discussed the advantages, we now analyze whether DMIFv2 has overhead problems. All DMIF signaling messages have a common header with the length of twelve bytes. This also seems to bring an overhead problem. However, given the way the DMIF

works, we note that DMIF does not have an overhead problem in general for the following reasons. First, as we have explained in Section 6.2.3, every time when a DMIFv2 entity wants to request information for some specific channels or sessions, it listens on the signaling channel for a short period of time to see whether the same request has been sent by others. This scheme helps to reduce the traffic. Second, unlike the session media data, DMIF messages are not transmitting all the time, for example, the sender multicasts DMIF messages at start up time, on receiver's requests, and when the channel information changes. The receiver sends DMIF messages only when he / she wants to request an addition of a channel. Moreover, DMIF messages take a separate channel so that the media transmission is not affected by the DMIF messages. So we conclude that the DMIF overhead is not a big concern in our project.

We have discussed why DMIF was our choice. In the next chapter, we will discuss how DMIF can fit into our project, what modifications have been made to make it more suitable for us. A typical interactive scenario is given to explain how the DAI primitives are used to implement the session management functions.

Chapter 7: Using DMIF for Session Management in Tele-Teaching Applications

7.1 A Typical Interactive Scenario

The DMIF of MPEG-4 was designed not only for use with MPEG-4 applications, but also for generic use in a much wider context. It defines a session level protocol that can be useful for any real-time multimedia applications using a number of concurrent streams in a distributed environment. DMIF defines a control plan that is used for applications to exchange session control messages, and a user plan, which may contain one or more data channels that carry the actual data of the session. DMIF ensures interoperability between end-systems in the control panel. Considering the example scenario in Figure 5.3, we see that we have to specify a mechanism for message exchange between the QoS Manager and the QoS Agents. We also compared several session protocol alternatives in the previous chapter and came to a conclusion that DMIF is the best choice.

Our tele-teaching application is developed in the context of the Internet. We assume that the underlying transport is provided by the UDP/IP protocols complemented with some multicasting facilities, such as IP tunneling, provided by MBone. There are two kinds of multicast channels: real-time transmission channels and a control message exchange channel. Each multicast channel is identified by a multicast IP address plus a UDP port number. We assume that RTP and RTCP are used for real time multimedia stream transmission. They are used for the purpose of synchronization and for monitoring of network-level QoS parameters. The DMIF network interface is therefore mapped onto the

multicast transport service provided by RTP/RTCP. The application lies over the DMIF layer.

The abstract scenarios shown in Figure 5.3 can be mapped to the DMIF primitives provided to the application through the DAI. A typical example is shown in Figure 7.1, which corresponds to the abstract scenario described in Section 5.4. Specifically, the figure shows the following interactions:

Point [1]: The QoS Manager initiates a session. The session's signaling channel address is identified by a DMIF-URL. The information of the potential stream variants (see Table 5.1) is encoded in the user data field *uuData* of the *DA_ServiceAttach* primitive and forwarded by the DMIF through the *SyncSource* and *SyncState* messages. *SyncSource* and *SyncState* should be broadcast over the signaling channel. *SyncSource* contains a session identifier. *SyncState* contains the session identifier and the state information of the session, in our case, the potential stream variant information encoded in *uuData*.

Point [2]: The session's signaling channel address DMIF-URL is available somehow to the receiver DMIF terminals, for example, by a session directory or email, etc. When a receiver wants to join the session, the QoS Agent sends a *DA_ServiceAttach* primitive to inform the local DMIF protocol entity. Since there may be a number of receivers involved in the session, to avoid requesting the information that other receivers just requested, the DMIF entity listens on the signaling channel for a reasonable time period, and collects the *SyncSource* and *SyncState* messages from the sender side. If these messages are not recieved during a given time period, the DMIF entity should request them. A *SyncSource* request is sent first to get a list of all the sources (senders) in this multicast session, in our module, only one. A *SyncState* request is sent afterwards to get

the stream variants information from the sender. The received information is passed to the QoS Agent in the *uuData* parameter of the service attach confirmation. The QoS Agent keeps this information locally. Whenever a channel activation or deactivation happens, the QoS Agent should refresh the local information.

Point [2']: According to the user's current profile, the QoS Agent selects an appropriate stream variant, and sends a *DA_ChannelAdd* response primitive to the DMIF entity. The *DA_ChannelAdd* response primitive is originally used to respond to the *DA_ChannelAdd* indication primitive, but it can also be used at any time during a session. In our case, the receiver does not have to connect to the new activated channel. This primitive can be used when the receiver wants to connect to a certain channel. The channel is indicated by the channel ID that is carried in this primitive. The DMIF will perform a group join operation at the DNI in order to join the MBone multicast group for the selected multimedia stream. Another primitive, *DA_ChannelMonitor,* is sent along with the channel add response primitive to inform the DMIF layer to start monitoring the network activities, and also defines the monitoring mode to the DMIF. There are three options for the QoS monitoring mode defined in the DMIF specification:

- **Periodic Monitoring**: The DMIF layer monitors the QoS of a particular channel and reports to the application periodically.
- **QoS Violation Detection**: DMIF layer reports to the application only when a QoS violation happens.
- **Application Request**: DMIF layer reports when the application requests it.

We use the second mode in our implementation.

Point [3]: The DMIF entity indicates a QoS violation, by sending a *DA_ChannelEvent* primitive, with detailed QoS information in *qosReport* parameter. If one of the other currently active streams is acceptable, the QoS Agent simply decides to switch to that stream.

Point [4]: If none of the currently active streams is acceptable, but an inactive stream is acceptable, the QoS Agent sends a *DA_UserCommand* primitive to ask for the activation of that stream, as specified in the *uuData* parameter. If neither active nor inactive streams are acceptable, the QoS Agent should suggest the user with three options: change his/her QoS profile, wait for some time and try again later, or abandon the session.

Figure 7.1 mainly shows the mapping from the abstract scenarios in Figure 5.3 to the corresponding DMIF primitives. It only contains the session setup and interactive operations during the session. To make the example more complete, the channel deactivation and session termination scenarios are shown in Figure 7.2.

Once the QoS Manager finds that there is no user of a particular channel, it may decide to deactivate it. The QoS Manager can get the channel's participants' information through the RTCP reports. Since each participant is required to send RTCP reports to all the other participants, including the sender, the QoS Manager in the sender node can assume that there is no user of a channel if no RTCP reports are received for a period of time. In this case, it broadcasts the *DA_ChannelDelete* primitive over the signaling channel to inform all the receivers to update their session stream information. A receiver can leave a session at any time. When a session is finished, the sender stops all the channels and broadcasts *DA_ServiceDatach* primitive over the signaling channel. If a receiver is still attached to this session, a message will be displayed to inform the receiver that the sender leaves.

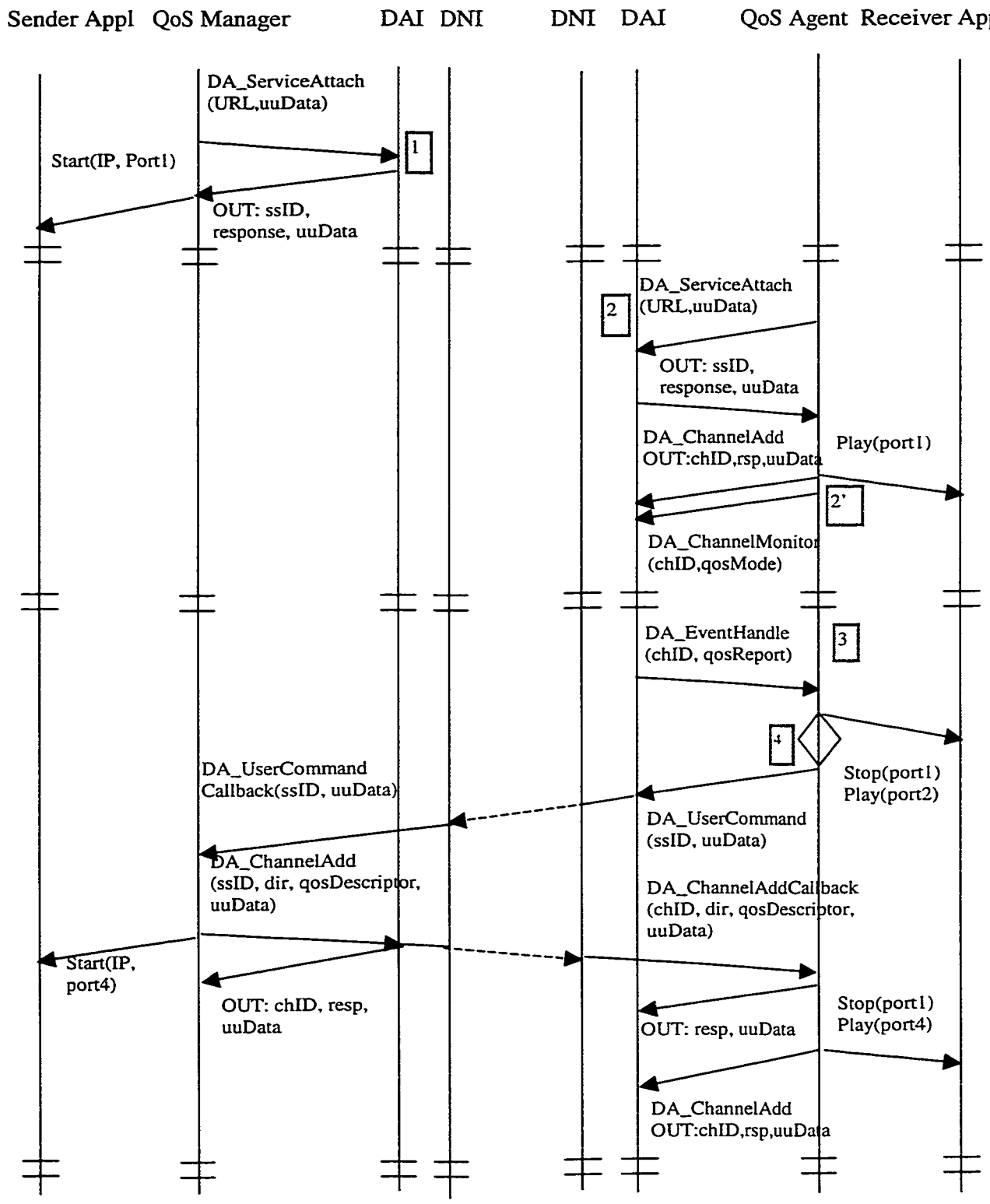

Figure 7.1 Multicast Session Interaction Scenario including DMIF (Part 1)

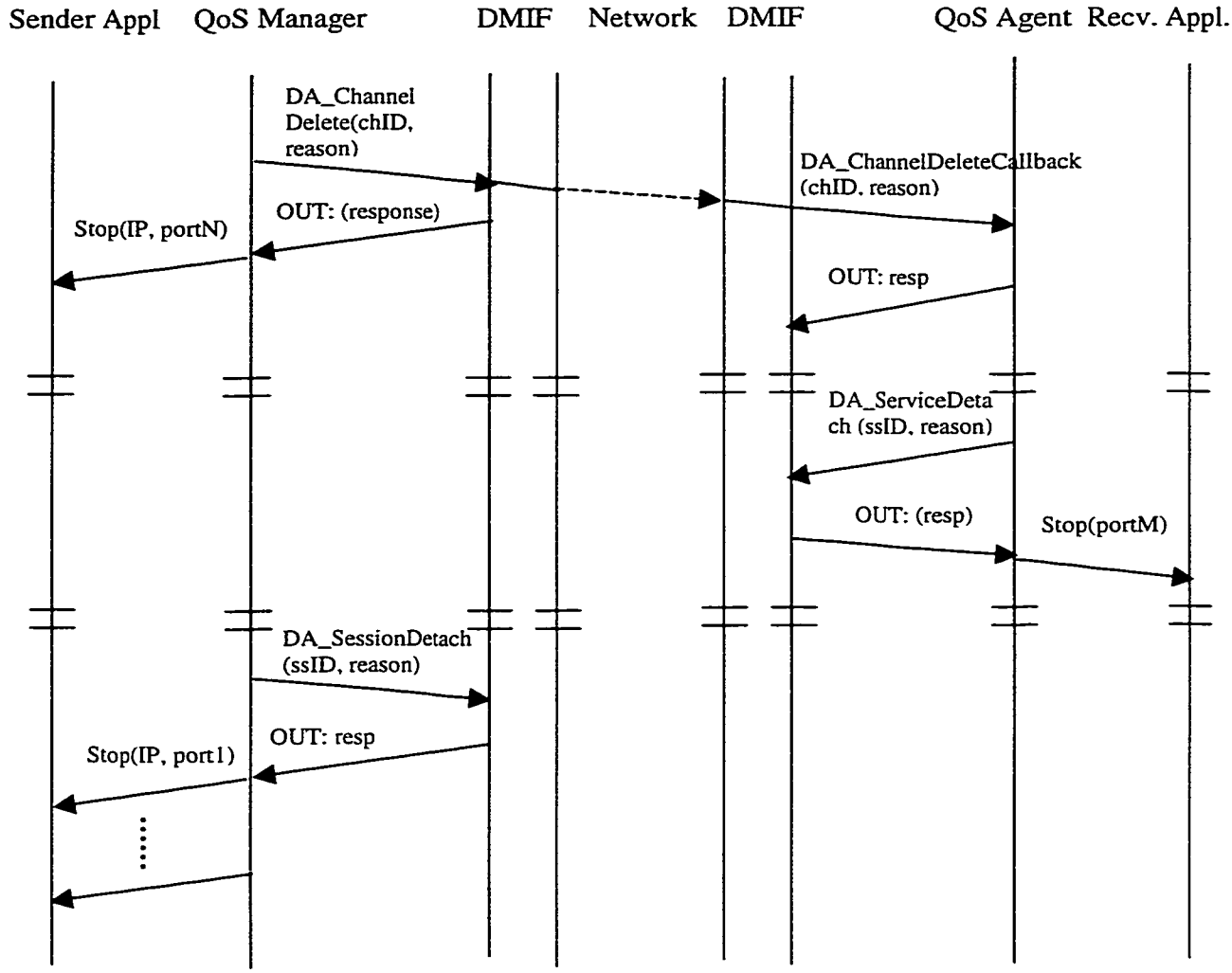

Figure 7.1 Multicast Session Interaction Scenario including DMIF (Part 2)

7.2 Changes to DMIF

The MPEG-4 standards are versioned, as well as the DMIF specifications. Our project is based on the ISO Working Draft of DMIF Version 2.

We found that the DMIF specifications of this version fit well with the session and QoS management functions that were required by our tele-teaching application. However, there are certain points that do not fit our requirements well. We mention in particular the following aspects:

- **SyncState message parameters:** The information about all potential stream variants (see Table 5.1) is encoded in the user data field *uuData* of the *DA_ServiceAttach* primitive invoked by the QoS and session manager in the sender node. When a receiver node joins the multicast session, this information is passed along in the *SyncState* message from the sender to the receiver DMIF protocol entity. As specified in the Working Draft, this primitive only contains the information of the active channels and the QoS information included provides only information about the transmission-level QoS parameters of the channels, but not the user-level parameters as we need in Table 5.1. This requires certain changes to the *SyncState* message parameters. The user-level parameters in Table 5.1 must be added to the *SyncState* message. In addition, the information for the inactive channels must be included.
- **Requesting the activation of a stream variant:** Another issue is the question how a QoS Agent in a receiver node can request that an inactive stream variant be activated. We have adopted the use of the *DA_UserCommand* primitives for this purpose. This primitive is designed to allow the transfer of user information from application to application, in our case from the QoS Agent in the receiver node to the QoS Manager in the sender node. The IP address and port number of the requested channel is carried in the parameter *uuData* of this primitive, which is sent from the QoS Agent to the QoS Manager. The QoS Manager should decide whether or not to activate this stream variant channel in a certain period of time. If it decides to activate the channel, it invokes a *DA_ChannelAdd* that leads to a multicast control message to all participating receiver nodes, in other words, a *DA_ChannelAdd* indication to all QoS Agents. If the QoS Manager does not activate the channel in the time period, as we

described before, QoS Agent in the receiver node will launch a message that suggests the user to change his/her profile or try again later on.

These adjustments are relatively minor. This shows that the DMIF protocol is very suitable for these kind of applications. It also shows that certain aspects of the DMIF protocol should be left open-ended to allow easy adaptation to various applications, such as ours.

Chapter 8: Prototype Implementation

8.1 Implementation Languages and Environment

Our implementation environment is based on MBone. One sender (teacher) node and several receiver (student) nodes communicate over the Internet via RTP/RTCP, UDP/IP and MBone protocols. At each side, there are two major parts: real time video transmission and session control message transmission. For the first part, we did not develop a completely new multimedia application, but used an existed MBone video conferencing tool VIC at the teacher side, and a Java application 'RTP player' at the student side. 'RTP player' is an example application of the Java Media Framework (JMF) [55], which provides APIs that specifies a unified architecture, messaging protocol and programming interface for playback of the timed media. It enables multimedia content in Java technology-based applets and applications, and allows for Web-based multimedia solutions that run in any Java compatible environment. Ideally, the teacher side should also use JMF APIs to capture the video from the camera and transmit to the student nodes. However, at the time this project was developed, only JMF version one was available, and the capture functions were not included in this version. So we chose a popularly used video tool on MBone, VIC, at the teacher side. The advantage of using VIC is that VIC provides the users the ability to specify the video transmit rate, which largely simplified our implementation. However, these tools have to be modified in order to be suitable for multicast applications and QoS management. The Java interfaces representing the abstract DAI primitives are specified in the DMIF (version 2) specification. We implemented these interfaces in the sender and receiver applications

with the necessary modifications (describe in Appendix B) for realizing the session and QoS management functions.

Due to the time limitation and some technical difficulties, the following functions are ignored in this implementation.

- The "Local network QoS monitor" module was not implemented. Instead, we let the sender decide which channel is activated at the beginning through the user interface.
- As a simple case study, only the video part was implemented. The audio part only has abstract structures, e.g. class definitions.
- The QoS adaptation was applied only in the context of frame rate. The sender application offers different frame rates by simply adding one parameter "-F rate" to the command, which initiates the VIC process.
- VIC was initially planned to be modified so that the sender application is able to capture data from one camera and send it over different channels using different frame rates. In final implementation, this feature was ignored but several cameras were used instead, one for each channel. VIC was used in the Windows operating system simply as a separate process, which is initiated by a system command line that is created and executed by the Java program.

In the earlier implementation, only the DMIF application interface (DAI) was implemented. The service provided by this interface was implemented by using Java RMI (Remote Method Invocation) [56]. Later, the DMIF protocol together with the DMIF network interface (DNI) was implemented by Mr. Darlagiannis as part of his MSc. project [58]. This software was integrated into our final prototype.

8.2 Key Design Issues

8.2.1 Global Program Structure

There are three main program modules: sender side, agent side, and user profile manager. This section gives the class diagrams of the first two modules. The user profile management module will be described in Section 8.2.4.

- **Notes Concerning the Sender Side Class Diagram** (see Figure 8.1)

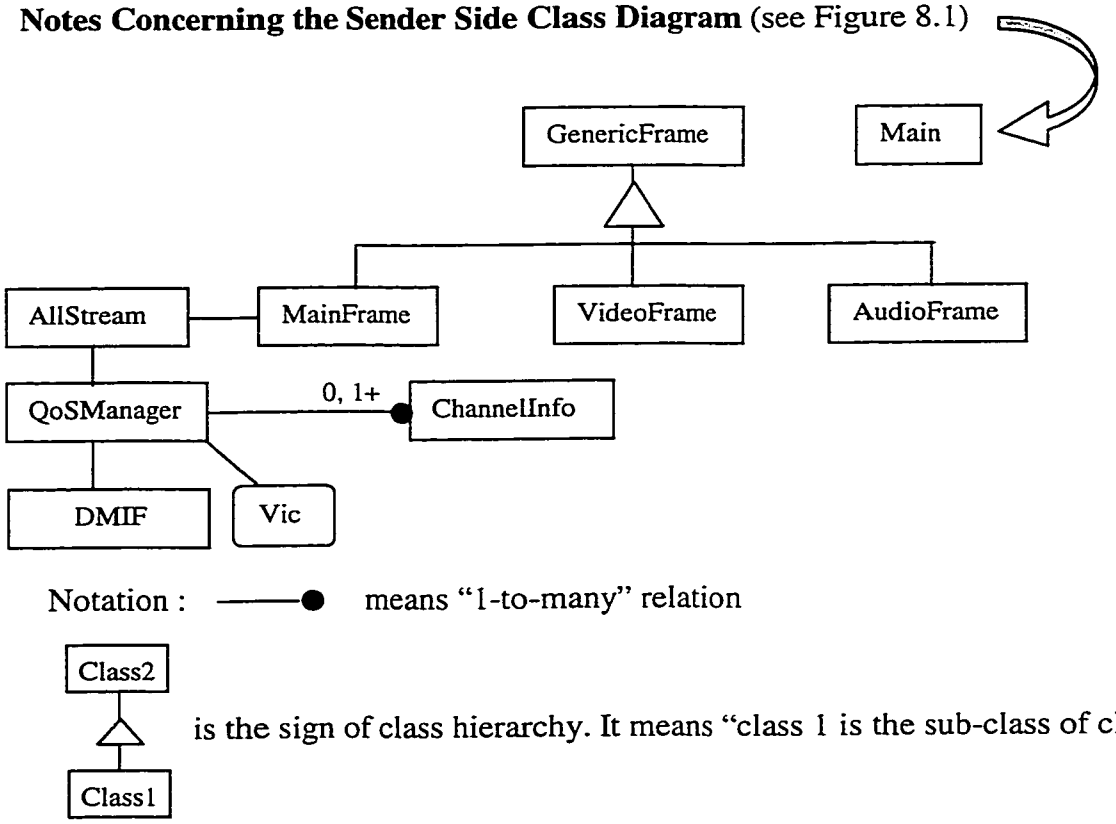

Figure 8.1 Sender Side Class Diagram

<1> Two DMIF APIs –*ApplicationSession* and *DMIFInstance* – are defined in the DMIF specification in the form of Java interfaces. They are implemented by two concrete classes: *QoSManager* implements *ApplicationSession*, and *DMIF* implements *DMIFInstance*.

<2> The program starts to execute from the *Main* class, which contains the *main* method.

<3> All user interface classes have the suffix *Frame*. *GenericFrame* is the super class of all user interface classes.

<4> *ChannelInfo* is used to maintain information for a channel, namely the port number and a channel ID, which is provided by the DMIF entity. It has the following simple attributes: *port* (integer) and *chID* (long integer). *QoSManager* keeps a collection of active channels (a Vector of *ChannelInfo*), which may contain 0 or more channels.

<5> VIC is called as a separate process under the Windows operating system.

<6> The subclasses of *AllStream* are explained in the next section.

- **Notes Concerning the Receiver Side Class Diagram** (see Figure 8.2)

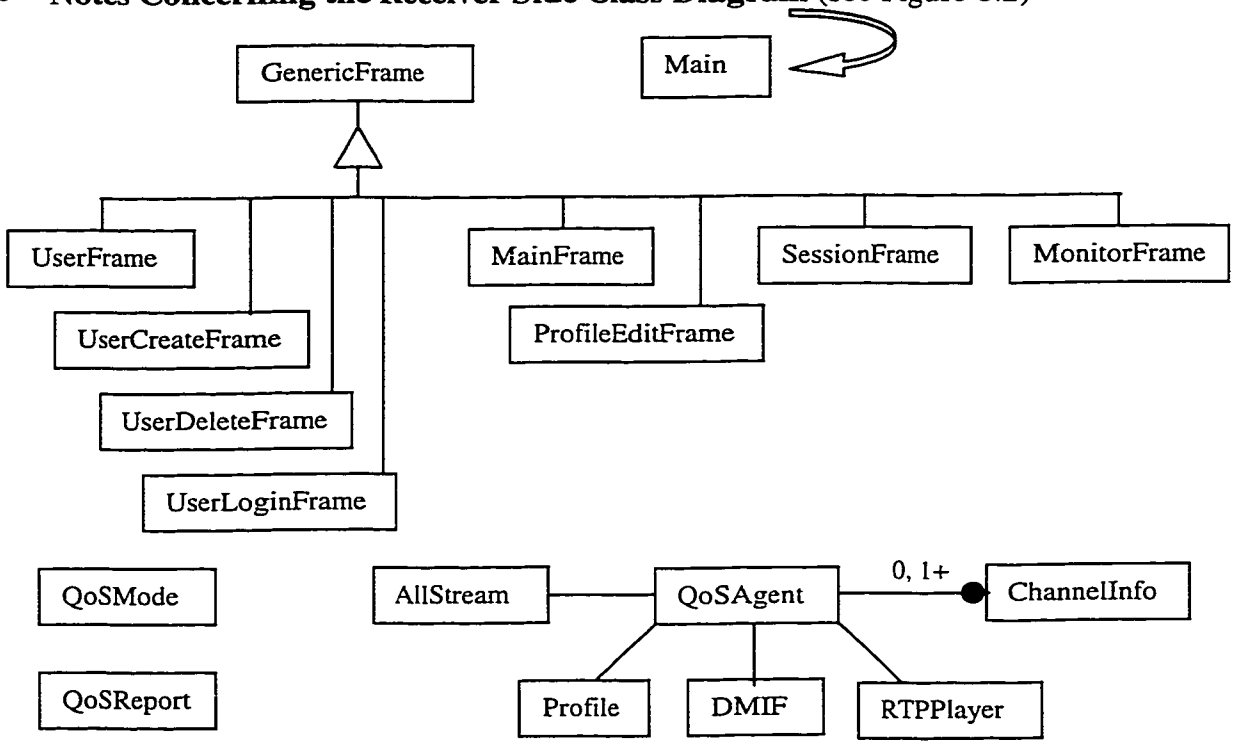

Figure 8.2 Receiver Side Class Diagram

<1> The notes 1, 2, 3, and 4 for the sender side class diagram apply here as well.

<2> The classes *QoSMode* and *QoSReport* are introduced in order to conform to the Java DMIF API specification. An object of *QoSMode* is carried in *DA_ChannelMonitor*

primitive. *QoSAgent* uses it to inform the *DMIF* which monitor mode should be used, in our case, report on violation mode was used. An object of *QoSReport* is carried in *DA_EventHandle* primitive. *DMIF* uses it to convey the report of violated QoS parameters to the *QoSAgent*. The format of these two classes will be described in Appendix B.

8.2.2 QoS Manager and QoS Agent Implementation

The QoS Manager and QoS Agent modules are implemented as two classes, which have the following features:

- **QoS Manager**:

<1> Keep the information about the QoS variants.

<2> Be aware of the user's input, since the user may overwrite the QoS manager's decision.

<3> Broadcast the information about the QoS variant at the beginning of a session.

<4> Listen to the control channel for the receivers' requests for channel activation.

<5> Decide whether or not to activate the channel when a receiver requests the activation of an inactive channel.

<6> Broadcast the information about the QoS variant whenever there is a change.

<7> Inform the application module of which channels should be started or stopped.

- **QoS Agent**:

<1> Get the information about the QoS variants at the beginning of a session.

<2> Get the user's preference from user's input or the User Profile Manager.

<3> Select the best variant for the user.

<4> Take an appropriate action when the best variant is currently not available .

<5> Inform the DMIF entity to monitor the selected channel.

<6> Select another variant when QoS violation occurs.

<7> Inform the application module of which channel should be played.

8.2.3 Format of QoS Alternatives

Video and audio QoS alternatives are shown in Table 5.1. To express them in a logic way, the following class diagram was designed:

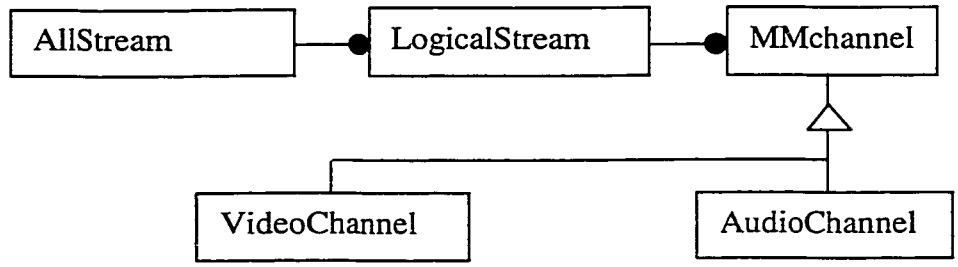

Figure 8.3 QoS Alternatives Class Diagram

AllStream is what a sender can offer to the receivers. It may contain several logical streams (*LogicalStream*), video and / or audio, etc. Each logical stream may contain several variants, which we call channels (*MMchannel*). Video and audio channels have some common parameters, such as IP address, port number, active flag, cost, and a value to express the network level QoS parameters, which will be used by the receiver DMIF entity to detect the QoS violation. There are also some differences between video and audio channels (*VideoChannel* and *AudioChannel*). Video channel parameters include parameters for frame rate, color, two-dimension resolution, and the coding scheme. Audio channel parameters include parameters for voice quality and coding scheme.

8.2.4 Format of DAI Primitive Parameters

The format of DAI primitive parameters is defined in the DMIF specification. However, the formats of some of the parameters need to be modified in order to be more suitable for our particular application. The modified primitives are listed in Appendix B.

8.2.5 User Profile Manager API

As described before, the User Profile Manager is a server application that is used to store user (student) profiles. A student who is a new user of the system should first create an account. For security reason, a password is required. Once a student creates an account on this server, he / she is automatically assigned a default profile, which can be modified later on, but cannot be deleted. A student may keep several profiles. Students are able to add new profiles, modify or delete existed profiles (except for the default profile). When joining a session, a student may either select one of the profiles or not select any, in which case the default profile will be used to for the new session.

A User Profile Manager maintains a number of users, and a user may have a number of profiles. The class diagram is shown in Figure 8.4.

The *Profile* class has attributes to describe the user's preferences concerning frame rates, colors, resolutions, audio qualities, etc. It includes a profile name and parameters in three categories: video, audio and cost. Each video or audio parameter has a desired and a minimum acceptable value. An importance factor is associated with each parameter in all three categories. This class definition is shown in Appendix C.

Figure 8.4 User Profile Manager Program Module Class Diagram

The user profile manager is a server program that should always run, since we do not know when a user may login to get a profile. In the case that the server has to be shut down, for maintenance or any other reason, the user and profile information will be stored in a file, and will be reloaded into the program when the server is run again. The users and profiles are stored in the object format provided by the Java Object Serialization [57] facility.

The client-server structure for the profile manager is implemented by using Java Remote Method Invocation (RMI). User Profile Manager is seen as a remote object on which the student application may invoke remote method calls. The thread usage in RMI works in a way where calls from different clients are executed in different threads. Therefore, in our case, more than one student applications may invoke the same remote method at the same time. The available methods (APIs) include user-related methods, such as *createUser*, *deleteUser*, *loginUser*; and profile-related methods, such as *getDefaultProfile*, *addNewProfile*, *getCertainProfile*, and so on. The exact definitions of these APIs are shown in Appendix C.

8.2.6 User Interface

- **Sender Side User Interface**

The user interface at the sender side is shown in Figure 8.5. Through this interface, the user at the sender side defines the potential stream variants of a session, including video and / or audio streams.

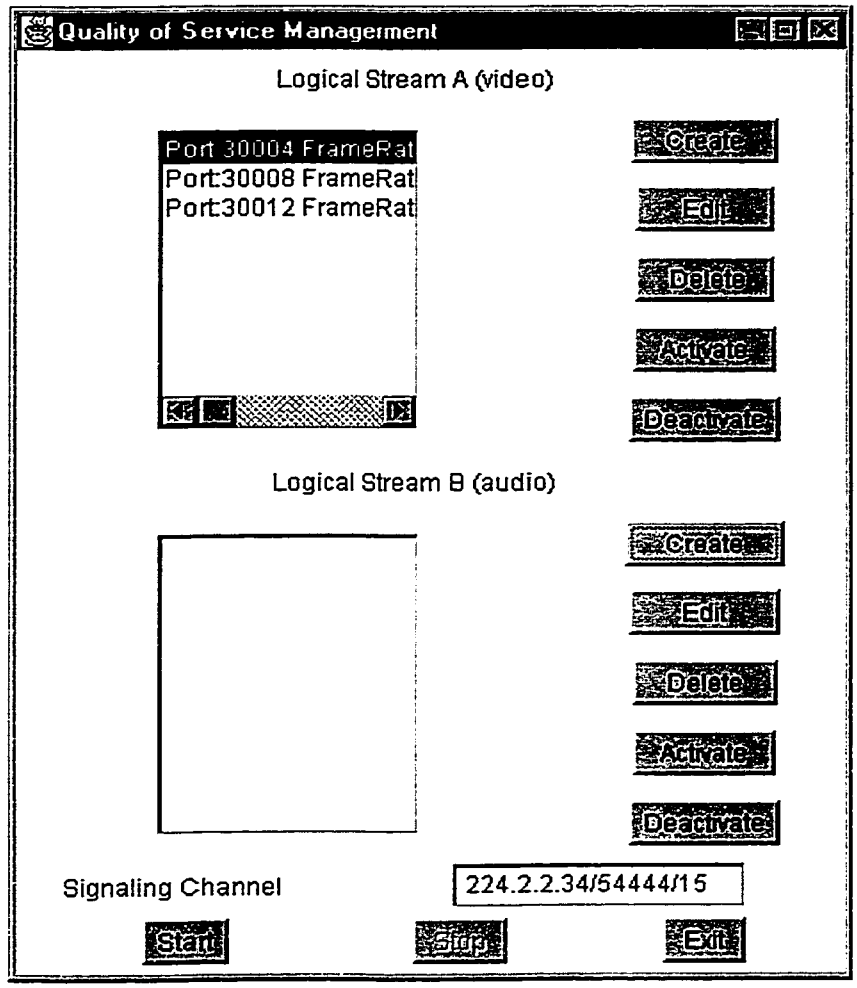

Figure 8.5 Sender Side Main User Interface

At the bottom of the screen, there is a text area that the user uses to specify the signaling channel's IP address, port number and TTL. We assume that for a given session the same IP address is used for all channels, including signaling and data channels. The different channels are distinguished by port numbers. This decision of using the same IP multicast address for all channels was taken for simplicity, however, this was a bad decision, since in this case all the data streams will be multicast to all receivers. Future versions of this system should correct this problem by letting the QoS manager select different IP addresses for the different data streams.

The user may create, edit or delete a stream variant. The user interface for defining the parameters of a video stream is shown in Figure 8.6.

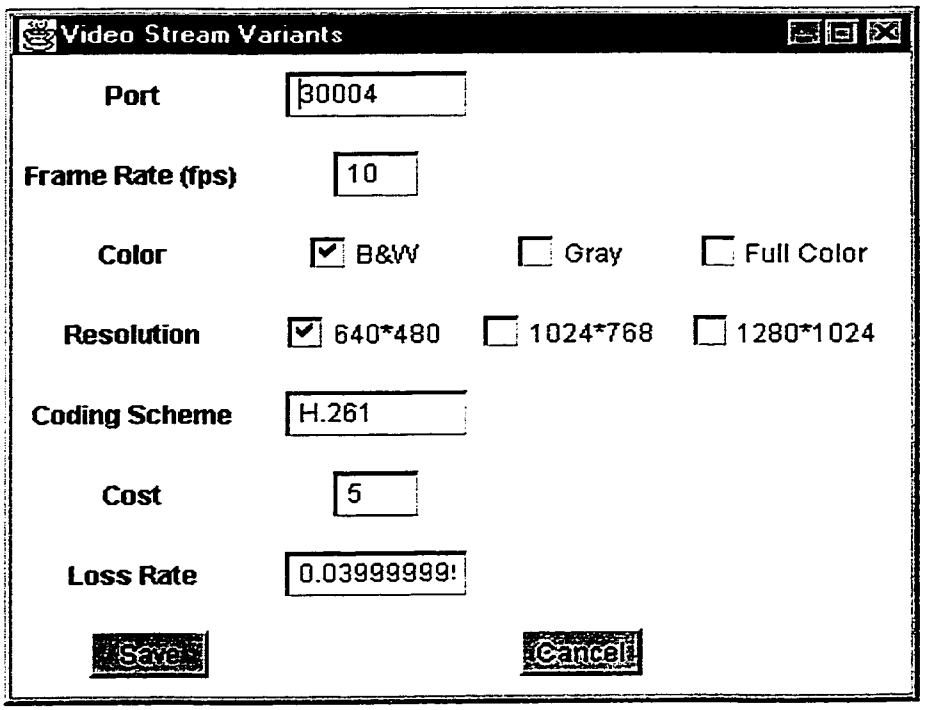

Figure 8.6 Video Variants Create and Edit

To activate or deactivate a given channel, the user needs to highlight the channel in the list, then click the button "Activate" or "Deactivate". The user may activate more than one channel at one time. Channel activation and deactivation can be done before the session begins or at any time during the session. If a channel is requested by a receiver, a message window is launched. An example is shown in Figure 8.7. It asks the user to click "No" within 20 seconds if he / she does not want to activate the channel. Otherwise, after 20 seconds, the channel will be automatically activated. The user may click "Yes" right away to activate the channel, instead of waiting for 20 seconds.

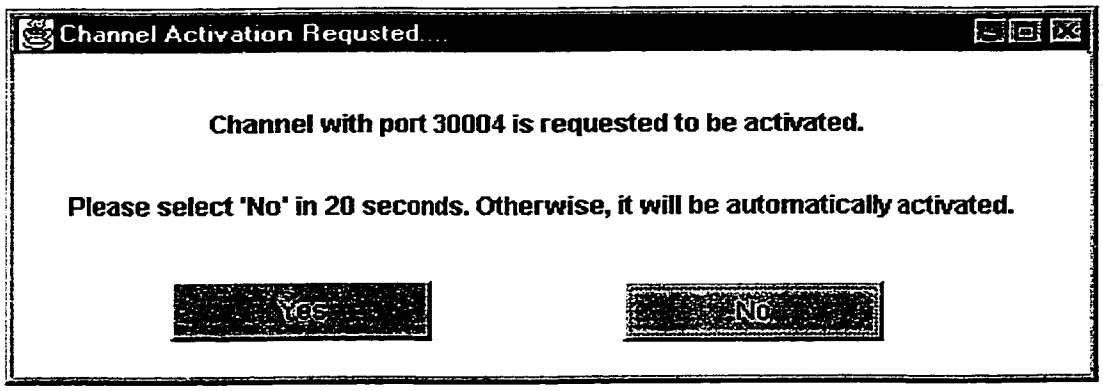

Figure 8.7 Channel Request by Receiver

When the button "Start" is clicked, the session begins. All the buttons are grayed (disabled), except for the "Activate" / "Deactivate" button and the "Stop" button. The reason of doing this is to ensure that the user is not able to create, edit or delete any channel during the session. The session has to be stopped, as the sender has to leave the multicast group explicitly, before exiting the whole program. When the "Stop" button is clicked, all buttons resume the enabled status. The user may edit the channel information and start a new session or exit the program.

- **Receiver Side User Interface**

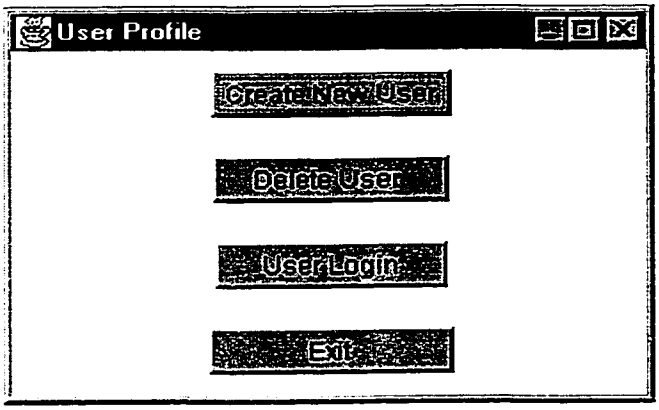

Figure 8.8 Receiver Side Main User Interface

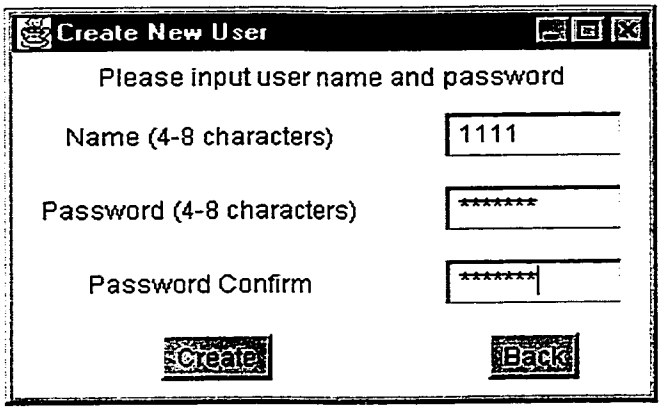

Figure 8.9 Create a New User

A user may choose to create a new account, delete an account, or login. The start menu and a user create account example are shown in Figure 8.8 and 8.9.

After the user logs in, a list of his profiles is displayed. The default profile is at the first position. The user may choose to create a new profile, edit or delete an existing profile, or join the session. To edit or delete a profile, the user should highlight the profile first.

Through the profile-editing interface, the user specifies his preferences. For each parameter, the user defines the desired value and the minimum accepted value, and a factor for the importance of improvement. Error message will be displayed in the following cases:

<1> The minimum value is bigger than the desired value.

<2> For the same parameter, more than one check box is checked, or none is checked.

<3> There is an empty text area.

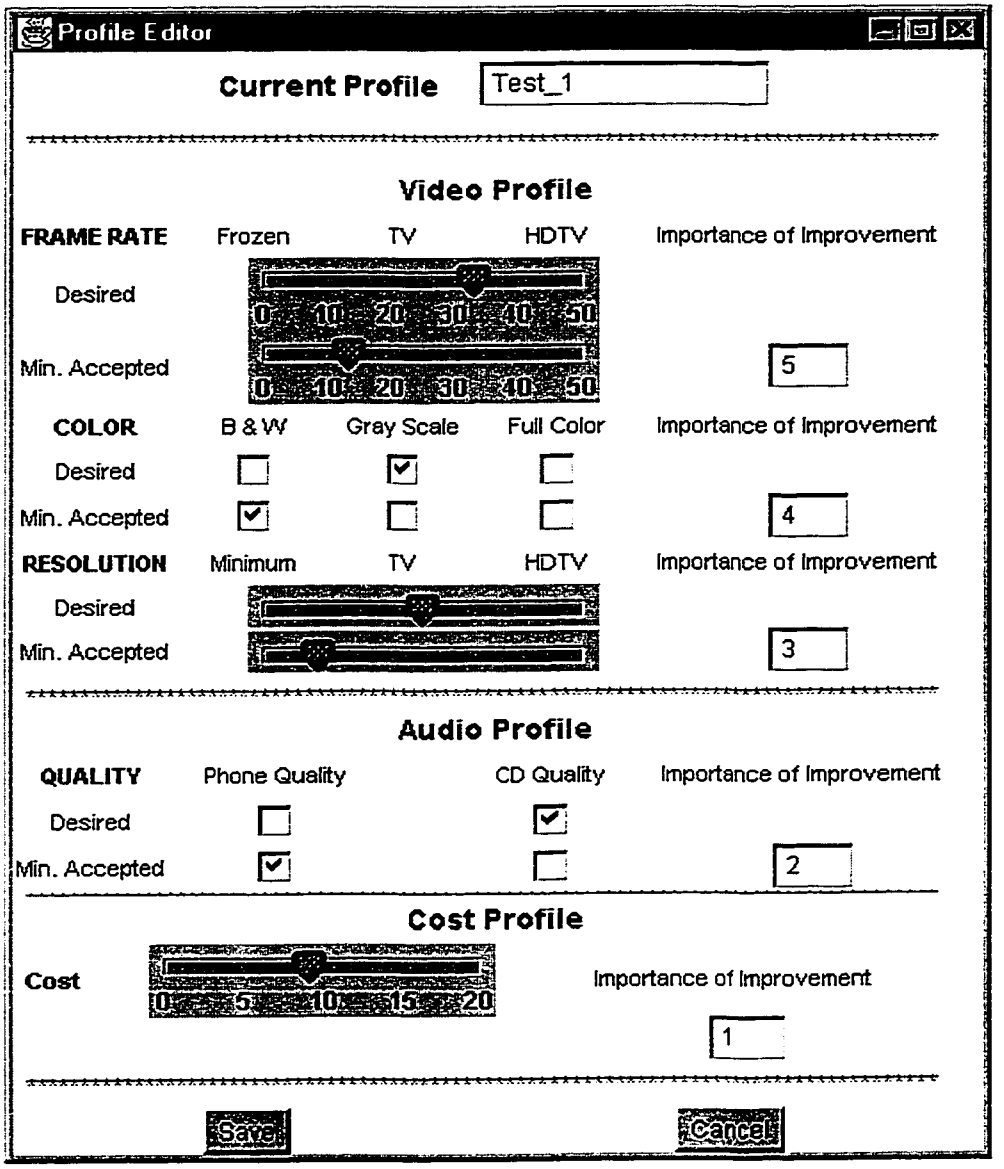

Figure 8.10 Profile Edit

The user selects a profile to join the session. This profile will be used by the QoS Agent to select the most suitable stream among the variants. The default profile is highlighted initially. The user does not have to select any profile, in which case, the default profile will be used (see Figure 8.11 below).

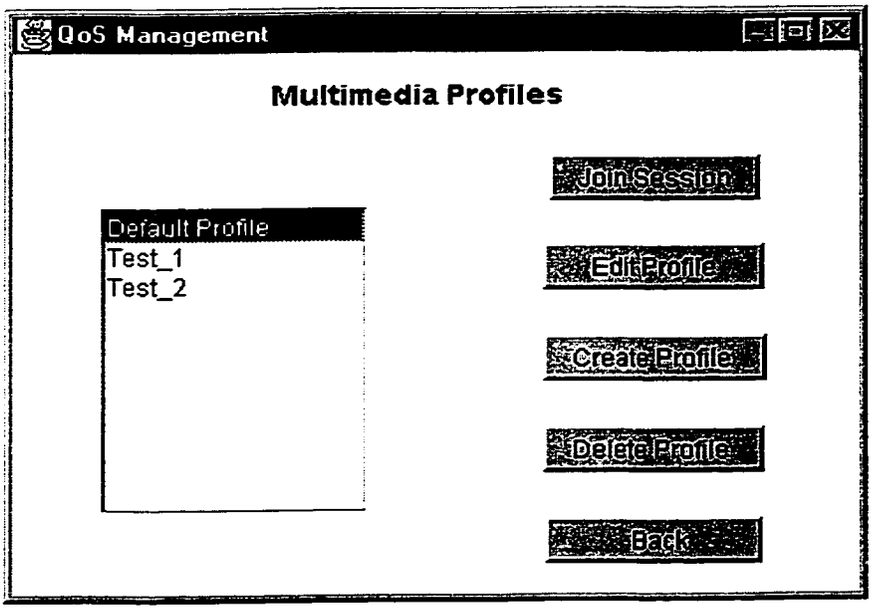

Figure 8.11 QoS Management

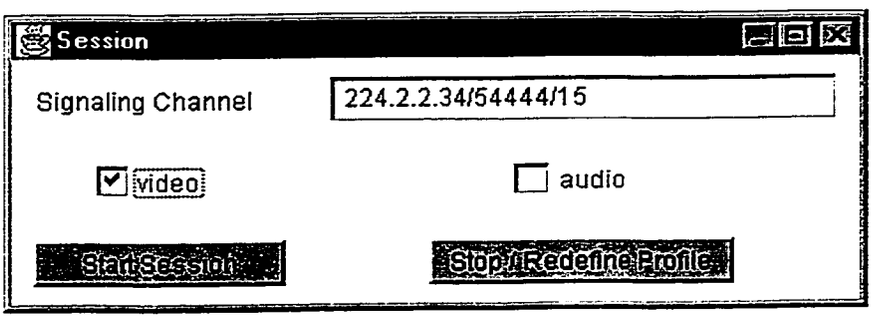

Figure 8.12 Session Initiation

The Monitoring Window (see Figure 8.13) continuously shows the updated session information. The first list shows the channel description, frame rate, color, resolution, etc. The active flag of each channel is displayed in the same line. In case of channel activation or deactivation, the flag changes immediately. Another column shows the operation status of each channel. For instance, a channel may be the currently connected channel, be requested to be activated, or refused to be activated, etc. For the current channel, the loss rate is displayed and updated every 5 seconds. This window may also be used by the user to overwrite the QoS Agent's automatic selection. The user can make his

own choice by simply double clicking the channel to be selected within the parameter list.

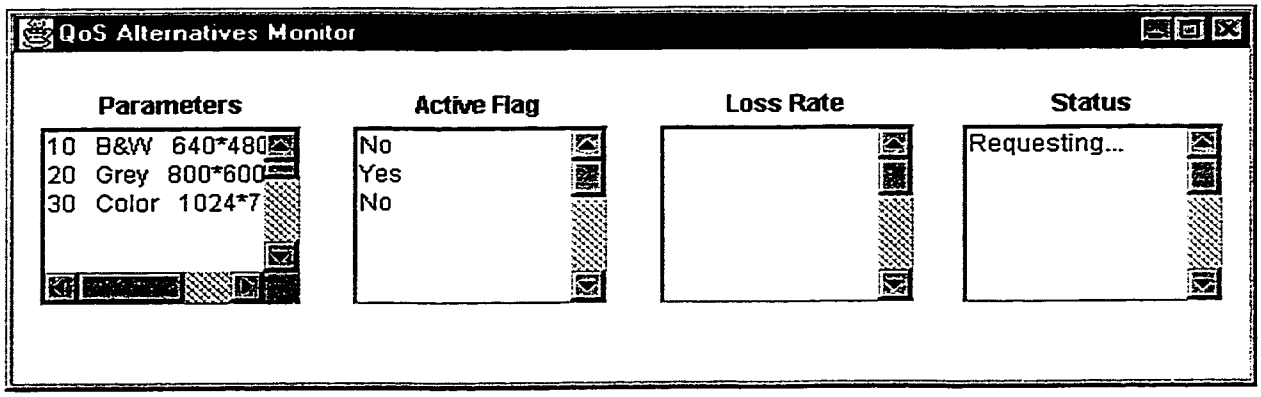

Figure 8.13 Monitoring Window

8.2.7 QoS Adaptation Algorithms

- **To select the most suitable channel at the beginning of a session**

At the beginning of a session, the QoS Manager broadcasts the channel information (*LogicalStream*) over the control channel. When a QoS Agent joins the session, it should first get this information. Then it checks for each channel whether the parameter values (frame rate, color, resolution, etc.) are higher than the minimum acceptable value from the user profile. The desired value is currently not used by the algorithm. It might be used to further optimize the selection. Two arrays are used to store all the information of acceptable channels, one is for the currently active channels, and one is used for the inactive channels. The QoS Agent scans all the channel parameters, and puts acceptable channels into the corresponding arrays. When this operation is done, the QoS Manager makes a decision according to the obtained result. The following cases may occur:

<1> Both of the arrays are still empty: The QoS Agent prompts a message, informing the user that none of the channel that the sender offers is suitable for him, and suggesting him to change his profile.

<2> The active channel array is not empty: We do not care whether the inactive channel list is empty or not. The best channel should be selected from the active channel list. The selection is based on the parameters' *importance factors*. All parameters are assigned an importance factor. The parameters are sorted according to this value, from the highest to the lowest. The parameter in the first place (with the highest importance factor) is used first for selection. The variant with the highest value of this parameter is assumed to be the most suitable one. If two variants have the same value for this parameter, the second parameter is used, and so on. Finally, the receiver system is connected to the selected channel and begins to receive the video.

<3> Active channel array is empty and inactive channel array is not empty: The most suitable potential channel is selected from the inactive channel list, and a user command is sent to the QoS Manager, asking to activate this channel. A message "Requesting....Please wait for 30 seconds" is displayed in the monitoring window at the receiver side. The QoS Agent waits for 30 seconds. If the sender activated the requested channel during this period, the video will start playing. Otherwise, a message will be displayed, suggesting the user to change the profile or wait some time to try again.

- **To select the most suitable channel when QoS violation happens**

One point needs to mention first is that we chose the loss rate as the measurement of the network QoS. Metrics that are used to measure the QoS violation are defined in the DMIF specification. In our implementation, we get the network information from the

RTCP receiver report. There are certain differences between these two specifications. A comparison is given in the following table. We decided to use only the *loss rate* (the only parameter that appears in both tables) to measure the QoS violation.

Metrics Defined in DMIF	Metrics in the RTCP Report
MAX_DELAY	DLSR (delay since last SR)
AVG_DELAY	Cumulative number of packet lost
LOSS_PROB	**Fraction lost**
Dejitter Buffer	inter-arrival jitter

Table 8.1 Two Different QoS Metrics

During a session, the loss rate is always monitored. This is done by a thread that calls the JMF APIs to get the loss rate from the RTCP receiver report every 5 seconds. The acceptable loss rate of a channel is given by the sender. If the observed value is higher than this value, a counter starts to work, and after 20 seconds, a QoS violation is announced. Since the loss rate might be fluctuating, we will not announce a QoS violation immediately when a high loss rate is detected. We consider that QoS violation occurs when several subsequent disappointing loss rates were measured. When a violation happens, the QoS Agent selects again a channel. We assume that when a violation happens, the system cannot support any stream with higher quality. The QoS Agent uses the same selection algorithm (as described under "To select the most suitable channel at the beginning of a session"), except that the selection is made among the channels whose parameter's value is lower than the current one. Again, there are three possibilities:

<1> Both of the arrays are empty: The QoS Agent prompts the following message to the user: "QoS violation happens, no channel is acceptable, please either change your profile, or wait for some time and try again."

<2> The active channel array is not empty: QoS Agent simply stops the current video and switches to the best stream in the array.

<3> Active channel array is empty and the inactive channel array is not empty: The QoS Agent selects the most suitable inactive channel, sends a user command to the sender, asking to activate that channel, and waits for 30 seconds. If the sender activates the requested channel during this period, the current video is stopped, and switched over to the new channel. Otherwise, a message will be displayed, suggesting the user to change the profile or wait some time to try again.

At the receiver side, there is a monitoring window. The user can easily see the channels offered by the sender, their parameters, which channels are active, which channel is currently used, and the loss rate of the currently used channel.

There is clearly some delay between the time when a QoS violation occurs and the time that the receiver has actually switched to another channel. This delay depends on various factors. First, it takes some time to determine that a QoS violation occurred, since the monitoring proceeds by measuring the loss rate over a fixed time period. The second delay is introduced by the QoS agent's action. If another suitable active stream available, the delay consists of the time the receiver takes to leave a multicast group and the time it takes to join another group. If the QoS agent requests the activation of an inactive channel, the QoS agent waits for a maximum of thirty seconds for the sender's decision. In the worst case, the delay consists of this thirty seconds and the time the receiver takes to leave one group and join another group. This delay appears to be too long. However, during this time, the old stream keeps playing, although with poor quality, and the user can still see the video.

8.3 Tests and Results

We have tested our prototype implementation in a simplified setting. The testing environment consisted of one camera connected to the sender machine, one receiver machine, and another camera connected to another machine at the sender side. This machine ran a background program that always gets ready to be invoked to start or stop another VIC application. Three video channels were defined in the following table:

No.	Port	Frame Rate	Color	Resolution	Coding Scheme	Loss Rate	Cost
1	30004	10	B&W	640*480	H.261	0.08	4
2	30008	20	Grey	800*600	H.261	0.05	6
3	30012	30	Color	1024*768	H.261	0.01	8

Table 8.2 Video Variants for Testing

A network analyzer was used to generate traffic in order to increase the loss rate. Since QoS adaptation was only applied for the frame rate, the user assigned the highest *importance factor* to the frame rate. At the beginning of the test, only channel 2 was activated. The following steps were applied, and the results are listed below.

No	Receiver Action	Sender Action	Rationale	Result
1	Define cost = 2		Define a cost parameter that is out of the offered range.	A message displayed at the receiver side: "Cannot find suitable channel, please change your profile."
2	Define acceptable frameRate=35		Define an unacceptable frame rate.	A message displayed at the receiver side: "Cannot find suitable channel, please change your profile."
3	Define acceptable		An inactive channel is	A message displayed at the sender side: "Channel with port

	frameRate=22		suitable for the user.	30012 is requested to be activated." In the receiver's monitor window, status of channel3 was "Requesting..."
4	Wait for 30 seconds.		Sender refuses to activate the requested channel.	A message displayed at the receiver side: "No currently active channel is suitable." In monitor window, the status of channel3 was "Refused".
5	Define acceptable frameRate=8		An active channel is suitable.	Video transmission began with port 30008. In the receiver's monitor window, the status of channel2 was "Current channel", and the loss rate begins to display.
After some time...				
6			QoS violation happens. One inactive channel with lower quality is suitable.	A message displayed at the receiver side: "QoS violation happened." In the monitor window, the status of channel1 was "Requesting...". At the sender side, a message displayed "Channel with port 30004 is requested to be activated."
7	Wait for 30 seconds.	Activate channel1 within 30 seconds.	Sender activates the requested channel	Video transmission switched to another channel. A message displayed: "Switched to another channel". In the monitor window, channel1's status was "Current Channel", and the loss rate began to display.

After some time...				
8		Deactivate channel2	Deactivate a channel.	In the receiver monitor window, the active flag of channel2 changed to "N".
9		Activate channel3	Activate a channel.	In the receiver monitor window, the active flag of channel3 changed to "Y".
10		Deactivate channel1	Deactivate the current channel.	Video transmission of this channel stopped. The message "Current channel is deactivated by the sender" was displayed. Channel3 was selected and the monitoring window showed channel3 as "Current channel". Video transmission resumed.

Table 8.3 Testing Results

Chapter 9: Conclusion and Future Work

We have introduced in this thesis an approach to implement quality of service (QoS) management for tele-teaching applications using the MPEG-4 / DMIF (Delivery Multimedia Integration Framework). Nowadays, most of the existing Internet video conferencing or tele-teaching applications do not support QoS adaptation, especially in the multicast context. Those that support dynamic QoS adaptation use a frame dropping scheme to reduce the effective throughput. Furthermore, few applications in this area consider the user's satisfaction; most of the work has been done from the network's point of view. We have chosen a different approach and also have implemented a system that provides QoS alternatives in a session and also considers the user's satisfaction. The main contribution of the work is that we have proved the feasibility of this approach that we believe can be used in other similar applications.

In this thesis, we first reviewed some background knowledge, such as the TCP/UDP/IP protocols, the MBone facilities, voice and video coding schemes, network QoS, adaptive applications, and so on. Then we discussed QoS management for multimedia applications with multicasting, which is a complex task, especially when a large number of users are involved. We introduced Fischer's approach in which media variants with different qualities are provided by the sender to adapt to different user needs. We developed our own prototype based on this approach. To implement this prototype, we specified the system requirements and also defined the general architecture. Then we discussed the protocol alternatives for both video transmission and session control, and discussed overhead issues. One of the most important issues was the selection of an appropriate

session control protocol. It appears that the DMIF standard for MPEG-4, which is presently extended for multicast applications, provides interesting session management functions for distributed multimedia applications in general, independently of the question whether MPEG-4 encoding is used. We described some popular session protocols, such as SAP/SDP and SIP, and discussed why DMIF was our final choice, how it benefited and simplified the implementation of our system.

We have shown how these DMIF functions can be used for session management of a tele-teaching application including different QoS alternatives for the participating users. In such an application, the sender node of the teacher provides different stream variants (with different QoS attributes) for each logical media stream. Each user participating as a student may then select one of these variants according to its QoS preferences. The detailed analysis of the DMIF protocol in the multicasting context has identified certain generalizations that would be useful in order to make it more generally usable for various distributed multimedia applications. This includes general means for distributing user information from the stream producers to the stream consumers, and some means for sending general user requests from a consumer to a particular producer.

We ended up with a prototype implementation of video multicasting with QoS alternatives, including the implementation of the MPEG-4 / DMIF application and network interfaces for session and QoS management. The test cases and results are described at the end.

Through the implementation, we have proved that the Cooperative QoS Adaptation approach, proposed in [51] is feasible. We believe that this work is valuable since it

provides a general architecture for the multicast multimedia applications with QoS adaptation, and this architecture is relatively easy to implement.

Providing QoS alternatives in a multicast session and therefore supporting dynamic QoS adaptation is a relatively new idea and is important in the study of this field. One of the important ongoing research issues in this field is to develop a simple and easily-implemented QoS architecture. Compared to other applications, we think that the advantage of our system is that it provides an easy implementation. We reused the DMIF protocol implementation that Mr. Darlagiannis developed in his project, and the functions provided by the Java Multimedia Framework APIs Related to the RTP protocol and video processing.

We believe that this system architecture can be used in other similar applications, for example, in a conference where a video tutorial needs to be shown from a central place, or for a video news or TV multicasting on the Internet, and so on.

However, due to time limitations and technical difficulties, some of the originally planed functions were simplified or not implemented. The first step of the future work is to complete and refine these functions. By the time the prototype was implemented, the Java Media Framework (JMF) (version 1.0) was available only for receiving media. So it was used only at the receiver side. At the sender side, we used instead an existing MBone video conferencing tool VIC, running as a separate process under the Windows NT environment. The JMF version 2.0 was released recently, which includes the video capture function. A next step would be to replace the VIC tool with the Java video capturing and sending functions, making use of the JMF 2.0 APIs. Furthermore, QoS alternatives provided by the sender are currently implemented by using different cameras.

This should be modified such that we can use a single camera and appropriate filtering functions to provide several video streams with different qualities. Another function that needs to be completed is to provide video QoS alternatives not only for frame rate, but also for color and resolution. Moreover, an attempt should be made to obtain a user interface for QoS preference selection which uses concepts readily understandable by ordinary users, without going too much into technical terms.An important feature of the future work is to integrate scalable encoding techniques into the current system. A number of scalable encoding schemes have been proposed, such as dropping a specified percentage of frames, dropping frames with lower priority, reducing the number of bits per pixel, multiplexing frames from multiple streams into one stream, etc. Some of these mechanisms are included in H.261, H.263, MPEG-1, MPEG-2 and MPEG-4. The future research will focus on how to apply certain techniques to provide the QoS adaptation from this perspective.

References

[1] RFC 791. J. Postel, "Internet Protocol", 1981.

[2] U. Black, "TCP/IP & Related Protocols (Second Edition)", 1994, McGraw-Hill Ryerson, Limited, ISBN: 0070055602

[3] RFC 793. J. Postel, "Transmission Control Protocol", 1981.

[4] RFC 768. J. Postel, "User Datagram Protocol", 1980.

[5] RFC 1112. S, Deering, "Host Extension for IP Multicasting", 1989.

[6] RFC 1700. J. Reynolds, J. Postel, "Assigned Numbers", 1994.

[7] RFC 2460. R. Hinden, S. Deering, "Internet Protocol, Version 6 (IPv6) Specification", 1998.

[8] RFC 2373. R. Hinden, S. Deering, "IPv6 Addressing Architecture", 1998.

[9] RFC 2375. R. Hinden, S. Deering, "IPv6 Multicast Address Assignment", 1998.

[10] D. R. Kosiur, "IP Multicasting", 1998, John Wiley & Sons Canada, Limited, ISBN: 0471243590.

[11] M. R. Macedonia, D. P. Brutzman, "MBone Provides Audio and Video Across the Internet"

http://www-atp.llnl.gov/atp/papers/HRM/references/mbone-av.html

[12] Frequently Asked Questions (FAQ) on the Multicast Backbone (MBONE)

http://www.cs.columbia.edu/~hgs/internet/mbone-faq.html

[13] MBone Software Archives

http://www.merit.edu/~mbone/index/titles.html

[14] RFC 2327. M. Handley, V. Jacobson, "SDP: Session Description Protocol", 1998.

[15] S. McCanne, V. Jacobson, "Vic: A Flexible Framework for Packet Video" http://www-nrg.ee.lbl.gov/vic

[16] RFC 2236. W. Fenner, "Internet Group Management Protocol, Version 2", 1997.

[17] M. Banikazemi, "IP Multicasting: Concepts, Algorithms, and Protocols" http://www.cis.ohio-state.edu/~jain/cis788-97/ip_multicast/index.htm

[18] C. Semeria, T. Maufer, "Introduction to IP Multicast Routing" http://www.3com.com/nsc/501303.html

[19] RFC 1075. D Waitzman, C. Partridge, S. Deering, "Distance Vector Multicast Routing Protocol", 1988.

[20] Mrouted manual page http://hegel.ittc.ukans.edu/topics/linux/man-pages/man8/mrouted.8.html#sect1

[21] RFC 1583. J. Moy, "OSPF Version 2", 1994.

[22] RFC 1584. J. Moy, "Multicast Extension to OSPF", 1994.

[23] RFC 2362. D. Estrin, D. Farinacci, A. Helmy, D. Thaler, S. Deering, M. Handley, V. Jacobson, C. Liu, P. Sharma, L. Wei, "Protocol Independent Multicast-Sparse Mode (PIM-SM): Protocol", 1998.

[24] V. Johnson, M. Johnson, "Higher Level Protocols Used with IP Multicast" http://www.ipmulticast.com/community/whitepapers/highprot.html

[25] RFC 1889. H. Schulzrinne, S. Casner, R. Frederick, V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", 1996.

[26] RFC 2326. H. Schulzrinne, A. Rao, R. Lanphier, "Real Time Streaming Protocol (RTSP)", 1998.

[27] RTSP Information Page

http://www.real.com/rtsp

[28] RFC 2205. R. Braden, L. Zhang, S. Berson, S. Herzog, S. Jamin, "Resource ReSerVation Protocol (RSVP) - Version 1 Functional Specification", 1997.

[29] K. Chang and C. Semeria, "Standards for Multimedia Applications on Converging Networks"

http://www.summitonline.com/netmanage/papers/3com2.html

[30] ISO/IEC 14496-1 "MPEG-4 Systems", ISO/IEC 14496-2 "MPEG-4 Visual", ISO/IEC 14496-3 "MPEG-4 Audio", ISO/IEC 14496-6 CD "DMIF"

From MPEG homepage http://drogo.cselt.it/mpeg

[31] "QoS" Definition in a Dictionary of Technology Terms on Web

http://whatis.com/qos.htm

[32] G. v. Bochmann, A. Hafid, "Some Principles for Quality of Service Management", Distributed Systems Engineering Journal 4, 1997, pages 16-27.

[33] A. Hafid, G. v. Bochmann, "Quality of Service Adaptation in Distributed Multimedia Applications", Multimedia Systems Journal (ACM), 1998, Vol. 6, No. 5, pages 299-315.

[34] D. B. Waldegg, "A temporal QoS Based CPU Scheduling Model for Multimedia Applications in General Purpose Operating Systems", Fifth IFIP International Workshop on Quality of Service, IWQOS'97 Proceedings, 1997, pages 157-160.

[35] K. Lakshman, R. Yavatkar, R. Finkel, "Integrated CPU and Network-I /O QoS Management in an End System", Fifth IFIP International Workshop on Quality of Service, IWQOS'97 Proceedings, 1997, pages 167-178.

[36] A. Hafid, "Providing a Scalable Video-on-Demand System Using Future Reservation of Resources and Multicast Communications", Fifth IFIP International Workshop on Quality of Service, IWQOS'97 Proceedings, 1997, pages 277-288.

[37] P. Ferguson, G. Huston, "Quality of Service: Delivery QoS on the Internet and in Corporate Networks", 1998, John Wiley & Sons Canada, Limited.

[38] Q. Ma, P. Steenkiste, "Quality of Service Routing for Traffic with Performance Guarantees", Fifth IFIP International Workshop on Quality of Service, IWQOS'97 Proceedings, 1997, pages 115-126.

[39] E. W. Fulp, M. Ott, D. Reininger, D. S. Reeves, "Paying for QoS: An Optimal Distributed Algorithm for Pricing Network Resources", Sixth IFIP International Workshop on Quality of Service, IWQOS'98 Proceedings, 1998, pages 75-84.

[40] R. Steinmetz, L.C. Wolf, "Quality of Service: Where are We?", Fifth IFIP International Workshop on Quality of Service, IWQOS'97 Proceedings, 1997, pages 211-222.

[41] "MPEG-2: Generic Coding of Moving Pictures and Associated Audio Information" http://drogo.cselt.stet.it/mpeg/standards/mpeg-2/mpeg-2.htm

[42] S. McCanne, V. Jacobson, M. Vetterli, "Receiver driven Layered Multicast", Proceedings of SIGCOMM'1996, http://www-nrg.ee.lbl.gov/nrg-papers.html.

[43] J. Gecsei, "Adaptation in Distributed Multimedia Systems", IEEE Multimedia, April-June 1997, pages 58-66.

[44] P. Moghe, A. Kalavade, "Terminal QoS of Adaptive Applications and Its Analytical Computation", Fifth IFIP International Workshop on Quality of Service, IWQOS'97 Proceedings, 1997, pages 369-380.

[45] D. Sisalem, "End-To-End Quality of Service Control Using Adaptive Applications", Fifth IFIP International Workshop on Quality of Service, IWQOS'97 Proceedings, 1997, pages 381-392.

[46] Jane Hunter, Varuni Witana, Mark Antoniades, "A Review of Video Streaming over the Internet".

http://archive.dstc.edu.au/RDU/staff/jane-hunter/video-streaming.html

[47] A Distributed Real-Time MPEG Video Audio Player

http://www.cse.ogi.edu/~scen/nossdav95/nossdav95.html

[48] E. Amir, S. McCanne, H. Zhang, "An Application Level Video Gateway", Proceedings of ACM Multimedia '95, November 1995, pages 255-265.

[49] SuperNOVA

http://archive.dstc.edu.au/SuperNOVA/

[50] S. Jo, P. E. Cantrell, "A Dynamic QoS Control Scheme for Videoconferencing in a Heterogeneous Internet".

http://www-itg.lbl.gov/~skjo/paper/inet99

[51] S. Fischer, A. Hafid, G. v. Bochmann, H. d. Meer, "Cooperative Quality of Service Management for Multimedia Applications", Proceedings of the 4th IEEE Internatinal Conference on Multimedia Computing and Systems, Ottawa, Canada, June 1997, pages 303-310.

[52] RFC 2508: Compressing IP/UDP/RTP Overheads

[53] RFC 2543. M. Handley, H. Schulzrinne, E. Schooler, J. Rosenberg, "SIP: Session Initiation Protocol", 1999.

[54] ISO/IEC 14496 – 6, "Information Technology – Very Low Bit-rate Audio-Visual

Coding, Part 6: Delivery Multimedia Integration Framework (DMIF)", (N2720).

From MPEG homepage: http://drogo.cselt.stet.it/mpeg

[55] Java Media Framework API

http://java.sun.com/products/java-media/jmf/index.html

[56] Java Remote Method Invocation (RMI) Interface

http://java.sun.com/products/jdk/rmi/index.html

[57] Java Object Serialization API Reference

http://java.sun.com/products/jdk/1.1/docs/guide/serialization

[58] V. Darlagiannis, "COSMOS: Collaborative System Based on MPEG-4 Objects and Streams", SITE, University of Ottawa, Masters thesis, December, 1999.

[59] C. Zaharia (University of Western Ontario), Z. Yang (University of Ottawa), "MBone Report", Oct. 1998.

ftp://beethoven.site.uottawa.ca/Publications/-REPORTS-THESES/

[60] J. W. Wong, D. Evans, N. D. Georganas, J. Brinskelle, G. Neufeld, D. Macaroff, "An MBone-based Distance Education System", Proceedings of ICCC'97, Cannes, Grance, Nov. 1997.

Appendix A: Multicast Routing Algorithms

A.1. Flooding

The flooding algorithm is the simplest multicast delivery algorithm. When a packet arrives at a router, the only thing the router checks is whether it has seen this packet before. If this packet reached here before, the router will discard the packet. If it is the first time that the router sees this packet, it will forward it to all the neighboring routers except for the one from which the packet was forwarded.

Although this algorithm is very simple, it has two major drawbacks. Routers generate a large number of duplicated packets that cause a big waste of network bandwidth. Routers have to keep track of each recently received packet to check whether a new arrived packet has been here. It is not an efficient use of the routers' memory resources.

A.2. Spanning Trees

This algorithm defines a tree structure that consists of a subset of network links. This tree spans to all nodes in the network. Consider the example in Figure A.1.

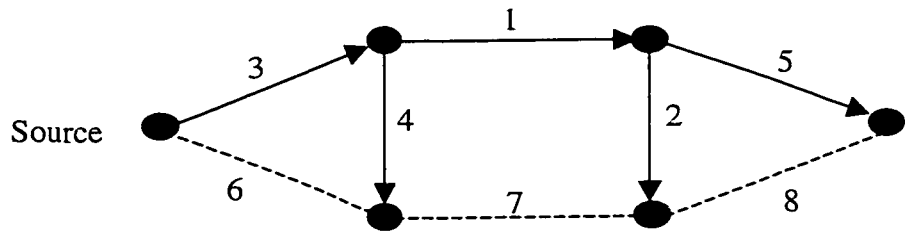

Figure A.1 Example of Spanning Tree Routing Algorithm

There are 6 nodes connected by 8 links. The cost associated with each link is labeled beside the link. A spanning tree from the source node is constructed in solid lines. When a multicast packet arrives at a router, the router will only forward the packet over the

links that are branches of the spanning tree, except for the one that the packet was from. This algorithm avoids too many duplicated packets flooding the network, and also improves the router memory usage since a router only keeps a boolean value for each interface whether it is in the spanning tree.

It still has two disadvantages: the traffic is centralized to a small set of network links, and the packets are forwarded to all routers no matter whether there are group members there.

A.3. Reverse Path Broadcasting (RPB)

The Reverse Path Broadcasting algorithm modifies the Spanning Tree algorithm by constructing a shortest path tree for each source, instead of only one tree for all the nodes in the whole network. When a router receives a packet from a link, it will see whether this link belongs to the shortest path back to the source. If it is, the router forwards the packet to all the links except for the previous one. See below the same small network structure as in Figure A.2. The tree is constructed differently.

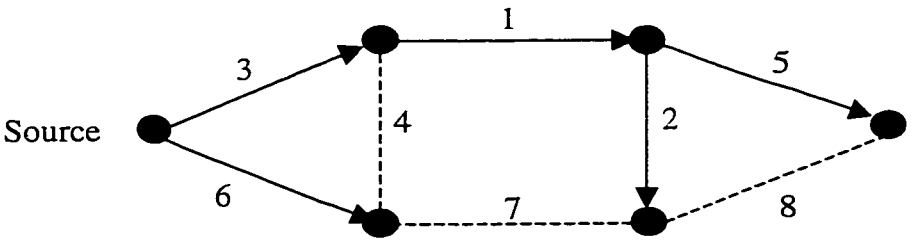

Figure A.2 Example of Reverse Path Broadcasting (RPB) Routing Algorithm

This algorithm is fast, since it uses the shortest path algorithm to build the tree, and it is easy to implement. But one problem still exists, as for Spanning Tree, it does not take into account the group membership information.

A.4. Truncated Reverse Path Broadcasting (TRPB)

As we introduced before, through IGMP, a router knows whether or not there are members of a particular group on its local network. The TRPB algorithm improves the RPB by introducing a truncation mechanism, which removes the leaf routers that do not have group members on their networks. However, only the leaf routers can be removed from the tree. This algorithm cannot prevent unnecessary traffic forwarding to non-leaf routers that do not have group members.

A.5. Reverse Path Multicasting (RPM)

The RPM algorithm was proposed to overcome some of the limitations of RPB and TRPB. Consider the example below:

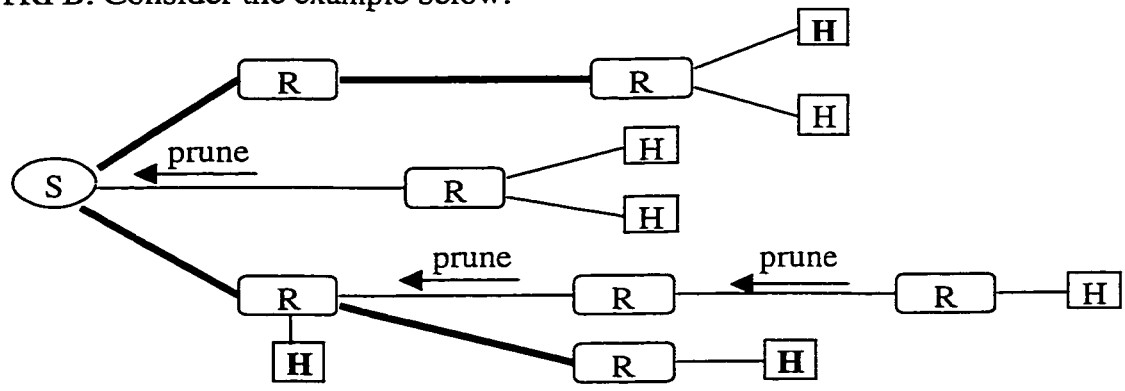

Figure 1.13 Example of Reverse Path Multicasting (RPM) Routing Algorithm

For each (source, destination group) pair, the first packet is forwarded based on the TRPB algorithm. If a router receives a packet and it does not have group members on its network, it will send a "prune" message one hop back towards the source. The upstream router needs to keep this "prune" message in its memory. In turn, if the upstream router receives a "prune" message, and it does not have recipients on its network, it will send a "prune" message back to its upstream router. The pruning continues until the source is

reached. Once the tree is constructed, the packets are sent along this tree. The tree is refreshed periodically by another packet flooding to the whole network.

Since each router has to keep the state of all the source destination pairs, this algorithm requires a big memory space on the router, so that it is not very scalable.

A.6 Core Based Tree (CBT)

The CBT builds a single tree for each group, regardless of where the sources are. A single router or a small set of routers is selected to be the "core" routers. A multicast packet is forwarded towards a core router by unicast until it reaches a router that belongs to the tree. Then the packet is forwarded over all links in the tree.

In this algorithm, the routers do not have to maintain much state information. But there are some other problems that may happen. Since there is only a single tree for each group, the traffic is centralized and there might be a bottleneck around the core routers with the result of delay in delivery.

Appendix B: Format of DAI Primitive Parameters

- **QoS Manager Side**

DA_ServiceAttachReq(String url, LogicalStream uuData)

This primitive can be issued by either a QoS Manager or a QoS Agent from the application layer to DMIF layer. The *url* has the following format: "*xdmif://<user><password>@<ip><port>/<service-entity-path>/<service-entity-role>*". The address of the session's signaling channel is defined by *<ip>* and *<port>*. The value of *<service-entity-role>* is used to identify the application role, either a QoS Manager or a QoS Agent.

The QoS Manager issues this primitive to the sender DMIF entity to explicitly join the multicast session. The potential stream information is encoded in *uuData*, in the format of an object (*AllStream*). On receiving this primitive, the DMIF entity should broadcast the *uuData* over the signaling channel through the DMIF network Layer.

DA_ServiceAttachCnf(long ssID, int response, int uuData)

This primitive is issued by the DMIF entity to respond to the *DA_ServiceAttachReq* primitive.

In the case of unicast, this primitive is initiated from the other end of the connection to report the connection status. In the multicast case, since the sender does not have knowledge of where the receivers are, the primitive is emulated by the local DMIF layer. The local DMIF randomly generates a session ID, which identifies the session uniquely such that the QoS Manager can refer to it in subsequent interactions. The *response* and *uuData* could be *null*.

DA_UserCommandInd(long ssID, int uuData)

This primitive is an indication of a channel activation request for a given session, which is identified by *ssID*. It is sent from the receiver side and issued by the sender DMIF layer to the QoS Manager, providing the *uuData*, which in our application contains the requested channel port number.

DA_ChannelAddReq(long ssID, long dir, QoSDescriptor qosInfo, int uuData)

This primitive is issued by the QoS Manager to announce the activation of a data channel in the context of a particular session identified by *ssID*. In our case, the QoS Manager provides information about the new activated data channel in the *uuData* parameter. The *qosInfo* expresses the network level QoS parameter of the channel. Since we have already included it in the logical stream information, this parameter could be *null* in this case. *dir* indicates the direction of the transmission. It has the value *UPSTREAM* when this primitive is sent from the receiver side to the sender side, and *DOWNSTREAM* in the reverse direction.

DA_ChannelAddCnf(long ssID, long chID, int response, int uuData)

Similar to *DA_ServiceAttachCnf*, this primitive is not sent from the other end, but from the local DMIF layer to emulate a *response* and *uuData*. These two parameters could be *null*. The DMIF also randomly generates a *chID*, which identifies the channel to which the QoS manger may refer to in subsequent interaction.

DA_ChannelDeleteReq(long chID, int reason)

This primitive is issued by the QoS Manager to announce the deactivation of a data channel, which is identified by *chID*.

DA_ChannelDeleteCnf(long chID, int response)

This primitive is sent by the local DMIF to emulate the channel deactivation response.

DA_SessionDetachReq(long ssID, int reason)

This primitive is issued by a QoS Manager, informing the local DMIF the session's termination. The DMIF program module clears its variables associated with this session.

DA_SessionDetachCnf(long ssID, int response)

The local DMIF emulates the confirmation of the service detach.

- **QoS Agent Side**

DA_ServiceAttachReq(String url, int uuData)

As mentioned before, this primitive can be issued by either QoS Manager or QoS Agent. The application role is indicated in the *url*. When issued by a QoS Agent, it is used for the receiver DMIF to initialize a session, getting the source information of the session, in our case, the potential stream variants from the manager. The format of the source data is described in Section 5.2.2. The multicast session control channel address is identified by the *url*. The *uuData* could be *null*.

DA_ServiceAttachCnf(long ssID, int response, LogicalStream uuData)

The local DMIF layer emulates the response to a service attach. It randomly generates a *ssID*, which identifies the session such that the QoS Agent can refer to it in subsequent interactions, and provides the received source data to the QoS Agent in *uuData*.

DA_ChannelAddInd(long ssID, long chID, long dir, QoSDescriptor qosInfo, int uuData)

This primitive is issued by the receiver's DMIF layer to inform the QoS Agent that a channel is activated in the context of a particular session identified by *ssID*. In our case, the information of the channel is provided to the QoS Agent in *uuData*. The local DMIF randomly generates a *chID*, which identifies the channel uniquely such that the QoS Agent can refer to it in subsequent interactions. *dir* and *qosInfo* have the same meaning as in the QoS Manager's *DA_ChannelAddReq* primitive. On receiving this primitive, the QoS Agent should update its local session stream variant information.

DA_ChannelAddRsp(long ssID, long chID, int response, int uuData)

This primitive can be used to respond to the *DA_ChannelAddInd* primitive, or to inform the DMIF to add a given channel (e.g. connect to a MBone channel). *chID* is used to differentiate these two cases. In the first case, the local DMIF emulates the response, carrying 0 as *chID*. In the second case, the *chID* is the actual ID of the channel that is to be connected to.

DA_ChannelMonitor(long chID, QoSMode mode)

This primitive is used to inform the DMIF to start monitoring the channel, identified by *chID*. The *mode* specifies the monitoring mode. There are three options defined in DMIF specification. Integer values are assigned to these options. We use the QoS violation indication option. The *QoSMonitorPeriod* is defined as 0, because we do not use that option. The *status* is "0". The monitoring is automatically stopped when another channel becomes the current channel or the session is ended.

```
public class QoSMode {
        int mode;
        int qosMonitorPeriod;
```

int status;

}

Mode	QosMonitorPeriod	status
NO_QOS_NOTIFICATION (0)	Frequency of QoS reports in milliseconds (0)	START (0)
PERIODIC_QOS_NOTIFICATION (1)		STOP (1)
QOSVIOL_DETECT_NOTIFICATION (2)		SINGLE (2)

Table B.1 QoS Mode

DA_EventHandle(long chID, QoSMode mode, QoSReport report)

This primitive is issued by the DMIF layer when a QoS violation is detected for a given data channel identified by *chID*. Detailed information is encoded in the *report*.

The structure of the *QoSReport* is shown below. The *qos_QualifierCount* is the count of how many metrics are used, in our case, only one (the *loss rate*). The *qos_QualifierTag* is the name of the metric. We assign an associated integer 0 to represent "loss rate". The *qos_QualityfierData* is the measured loss rate.

public class **QoSReport** {

int qos_QualifierCount;

int qos_QualifierTag[ ];

long qos_QualifierData[ ];

}

DA_UserCommandReq(long ssID, int uuData)

This primitive is intended to support the delivery of control information (*uuData*) to the remote peer. In our case, it is issued by a QoS Agent to request a channel activation. The channel's IP address and port number can be provided in the *uuData*.

DA_ChannelDeleteInd(long chID, int reason)

This primitive is issued by the DMIF entity to inform the QoS Agent about a channel deactivation. On receiving of this primitive, the QoS Agent should update the local stream variant information.

DA_ChannelDeleteRsp(long chID, int response)

The local DMIF emulates the response.

DA_ServiceDetachReq(long ssID, int reason)

This primitive is used by the QoS Agent to leave a session. In multicast applications, the receivers can leave a session at any time; this primitive does not have any effect on the sender side. The QoS Agent program module clears all the variables, and sends this primitive to the local DMIF. The DMIF program module clears all its variables as well.

DA_ServiceDetachCnf(long ssID, int response)

The local DMIF emulates the confirmation of a service detach.

Appendix C: User Profile Management APIs

boolean createUser (String usrName, String pwd)

Given the user name and password, create a new user. If the user name was already used by another user, a boolean value "false" is returned. Otherwise, "true" is returned.

boolean deleteUser (String usrName, String pwd)

Given the user name and password, delete the user. If either the user name or the password does not exist, a boolean value "false" is returned. Otherwise, "true" is returned.

int loginUser (String usrName, String pwd)

Given the user name and the password, check whether a user already has an account on this server. If user name and password are correct, return 0, otherwise, return –1. A user has to login correctly first to be able to edit his profile and join the session.

Profile getDefaultProfile (String usrName, String pwd)

For security reason, the user needs to give his name and password again to get his profiles. Given the user name and the password, this method returns the user's default profile. The default profile is always at the first position in the user's profile array.

Profile getCertainProfile (String usrName, String pwd, String profileName)

Given the user name, the password, and the profile name, return the required profile.

addNewProfile (String usrName, String pwd, Profile pro)

Given the user name, the password, and the newly created profile, put this profile into the user's profile array.

deleteProfile (String usrName, String pwd, String profileName)

Given the user name, the password, and the profile name, deleted this profile from the user's profile array.

saveProfile (String usrName, String password, String oldProfileName, Profile pro)

Given the user name, the password, the old profile name and the new profile, replace the old profile with the new one.

Vector getProfileList (String usrName, String pwd)

Given the user's name and the password, return a list of all profile names.

Note: The class Profile has the following definition:

```
public class Profile{
        public String profileName;
        public int frameRateD, colorD, resolutionXD, resolutionYD, audioD;
                /* desired values
        public int frameRateA, colorA, resolutionXA, resolutionYA, audioA;
                /* acceptable values
        public int cost;
        public int frameRatePri, resolutionPri, colorPri, audioPri, costPri;
                /* importance factor
}
```