NAVAL POSTGRADUATE SCHOOL
MONTEREY, CALIFORNIA

THESIS

VIRTUAL REALITY TRANSFER PROTOCOL (VRTP): IMPLEMENTING A MONITOR APPLICATION FOR THE REAL-TIME TRANSPORT PROTOCOL (RTP) USING THE JAVA MEDIA FRAMEWORK (JMF)

by

Francisco Afonso

September 1999

Thesis Advisor:	Don Brutzman
Second Reader:	Don McGregor

Approved for public release; distribution is unlimited.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC

1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** **Error! Reference**	**3. REPORT TYPE AND DATES COVERED** Master's Thesis

4. TITLE AND SUBTITLE
Virtual Reality Transfer Protocol (vrtp): Implementing a Monitor Application for the Real-time Transport Protocol using the Java Media Framework (JMF)

5. FUNDING NUMBERS

6. AUTHOR
Error! Reference source not found.

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT

The Real-time Transport Protocol (RTP) supports the transmission of time-based media, such as audio and video, over wide-area networks (WANs), by adding synchronization and quality-of-service (QoS) feedback capabilities to the existing transport protocol. RTP has been widely used in the Multicast Backbone (MBone), a virtual network that has become a shared worldwide medium for Internet multicast communications.

This work presents the design patterns, architecture and implementation of an RTP monitor application using the Java Media Framework (JMF), a new Java Application Programming Interface (API) for multimedia support. An RTP monitor is an application that receives packets from all participants in a multicast session in order to estimate the quality of service for distribution monitoring, fault diagnosis and both short and long-term statistics.

This new RTP monitor is available as a component of the Virtual Reality Transfer Protocol (vrtp), a protocol being developed to support large-scale virtual environments (LSVEs) over the Internet. Initial test results are satisfactory for audio and video streams, as well as prototype RTP-compliant Distributed Interactive Simulation (DIS) protocol streams. Future work includes automated monitoring across WANs and standardizing structured data formats to comply with Management Information Base (MIB) requirements using Extensible Markup Language (XML) target set definitions.

14. SUBJECT TERMS
Multicasting, Real-time Transport Protocol, Virtual Reality, Java, Multimedia, RTP, vrtp

15. NUMBER OF PAGES
232

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	**18. SECURITY CLASSIFICATION OF THIS PAGE**	**19. SECURITY CLASSIFICATION OF ABSTRACT**	**20. LIMITATION OF ABSTRACT**
Unclassified	Unclassified	Unclassified	UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18, 298-102

Approved for public release; distribution is unlimited.

VIRTUAL REALITY TRANSFER PROTOCOL (VRTP): IMPLEMENTING A MONITOR APPLICATION FOR THE REAL-TIME TRANSPORT PROTOCOL (RTP) USING THE JAVA MEDIA FRAMEWORK (JMF)

Francisco Carlos Afonso
Lieutenant Commander, Brazilian Navy
B.S.E.E., University of Sao Paulo, 1988

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
September 1999

Author: _____
Francisco Carlos Afonso

Approved by: _____
Don Brutzman, Thesis Advisor

_____
Don McGregor, Second Reader

_____
Dan C. Boger, Chair
Department of Computer Science

ABSTRACT

The Real-time Transport Protocol (RTP) supports the transmission of time-based media, such as audio and video, over wide-area networks (WANs), by adding synchronization and quality-of- service (QoS) feedback capabilities to the existing transport protocol. RTP has been widely used in the Multicast Backbone (MBone), a virtual network that has become a shared worldwide medium for Internet multicast communications.

This work presents the design patterns, architecture and implementation of an RTP monitor application using the Java Media Framework (JMF), a new Java Application Programming Interface (API) for multimedia support. An RTP monitor is an application that receives packets from all participants in a multicast session in order to estimate the quality of service for distribution monitoring, fault diagnosis and both short and long-term statistics.

This new RTP monitor is available as a component of the Virtual Reality Transfer Protocol (vrtp), a protocol being developed to support large-scale virtual environments (LSVEs) over the Internet. Initial test results are satisfactory for audio and video streams, as well as prototype RTP-compliant Distributed Interactive Simulation (DIS) protocol streams. Future work includes automated monitoring across WANs and standardizing structured data formats to comply with Management Information Base (MIB) requirements using Extensible Markup Language (XML) target set definitions.

TABLE OF CONTENTS

I. INTRODUCTION 1

A. BACKGROUND 1

B. MOTIVATION 1

C. OBJECTIVES 2

D. THESIS ORGANIZATION 3

II. RELATED WORK 5

A. INTRODUCTION 5

B. RTP MONITORING IN MEDIA CONFERENCE APPLICATIONS 5

1. Video Conferencing Tool (VIC) 5
2. Robust Audio Tool (RAT) 7

C. DEDICATED MONITORS 8

1. Mtrace 8
2. Session Directory (SDR) Monitor 8
3. Rtpmon 9
4. MultiMON 11
5. MHealth 12

D. VRTP 13

E. INTERNET 2 SURVEYOR 15

F. DESIGN PATTERNS 15

III. REAL-TIME TRANSPORT PROTOCOL (RTP) 17

A. INTRODUCTION 17

B. OVERVIEW OF TRANSPORT RELATIONSHIPS 17

C. RTP 18

1. RTP Units: Mixer, Translator and Monitor 18
2. RTP Header 19
3. Profiles and RTP Header Extension 20

D. RTP SESSION ADDRESSING 21

E. RTP CONTROL PROTOCOL (RTCP) 22

1. Sender Report (SR) 24
2. Receiver Report (RR) 27
3. Source Description (SDES) 29
4. Goodbye (BYE) 32

F. MULTIMEDIA IN RTP 33

G. ANALYSIS OF SR AND RR REPORTS 33

H. RTP PROFILES AND PAYLOAD FORMAT SPECIFICATIONS 34

1. Profile Specification Documents 34
2. Payload Format Specification Documents 34

I. SUMMARY 35

IV. JAVA MEDIA FRAMEWORK (JMF) 37

A. INTRODUCTION 37

B. OVERVIEW 37

C. JMF ARCHITECTURE 38

D. RTP SESSION MANAGER API 42

1. RTP Streams 42
2. RTP Participants 43
3. RTCP Source Description 43
4. RTCP Report 44
5. Event Listeners 46
6. RTP Media Locator and RTP Session Address 47
7. RTP Session Manager 49
8. Receiving and Presenting RTP Media Streams 50
9. Transmitting RTP Streams 51
10. RTP Statistics 53

D. SUMMARY 54

V. DESIGN AND IMPLEMENTATION OF THE RTPMONITOR APPLICATION 55

A. INTRODUCTION 55

B. RTPMONITOR FUNCTIONALITY AND INTERFACE 55

1. Graphical User Interface (GUI) 55
2. Statistics Display 56
3. Statistics Recording 57
4. Media Presentation 57
5. Command Line Operation 57

C. RTPMONITOR CLASS DESIGN 58

1. RtpMonitorManager and RtpUtil 58
2. RecordTask and FileCatalog 60
3. RtpPlayerWindow 61
4. RtpMonitor 63
5. RtpMonitor Applet 63
6. RtpMonitorCommandLine 63

D. SUMMARY 64

VI. RTP MANAGEMENT INFORMATION BASE (MIB) 65

A. INTRODUCTION 65

B. NETWORK MANAGEMENT OVERVIEW 65

C. RTP MIB DESCRIPTION 66

D. COMPATIBILITY WITH JMF STATISTICS 66

E. SUMMARY 67

VII. EXPERIMENTAL RESULTS 69

A. INTRODUCTION 69

B. TEST RESULTS 69

C. OBSERVED PROBLEMS 70

D. EXTENDING DIS-JAVA-VRML PDU HEADER 70

E. SUMMARY 71

VIII. CONCLUSIONS AND RECOMMENDATIONS 73

A. RESEARCH CONCLUSIONS 73

B. RECOMMENDATIONS FOR FUTURE WORK 73

1. Participants Information 73
2. Extensible Markup Language (XML) Recording 74
3. Recorded Data Analysis and Presentation 74
4. Session Description Protocol (SDP) Reception 74
5. RtpMonitor Activation from SDR 74
6. JMF Extensibility for Other Media 75
7. Automated Network Monitoring of RTP Streams for VRTP. 75
8. Design Patterns Course in Computer Science Curriculum 75

APPENDIX A. PREPARING UML DIAGRAMS USING RATIONAL ROSE 77

APPENDIX B. RTPMONITOR USER MANUAL 79

APPENDIX C. RTPMONITOR JAVADOC 91

APPENDIX D. RTPMONITOR SOURCE CODE 137

APPENDIX E. COMPARISON RTP MIB VERSUS JMF STATISTICS..... 191

APPENDIX F. RTPHEADER JAVADOC 192

APPENDIX G. RTPHEADER SOURCE CODE 207

LIST OF REFERENCES 213

INITIAL DISTRIBUTION LIST 217

ACKNOWLEDGEMENTS

I would like to thank my wife Helena for the continuous support and patience regarding my academic work at NPS.

To Professor Brutzman, I offer my thanks for the friendship and enthusiasm I have experienced during this time.

I. INTRODUCTION

A. BACKGROUND

The Multicast Backbone (MBone) is a virtual network that has been in operation since 1992. It was initially used to multicast audio and video from meetings of the Internet Engineering Task Force, but nowadays it has become a shared worldwide medium with many diverse channels for Internet multicast communications (Macedonia, et al., 94).

The Real-time Transport Protocol (RTP) was developed to support time-based media, such as audio and video, over multicast-capable networks (Schulzrinne, et al., 99). RTP is the basic packet header format for MBone application streams. Using RTP, a multicast session between several participants can be established, making possible the correct synchronization of the exchanged media and the feedback of each participant about the quality of reception. The RTP Control Protocol (RTCP) performs the feedback and control mechanisms of RTP.

Java Media Framework (JMF) is a new Java Application Programming Interface (API) developed by Sun and other companies to allow Java programmers use multimedia features in applications and applets (Sun, 99). JMF supports RTP transmission and reception of audio and video streams. This thesis examines network-monitoring issues relevant to RTP through implementation and testing of a JMF application.

B. MOTIVATION

Since its inception, RTP has been mostly used in audio and video conferences. However, a diverse set of multicast applications can take benefit of RTP mechanisms for

synchronization, such as exchanging simulation data over a wide-area network (WAN). RTP is particular important because it is used by backbone routers to support Quality of Service (QoS) related performance optimization.

The Virtual Reality Transfer Protocol (vrtp) is developed to provide client, server, multicast streaming and network-monitoring capabilities in support of internetworked 3D graphics and large-scale virtual environments (LSVEs) (Brutzman, 99). RTP is an integral part of the vrtp architecture, used in both the streaming and monitoring components.

JMF is a possible solution for the implementation of the RTP protocol in vrtp. JMF is a free package and is an approved extension of the Java language application programming interface (API).

C. OBJECTIVES

The goal of this thesis is the implementation of a monitor application for the Real-time Transport Protocol (RTP) using Java Media Framework (JMF). An RTP monitor is an application that receives packets sent by all participants in order to estimate the quality of service for distribution monitoring, data recording, statistics analysis and fault diagnosis (Schulzrinne, et al., 99). The emphasis is in both short-term and long-term statistics by having recording capabilities for future analysis. As a result of this work, an example set of classes for monitoring RTP sessions can be provided to vrtp applications.

D. THESIS ORGANIZATION

The remaining chapters of this thesis are organized as follows. Chapter II describes work related to RTP monitoring. Chapter III presents the RTP protocol functionality and packet formats. Chapter IV provides an overview of the JMF architecture with emphasis on the RTP classes. Chapter V describes the design and implementation of the RTP monitor application, and the interdependency between the RTP Monitor and JMF classes. Chapter VI contains a study of compatibility between JMF statistics and the proposed RTP Management Information Base (MIB) (Baugher, et al., 99). Chapter VII contains the experimental results achieved and problems observed. Finally, Chapter VIII presents conclusions and provides recommendations for future work.

THIS PAGE LEFT INTENTIONALLY BLANK

II. RELATED WORK

A. INTRODUCTION

This chapter presents some related work in the area of RTP monitoring. RTP monitoring is closely related to the RTCP protocol, but some applications discussed here combine RTCP data with multicast route tracing, improving the monitoring quality. Applications are considered in into two categories: media conference applications with monitoring features, and dedicated monitors. This chapter also contains pertinent information about the vrtp protocol and the Internet 2 Surveyor project.

B. RTP MONITORING IN MEDIA CONFERENCE APPLICATIONS

Most conference applications used in MBone can display some sort of RTP monitoring data. As the emphasis of these applications is media presentation and stream quality assessment, the statistical data is usually not enough for network administration or diagnostic purposes. The RTP statistics features of some conference applications are described here.

1. Video Conferencing Tool (VIC)

VIC is a video conferencing application developed by the Network Research Group at the Lawrence Berkeley National Laboratory (LBNL) in collaboration with the University of California, Berkeley (UCB). The software was improved by University College of London (UCL). The latest version is 2.8ucl4. (UCL, 99)

The RTP statistics window of each VIC video stream consists of a grid with three columns (Figure 2.1). The first column, *EWA* (Exponentially Weighted Average), is

the change since the last sampling period (i.e. change over the last second); the middle column, *Delta*, is a smoothed version of the EWA; and the last column, *Total*, is the accumulated value since start-up. Clicking any of the buttons in the left column opens a *Plot Window* displaying the statistics for that parameter graphically. (UCL, 99)

Figure 2.1 contains example RTP statistics for video VIC streams.

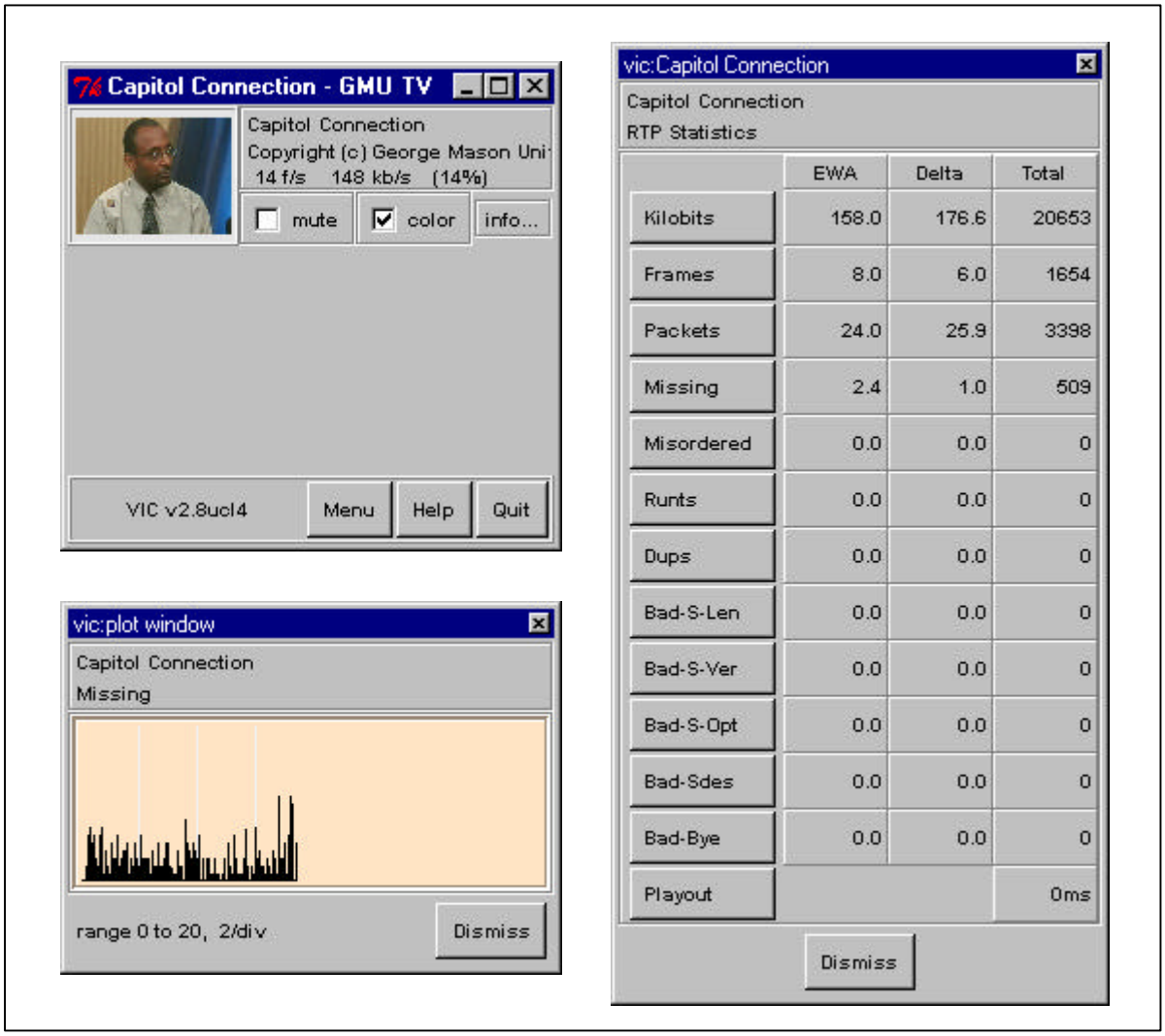


Figure 2.1 RTP Statistics in VIC showing video stream, available statistics (right side) and a 10-second trace of missing-packet data (lower-left corner). Recorded July 1999.

2. Robust Audio Tool (RAT)

RAT is an audio conference application developed by UCL. The latest version is 4.0.3 (UCL, 99). Figure 2.2 shows an example of RTP statistics available in RAT. Each set of statistics is related to one receiving audio stream.

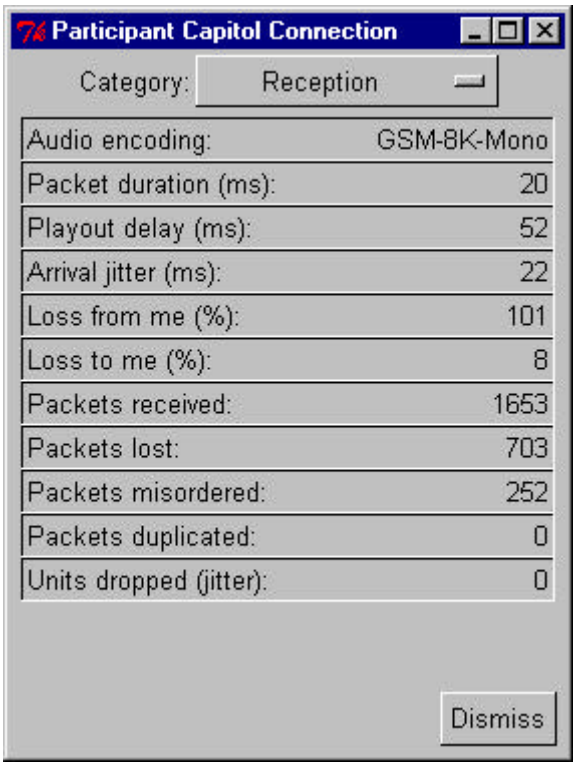


Figure 2.2 RTP Statistics in RAT.

C. DEDICATED MONITORS

1. Mtrace

Mtrace allows the user trace a route from a receiver to a source working backwards using a particular multicast address. It runs only on Unix systems. If either the receiver or the source is not participating in a multicast on the specified address then mtrace may not work. Vic and vat can automatically launch the mtrace utility. Mtrace is included in the mrouted distribution and can be downloaded at the following site:

http://www.cs.unc.edu/~wangx/MBONE/mbonetoolarchive.html#mtrace

2. Session Directory (SDR) Monitor

SDR Monitor, short for SDR Global Session Monitoring Effort, is an effort to track, manage, and present information about the availability of world-wide multicast sessions using the SDR tool (Sarac, 99). SDR is a session directory tool designed to allow the advertisement and joining of multicast conferences on the MBone (UCL, 99).

The basic idea of the SDR Monitor is that the SDR application can periodically send an an email to the project control containing all the session announcements that are being received at the user site. All data collected is made available in world-wide-web page, using a tabulated form that conveys information about the session reachability.

3. Rtpmon

Rtpmon is a third-party RTP monitor written in C++ by the Plateau Multimedia Research Group at Berkeley. Version 1.0 was the last release in 1996. It works only on Unix systems. Both source code and binaries are available. It is possible to integrate rtpmon with vic and/or vat by modifying the files .vic.tcl and/or .vat.tcl. This will add a monitor button to vic and vat that when pressed will launch rtpmon. (Agarwal, 97)

Rtpmon provides a number of useful capabilities for sorting, filtering and displaying the statistics generated in an RTP session (David, 96). Figure 2.3 shows an example of the rtpmon Graphical User Interface (GUI). It displays RTCP statistics in a table with senders listed along the top and listeners along the left. Each entry in the table corresponds to the data obtained about a single sender-receiver pair. The program listens to RTCP messages on a multicast address specified at startup time; there is no provision for displaying data from multiple sessions. Information about each participant can be obtained by clicking the participant's name in the table display; doing so brings up a window with the session description items provided in the member's RTCP reports. The information window also contains a button, which spawns mtrace for detailed single-sender-receiver multicast routing data. Rtpmon can also display a brief history of the statistics from each sender-receiver pair. Clicking on a data element in the main table brings up charts for each of the statistics values that rtpmon tracks.

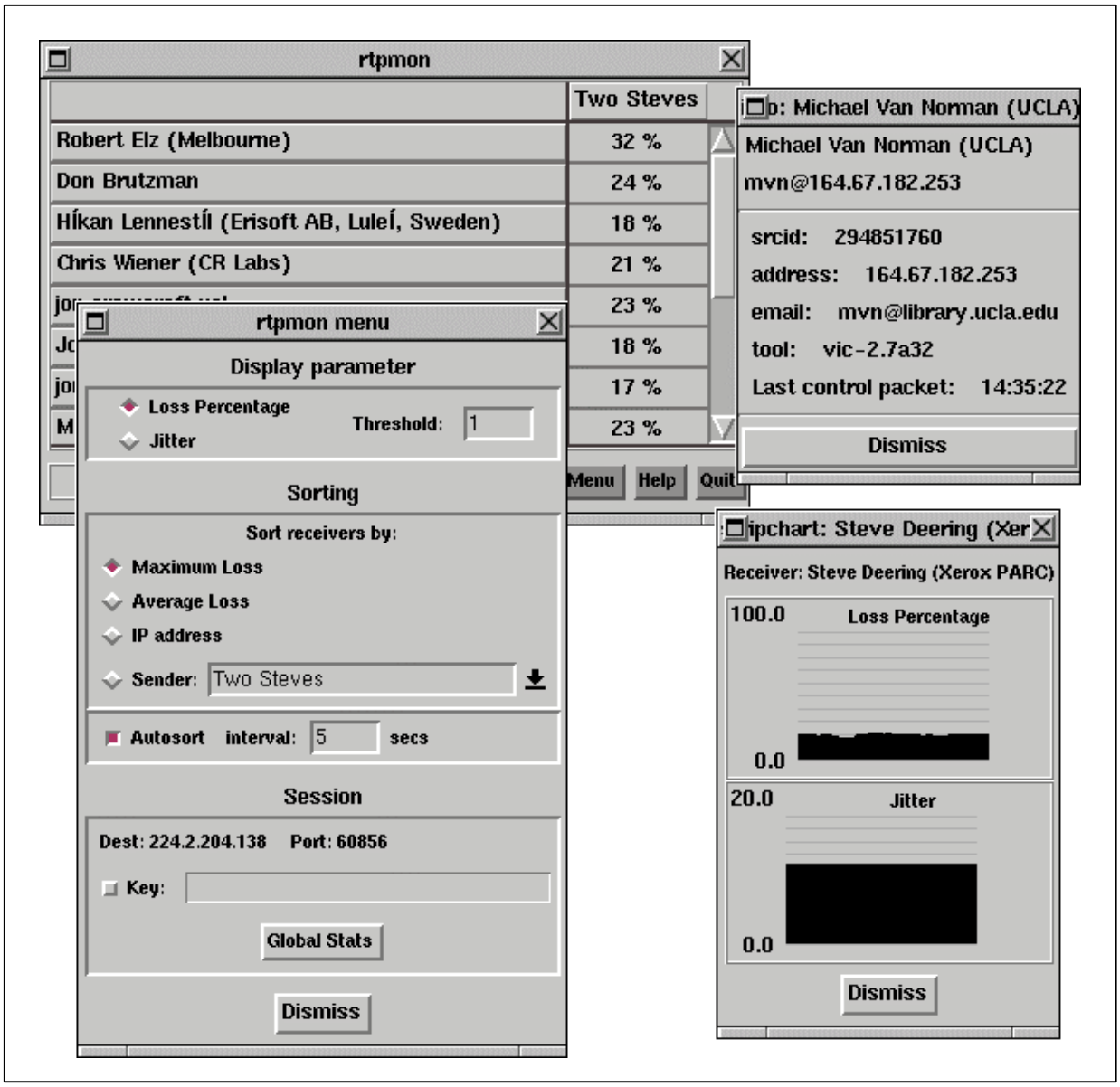


Figure 2.3 Rtpmon graphical user interface (GUI).

4. MultiMON

MultiMON is a client/server monitor that collects, organizes and displays all the IP multicast traffic that is detected at the location of a server (Robinson, et al, 96). It was developed by the Communications Research Center at Ottawa. Last release was version 2.0 in July 1998. It can be downloaded at ftp://debra.dgbt.doc.ca/pub/mbone/multimon/.

MultiMON is a general-purpose multicast monitoring tool. It is intended to monitor multicast traffic on local-area network (LAN) segments and assist a network administrator in managing the traffic on an Intranet. The client main window displays the total bandwidth occupied by the multicast traffic and gives a graphical breakdown of the traffic by application type, as shown in Figure 2.4.

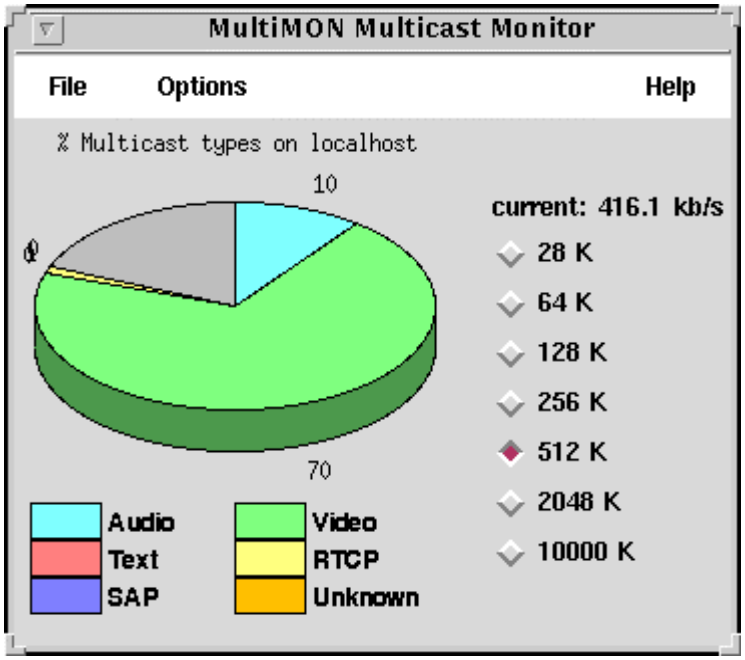


Figure 2.4 MultiMON client window (Robinson, et al, 96).

The software includes an RTCP monitoring and recording tool (MERCInari) that allows an analysis of the RTCP data for QoS management. A session can be recorded for later analysis. MultiMON is written in tk/tcl, but needs tcpdump, xplot, the distributed processing additions and the object-oriented additions to tk/tcl. It runs in Sun workstations, but it can be ported to other platforms including Windows 95/98 and NT.

5. MHealth

MHealth, the Multicast Health Monitor, is a graphical multicast monitoring tool (Makofske, et al., 99). By using a combination of application-level protocol data for participant information and a multicast route-tracing tool for topology information, MHealth is able to present a multicast tree's topology and information about the quality of received data. Figure 2.5 contains a screenshot of MHealth.

MHealth also provides data-logging functionality for the purpose of isolating and analyzing network faults. Logs can be analyzed to provide information such as receiver lists over time, route histories and changes, as well as the location, duration, and frequency of packet loss (Makofske, et al., 99). MHealth was written in Java but needs mtrace for its operation. Version 1.0 can be downloaded at: http://steamboat.cs.ucsb.edu/mhealth/download-v1.0/.

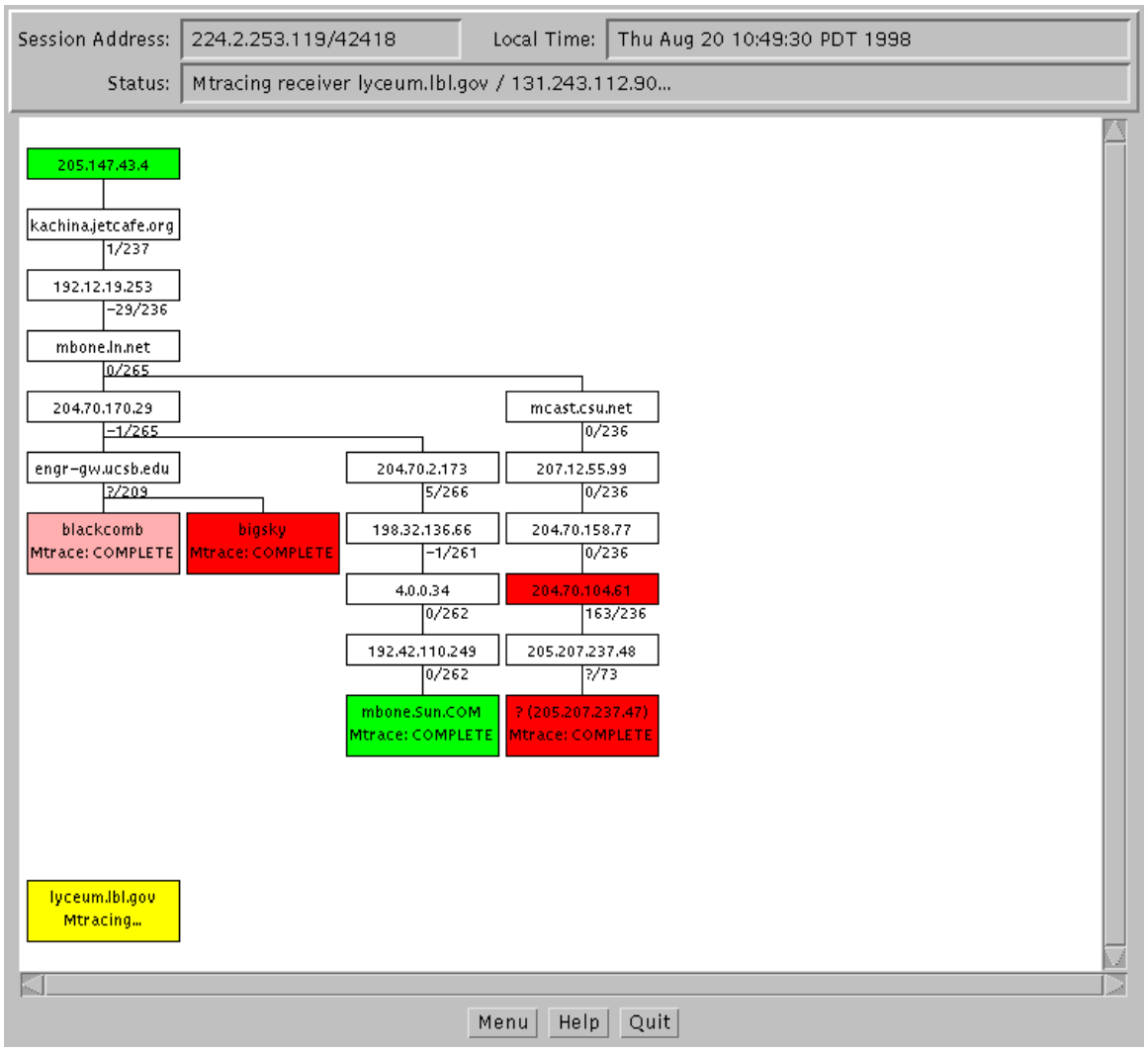


Figure 2.5 MHealth display showing participants in a multicast tree (Robinson, et al., 96).

D. VRTP

The Virtual Reality Transfer Protocol (vrtp) is a protocol being developed to provide client, server, multicast streaming and network-monitoring capabilities in support of internetworked 3D graphics and large-scale virtual environments (LSVEs) (Brutzman, 99). vrtp is designed to support interlinked VRML worlds in the same manner as http was designed to support interlinked HTML pages. The intent is to develop a free library

to provide any machine with client, server, peer-to-peer and network monitoring capabilities to navigate and join large, interactive, fully internetworked 3D worlds.

RTP is an integral part of the vrtp architecture, used in both the streaming and monitoring components. Figure 2.6 shows the functional design of the vrtp streaming behaviors component.

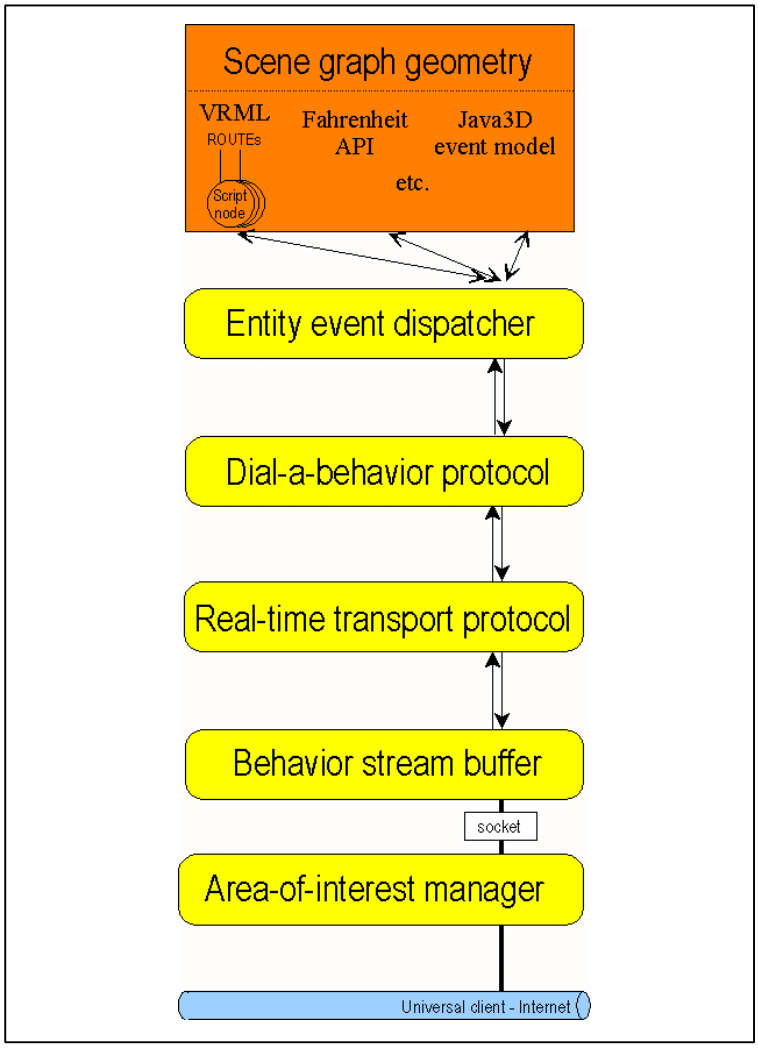


Figure 2.6 vrtp streaming behaviors component (Brutzman,99).

E. INTERNET 2 SURVEYOR

The university community has joined together with government and industry partners to accelerate the next stage of Internet development. The Internet2 project is bringing focus, energy and resources to the development of a new family of advanced applications to meet emerging academic requirements in research, teaching and learning (Advanced Networks, 97).

Surveyor is a measurement infrastructure that is being currently deployed at participating sites around the world. Based on standards work being done in the Internet Engineering Task Force (IETF), Surveyor measures the performance of the Internet paths among partcipating organizations. The project is also developing methodologies and tools to analyze the performance data. The project aims to create the infrastructure and tools that will improve the ability to understand and effectively engineer the Internet infrastructure (Advanced Networks, 97). Surveyor monitoring workstations installed at NPS are expected to provide a controlled network environment supporting RtpMonitor work.

F. DESIGN PATTERNS

Design Patterns are reusable solutions to recurring problems that occur during software development. In 1995, the book "Design Patterns" (Gamma, 95) has started popularizing the idea of patterns. In recent years, new patterns have been proposed and several books about patterns have been released. Frequently, books about Design Patterns give examples of pattern implementations in some computer language. A good book for studying patterns in Java was written by Mark Grand (Grand, 98). Sun Microsystems' Java Application Programming Interfaces (APIs) contain plenty of examples of pattern

use. Design patterns provided great benefit during the software analysis, design and implementation work in this thesis.

III. REAL-TIME TRANSPORT PROTOCOL (RTP)

A. INTRODUCTION

This chapter presents the Real-time Transport Protocol (RTP) functionality and packet format. RTP is a header format and control protocol designed to support applications transmitting real-time data (such as audio, video, or simulation data), over multicast or unicast network services. RTP was proposed by RFC1889 (Schulzrinne, et al., 99) and has received wide acceptance.

B. OVERVIEW OF TRANSPORT RELATIONSHIPS

The transport layer provides a flow of data between hosts. In the TCP/IP protocol suite there are two vastly different transport protocols: the Transmission Control Protocol (TCP) and the User Datagram Protocol (UDP). While TCP provides a reliable flow of data between two hosts, by using packet acknowledgements and retransmissions, UDP just sends single packets with no guarantee that the packets will be received at the other side.

RTP does not provide all the functionality required by a transport protocol. It is intended to run over some transport protocol, such as UDP, primarily in multicast mode. TCP is not suitable for real-time audio/video transfers because packet loss implies in retransmissions that affect delay-sensitive data. Also TCP does not support multicasting. Despite the implication of its name, RTP does not provide any means to ensure timely delivery or to guarantee a desired quality of service. In fact, RTP was designed to satisfy the needs of multi-participant multimedia conferences, where the loss of some packets or some small delay will not affect the session significantly.

Basically, the RTP header provides a sequence number and a timestamp in each packet, allowing the timing reconstruction of the receiving stream. Additionally, RTP header specifies a payload type, allowing different data formats to originate from different senders in a single session. Further parameters in the RTP packet header are defined in a media-specific manner.

RTP is used together with the RTP Control Protocol (RTCP), which provides mechanisms of synchronization, source identification and quality-of-service feedback.

C. RTP

This section summarizes RTP, which is formally described in RFC1889 (Schulzrinne, et al., 99).

1. RTP Units: Mixer, Translator and Monitor

A Mixer is an intermediate system that receives streams of RTP data packets from one or more sources, possibly changes the data format, combines the streams in some manner and then forwards the combined stream. A mixer sends its data just as if it were a new source.

A Translator is an intermediate system that forwards packets without changing its source description. Translators can be used as devices to convert encodings, such as replicators from multicast to unicast, and such as application filters in firewalls.

A Monitor is an application that receives RTCP packets sent by all participants to estimate the quality of service for distribution monitoring, fault diagnosis and long-term

statistics. The monitor is likely to be built into the applications participating in a session, but it may also be a separate application that does not send or receive RTP data packets such as third-party monitors.

RTP is designed to connect several end-systems in single or multiple sessions. An end-system sends and/or receives RTP data packets. In addition, RTP supports the notion of Translators and Mixers, which can be considered intermediate systems at the RTP level. The need for these systems has been established by experiments with multicast and video applications, especially for dealing with low-bandwidth connections and firewalls.

2. RTP Header

A RTP header precedes each RTP packet. Figure 3.1 represents an RTP header.

0	4	8	9 16	31
V	PX	CC	M PT	sequence number
timestamp				
synchronization source(SSRC) identifier				
contributing source (CSRC) identifiers				

Figure 3.1 RTP Header Contents (Schulzrinne, et al., 99).

The following information is part of the header:

- Version (V): 2 bits - Identifies the version of RTP. The current version defined in RFC1889 is two (2).
- Padding (P): 1 bit - If the padding bit is set, the packet contains one or more additional padding bytes. The last byte contains the number of padding bytes, including itself. The RTP header has no packet length field.

- Extension (X): 1 bit - If this bit is set the header will be followed by a header extension.

- CSRC count (CC): 4 bits - Contains the number of contributing source identifiers in the header.

- Marker (M): 1 bit – Can be used by a profile.

- Payload type (PT): 7 bits - Identifies the type of data (payload) carried by the packet. The payload types for standard audio and video encodings are defined in RFC1890 (Schulzrinne, 99).

- Sequence number: 16 bits - It is incremented each time a packet is sent. The initial value is randomly generated.

- Timestamp: 32 bits - This field reflects the sampling instant of the first byte of data. The clock frequency depends of the data type. For example, if audio data is being sampled at 8KHz, and each audio packet has 20 ms of samples (160 sample values), the timestamp should be increased by 160 each 20 ms. The timestamp initial value should be random.

- SSRC: 32 bits - This field identifies the synchronization source. It is a number chosen randomly. It must be unique among all participants in a session. There is a mechanism to solve conflicts when two sources choose the same number, described in Session 8 of RFC1889. A participant need not use the same SSRC identifier in all sessions of a multimedia session.

- CSRC list: 0 to 15 items, 32 bits each - The CSRC list identifies the contributing sources for the payload contained in this packet. It is used only by mixers, which combine several streams from different sources in a single stream.

3. Profiles and RTP Header Extension

The RTP protocol is designed to be malleable and to allow modifications and additions as defined by the corresponding media profile specification. Typically each application only operates under one profile. For example, RFC1890 (Schulzrinne, et al., 99) defines a profile for audio and video conferences.

In the RTP fixed header, the marker bit (M) and the payload type (PT) fields carry profile-specific information. If the marker bit is set, a header extension will be added to the RTP fixed header to provide the additional data functionality required for the profile.

The header extension has the following format seen in Figure 3.2.

0	16 31
defined by the profile	Length (# of 32-bit words)
header extension data	

Figure 3.2 RTP Header Extension (Schulzrinne, et al., 99).

The length field defines the number of 32-bit words of the header extension, excluding the first one. Therefore, zero is a possible value. Only a single variable-length header extension can be appended to the RTP header.

D. RTP SESSION ADDRESSING

A session is an association among participants. It is defined by two transport parameter pairs (two network addresses plus corresponding port numbers). One transport address is used for transmitting RTP packets and the other is used for RTCP packets. The destination transport address pair may be the same for all participants when using multicast, or may be different for each when using unicast. For multicast UDP, the multicast address for RTP and RTCP in a session must be the same, the RTP port must be even, and the RTCP port must be the next higher (odd) port number. Multiple RTP sessions are distinguished by different port number pairs and/or different multicast addresses.

Some applications like audio and videoconferences require the use of two sessions: one for audio and the other for video. Distinct media types should not be carried in a single session, because of the different timing and bandwidth requirements.

Figure 3.3 shows three different multicast conferences configurations. Part a) is a single media conference, represented by a single session. Part b) and c) represent two possible configurations for a conference with two types of media, e.g. audio and video.

E. RTP CONTROL PROTOCOL (RTCP)

RTCP provides back-channel monitoring and synchronization for use with RTP streams. RTCP provides a periodic transmission of control packets to all participants in the session. RTCP performs the following functions:

- Feedback on the quality of the data distribution. Receivers must periodically send RTCP packets containing a set of information related to the quality of each sender transmission, such as the number of lost packets, fraction of lost packets, delay times and jitter (variations in delay).

- Provides information about the participants in the session. Each participant must periodically send their canonical name (CNAME), e-mail address, telephone number and so on. The canonical name must be unique among the participants of a multimedia session (a group of sessions). The CNAME is sent together with the SSRC identifier, allowing the detection of any collision in choosing the SSRC identifier in a session.

- Media synchronization. RTCP conveys information about the absolute time (wallclock) for each participant. If the senders are synchronized, it will be possible to synchronize the different media in a session.

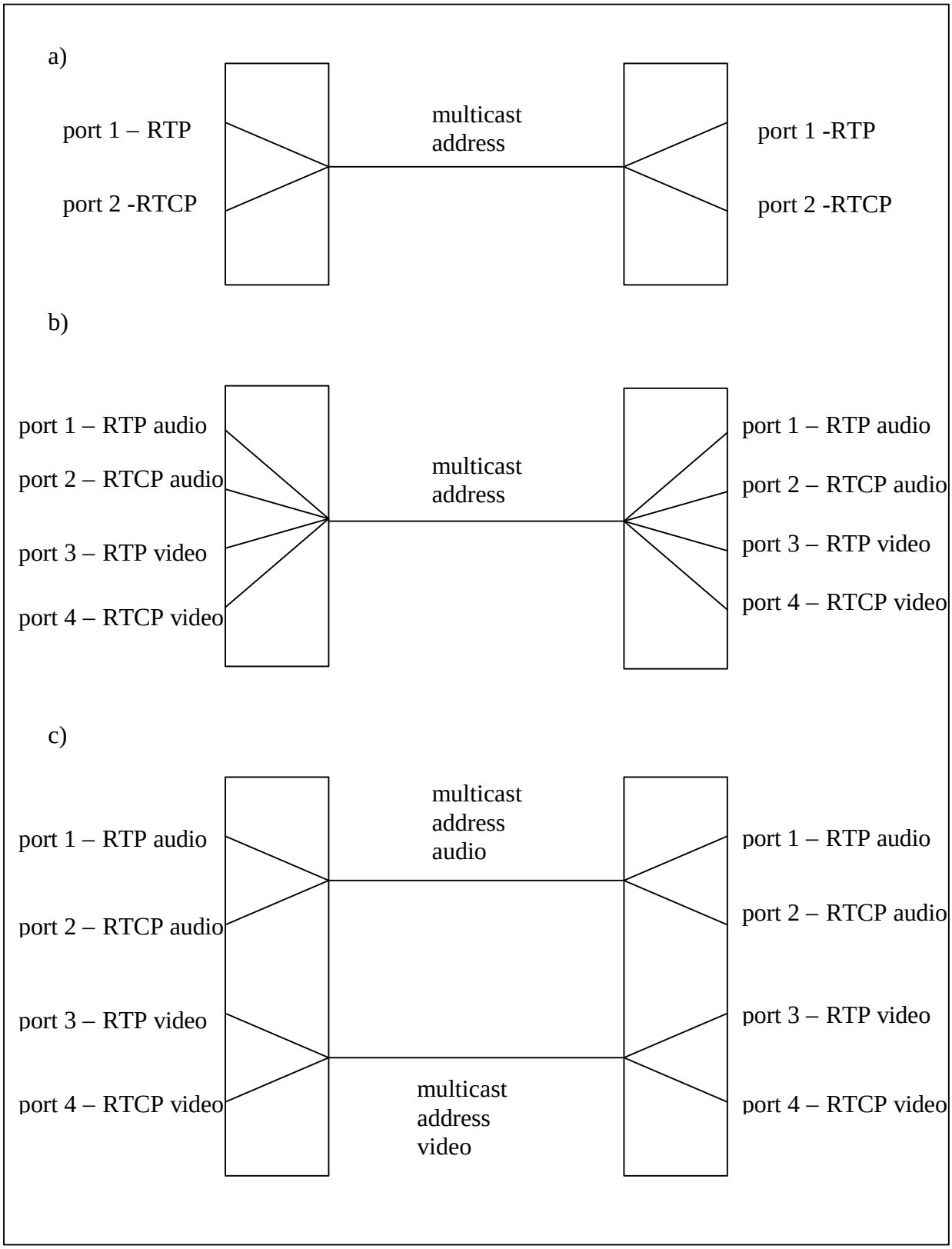


Figure 3.3 RTP Addressing Configuration Examples.

As all participants must send RTCP packets, the rate must be controlled to avoid excessive RTCP traffic as the number of participants scales up. This, each participant must control his RTCP rate to guarantee that RTCP packets correspond to less than 5% of the RTP session. Also, it is recommended that at least ¼ of the RTCP bandwidth would be dedicated to senders. When the proportion of senders is greater than ¼ of the participants, the sender gets its proportion from the full RTCP bandwidth. It is further recommended that the interval between RTCP transmissions by each participant should always be greater than 5 seconds.

RFC1889 defines five types of RTCP packets:

- SR: Sender Report, for transmission and reception of statistics from participants that are active senders.
- RR: Receiver Report, for reception statistics from participants that are not active senders.
- SDES: Source Description**,** for each participant transfer information about himself.
- BYE: to notify the end of participation.
- APP: application specific functions.

Multiple RTCP packets need to be concatenated without any separator to form a compound RTCP packet. Each participant must send RTCP compound packets with at least one SR or RR and one SDES with his CNAME.

1. Sender Report (SR)

A Sender Report is issued if the participant has sent any RTP data packets since the last report, otherwise a Receiver Report (RR) is issued. A sender report, besides having statistical information about itself, may contain reception statistics of a maximum

of 31 other senders. If a sender has more than 31 receiver statistics to send then, it must add a RR to the compound packet. The format of the Sender Report is shown in Figure 3.4.

The Sender Report packet consists of three sections: a header section, a sender information section, and a receiver information section. A fourth profile-specific extension section can be defined. The fields have the following meaning:

- Version (V): 2 bits - Identifies the version of RTP. The version defined in RFC1889 is two (2).

0	3	8	16	31
V P	RC	PT = 200 (SR)	length	
SSRC of sender				
NTP timestamp, most significant word				
NTP timestamp, least significant word				
RTP timestamp				
sender’s packet count				
sender’s byte count				
SSRR_1 (SSRC of the first source)				
fraction loss	cumulative number of packets loss			
extended highest sequence number received				
interarrival jitter				
last SR (LSR)				
delay since last SR (DLSR)				
SSRR_2 (SSRC of the first source)				
...				
profile-specific extensions				

Figure 3.4 RTCP Sender Report (SR) Format (Schulzrinne, et al., 99).

- Padding (P): 1 bit - If padding bit is set, the packet contains one or more additional padding bytes. The last byte contains the number of padding bytes, including itself. The padding bytes are included in the length field. Only the last packet of a compound packet can use padding.

- Reception report count (RC): 5 bits - The number of reception report blocks contained in this packet.

- Packet type (PT): 8 bits - It is set to 200 to identify a SR packet.

- Length: 16 bits - The length of the SR packet in 32-bit words minus one, including the header and any padding.

- SSRC: 32 bits - The synchronization source identifier for this originator of this SR packet.

- NTP timestamp: 64 bits - Indicates the wallclock time when this report is sent. The wallclock represents the absolute day and time using the timestamp format of the Network Time Protocol (NTP), described in RFC1305. The full resolution NTP timestamp is a 64 bit unsigned number with 32 bits for the integer part and 32 bits for the fractional part. It represents the number of seconds relative to 0h UTC on 1 January 1900.

- RTP timestamp: 32 bits - Corresponds to the same time as the NTP timestamp above, but in the same units and with the same random offset as the RTP timestamp in data packets. This correspondence may be used for intra-media and inter-media synchronization for sources whose NTP timestamps are synchronized.

- Sender's packet count: 32 bits - The total number of packets transmitted by the sender since starting transmission until the time the SR packet was generated. The count should be reset if the sender changes its SSRC identifier.

- Sender's byte count: 32 bits - The total number of payload octets (i.e. not including header or padding) transmitted in the RTP data by the sender since starting the transmission until this SR packet was generated. The count should be reset if the sender changes its SSRC identifier.

- SSRC_n: 32 bits - The SSRC identifier of the source to which the information in this block pertains.

- Fraction lost: 8 bits - The fraction of RTP data packets from source SSRC_n lost since the previous SR or RR packet. It is represented as a fixed-point number with the binary point at the left edge of the field.

- Cumulative number of packets lost: 24 bits - The total number of RTP data packets from source SSRC_n that have being lost since the beginning of the reception.

- Extended highest sequence number received: 32 bits - The low 16 bits contain the highest sequence number received in an RTP data packet from source SSRC_n. The most significant 16 bits extend that sequence number with the corresponding count of sequence number cycles.

- Interarrival jitter: 32 bits - An estimate of the statistical variance of the RTP data packet interarrival time, measured in timestamp units and expressed as an unsigned integer.

- Last SR timestamp (LSR): 32 bits - The middle 32 bits out of 64 bits in the NTP timestamp received as part of the most recent RTCP SR packet from the source SSCR_n.

- Delay since last SR (DLSR): 32 bits - The delay, expressed in units of 1/65536 seconds, between receiving the last SR packet from source SSRC_n and sending this reception report block. Based on this information and the LSR the sender can compute the round trip propagation delay to this receiver. This can be done by the following formula: Round Trip = A- LSR- DLSR, where A is the time the receiver gets the RR message. Figure 3.5 is a time diagram that represents the DLSR.

2. Receiver Report (RR)

The format of the Receiver Report (RR) is the same of the Sender Report (SR) except that the packet type field contains the value 201 and that there is no sender information section.

An empty RR packet (RC=0) must be put at the head of a compound packet when there is no data transmission or reception to report. Figure 3-6 shows the Receiver Report packet format. Figure 3.6 represents a Receiver Report packet format.

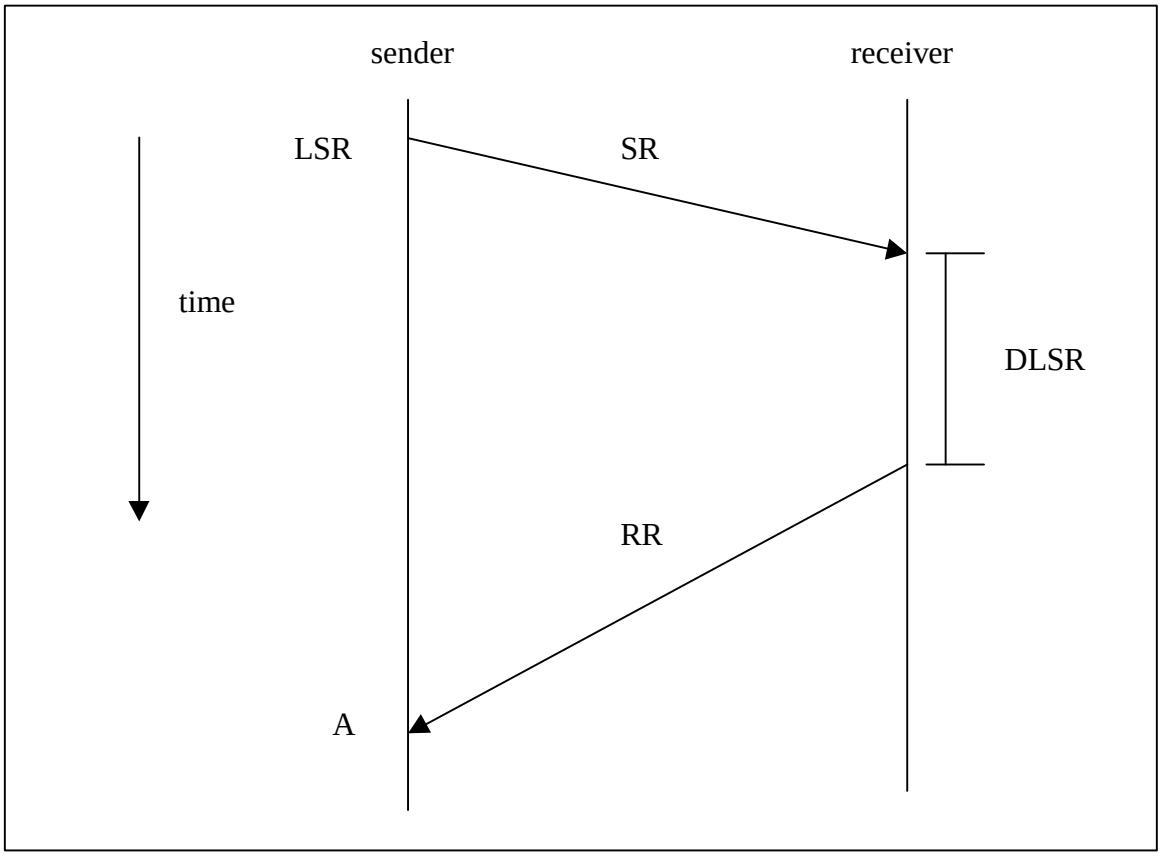


Figure 3.5 Delay Since Last SR (DLSR) Time Diagram.

0	3	8	16	31
V P	RC	PT = 201 (RR)	length	
SSRC of sender				
SSRR_1 (SSRC of the first source)				
Fraction loss	cumulative number of packets loss			
Extended highest sequence number received				
interarrival jitter				
last SR (LSR)				
delay since last SR (DLSR)				
SSRR_2 (SSRC of the first source)				
...				
profile-specific extensions				

Figure 3.6 Receiver Report (RR) Format (Schulzrinne, et al., 99).

3. Source Description (SDES)

The SDES packet consists of a header section and zero or more chunks of data. Each chunk is composed of items describing the source identified in that chunk. The SDES packet format is shown in Figure 3.7.

0	3	8	16 31
V P	SC	PT = 202 (SDES)	length
SSRR_1 /CSRC_1			
SDES items			
SSRR_2 /CSRC_2			
SDES items			
...			

Figure 3.7 Source Description (SDES) Packet Format (Schulzrinne, et al., 99).

The fields in the SDES packet are:

- Version (V), Padding (P) and length - As described for SR and RR packets.
- Packet type (PT): 8 bits - It is set to 202 to identify a SDES packet.
- Source count (SC): 5 bits - The number of SSRC/CRCS chunks contained in this packet.

The SDES item format is shown in Figure 3.8.

0	8	16 n
type	length	content

Figure 3.8 SDES Item Format (Schulzrinne, et al., 99).

The fields in each SDES item are:

- Type: 8 bits - Contains a value to identify a type of description. The following values are defined in RFC1889:
  Canonical Name – 1
  Name – 2
  Email –3
  Phone – 4
  Location – 5
  Tool – 6
  Note – 7
  Private Extensions – 8

- Length: 8 bits - The length of the chunk content, in bytes.

- Content - Consists of text encoded in UTF-8 encoding specified in RFC2279. This field is continuous and it is not limited to a 32-bit boundary. The field must be terminated with a null octet, and followed by zero or more null octets until the next 32-bit boundary. This kind of padding is separated from the one specified with the Padding bit in the header.

Canonical name is the only mandatory SDES item. It must be sent on each RCTP packet. As the canonical name will be used to identify a participant in single and multiple RTP sessions, it must be unique. CNAME will be use to solve collisions with the SSRC identifiers. A participant may have multiple SSCR identifiers, one for each related session he is in, but he must have only one CNAME. As CNAME should provide information in order to locate a source, its recommended formats are user@full hostname or user@IPaddress.

4. Goodbye (BYE)

The BYE packet indicates that one or more participants are no longer active. The packet has the format shown in Figure 3.9.

0	3	8	16	31
V P	SC	PT = 203 (BYE)	length	
SSRR_1 /CSRC_1				
SSRR_2/CSRC_2				
....				
length	reason for leaving			...

Figure 3.9 BYE Packet Format (Schulzrinne, et al., 99).

The header section has the same format of the previous RTCP packets. Following the header there is list of all participants that are leaving the section. The reason for providing the capability for more than one source in the BYE packet is the Goodbye packet sent by a mixer. If a mixer shuts down, it must send a BYE packet listing its SSRC and the CSRC of all sources it handles.

The last section is optional. It gives a reason by leaving a section, like “bad reception” or “time for lunch.” The “length” and “reason for leaving” fields have the same behavior as “length” and “context” fields of the SDES item.

F. MULTIMEDIA IN RTP

Each media must be carried in a distinct RTP session. Interleaving packets with different RTP media types but using the same session and SSRC is not permitted, in order to avoid the following problems:

- An SSRC has only one single timing and sequence number. Different payload types would require distinct timing spaces. Also there would be no means to identify which media suffered losses.
- The RTCP sender and receiver reports can only describe one timing and sequence number space per SSRC and do not carry a payload type field.
- An RTP mixer would not be able to combine interleaved streams of incompatible media onto one stream.
- It would not be possible to use different network paths or resource allocations for each media.

G. ANALYSIS OF SR AND RR REPORTS

It is expected that reception quality feedback data will be useful not only for the senders but also for receivers and third-part monitors. The senders can modify its transmission based of the feedback; receivers can determine where the problems are local, regional or global; network managers may use profile-independent monitors that receive only the RTCP packets and not the corresponding RTP data streams to evaluate the performance of their networks for multicast distribution. These mechanisms also support protective and corrective mechanisms such as congestion avoidance and congestion control.

The interarrival jitter field provides a short-term measure of network congestion. The packet loss metric tracks persistent congestion, while the jitter measure tracks

transient congestion. The jitter measure may indicate congestion condition forming before it leads to packet loss.

H. RTP PROFILES AND PAYLOAD FORMAT SPECIFICATIONS

RTP is intended to be tailored through modifications and/or additions to the headers as needed. Multiple profiles have been defined, each with different characteristics. Therefore, for a given application, a complete specification will require one or more companion documents, as follows.

1. Profile Specification Documents

A Profile Specification Document defines a set of payload type codes and their mapping to payload types. A profile may also define extensions or modifications to RTP that are specific for that kind of applications. The only profile document issued as of this writing is RFC 1890 - RTP Profile for Audio and Video Conferences with Minimal Control.

2. Payload Format Specification Documents

A Payload Format Specification Document defines how a particular payload, such an audio or video encoding, is to be carried in RTP. There is an Internet-Draft containing guidelines to be followed by Payload Format Specification Documents (Handley, 99). Several payload format specifications were proposed so far, most of them related to audio and video encodings. Below is a list of payload format RFCs and Internet-Drafts:

- RFC 2032 - RTP Payload Format for H.261 Video Streams.
- RFC 2029 - RTP Payload Format of Sun's CellB Video Encoding.
- RFC 2190 - RTP Payload Format for H.263 Video Streams.
- RFC 2198 - RTP Payload for Redundant Audio Data.

- RFC 2250 - RTP Payload Format for MPEG1/MPEG2 Video.
- RFC 2343 - RTP Payload Format for Bundled MPEG.
- RFC 2429 - RTP Payload Format for the 1998 Version of ITU-T Rec. H.263 Video.
- RFC 2431 - RTP Payload Format for BT.656 Video Encoding.
- RFC 2435 - RTP Payload Format for JPEG Compressed Video.
- RTP Payload for Dial-Tone Multi-Frequency (DTMF) Digits.
- RTP Payload Format for X Protocol Media Streams.
- RTP Payload Format for MPEG-4 Streams.
- RTP Payload Format for User Multiplexing.
- RTP Payload Format for Reed-Solomon Codes.
- RTP Payload Format for Interleaved Media.
- RTP Payload Format for Telephone Signal Events.
- RTP Payload Format for DVD Format Video.

Other RTP related RFCs and Internet-Drafts include:

- RFC 2354 - Options for Repair of Streaming Media.
- RFC 2508 - Compressing IP/UDP/RTP Headers for Low-Speed Serial Links.
- Real-Time Protocol Management Information Base.
- Sampling the Group Membership in RTP.
- Issues and Options for RTP Multiplexing.
- Conformance Texts for RTP Scalability Algorithms.
- RTP Testing Strategies.

I. SUMMARY

The Real-Time Transport Protocol (RTP) provides a way to transmit time-based media over wide-area networks (WAN), adding synchronization and feedback features over the existing transport protocol. Although RTP was initially devised for application in audio and video conferences, this protocol can be applied to convey other types of streamed media across the network.

THIS PAGE LEFT INTENTIONALLY BLANK

IV. JAVA MEDIA FRAMEWORK (JMF)

A. INTRODUCTION

This chapter provides an overview of the Java Media Framework (JMF) basic architecture and the specialized set of classes to manage RTP transmission, reception and control. Several Unified Modeling Language (UML) (Booch, et al, 97). diagrams are used to illustrate JMF classes interdependencies and behavior. The software application Rational Rose 98i (Rational, 99) was used to import the class components from the JMF API and to draw these diagrams. Appendix A contains a description of how UML diagrams can be prepared using the software Rational Rose 98i.

B. OVERVIEW

Java Media Framework (JMF) is a Java Application Programming Interface (API) developed by Sun Microsystems in partnership with other companies to allow Java programs deal with time-based media, especially audio and video (Sun, 99). Time-based media can be defined as any data that changes meaningfully with respect to time. It is also referred to as streaming media, since it is delivered in a steady stream of packets that must be received and processed within a particular timeframe to produce acceptable results.

JMF 1.0, the 1998 version, supports playback of several media types and RTP stream reception. JMF 2.0 is being developed by Sun and IBM, and is currently in public beta testing. JMF 2.0 provides media capture functionality, file saving and transmission of RTP streams, together defining a plug-in API that is intended to enable developers to customize and extend JMF functionality. JMF 2.0 early access version was released in

June 1999. The beta version was released in August 1999 and the final release is being expected in Fall 99. This chapter was written based on JMF2.0 early access (Sun, 99), and the software development was performed using JMF 1.0, 2.0 early access and 2.0 beta.

C. JMF ARCHITECTURE

The JMF API can be divided into two parts. A higher-level API, called the JMF Presentation and Processing API, manages the capturing, processing and presentation of time-based media. A low-level API, called the JMF Plug-in API, allows customization and extension. Developers working on new capabilities are expected to add software elements to the JMF Plug-in API, thereby extending JMF functionality and supporting new media types.

JMF 2.0 uses the following basic classes/interfaces to model the high-level API:

- MediaLocator - describes the location of a media content. A MediaLocator is closely related to an URL, but identifies stream parameters.

- DataSource – represents the media itself. A DataSource encapsulates the media stream much like videotape does for a video movie. This class is created based on a Media Locator.

- Player – provides processing and control mechanisms over a DataSource just like a VCR does for a videotape. A Player can also render the media to the appropriate output device (e.g. monitor or speakers).

- Processor – is a specialized type of Player that provides control over what processing is done on the input media stream. Processor supports a programmatic interface to control the processing of the media data, and also provides access to the output data streams.

- DataSink – represents an output device other than a monitor and speaker, the most common destinations for media output. For example, a DataSink can be used to save the media to a file or to retransmit to a network.

- Manager – handles the construction of Player, Processor, DataSource and DataSink objects.

Figure 4.1 shows some possible connections between the above elements. Part a) is the configuration used to play a media stream. In part b) we see a processor creating a DataSource object from another DataSource. In part c) a DataSource is passed to a DataSink that writes it to a file. Note that multiple DataSource objects can be connected via Processors.

A Player or Processor generally provides two standard user interface components: a visual component and a control-panel component. These components can be accessed by calling the getVisualComponent and getControlPanelComponent methods.

A DataSource represents a media stream which can have multiple channels of data called tracks. For example, a Quicktime (Apple, 99) file might contain both audio and video tracks. Demultiplexing is the process of separating out the individual tracks of a complex stream.

Inside a Player or Processor several operations can take place. For each operation there is a dedicated piece of software called a "plug-in." There are five types of plug-ins:

- Demultiplexers - extract individual tracks of media from a multiplexed media stream.
- Multiplexers - join individual tracks into a single stream of data.
- Codecs – perform media data encoding and decoding.
- Effect filters – modify the track data in some way, often creating some special effect. They can be classified as post-processing or pre-processing effect filters, depending on when they are applied in relation to the codec plug-in.
- Renderers – delivers the media data in a track to presentation device. For video, the presentation device is typically the computer screen. For audio, the presentation device is typically an audio card.

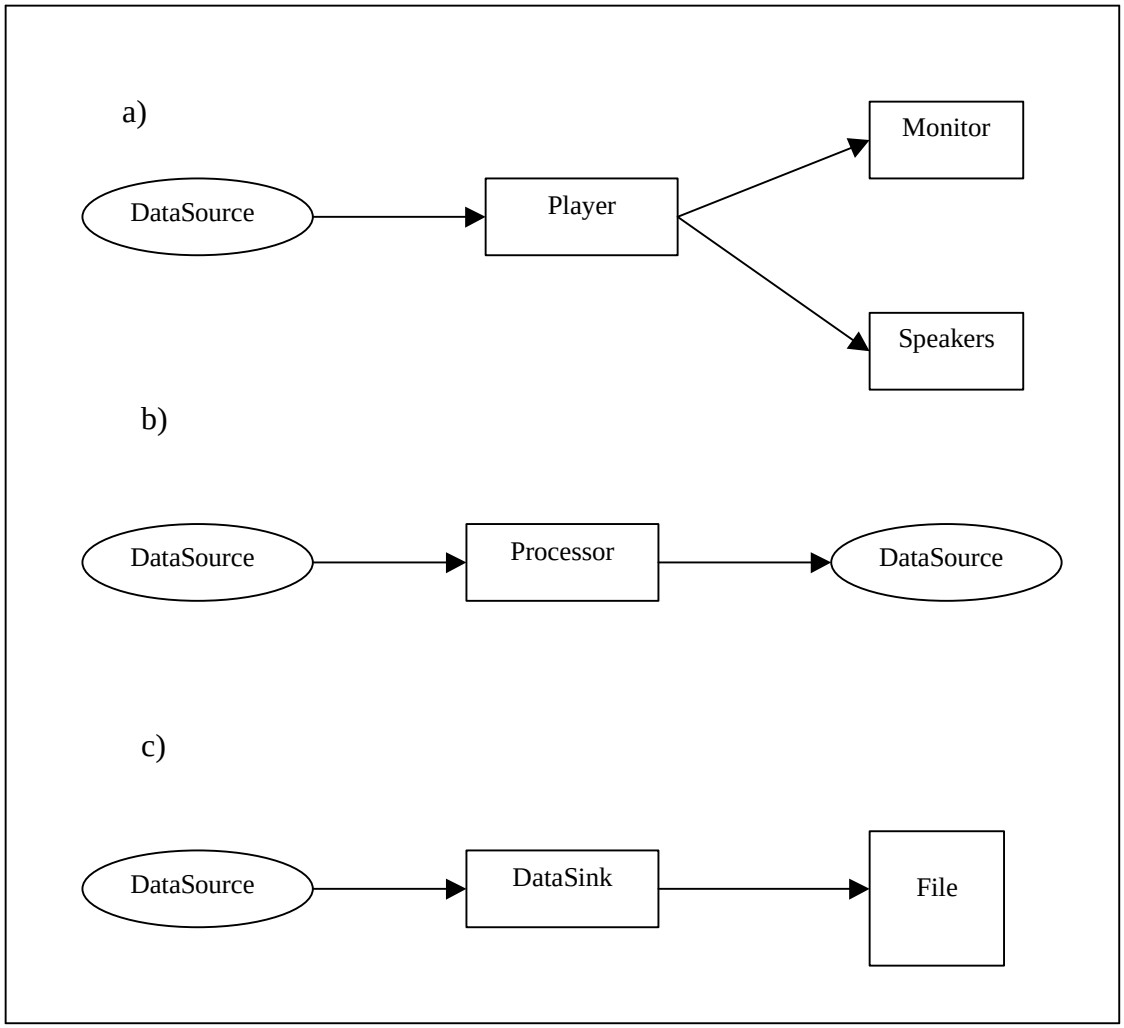


Figure 4.1 JMF High-level API Connection Examples.

Figure 4.2 shows an example of a Processor internal operation. In this example the media has one audio and one video track that are demultiplexed and processed individually. At the end they are multiplexed again and become available as another DataSource.

Manager provides access to a protocol-independent and media-independent mechanism for constructing and connecting DataSources, Players, Processors and DataSinks. A DataSource can be created by the method createDataSouce, with a

parameter specifying either a MediaLocator or URL. A Player or Processor can be created by the method createPlayer or createProcessor. The argument may be a DataSource, a MediaLocator or a URL. In order to create a Player or Processor from a MediaLocator or URL, the Manager first tries to create a DataSource. At last, for creating a DataSink a Manager has to receive a DataSource and a MediaLocator as argument. The DataSource represents the input to the DataSink and the MediaLocator describe the destination of the media to be handled by the DataSink.

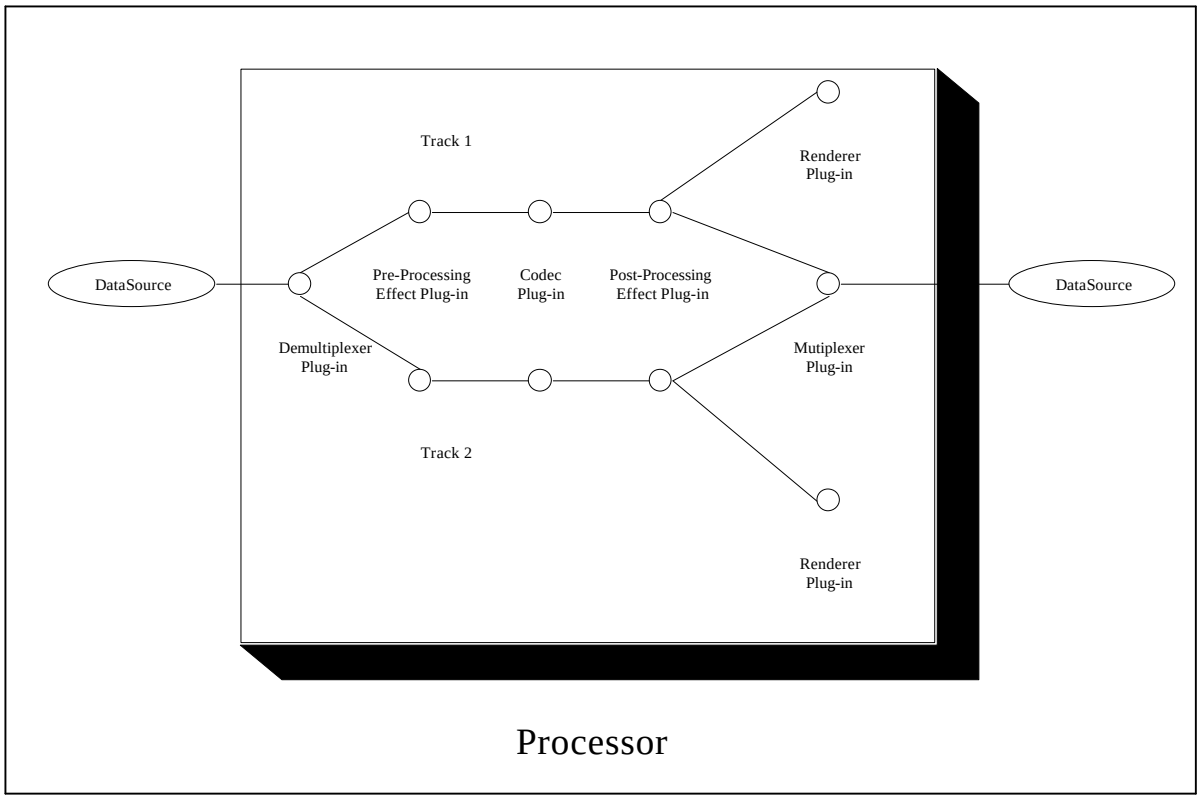


Figure 4.2 Processor Decomposition (Sun, 99).

The JMF class hierarchy can be extended by implementing new plug-in interfaces to perform custom processing on a track, or by implementing new DataSources, Players, Processors or DataSinks. New plug-ins must be registered with the class PlugInManager.

New Players, Processors, DataSources and DataSinks must be registered with the class PackageManager.

D. RTP SESSION MANAGER API

The RTP Session Manager API is the part of JMF API that deals with RTP/RTCP transmission and reception. It is contained in two packages: `javax.media.rtp` and `javax.media.rtp.session`.

The RTPSessionManager interface is responsible for entering mandatory conventions for creating, maintaining and closing an RTP session. It detects incoming RTP streams, maintains a list of RTP participants and transmits outgoing streams. It also keeps track of global statistics about the session.

Since `javax.media.rtp.session.RTPSessionManager` is an interface, an implementation is provided by Sun in the file jmf.jar. The class name is RTPSessionMgr. There is currently no source code available for this package. The following line creates an RTPSessionManager:

```
RTPSessionManager mgr = new com.sun.media.RTP.RTPSessionMgr();
```

1. RTP Streams

RTPStream is an interface that represents a series of data packets originated from a single host. There are two sub-interfaces of RTPStream: RTPRecvStream and RTPSendStream. The first represents a stream that is being received from a remote participant. RTPSessionManager creates RTPRecvStream objects automatically when new receiving streams are detected. The second represents a stream being sent by a local

participant. RTPSessionManager creates new RTPSendStream objects when the method createSendStream is called, using a DataSource object as argument.

2. RTP Participants

RTPParticipant is an interface that represents one participant in an RTP session. A participant may be the source of zero or more streams. The method getStreams returns a vector containing all RTPStream objects owned by the participant.

RTPParticipant has two sub-interfaces: RTPRemoteParticipant and RTPLocalParticipant. These are only marker interfaces, with no extra functionality. The RTPSessionManager creates a new RTPRemoteParticipant whenever a new RTCP packet arrives that contains a CNAME that has not been seen before. The association between the RTPParticipant object and a RTPRecvStream is done using the SSRC identifier. It is possible to have an unassociated RTPRecvStream as the source can start sending RTP packets before a CNAME RTCP packet is sent.

A participant that sends no data is called a Passive Participant. Otherwise it is called an Active Participant. The method getAllParticipants of RTPSessionManager returns a vector with all RTPParticipants. Similar methods exist to return vectors with remote, local, active and passive participants.

3. RTCP Source Description

RTCPSourceDescription is a class that contains one description information related to a participant, as received by the RTCP SDES packets. So, associated with a RTPParticipant object there may be several RTCPSourceDescription objects, each representing one description information, as CNAME, name, e-mail, location, tool, etc.

The method getSourceDescription of RTPParticipant returns a vector with the RTCPSourceDescription objects related to the participant.

4. RTCP Report

Passive participants send RTCP RR packets as a feedback about the reception of incoming streams. Active participants send RTCP SR packets that give information about the stream being sent and also include feedback about the reception of incoming streams. So, a SR packet includes the RR packet information. A RTCP SR or RR packet is originated from each SSRC identifier. A participant may have more than one SSRC identifier if it is source of more than one stream in the same session. Each SSRC is related to one stream. In spite of it, each participant has only one CNAME. RTCP SDES packets allow the correlation between SSRC and CNAME.

In JMF there is an interface called RTCPReport to represent both RTCP SR and RTCP RR packets. RTCPReport has two sub-interfaces: RTCPSenderReport and RTCPReceiverReport. RTCPReceiverReport is a marker interface. All of its functionality is contained in RTCPReport. RTCPSenderReport extends RTPCReport to provide methods for retrieve sender report information. The method getReports of RTCPParticipant returns a vector containing the last SR or RR reports sent by the participant. This method will usually return only one report: SR if the participant is active and it sends only one stream, and RR if the participant is passive. If the participant sends more than one stream this method will return the same number of SR reports, because each SR report is associated to one stream/SSRC.

RTCPReport has a method called getFeedback that returns a vector of RTCPFeedback objects. Each RTCPFeedback object conveys information about the

reception of one incoming stream. In other words, it represents a feedback from a SSRC about other SSRC that originates a stream. Feedback information includes fraction lost, cumulative number of packets lost, etc.

Figure 4.3 contains a class diagram describing the relationship between RTPParticipant and other classes/interfaces discussed so far. This class diagram uses the standard notation specified for the Unified Modeling Language (UML) (Booch, et al, 97).

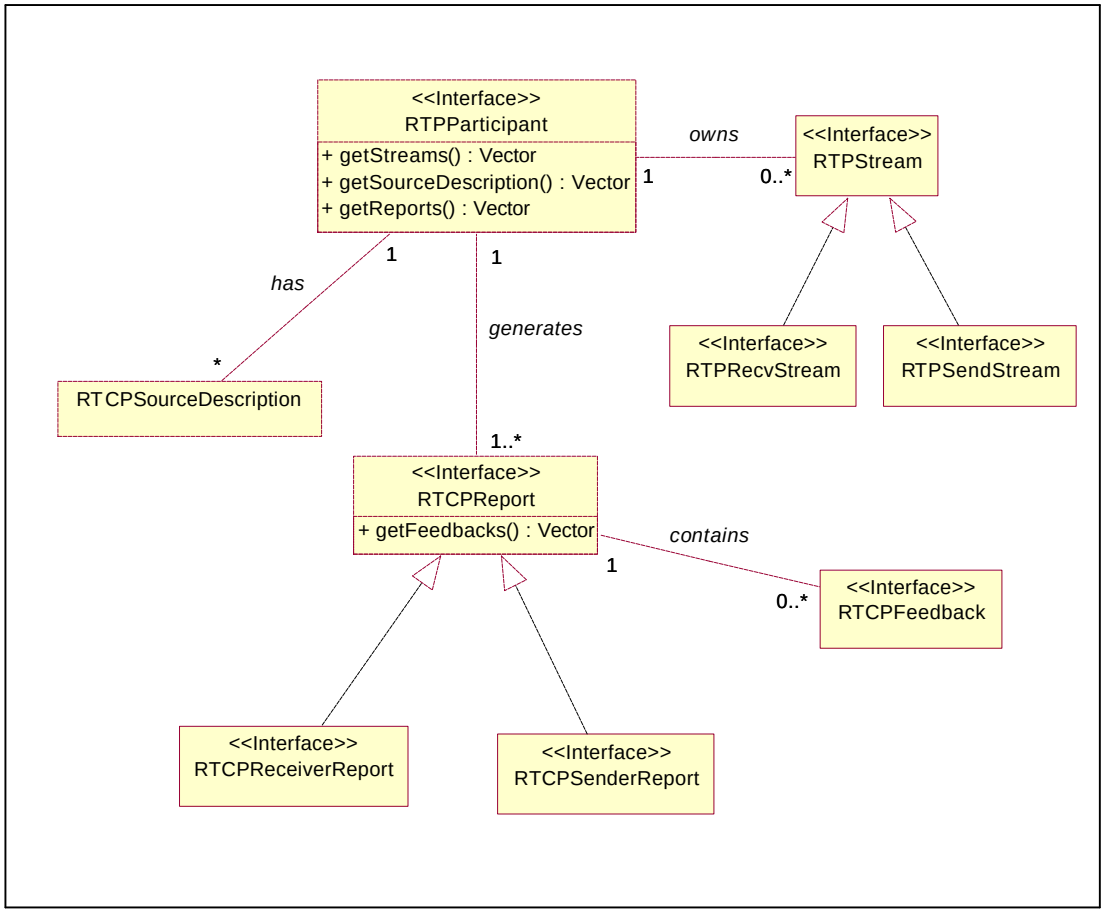


Figure 4.3 RTPParticipant Class Diagram.

5. Event Listeners

The RTP API has four types of event listeners: RTPSessionListener, RTPSendStreamListener, RTPReceiveStreamListener and RTPRemoteListener. These listeners provide a mechanism of event notification on the state of the RTP session and its streams.

The RTPSessionListener interface detects events related to the RTP session as a whole rather than a particular stream or participant. Two types of events can be posted: NewparticipantEvent, generated when a RTCP packet from an unknown participant was received, and LocalCollisionEvent, generated when a SSRC collision was detected between the local participant and a remote one.

The RTPSendStreamListener interface detects state transitions that occur on a RTPSendStream. Four types of events can be posted:

- NewSendStreamEvent – generated when a new transmitting stream has been created.
- ActiveSendStreamEvent – generated when the transfer of data from the DataSource object has started arriving after a previous stop.
- InactiveSendStreamEvent – generated when the transfer of data from the DataSource object has stopped.
- SendPayloadChangeEvent – generated when the payload type of the DataSource object has changed.

The RTPRecvStreamListener interface detects state transitions that occur on a RTPRecvStream. Seven types of events can be posted:

- NewRecvStreamEvent – generated when a new incoming stream has been detected. This means that RTP data packets has been received from a SSRC that had not previously been sending data.

- ActiveRecvStreamEvent – generated when the stream data packets have started arriving after a previous stop.
- InactiveSendStreamEvent – generated when the stream data packets have stopped arriving.

- PayloadChangeEvent – generated when a remote sender has changed the payload type of a data stream.

- TimeOutEvent – generated when a remote sender has not sent packets for a while and can be considered timed-out. A time-out has the same effect as if the participant has sent the RTCP BYE packet.

- RecvStreamMappedEvent – generated when a recently created stream has been associated with a participant after the first RTCP packet has been received.

- AppEvent – generated when an RTCP APP packet has been received.

The RTPRemoteListener interface detects events related to RTCP control messages received from remote participants. This interface can be used for monitoring applications that do not need to receive each stream, but only RTCP reports. Three types of events can be posted:

- RecvReceiverReportEvent – generated when a new RTCP RR report has been received.

- RecvSenderReportEvent – generated when a new RTCP SR report has been received.

- RemoteCollisionEvent – generate when two remote participants are using the same SSRC simultaneously. Upon detecting the collision, both remote participants should start sending data with new SSRCs.

6. RTP Media Locator and RTP Session Address

RTPMediaLocator is a class that stores information about the session address and other settings used in a session like TTL and SSRC. The format is similar to an URL. The RTP MediaLocator string is of the form:

- rtp://address:port[:SSRC]/content-type/[TTL]

Optional parameters are enclosed in brackets. Address is the IP address of the session. Port is the port number used for RTP packets. It must be an even number according RFC1889. SSRC is optional. Content-type can be either “audio” or “video.” TTL is the time-to-live, also optional. The string above is used to create a RTPMediaLocator object. If the media locator is invalid a MalformedRTPMRLException will be thrown.

The class RTPMediaLocator has several methods to retrieve the above information about a session. For example, the method getSessionAddress returns a string with the IP address and the method getSessionPort returns an integer with the RTP port number. However, to create a session and instantiate a RTPSessionManager under JMF, another object has to be created first: an RTPSessionAddress.

RTPSessionAddress is a class that encapsulates a pair of multicast addresses, each constituting of an IP address and a port number. One multicast address is used by RTP and the other by RTCP. (In fact, this definition of a RTPSessionAddress is too broad, because RFC1889 says that the RTCP multicast address must have the same IP address and a immediately higher port with relation the RTP address.) Also, RTP port must be even by RFC1889.

In order to create a RTPSessionAddress object for representing a session address, two InetAddress objects are required as arguments, one for RTP and other for RTCP. The InetAddress class is part of the java.net package and encapsulates an IP address. The static method getByName can create a InetAddress object given a string with the IP address. Figure 4.4 contains a sequence diagram that describes the creation of a RTPSessionAddress object.

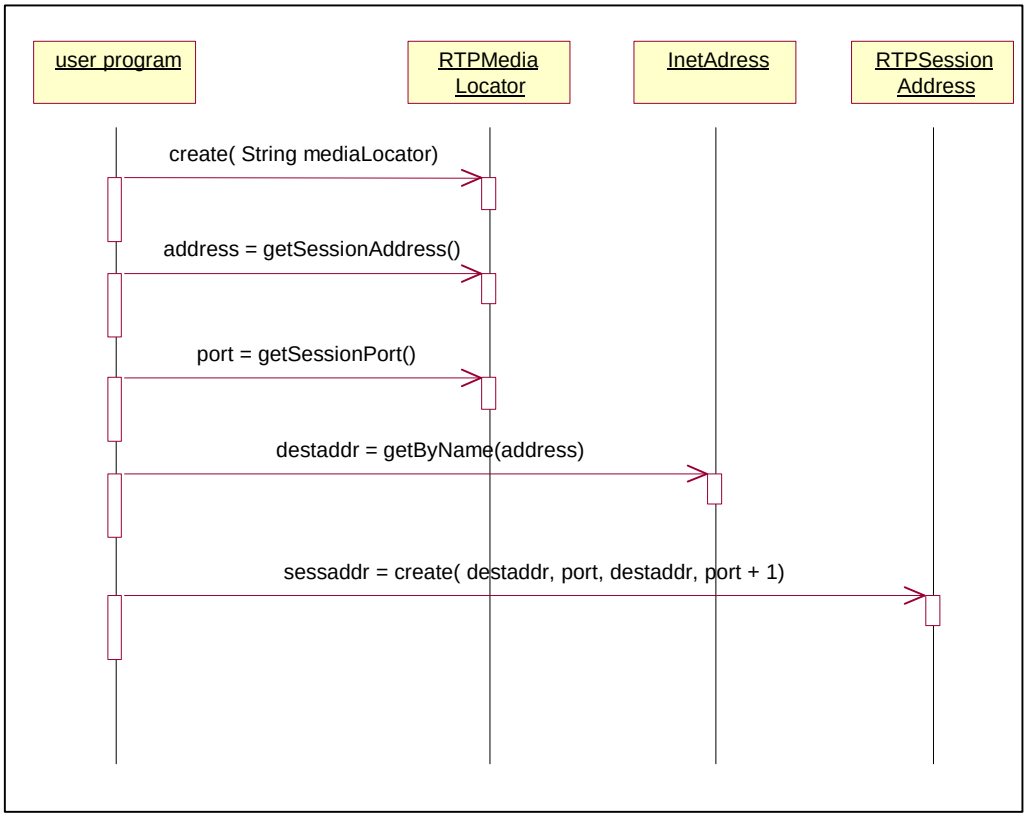


Figure 4.4 RTPSessionAddress Creation UML Sequence Diagram.

7. RTP Session Manager

After creating an empty RTPSessionManager object as described in item C), a RTPSessionManager must be initialized by calling the method initSession. The required parameters are:

- The local IP address as a RTPSessionAddress object. This object can be created by calling the RTPSessionAddress constructor with no arguments.
- An array of RTPSourceDescription objects containing information about the local participant.
- The fraction of the bandwidth to be allocated to RTCP. RFC1889 recommends 5% of the RTP bandwidth.

- The fraction of the RTCP bandwidth to be allocated to Sender Reports. RFC1889 recommends 25% of the RTCP bandwidth.

At this moment the RTPSessionManager is still not active. The method startSession must be called in order to cause RTCP reports be generated and callbacks to the several listeners to be made. This method must be called prior to the creation of any streams on a session. The required parameters are:

- The session address as a RTPSessionAddress object.
- The time-to-live (TTL)
- A RTPEncryptionInfo object if any encryption is desired.

8. Receiving and Presenting RTP Media Streams

After RTPSessionManager has been started, any receiving stream will generate a RTPRecvStream object and a NewRecvStreamEvent event. In the update method of the interface RTPRecvStreamListener, the RTPRecvStream object can be obtained by the method getRecvStream of NewRecvStreamEvent. By retrieving the DataSource from the RTPRecvStream object and passing it to the Manager we can create a Player for the received stream. The class PlayerWindow, supplied by Sun, receives a Player, plays back the media, audio or video, and creates a control window. Figure 4.5 contains a collaboration diagram describing the sequence above. This collaboration diagram uses standard notation for showing logical relationship between instantiated objects, as specified by UML (Booch, et al, 97).

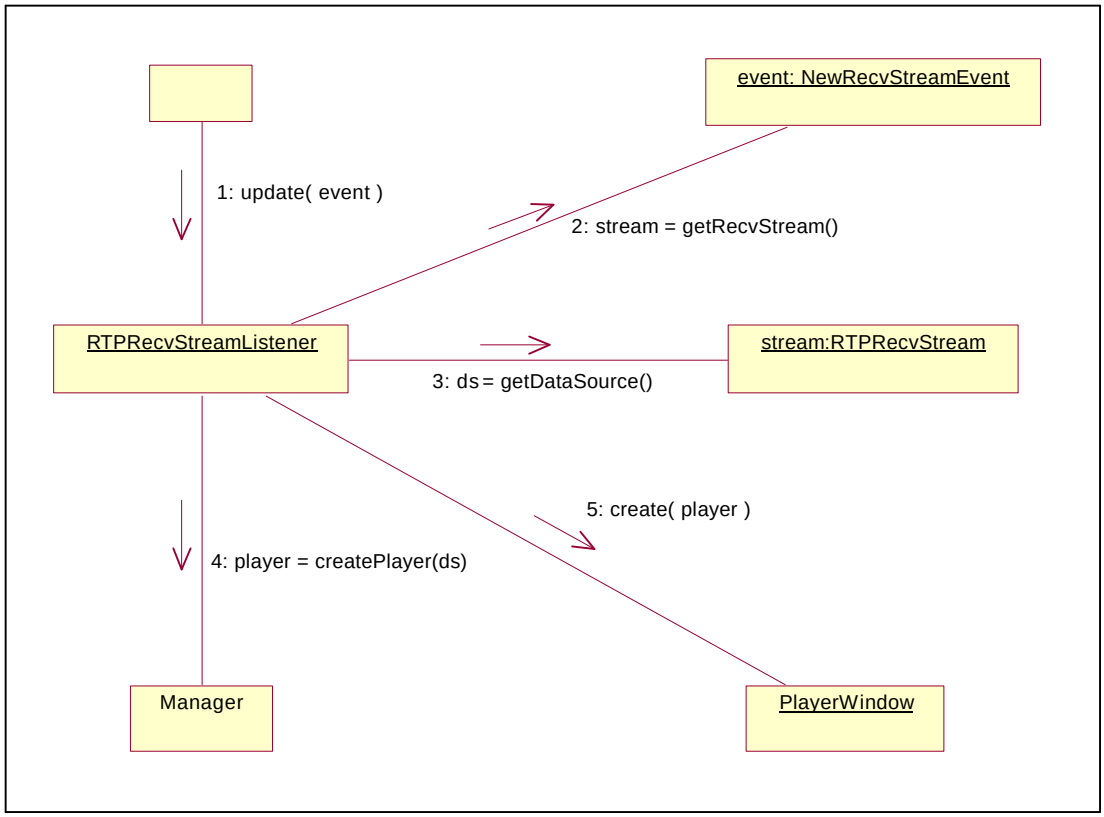


Figure 4.5 RTP Media Presentation Collaboration Diagram. Numbered arrows indicate the sequence of method calls occurring between objects.

9. Transmitting RTP Streams

RTP stream can be transmitted by passing a DataSource object to the Session Manager using the method createSendStream. As a DataSource can contain multiple streams/tracks, an index of the stream must be specified. The DataSource is usually obtained as an output of a Processor. In this case the Processor has to generate RTP-encoded data because the Session Manager does not perform that function. So each track to be transmitted has to be set to an RTP-specific format. The method setFormat of the TrackControl interface allows a Processor to set the format of a track. If the format can not be applied to the track, an IncompatibleFormatException is thrown.

If the physical source of the stream is intended to be camera or microphone, a CaptureDeviceInfo object must be created by using the method getDevice of CaptureDeviceManager. The argument passed to the method getDevice is a Format object, which describes a media format. The CaptureDeviceManager searches for a device in the system that supports the desired format. The method getLocator of CaptureDeviceInfo returns a Media Locator that can be used to create a Processor.

Figure 4.6 contains a UML collaboration diagram that describes a JMF procedure to capture the media and transmit it in a RTP session.

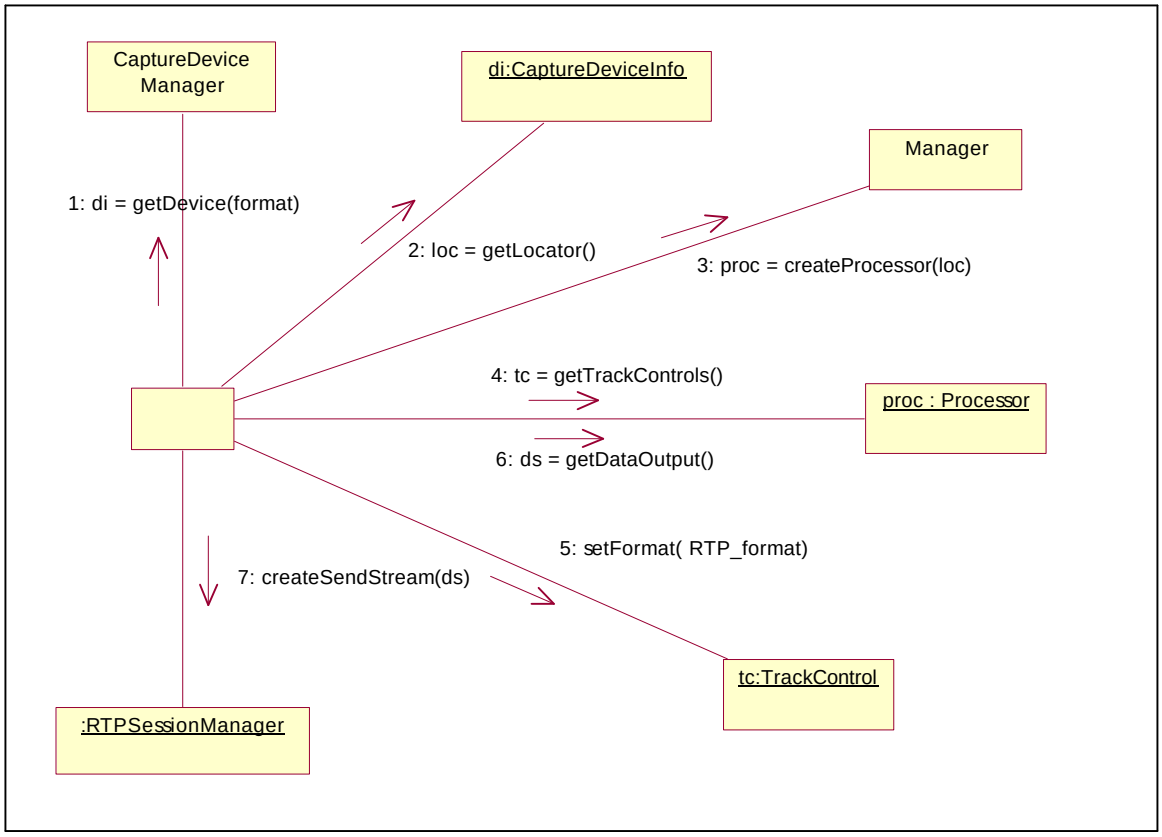


Figure 4.6 Media Capture and Transmission Collaboration Diagram.

10. RTP Statistics

JMF maintains several statistics about the RTP session and its streams. Statistics about the session as a whole are obtained through the RTPSessionManager by the methods getGlobalTransmissionStats and getGlobalReceptionStats. These methods retrieve GlobalReceptionStats and GlobalTransmissionStats objects respectively, which have methods to get each statistic. Individual stream statistics are maintained within RTPSendStream and RTPRecvStream objects by the RTPTransmissionStats and RTPReceptionStats interfaces. Figure 4.7 contains a UML class diagram showing the classes mentioned above.

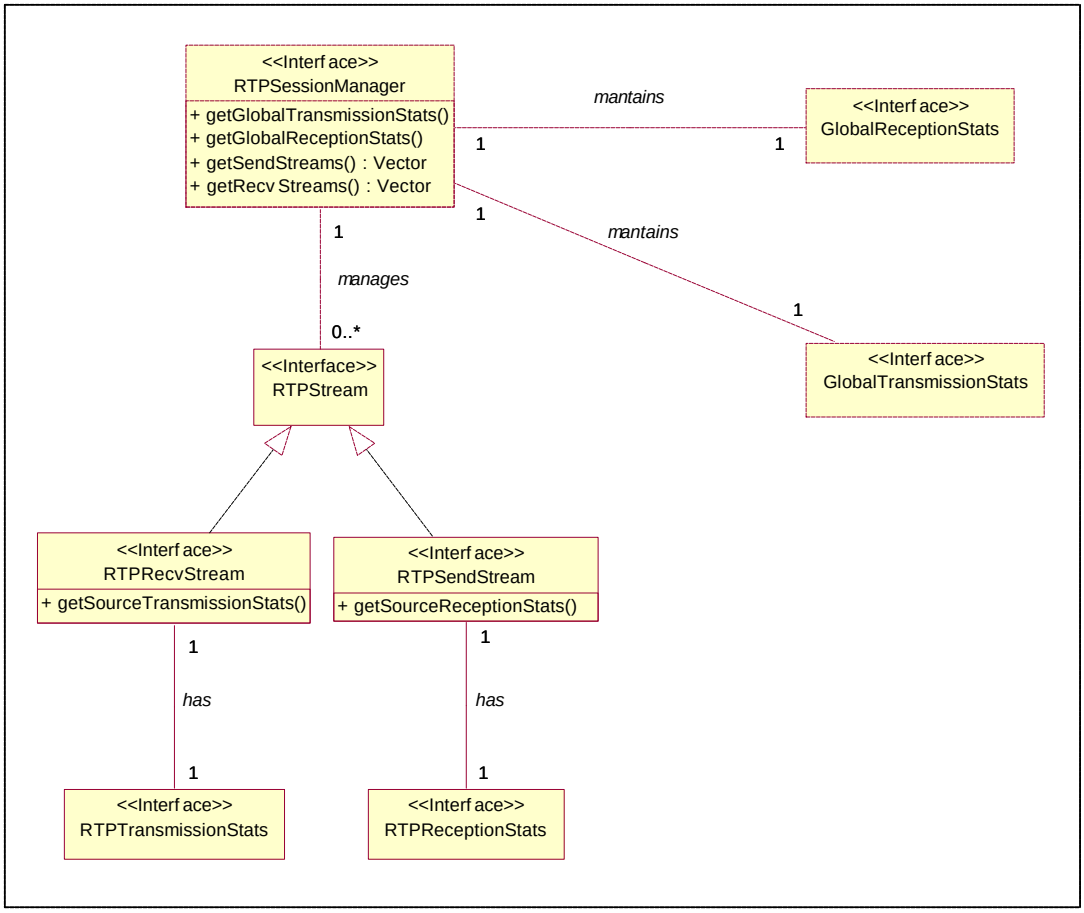


Figure 4.7 RTP Statistics Class Diagram.

D. SUMMARY

The Java Media Framework (JMF) architecture is intended to support multimedia in a variety of applications. It is based in a high level API, containing generic classes to capture and present media, and a low-level API to allow customization and extension. The RTP API supports RTP transmission and reception, as well as retrieval of RTP statistics. Future work on JMF can include the transmission of time-based media other than audio and video, such as the Distributed Interactive Simulation (DIS) protocol and other behavior-based streams.

V. DESIGN AND IMPLEMENTATION OF THE RTPMONITOR APPLICATION

A. INTRODUCTION

This chapter covers the functionality and class design of the program developed as the main goal of this thesis, called rtpMonitor. rtpMonitor is a Java application that presents and records RTP statistics about a single RTP session. It can also present the media being received, either audio or video, by launching audio/video playback windows. All Unified Modeling Language (UML) diagrams in this chapter conform to the UML Specification (Booch, 97) and were prepared using the Java to UML facilities of Rational Rose 98i (Rational, 99).

B. RTPMONITOR FUNCTIONALITY AND INTERFACE

This section contains a summary of the rtpMonitor functionality. Additional information can be found in the Appendix B, the rtpMonitor User Manual.

1. Graphical User Interface (GUI)

Figure 5.1 shows the rtpMonitor main window. In the "Bookmark" menu, the user can select, add or delete a bookmark related to the session name/address. In the "Preferences" menu the user can select whether the monitor will participate in the session, play streams and/or record statistics. The user can also be selected the duration of the monitoring session, the recording interval and the presentation interval. In the "Output files" menu the user can launch an external viewer to see the contents of the statistic files generated by the program.

The upper part of the window contains the session address, according the RtpMediaLocator format and the session name. The user enters the session name when a new bookmark is inserted. This name does not necessarily have any relation with the session name of the Session Description Protocol (SDP) announcements. The rest of the window contains the current RTP statistics.

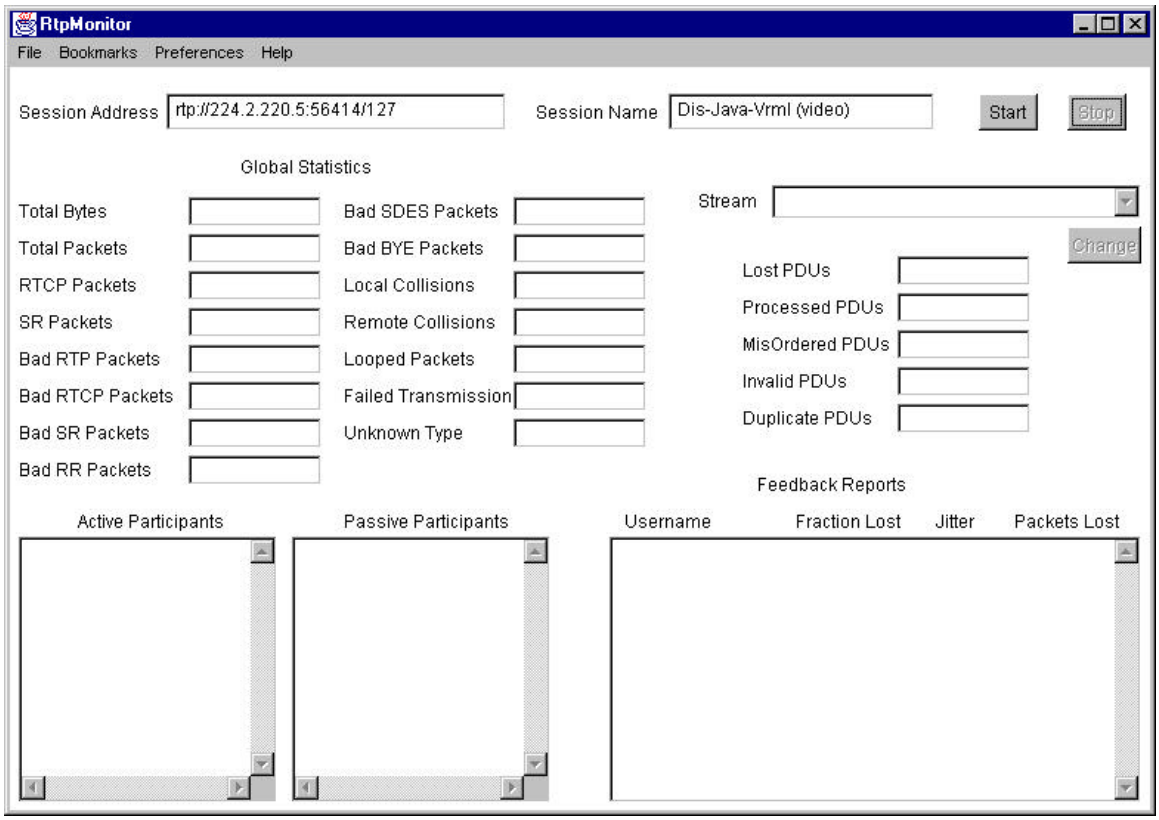


Figure 5.1 rtpMonitor Main Window.

2. Statistics Display

The program periodically displays the following information:

- Global statistics: general reception information about the whole session.
- Stream statistics: about the reception of a single stream.

- Feedback: the RTCP Receiver Report information from all participants about the stream being monitored.

3. Statistics Recording

rtpMonitor can record statistics in text files using a recording interval defined by the user. Several files are created and managed to simplify further retrieval. There are files to hold data from the last five minutes, previous five minutes, last hour, previous hour and different dates. Multiple file sizes and periodicities are employed to avoid excessive file sizes when performing extended monitoring. File name conventions are presented in Appendix A.

4. Media Presentation

rtpMonitor can present the incoming video stream on the computer screen or the incoming audio streams on the computer speakers. This option is selected in the "Preferences" menu. In case of video, each stream is presented in a separate window. In case of audio, a toolbar window is opened to allow audio controls (such as the mute function). The launched applications are part of the JMF API.

5. Command Line Operation

If the program is called with any argument in the command line the GUI will not be launched. In this case the session address and preferences must be passed by the command line.

The following command line is an example of invocation of the rtpMonitor to record statistical data and present RTP session streams during a period of 24 hours:

```
java org.web3d.vrtp.rtp.RtpMonitor  rtp://224.120.67.46/64542/12
     -play -record -e 24
```

Using the command line version of rtpMonitor no statistics are sent to the console. A monitoring session can be stopped by pressing "Crtl-C".

C. RTPMONITOR CLASS DESIGN

The rtpMonitor class design goal was to create a set of basic classes that could perform the RTP monitoring tasks with minimal access from user applications to the Java Media Framework API. Figure 5.2 contains a data flow diagram showing the exchange of data (statistics, settings and commands) between classes in rtpMonitor. Each bubble in the diagram will be discussed in the remainder of this session. Appendix C contains the rtpMonitor Javadoc, and Appendix D contains the rtpMonitor source code.

1. RtpMonitorManager and RtpUtil

The RtpMonitorManager class is the main interface between a user application and JMF. It performs the following functions:

- Creates and starts a session, represented by a RTPSessionManager object in JMF.
- Records session statistics in files.
- Presents (i.e. plays) incoming media streams.

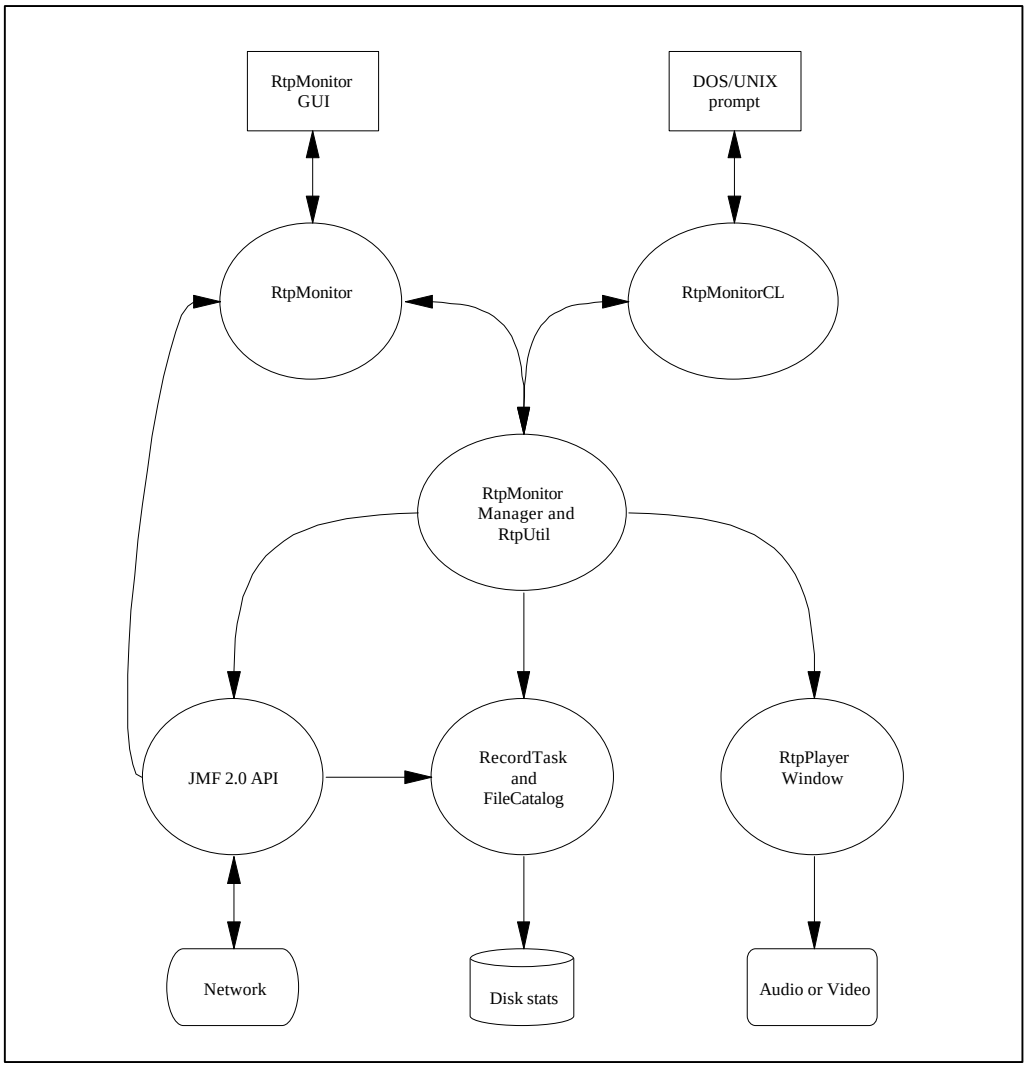


Figure 5.2 RTPMonitor Data Flow Diagram.

RtpMonitorManager does not have methods to retrieve each single statistic in an RTP session. If an application needs access to individual statistics, e.g. for display purposes, it is necessary to retrieve the RTPSessionManager object and use its methods to get the desired set of statistics.

The following parameters are necessary for instantiating a new RtpMonitorManager object:

- The session address string, e.g. rtp://224.2.134.67:50980/127
- A Boolean variable indicating if the statistics are to be recorded on files.
- A Boolean variable indicating if the incoming streams are to be played.
- A Boolean variable indicating if the monitor will actively participate in the session (i.e. send RTCP packets).

During instantiation, RtpMonitorManager creates a RTPSessionManager object in JMF. Figure 5.3 shows a UML sequence diagram representing the steps taken by RtpMonitorManager in creating and initializing the session manager. Figure 5.4 shows a class diagram of the classes related to RtpMonitorManager.

RtpUtil is a class that has only static methods for performing some extra functionality to JMF. For example, there is a method to return the username given a RTPParticipant object.

2. RecordTask and FileCatalog

RecordTask is a class used by RtpMonitorManager objects to write the statistics periodically to disk. It is created as a separate thread that waits for a fixed period of time after writing data to disk. This class writes data to the file called "`statisticsLastReport.txt`" that contains the last single report only. The FileCatalog class is actually responsible for transferring data to other files as well (last five minutes, last hour, and so on).

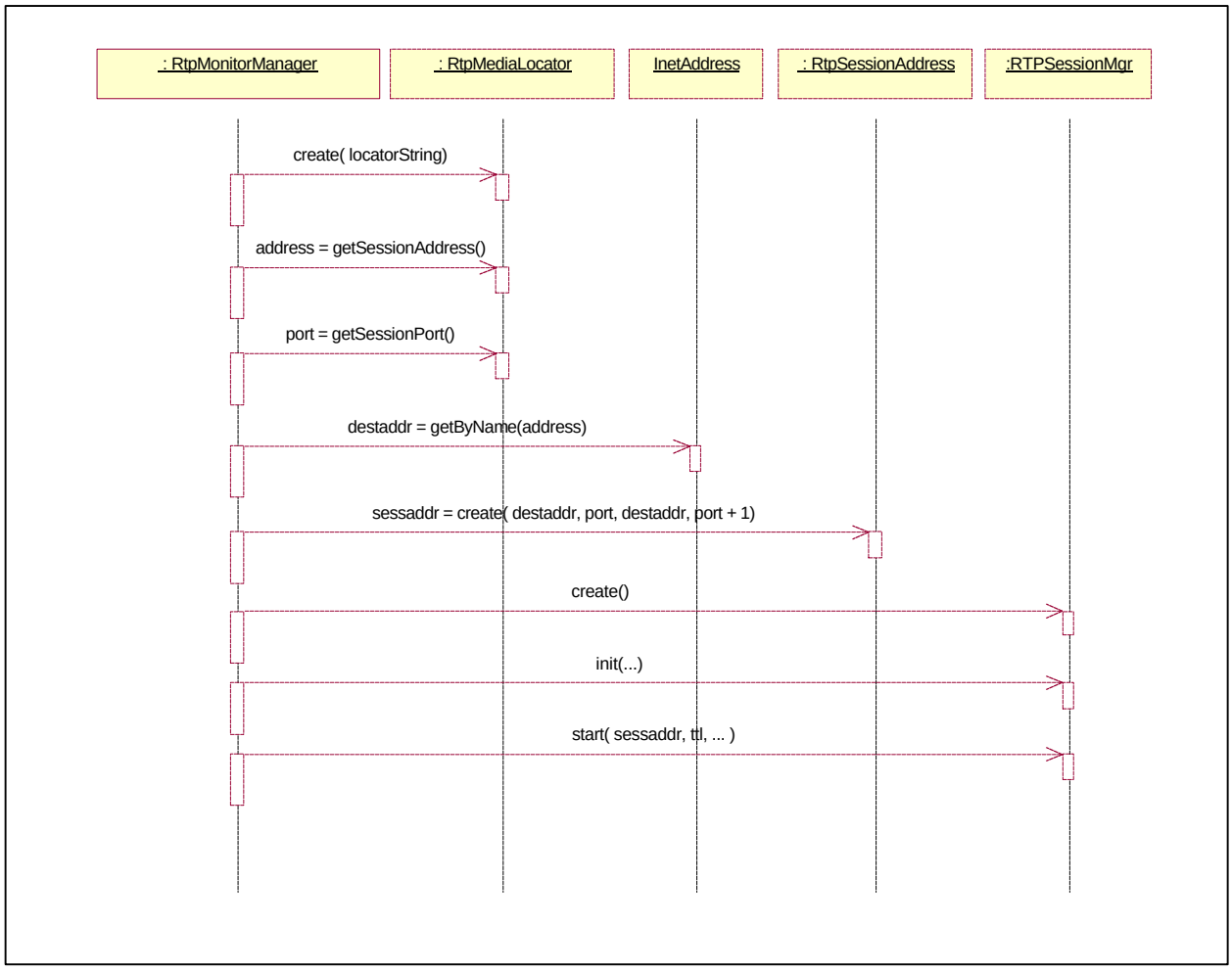


Figure 5.3 RTPSessionManager Initialization UML Sequence Diagram.

3. RtpPlayerWindow

RtpPlayerWindow is a class used to create a window for playing an audio/video stream. It is a subclass of PlayerWindow, adding the capacity of modifying the window name. Both classes were developed by Sun. RtpPlayerWindow came with JMF1.1 sample code and PlayerWindow is in the file `jmf.jar`.

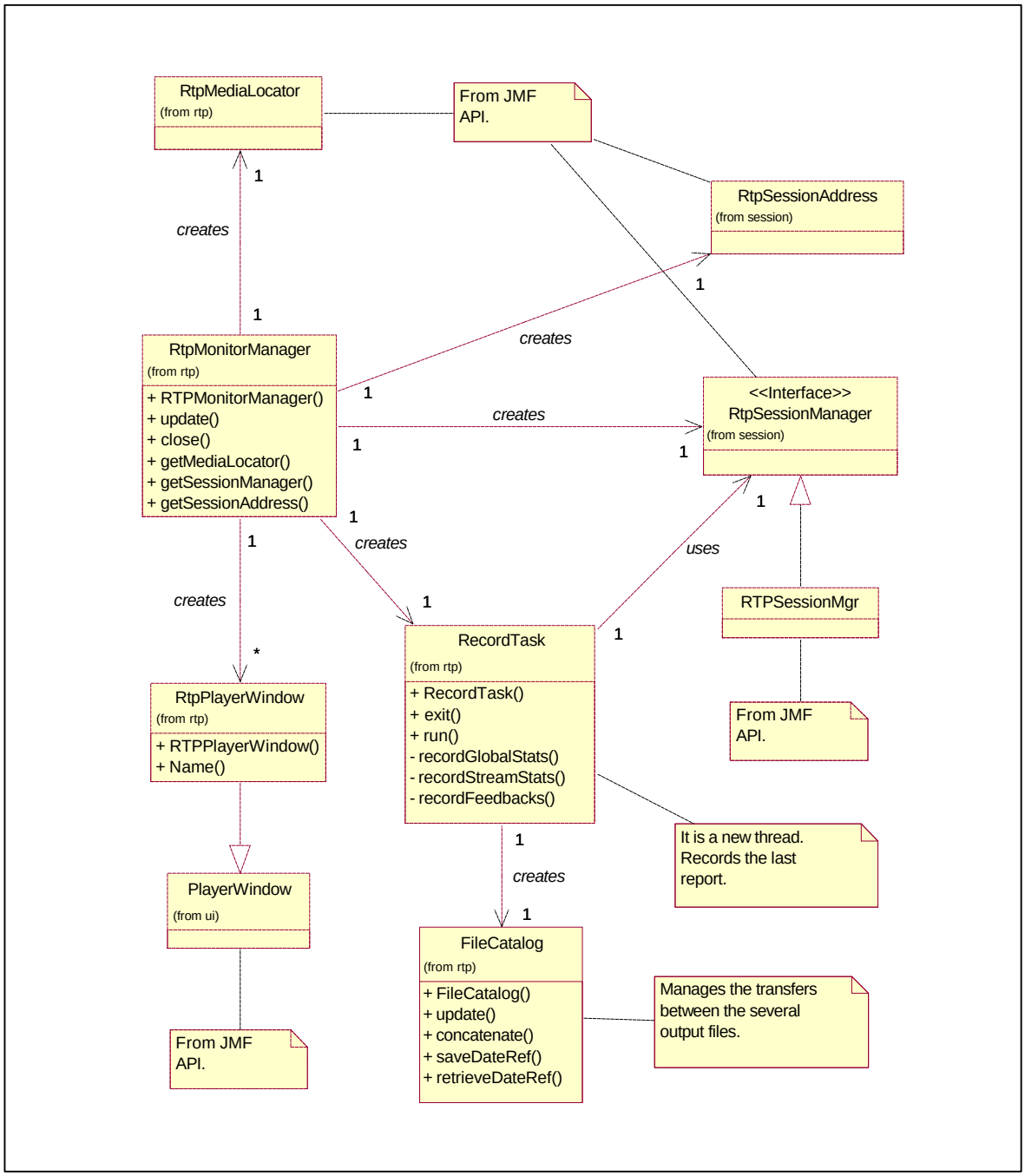


Figure 5.4 RTPMonitorManager UML Class Diagram.

4. RtpMonitor

The main RtpMonitor class extends a Frame and implements the rtpMonitor Graphical User Interface (GUI). RtpMonitor and RtpMonitorCommandLine can be considered classes in the application level. They use the services of RtpMonitorManager to provide some user level functionality. RtpMonitor has the following functions:

- Collects user preferences. It uses the ModifyPreference class as the dialog box.
- Selects, adds and deletes bookmarks. It uses the SelectBookmark, AddBookmark and DeleteBookmark classes as dialog boxes.
- Displays statistics periodically. It creates a DisplayTask object, which is a new thread that sleeps for a user-defined time, to call its methods for updating the statistics on screen.

5. RtpMonitor Applet

A good idea might be to make an applet version in a web page. To do this, security problems must be addressed, as the monitor must write data to disk and open network connections. The implementation of an RTP Monitor version running as an applet was left as future work.

6. RtpMonitorCommandLine

RtpMonitorCommandLine objects are instantiated by the main method of RtpMonitor when some parameter is passed. This class performs the creation of a RTPSessionManager object but does not present statistics on screen.

Several options can be passed by the command line call to RtpMonitor. They are basically the same available in RtpMonitor preferences dialog.

D. SUMMARY

The rtpMonitor application allows the monitoring of an RTP session by presenting session and stream statistics on screen as well as recording statistics on files for future analysis. The program also supports Media presentation of audio and video.

The design goal of the rtpMonitor was to provide a basic set of classes for RTP statistics recording with minimal interfacing with JMF.

VI. RTP MANAGEMENT INFORMATION BASE (MIB)

A. INTRODUCTION

The RTP Management Information Base (MIB) defines Simple Network Management Protocol (SMNP) objects for managing RTP systems. This work is produced by the Audio Video Transport (AVT) Group of the Internet Engineering Task Force (IETF) (Baugher, et al., 99) as broad guidance for all applications collecting RTP statistics. This chapter describes some basic concepts of the RTP MIB and compares its attributes with the existing set of Java Media Framework used by the rtpMonitor application.

B. NETWORK MANAGEMENT OVERVIEW

A network management system is a collection of tools for networking monitoring and control, including hardware and software (Stallings, 97). The key elements of a network management system are:

- Management station – the interface to the network manager.
- Agent – responds to requests for information and for taking actions. Typically, the agent software is installed in routers, bridges, hubs and hosts.
- Management information base (MIB) – represents a collection of objects (data variables) managed by agents.
- Network management protocol – links the management station with the agents.

The Simple Network Management Protocol (SNMP) is the most widely management protocol in use for TCP/IP networks. SNMPv2 (version 2) is described in

RFC 1901 (Case, et al., 96). SMNP include mechanisms for retrieving data from agents, set values of objects on agents, and notify the management station of significant events.

C. RTP MIB DESCRIPTION

RTP agents running this MIB can be either RTP hosts (end systems) or RTP Monitors. The objective is to collect statistical data about RTP sessions and its streams, for diagnostics and management purposes. Each agent maintains a MIB that can be queried by a Manager. Only the last updated statistic is stored in the MIB.

RTP MIB has three tables:

- **rtpSessionTable** – contains objects that describe active sessions at the host, intermediate system or monitor. There is an entry in this table for each RTP session on which packets are being sent, received and/or monitored.

- **rtpSenderTable -** contains information about senders of the RTP session. RTP sender hosts must have an entry in this table for each stream being sent, but RTP receiving hosts do not have to maintain this table. RTP Monitors must create an entry for each observed stream.

- **rtpRcvrTable -** contains information about receivers of the RTP session. RTP receivers must create an entry in this table for each received stream. RTP senders do not have to maintain this table. RTP monitors must have an entry for each pair sender/receiver in the sessions being monitored.

D. COMPATIBILITY WITH JMF STATISTICS

Appendix E is a table that compares the fields in the RTP MIB tables with the JMF statistics used by RTPMonitor. This comparison has two objectives:

- Detect what MIB statistics are not supplied by JMF. If JMF does not supply all the required MIB data, an RTP MIB agent can not be implemented using JMF.

- Detect what JMF statistics are not part of RTP MIB. Some of the JMF statistics can be added to the RTP MIB.

The comparison was sent to the JMF and IEFT-AVT mailing lists. The comments made by Bill Strahm, one of the authors of the RTP MIB Internet Draft are included in the table of Appendix E. Sun Microsystems software engineers have not replied with any comments.

E. SUMMARY

The Real-time Transport Protocol (RTP) Management Information Base (MIB) consists of a new Internet-Draft proposed by the AVT group of the IETF to be applied in RTP network management with SNMP.

A number of significant differences were found between JMF 2.0 (Sun, 99) and the RTP MIB (Baugher, et al., 99). Further work will need to be performed by one or both organizations to resolve these discrepancies.

THIS PAGE LEFT INTENTIONALLY BLANK

VII. EXPERIMENTAL RESULTS

A. INTRODUCTION

This chapter presents the experimental results achieved with the rtpMonitor application and describes problems faced during the testing phase. Furthermore, this chapter presents the work on an RTP header for the Distributed Interactive Simulation (DIS) protocol (IEEE, 95).

B. TEST RESULTS

The Java application rtpMonitor, described in Chapter V, was tested using the versions 1.1.7, 1.2.1 and 1.2.2 of the Java Development Kit (JDK), in combination with versions 1.0, 2.0 Early Access, and 2.0 Beta of the Java Media Framework. JMF Beta has three subversions available:

- Pure Java: includes binaries written entirely in the Java programming language that can be installed on any operating system supported by the Java platform.
- Solaris Performance Pack: an optimized version for the Solaris platform that includes binaries for this operating environment.
- Windows Performance Pack: an optimized version for the Windows platforms that includes binaries for this operating environment.

rtpMonitor was tested only in Windows NT platforms using the Windows Performance Pack. The program was tested in all its functionality, with special emphasis on the recording capabilities. It proved to be robust, running continuously for the maximum allowed session duration (one week) several times. It has generated correct output files of more than 40 Mbytes for a single session.

C. OBSERVED PROBLEMS

Several errors (i.e. software bugs) were detected in JMF 1.0 through the development of rtpMonitor. The following errors were reported in the JMF mailing list and were corrected in JMF2.0 Early Access:

- The number of lost PDU in the stream data was usually wrong. It returned a high number of lost PDU, even greater than the actual number of packets sent by the source.
- When a non-participating option had been selected the information about the active and passive participants was inconsistent. Usually no participants were presented even though they might have existed.
- Individual video windows could not be closed.
- After a session had been stopped and a new session had been initiated, the global statistics about the previous session were still being considered.

In JMF 2.0 Early Access and JMF 2.0 Beta, the only observed error is related to the Cumulative Number of Packets Lost (Packets Lost in the Feedback Area) which returns wrong values after some time. This problem has been reported to Sun's JMF-bugs e-mail box (jmf-bugs@sun.com) and to Sun's JMF-interest mailing list (jmf-interest@java.sun.com).

D. EXTENDING DIS-JAVA-VRML PDU HEADER

Protocol Data Units (PDU) currenty used by the vrtp protocol (Brutzman, 99) are based on the Distributed Interactive Simulation (IEEE, 95) standard. As the vrtp protocol intends to use the Real-Time Transport Protocol as the transport protocol, an RTP Header should be added to the existing DIS PDU. Although such a modification is no longer strictly compliant with DIS over-the-wire formats, it nevertheless provides and interesting opportunity for research and testing.

As a product of this thesis, a new Java class for extending the DIS PDU with RTP header information was created. This class is specifically designed to be part of the DIS package (`mil.nps.navy.dis`) used by the DIS-Java-VRML application, a component of the vrtp protocol (Brutzman, 99). This class was named RtpHeader. Appendix F contains the RtpHeader Javadoc and Appendix G contains the RtpHeader source code.

The RtpHeader class was briefly tested during the period of this thesis. Client software automatically discriminates among DIS PDUs with and without RTP headers. This is an excellent result. Further testing is required as a future work.

E. SUMMARY

The rtpMonitor application has successfully been tested with different versions of the Java Development Kit (JDK) and Java Media Framework (JMF). The program proved to be robust in several long-term monitoring sessions. Despite most JMF problems have been solved in version 2.0 Early Access, additional fix should be done.

A new header for the Protocol Data Units (PDU) of the DIS-Java-VRML application was developed and initial test produced excellent results.

THIS PAGE LEFT INTENTIONALLY BLANK

VIII. CONCLUSIONS AND RECOMMENDATIONS

A. RESEARCH CONCLUSIONS

The RTP monitor application has been successfully implemented using Java Media Framework (JMF). The monitor can be applied to help detecting problems in RTP based multicast session by adding statistics recording capabilities, not available in the existing individual conference applications. The presentation of RTP statistics and feedback reports in a single screen is also a good feature for on-line monitoring. The RtpMonitor class package can be used by future RTP applications, as a simple means to record statistics with no direct access needed to JMF resources. This is an important new capability, since core Java classes do not provide access to IGMP (Deering, 89) packets on the network layer and would otherwise require additional programming of native code (e.g. C source code) via the Java Native Interface (JNI) (Sun, 99) to achieve RTCP capabilities.

B. RECOMMENDATIONS FOR FUTURE WORK

A number of excellent opportunities for future work are now possible.

1. Participants Information

RTCP Session Description Reports convey several data about each participant, as name, e-mail and tool being used. This information can be added to RTPMonitor for presentation and recording purposes. JMF contains the necessary classes and methods to set, send and retrieve this data.

2. Extensible Markup Language (XML) Recording

rtpMonitor records the RTP session data as text files. The use of Extensible Markup Language (XML) (Word Wide Web Consortium, 99) for recording the data would allow an easier data retrieval. Java-based XML parsers are available and can provide an efficient way to read and write Java data structures as XML documents and vice-versa.

3. Recorded Data Analysis and Presentation

A tool to retrieve remotely (or receive) the rtpMonitor output file statistics is needed. It should be able to present the each individual data in a graphical format, covering a period of time defined by the user. Again XML provides browsing options for automatic presentation of such data by any web browser.

4. Session Description Protocol (SDP) Reception

The Session Description Protocol (SDP) is intended for describing multimedia sessions for the purposes of session announcement, session invitation, and other forms of multimedia session (IETF, 98). In rtpMonitor the user has to enter the RTP session address. Future work needs to add SDP reception capability to simplify starting a monitoring session and allowing the detection of ongoing sessions, in addition to the current approach.

5. RtpMonitor Activation from SDR

Session Directory (SDR) is a session directory tool designed to allow the advertisement and joining of multicast conferences on the MBone (UCL, 99). It is

possible to launch external application from SDR by modifying some of the SDR configuration files. Providing a plug-in file to SDR might result in an easier activation of the rtpMonitor application.

6. JMF Extensibility for Other Media

Java Media Framework (JMF) distribution comes with a set of concrete classes to implement the RTP transmission, reception and playback of audio and video. However, the extension of JMF to support other types of media is highly desirable. An important area of future work is the implementation of simulation data transmission and reception using the RTP API of Java Media Framework. This work will have direct application in the vrtp streaming behaviors stack.

7. Automated Network Monitoring of RTP Streams for VRTP.

The rtpMonitor class library can be integrated with the vrtp protocol to allow the automatic monitoring capability in vrtp sessions. For this purpose the rtpMonitor library must be updated to support transmission statistics and direct activation by vrtp components. This is a good area of study for agent-based network monitoring, diagnosis and problem correction.

8. Design Patterns Course in Computer Science Curriculum

Design Patterns study helps to solve recurring design problems by using common adopted solutions. Furthermore, the Design Patterns nomenclature provides a precise way of communicating design ideas among software engineers. The inclusion of a Design Patterns course in the Computer Science and MOVES Curricula is highly recommended.

THIS PAGE LEFT INTENTIONALLY BLANK

APPENDIX A. PREPARING UML DIAGRAMS USING RATIONAL ROSE

UML Diagrams Preparation

Rational Rose version 98i can represent the following types of UML diagrams:

- Use Case Diagrams
- Class Diagrams
- Collaboration Diagrams
- Sequence Diagrams
- Component Diagrams
- Deployment Diagrams

The program is capable of handling different models (projects), each model being a set of basic components (classes, interfaces, actors and associations) and diagrams. For better visualization the project is organized in a tree-like structure having the following main branches:

- Use Case View – usually contains the Use Case Diagrams and actors components.
- Logical View – usually contains the Class Diagrams, Collaboration Diagrams Sequence Diagrams, Class Components and Interface Components.
- Component View – contains the Component Diagrams.
- Deployment View – contains a Deployment Diagram.

To create a new diagram, the user has to select the branch he wants the diagram in, click with the mouse right button and select what type of diagram is to be created. A new blank diagram is shown and the components can be created and placed on the diagram by selecting the appropriate component in a tool bar and clicking in the place the component must be positioned in the diagram. Depending on the type of the component different information should be provided. By right clicking on a component a pull-down menu is shown and the user can invoke the component specification as long as the option-setting feature.

Reverse Engineering of Java Source Code

Reverse engineering is the process of creating or updating a model by analyzing Java source code. As Rational Rose reverse engineers each .java or .class file, it finds the classes and objects in the file and includes them in the model.

To reverse engineer all or part of a Java application:

1. If you are updating an existing model, open the model, otherwise create a new model.

2. On the Tools menu, point to Java, and then click Reverse Engineer.

3. From the directory structure , select the classpath setting and folder where the files you're reverse engineering are located. (If the list is empty, check your classpath. For details, see How Rose J Models the Classpath and Extending the Java Classpath .)

4. Set the Filter to display the type of the Java files whose code you want to reverse engineer (.java or .class files).

5. Do one of the following to place the Java files of the type you selected into the Selected Files list:

- In the File list box, select one or more individual files and click Add.
- Click Add All
- Click Add Recursive

6. Select one or more files in the Selected Files box or click Select All to confirm the list of files to reverse engineer.

7. Click Reverse to create or update your model from the Java source you specified. An error dialog displays, if any errors occur during reverse engineering.

8. Check the Rose Log for a listing of any errors that might have occurred.

The procedure above is described in the Rational Rose Help. An additional requirement for reverse engineering of Java code is that the JDK API definitions must be available by adding the file `c:\jdk1.2.2\jre\lib\rt.jar` to the classpath.

Using the reverse engineering feature, Rational Rose will import from Java code all classes, interfaces and associations, but the diagrams are not automatically generated.

rtpMonitor User Manual
version 1.0

Contents:

1. Execution
2. Defining a session
3. Session Bookmarks
4. Preferences
5. Starting a session
6. Statistics
7. Stopping a session
8. Recording monitoring data
9. Using the monitor without GUI
10. Wrong behaviors and results (bugs)
11. Running the program in other directory
12. Reinstallation recommendations

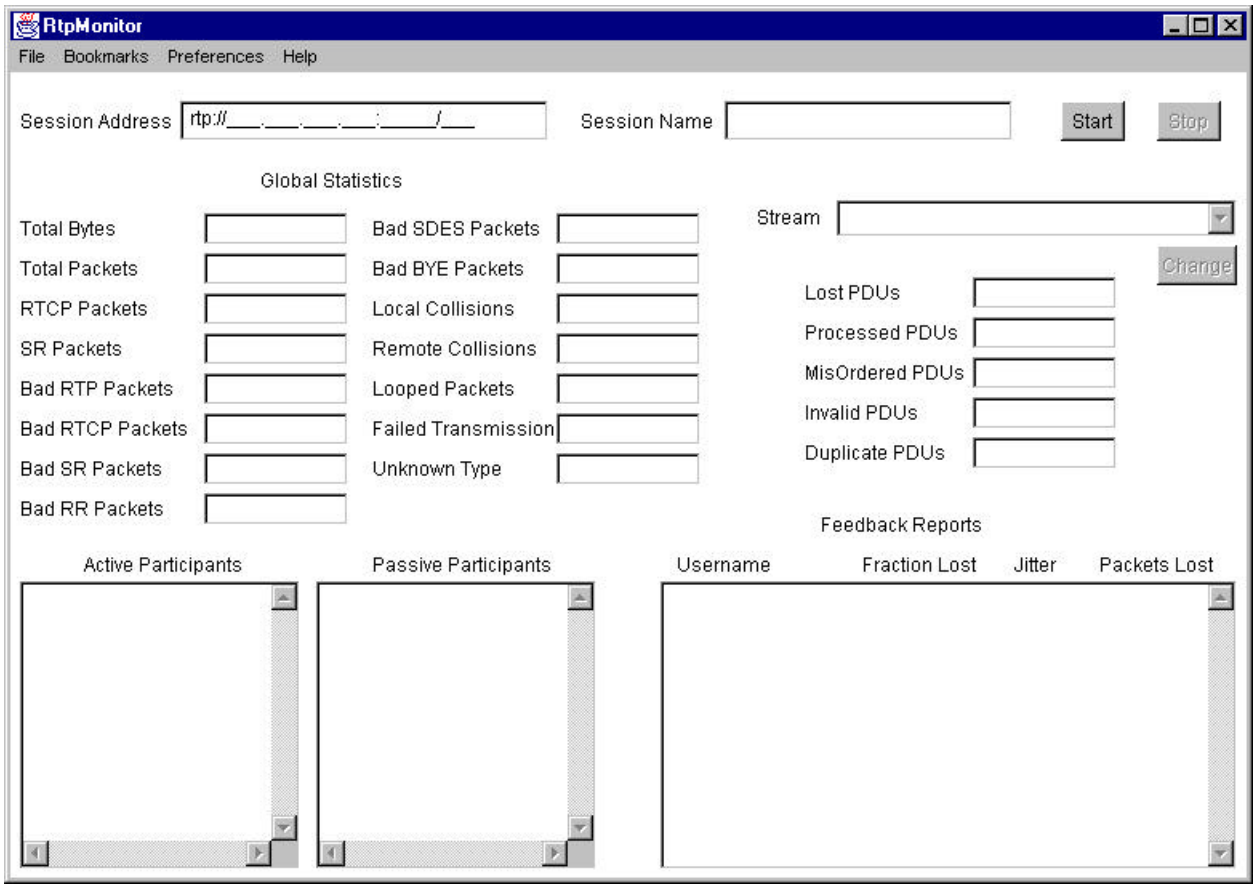
1. Execution

a) In the command line type:

```
cd \vrtp\rtpMonitor
java org.web3d.vrtp.rtp.RtpMonitor
```

b) or run the batch file rtpMonitor.bat in the c:\vrtp\rtpMonitor directory. There is a Windows shortcut to this file in the same directory. This shortcut can be copied to the Desktop to create an icon for the rtpMonitor program.
Using this call with no arguments the GUI version of the program will be executed. The following window is presented:

2.



Defining a session

In the session box enter the session address/port/ttl as in the example below:

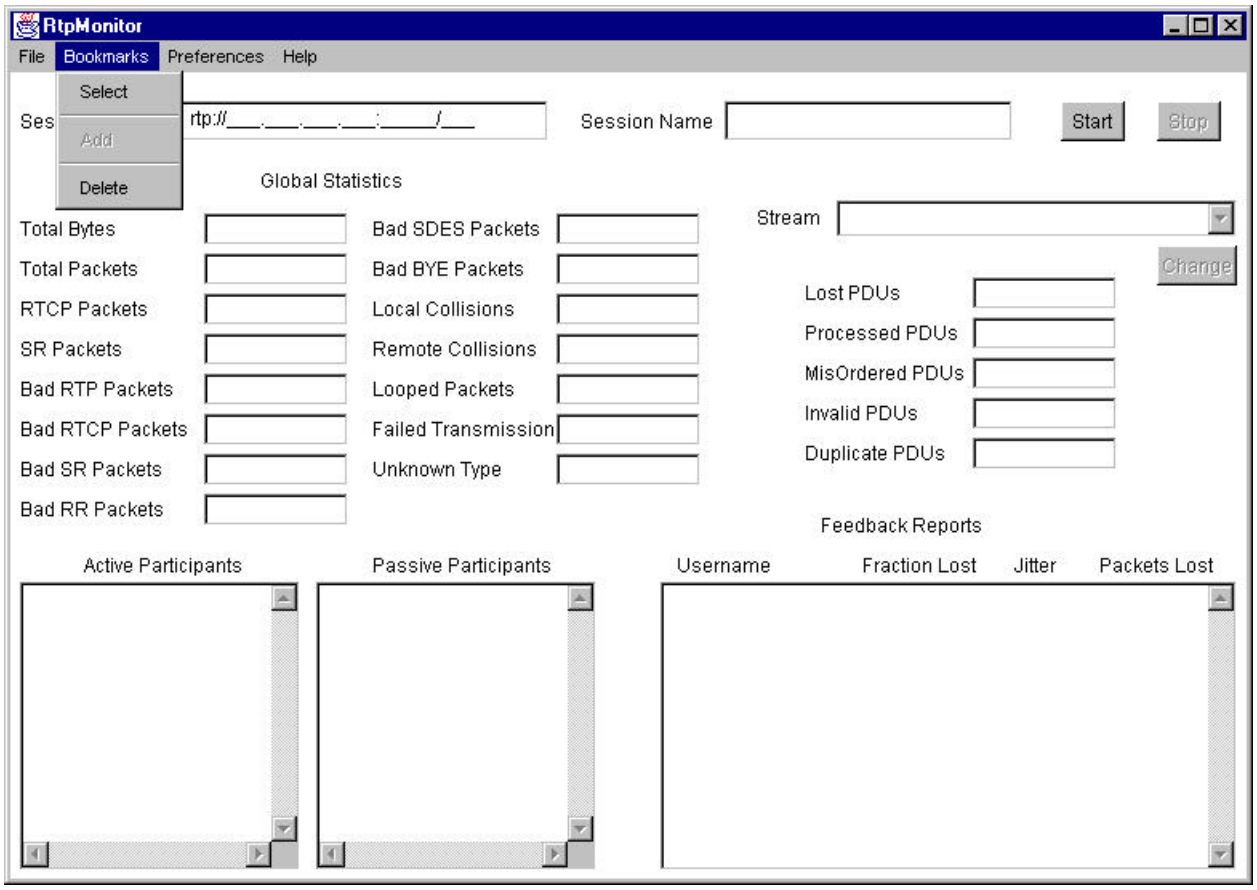
rtp://224.2.125.60:55690/127

where :

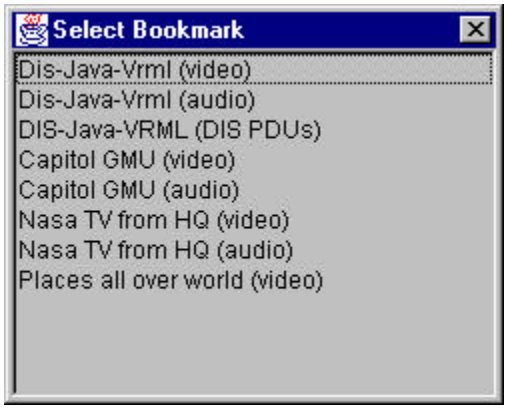
Multicast IP address: 224.2.128.60
RTP port: 55690
TTL : 127

3. Session Bookmarks

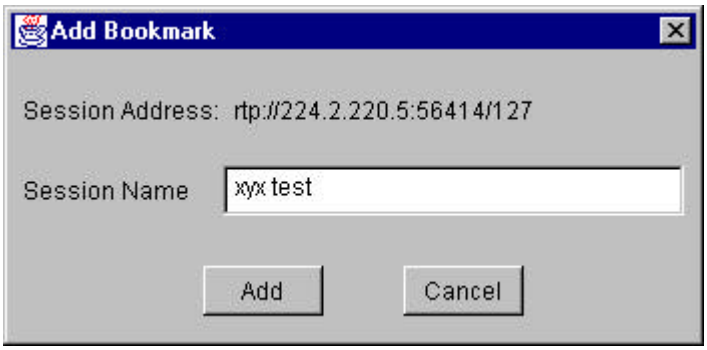
As an option to writing the session address, it is possible to select a session bookmark. The program already comes with some pre-defined session bookmarks. Bookmarks can be added and deleted.



The option "Select" displays a window with the pre-defined session bookmarks. Click over the desised session to select it.



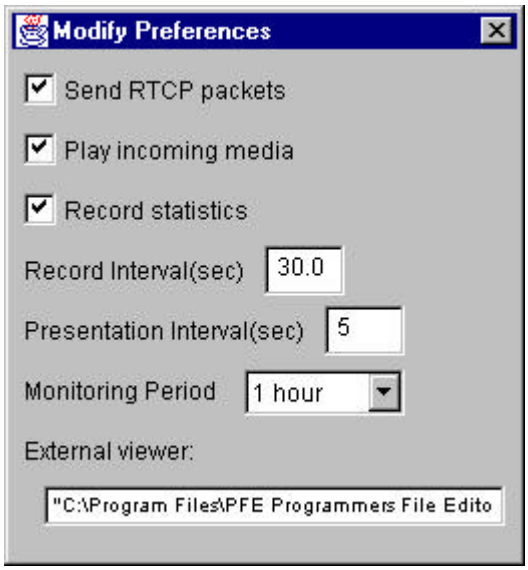
The option "Add" allows the insertion of a new session bookmark. This option is enabled only when a session has been started. The session address is the session of the current session. The session name will be the bookmark name.



The option "delete" allows the deletion of a bookmark. A window with the existing bookmarks will be presented and the user can select the bookmark to be deleted and click the "Delete" button.

4. Preferences

Before starting to monitor a session, the user should set up the program preferences. The menu "Preferences" allows the verification and modification of the program preferences. Selecting Preferences -> Modify the following window will be presented:



The options are:

- Send RTCP packets checkbox: defines if the monitor will participate in the session, sending RTCP packets.
- Play incoming media checkbox: defines if the monitor will play the received streams (audio or video). For video, a new playing window will be created for each active participant stream. For audio, only one playing window will be created.
- Record statistics checkbox: defines if the monitor will record the session statistics in files.
- Record Interval textbox: the user can enter the interval between recorded data, in seconds (default = 30 sec).
- Presentation Interval textbox: the user can enter the interval between data updates on screen, in seconds (default = 5 sec).
- Monitoring Period choicebox: allows the user to specify the duration of the monitoring session. After the time is over the program will exit automatically. The maximum allowed duration is one week (default = 1 hour).
- External viewer textbox: defines an external text editor that will be called to present the output files generated by the monitor (default = MS Windows Wordpad).

The selected preferences will be saved on file and will be available in the next time the program is executed.

5. Starting a session

After entering the session address and the desired option, the user can click on the start button to start a monitoring session. If the supplied session address is invalid or if any other problem in establishing a session occurs an error message will be displayed in the feedback text area.

6. Statistics

The program displays the following types of information about the session:

- Global statistics: general information about the whole session.
- Active Participants: the username of the participants actually sending data streams.
- Passive Participants: the username of the participants that do not send any stream.
- Stream: the stream that is currently being monitored. That stream can be changed in a session with multiple incoming streams by clicking in the "change" button and selecting another stream in the stream selection box.
- Feedback: that display area presents the RTCP feedback data related to the selected stream. Usually the participants in the session send feedback data about all streams.

7. Stopping a session

Clicking the "stop" button can stop a session. After stopping a session, the user is allowed to change the session address and preferences before starting monitoring again.

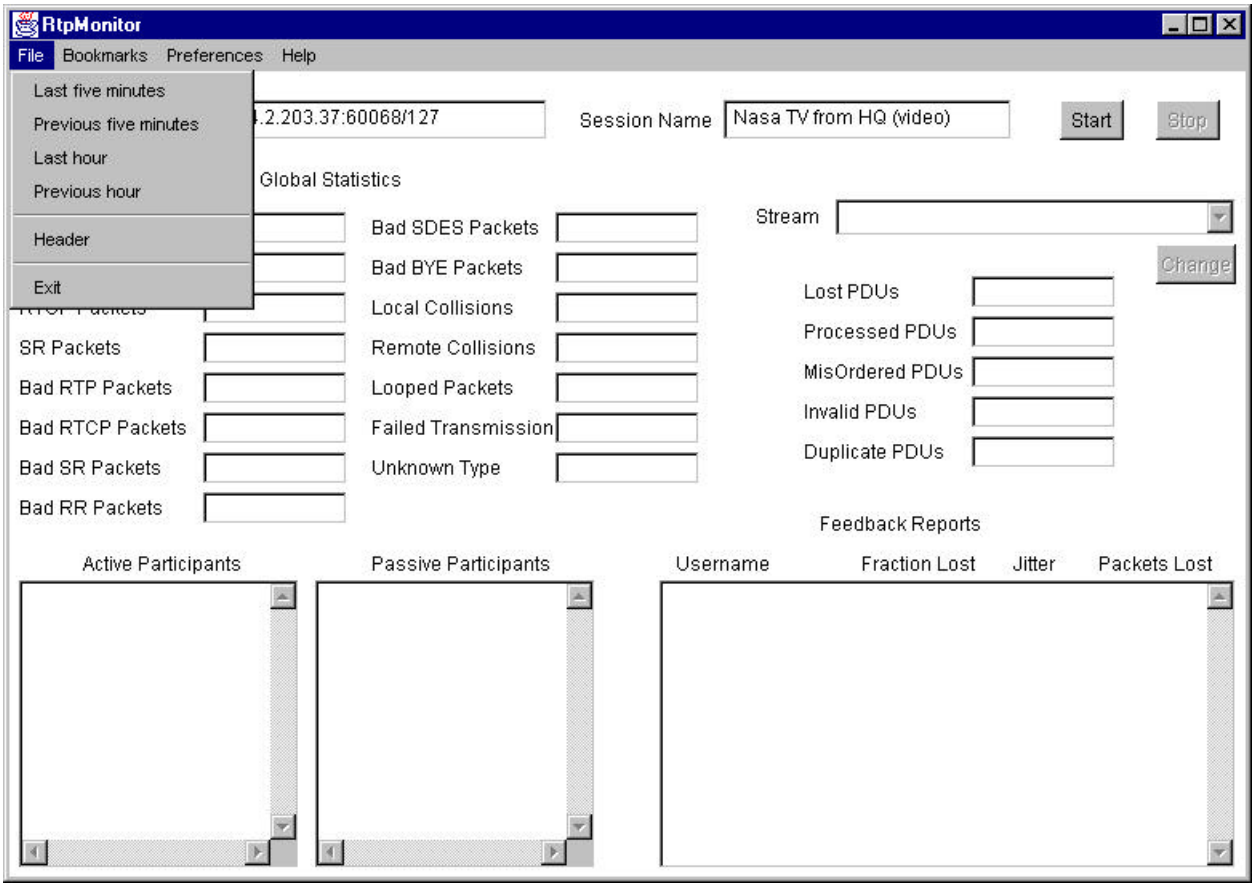
8. Recording monitoring data

A new subdirectory is created for each session address, with the following name: session [IPaddress] port [port number]

Inside this subdirectory several files will be generated and updated during a recording session. They are:

- statisticsHeader.txt: contains the description of the fields being stored.
- statisticsLastFiveMinutes.txt: contains the last five minutes block of statistics.
- statisticsPreviousFiveMinutes.txt: contains the previous five minutes block of statistics.
- statisticsLastHour.txt: contains the last hour block of statistics.
- statisticsPreviousHour.txt: contains the previous hour block of statistics.
- statisticsDateMM-DD-YYYY: contains statistics taken in described date.
- statisticsLastReport.txt – contains the last single report.

There is also a file called LastDateRef.txt, which contains the last monitoring date for that session. It is possible to see some of these files using the menu "File".



The external viewer defined in the preferences will be called to present the output files. These options can be executed during a monitoring session. Below is an example of the Last five minutes file using MS Windows WordPad editor as an external viewer. The format of the data is stored in the file statisticsHeader.txt, also shown below.

statisticsLastFiveMinutes.txt - Notepad

File Edit Search Help

```
D1 10:25:27 9 8 1 5651647 6334 358 27 0 0 0 0 0 0 0 0 0 0 0
D2 ellery@131.182.10.250 1564101640 903 5976 0 0 0
D3 joelja@128.223.214.27 1569838982 1564101640 0.0 8 0
D3 brutzman@accelerate 4285065786 1564101640 0.0390625 803 3026
D3 ellery@131.182.10.250 1564101640 1564101640 0.0 1 0
D3 miyake@128.223.83.29 1568747521 1564101640 0.0 40 0
D3 mmei@141.78.3.242 398571242 1564101640 0.0546875 17478 0
D1 10:25:57 9 8 1 6893963 7723 428 33 0 0 0 0 0 0 0 0 0 0 0
D2 ellery@131.182.10.250 1564101640 967 7296 0 0 0
D3 joelja@128.223.214.27 1569838982 1564101640 0.0 8 0
D3 brutzman@accelerate 4285065786 1564101640 0.09375 918 3047
D3 ellery@131.182.10.250 1564101640 1564101640 0.0 1 0
D3 miyake@128.223.83.29 1568747521 1564101640 0.0 42 0
D3 mmei@141.78.3.242 398571242 1564101640 0.1015625 17614 0
D1 10:26:27 9 8 1 8225068 9184 497 38 0 0 0 0 0 0 0 0 0 0 0
D2 ellery@131.182.10.250 1564101640 1069 8688 0 0 0
D3 joelja@128.223.214.27 1569838982 1564101640 0.0 8 0
D3 brutzman@accelerate 4285065786 1564101640 0.0546875 1023 2953
D3 ellery@131.182.10.250 1564101640 1564101640 0.0 1 0
D3 miyake@128.223.83.29 1568747521 1564101640 0.0 44 0
D3 mmei@141.78.3.242 398571242 1564101640 0.08984375 17730 0
D1 10:26:57 9 8 1 9499537 10587 556 43 0 0 0 0 0 0 0 0 0 0 0
D2 ellery@131.182.10.250 1564101640 1214 10031 0 0 0
D3 joelja@128.223.214.27 1569838982 1564101640 0.0 8 0
D3 brutzman@accelerate 4285065786 1564101640 0.015625 1043 3332
D3 ellery@131.182.10.250 1564101640 1564101640 0.0 1 0
D3 miyake@128.223.83.29 1568747521 1564101640 0.0 44 0
D3 mmei@141.78.3.242 398571242 1564101640 0.08984375 17866 0
D1 10:27:27 9 8 1 10912065 12114 627 47 0 0 0 0 0 0 0 0 0 0 0
D2 ellery@131.182.10.250 1564101640 1230 11488 0 0 0
D3 joelja@128.223.214.27 1569838982 1564101640 0.0 8 0
```

statisticsHeader.txt - Notepad

File Edit Search Help

```
H1 Time TotalParticipants RemoteParticipants ActiveParticipants TotalBytes
H2 CNAME SSRC LostPDU ProcessedPDU MisorderedPDU InvalidPDU DuplicatePDU
H3 CNAME FromSSRC AboutSSRC FractionLost PacketsLost Jitter
```

Each line of data is preceded with a header indicator (D1, D2 or D3), which indicates the line of the statisticsHeader file that contains the description of the data being stored (H1, H2 or H3).

9. Using the monitor without GUI

The program can be executed without the GUI (no statistics are presented) by passing the session address and options data via the command line. The format is:

java org.web3d.vrtp.rtp.RtpMonitor sessionAddress [options]

The options are:

-part : the monitor participates in the session (sends RTCP packets as a receiver in the session)

-play : the monitor play streams

-record : the monitor records statistics

-i nnn : nnn defines the recording interval in seconds (default 30s)

-e ppp : ppp defines the monitoring duration in hours (default: 168 hours = 1 week)

-help: displays the options on the console.

Example: *java org.web3d.vrtp.rtp.RtpMonitor rtp://224.120.67.46/64542/127 -play -record -e 24*

Action: runs the program for monitoring the session in the IP address 224.120.67.46, port 64542, with TTL = 127. It does not participate in the session, but plays the incoming streams and records statistics on files. The recording interval will be 30 seconds and the monitoring duration will be 24 hours.
To stop the program press <Ctrl-C->

10. Wrong behaviors and results (bugs)

Several early bugs were related to JMF1.1 and were corrected in JMF2.0 Early Access. A new bug was observed in JMF 2.0 Early Access: the Cumulative Number of Packets Lost (Packets Lost in the Feedback Area) is returning wrong values after some time. This problem was reported to Sun's JMF-bugs list in 08-Jun-1999, but it is still present in JMF2.0 Beta.

11. Running the program in other directory

To run the program in another directory copy the files "bookmarks.txt" and "Header.txt" to the new directory. Then run rtpMonitor from this directory.
Example:

```
copy bookmarks.txt c:\Mydir
copy Header.txt c:\Mydir
cd \Mydir
java org.web3d.vrtp.rtp.RtpMonitor
```

12. Reinstallation recommendations

In order to continue using the previous bookmarks and preferences, the files "bookmark.txt" and "preferences.txt" in the directory "\vrtp\rtpMonitor" should be saved and restored after the new installation.

To reinstall the program it is recommended the deletion of the following directories:

- \vrtp\javadoc\rtpMonitor
- \vrtp\org\web3d\vrtp\rtp
- \vrtp\rtpMonitor

THIS PAGE LEFT INTENTIONALLY BLANK

APPENDIX C. RTPMONITOR JAVADOC

The rtpMonitor Javadoc is available at:

http://www.web3D.org/WorkingGroups/vrtp/javadoc/rtpMonitor/index.html

APPENDIX D. RTPMONITOR SOURCE CODE

```
package org.web3d.vrtp.rtp;

import java.awt.*;
import java.awt.event.*;
import java.util.*;

/**
 * A Dialog to display information about the program
 * e.g. version, date
 * <P>
 *
 * @author  Francisco Afonso (afonso@cs.nps.navy.mil)
 * @version 1.0
 */
public class About extends Dialog {

    /**
     * Constructor.
     * <P>
     * @param parent the parent frame
     */
    public About(Frame parent){
        super( parent, "About" , true );
        setSize( 270 , 200 );
        setLayout(null);
        setBackground(Color.lightGray);

        addWindowListener( new CloseWindow() );

        Label versionLabel = new Label( "rtpMonitor version 1.0");
        versionLabel.setBounds( 20, 30, 150, 25 );
        add(versionLabel);

        Label versionDateLabel = new Label( "Version date: 30 August 1999");
        versionDateLabel.setBounds( 20, 60, 180, 25 );
        add(versionDateLabel);

        Label npsLabel = new Label( "Naval Postgraduate School");
        npsLabel.setBounds( 20, 90, 150, 25 );
        add(npsLabel);

        Label vrtpLabel = new Label( "virtual reality transfer protocol (vrtp)");
        vrtpLabel.setBounds( 20, 120, 200, 25 );
        add(vrtpLabel);

        Label siteLabel = new Label( "http://www.web3d.org/WorkingGroups/vrtp/");
        siteLabel.setBounds( 20, 150, 240, 25 );
        add(siteLabel);
    }

} // end of class About
```

```
package org.web3d.vrtp.rtp;

import java.awt.*;
import java.awt.event.*;
import java.util.*;
import java.io.*;

/**
 * The Dialog to add a session bookmark.
 * <P>
 *
 * @author  Francisco Afonso (afonso@cs.nps.navy.mil)
 * @version 1.0
 */
public class AddBookmark extends Dialog
                            implements ActionListener{

    Vector sessionAddressVec, sessionNamesVec;
    Label sessionAddLabel, sessionAddRealLabel, sessionNameLabel;
    TextField sessionNameText;
    Button addButton, cancelButton;
    RtpMonitor theParent;


    /**
     * Constructor.
     * <P>
     * @param parent the parent frame
     */
    public AddBookmark(Frame parent){
        super( parent, "Add Bookmark" , true );
        setSize( 350 , 170 );
        setLayout(null);
        setBackground(Color.lightGray);

        addWindowListener( new CloseWindow() );

        theParent = (RtpMonitor) parent;

        loadBookmarks();

        sessionAddLabel = new Label( "Session Address:");
        sessionAddLabel.setBounds( 10, 40, 100, 25 );
        add(sessionAddLabel);

        sessionAddRealLabel = new Label( theParent.sessionText.getText());
        sessionAddRealLabel.setBounds(115,40, 230, 25 );
        add(sessionAddRealLabel);

        sessionNameLabel = new Label( "Session Name");
        sessionNameLabel.setBounds( 10, 80, 100, 25 );
        add(sessionNameLabel);

        sessionNameText = new TextField(theParent.sessionNameText.getText());
        sessionNameText.setBounds(110, 80, 230, 25);
        add(sessionNameText);
```

```
     addButton = new Button("Add");
     addButton.setBounds(100,130,60,25);
     addButton.addActionListener(this);
     add(addButton);

     cancelButton = new Button("Cancel");
     cancelButton.setBounds(200,130,60,25);
     cancelButton.addActionListener(this);
     add(cancelButton);

 }


 /**
  * Takes action when buttons are selected.
  *
  */
public void actionPerformed( ActionEvent e )
{
    if( e.getSource() == addButton){
        String sessAdd  = theParent.sessionText.getText().trim();
        String sessName = sessionNameText.getText().trim();
        if( !sessName.equals("") ){
            sessionNamesVec.addElement(sessName);
            sessionAddressVec.addElement(sessAdd);
            saveBookmarks();
            theParent.sessionNameText.setText( sessName );
            setVisible(false);
        }
    }

    if( e.getSource() == cancelButton){
        setVisible(false);
    }
 }


/**
 * Loads the session bookmarks from file "bookmarks.txt"
 */
private void loadBookmarks(){

   sessionAddressVec = new Vector();
   sessionNamesVec = new Vector();


   try{
      BufferedReader input = new BufferedReader( new
                                      FileReader("bookmarks.txt") );
      String line;

      while( (line = input.readLine()) != null){
         int pos = line.lastIndexOf("rtp://");
         if (pos != -1){
            sessionAddressVec.addElement( line.substring(pos).trim() );
            sessionNamesVec.addElement( line.substring( 0 , pos ).trim() );
         }
```

```
         }
         input.close();
      }
      catch ( FileNotFoundException e){
         System.out.println( "Add Bookmarks: " + e.getMessage() );
      }
      catch ( IOException e){
         System.err.println( "Exception reading bookmark: " + e.getMessage() );
      }

   }

   /**
    * Saves the session bookmarks to file "bookmarks.txt"
    */
    private void saveBookmarks(){
       PrintStream output;

       try {
          output = new PrintStream(new FileOutputStream("bookmarks.txt", false) );

          for(int ii=0; ii < sessionAddressVec.size(); ++ii){
              output.print( (String) sessionNamesVec.elementAt(ii));
              output.print( " " );
              output.println( (String) sessionAddressVec.elementAt(ii) );
          }

          output.close();
        }
        catch (FileNotFoundException e){
          System.out.println( "Saving bookmarks : " + e.getMessage() );
        }
        catch ( IOException e){
          System.err.println( "Exception writing bookmark: " + e.getMessage() );
        }


    }

} // end of class AddBookmark
```

```
package org.web3d.vrtp.rtp;

import java.awt.event.*;

/**
 * This class is used to set a frame/dialog as not visible
 * when the close icon is clicked.
 * @author  Francisco Afonso (afonso@cs.nps.navy.mil)
 * @version 1.0
 */
public class CloseWindow extends WindowAdapter {


   /**
    * set frame/dialog as not visible. This method is activated when the window
    * is closed.
    *
    */
   public void windowClosing( WindowEvent e )
   {
      e.getWindow().setVisible( false );
   }

} // end of class CloseWindow
```

```
package org.web3d.vrtp.rtp;

import java.awt.*;
import java.awt.event.*;
import java.util.*;
import java.io.*;

/**
 * A Dialog to delete a session bookmark.
 * <P>
 *
 * @author  Francisco Afonso (afonso@cs.nps.navy.mil)
 * @version 1.0
 */
public class DeleteBookmark extends Dialog
                            implements ActionListener{

    java.awt.List sessionNamesList;
    Vector sessionAddressVec;
    Button deleteButton, cancelButton;

    /**
     * Constructor.
     * <P>
     * @param parent the parent frame
     */
    public DeleteBookmark(Frame parent){
        super( parent, "Delete Bookmark" , true );
        setSize( 250 , 300 );
        setLayout(null);
        setBackground(Color.lightGray);

        addWindowListener( new CloseWindow() );

        sessionNamesList = new java.awt.List( 5 , false);
        sessionNamesList.setBounds(10,30,230,210);

        loadBookmarks();

        add(sessionNamesList);

        deleteButton = new Button("Delete");
        deleteButton.setBounds(50,260,60,25);
        deleteButton.addActionListener(this);
        add(deleteButton);

        cancelButton = new Button("Cancel");
        cancelButton.setBounds(130,260,60,25);
        cancelButton.addActionListener(this);
        add(cancelButton);

    }
```

```
 /**
  * Takes action when buttons are selected.
  *
  */
public void actionPerformed( ActionEvent e )
{
    if( e.getSource() == deleteButton){
        int index = sessionNamesList.getSelectedIndex();
        if( index >= 0 ){
            sessionNamesList.remove(index);
            sessionAddressVec.remove(index);
            saveBookmarks();
        }
    }

    if( e.getSource() == cancelButton){
        setVisible(false);
    }
 }


/**
 * Loads the session bookmarks from file "bookmarks.txt"
 */
private void loadBookmarks(){

   sessionAddressVec = new Vector();

   try{
      BufferedReader input = new BufferedReader( new
                                          FileReader("bookmarks.txt") );
      String line;

      while( (line = input.readLine()) != null){
         int pos = line.lastIndexOf("rtp://");
         if (pos != -1){
            sessionAddressVec.addElement( line.substring(pos).trim() );
            sessionNamesList.add( line.substring( 0 , pos ).trim() );
         }
      }
      input.close();
   }
   catch ( FileNotFoundException e){
      System.out.println( "Delete Bookmarks: " + e.getMessage() );
   }
   catch ( IOException e){
      System.err.println( "Exception reading bookmark: " + e.getMessage() );
   }

}
```

```
    /**
     * Saves the session bookmarks to file "bookmarks.txt"
     */
     private void saveBookmarks(){
        PrintStream output;

        try {
           output = new PrintStream(new FileOutputStream("bookmarks.txt", false) );

           for(int ii=0; ii < sessionAddressVec.size(); ++ii){
               output.print( sessionNamesList.getItem(ii));
               output.print( " " );
               output.println( (String) sessionAddressVec.elementAt(ii) );
           }

           output.close();
         }
         catch (FileNotFoundException e){
           System.out.println( "Saving bookmarks : " + e.getMessage() );
         }
         catch ( IOException e){
           System.err.println( "Exception writing bookmark: " + e.getMessage() );
         }


     }

} // end of class DeleteBookmark
```

```
package org.web3d.vrtp.rtp;

import javax.media.rtp.*;
import javax.media.rtp.event.*;
import javax.media.rtp.rtcp.*;

import java.awt.*;
import java.util.*;

/**
 * A class used by RtpMonitor objects to periodically launch their showStats
 * methods (screen updates).
 * <P>
 *
 * @author  Francisco Afonso (afonso@cs.nps.navy.mil)
 * @version 1.0
 */
public class DisplayTask implements Runnable {

    RtpMonitor myMon;
    SessionManager mymgr;
    Thread thread;
    int intervalParam;

    /**
     * Constructor. It creates a new thread of execution and calls
     * the method run().
     *
     * @param mon the RtpMonitor object that will be called back for screen
     * updates.
     * @param interval the interval between screen data updates, in seconds.
     */
     public DisplayTask(RtpMonitor mon, double interval){

        myMon = mon;
        intervalParam = (int) (interval*1000);

        thread = new Thread(this,"DisplayTask thread");
        thread.setDaemon(true);
        thread.start();

    }
```

```
    /**
     * This method runs continuously calling the showStats methods of RtpMonitor
     * in the proper presentation interval, until the RtpMonitor stops the
     * session.
     */
     public void run(){
        while(myMon.isMonitoring()){

              myMon.showGlobalStats();
              myMon.showParticipants();
              myMon.showStreamStats();
              myMon.showFeedbacks();

            try{
                Thread.sleep(intervalParam);
            }
            catch (InterruptedException e){}

        }
    }

} // end of class DisplayTask
```

```
package org.web3d.vrtp.rtp;

import java.io.*;
import java.util.*;

/**
 * A class used by RecordTask objects to organize statistics
 * in several files ( five minutes, hour, and day ).
 * <P>
 * This class writes to the file following files:
 * statisticsLastFiveMinutes.txt, statisticsPreviousFiveMinutes.txt,
 * statisticsLastHour.txt, statisticsPreviousFiveMinutes.txt, and
 * statisticsDateMM-DD-YYYY.txt.
 *
 * @author  Francisco Afonso (afonso@cs.nps.navy.mil)
 * @version 1.0
 */
public class FileCatalog{

   Calendar rightNow;
   String prefix;
   String dateRef;
   int minuteRange, hourRange;
   boolean flgFirstDataMin;
   File min5, min5last,hour, hourlast, header;

   /**
    * Constructor. It checks if there are previous data from the same session
    * and transfers these data to the correct statisticsDateMM-DD_YYYY.txt file.
    * Then it clears all "five minutes" and "hour" data files.
    * <P>
    * @param pref a string containing the directory where the data must be written
    */
   public FileCatalog( String pref){

      prefix = pref;

      // creates File objects
      min5 = new File( prefix + "LastFiveMinutes.txt");
      min5last = new File( prefix + "PreviousFiveMinutes.txt");
      hour = new File( prefix + "LastHour.txt");
      hourlast = new File( prefix +"PreviousHour.txt");
      header = new File( prefix + "Header.txt");

      // retrieves the date of last record for this session
      String lastDateRef = retrieveDateRef();

      // copies the header file to the session directory if it does not exit
      if( ! header.exists() ){
         concatenate( prefix + "Header.txt" , "Header.txt" );
      }
```

```
   if( min5.exists()){

      // if there is a last five minutes file, transfers its contents to the
      // last hour file and deletes the last five minutes file
      if( lastDateRef != null ){
        concatenate(prefix + "LastHour.txt" , prefix + "LastFiveMinutes.txt" );
      }
      min5.delete();
   }

   // deletes the previous five minutes file if it exists
   if( min5last.exists()){
      min5last.delete();
   }

   if( hour.exists()){

      // if there is a last hour, transfers its contents to the
      // related date file file and deletes the last hour file
      if( lastDateRef != null ){
         String dateFileName = new String(prefix +"Date" + lastDateRef+".txt");
         concatenate( dateFileName , prefix + "LastHour.txt" );
      }
      hour.delete();
   }

   // deletes the previous hour file if it exists
   if( hourlast.exists()){
      hourlast.delete();
   }

   flgFirstDataMin = true;
}




/**
 * Method that will update the files, transfering data from
 * statisticsLastReport.txt to the appropriate files.
 *
 * @param time the current time as a Calendar object.
 */
public void update( Calendar time){

   rightNow = time;

   // displays the current time on the console
   System.out.println( rightNow.getTime().toString());
   // minuteRangeNow represents a block of five minutes
   int  minuteRangeNow = rightNow.get(Calendar.MINUTE) / 5;

   int  hourRangeNow = rightNow.get(Calendar.HOUR_OF_DAY) ;
   String dateNow = new String( (rightNow.get(Calendar.MONTH) + 1) + "-" +
                                rightNow.get(Calendar.DAY_OF_MONTH)+ "-" +
                                rightNow.get(Calendar.YEAR) );
```

```
   // if it is the first report of the session
   if(flgFirstDataMin){
      minuteRange = minuteRangeNow;
      hourRange = hourRangeNow;
      dateRef = dateNow;
      saveDateRef(dateRef);
      flgFirstDataMin = false;
   }

   if( minuteRangeNow != minuteRange ){

      // if it is new block of five minutes

      // appends data from the five minutes file to the last hour file
      concatenate( prefix + "LastHour.txt" , prefix + "LastFiveMinutes.txt");
      if( min5last.exists()){
         min5last.delete();
      }

      // rename last five minutes file to previous five minutes file
      min5.renameTo(min5last);
      minuteRange = minuteRangeNow;

      if( hourRangeNow != hourRange ){

         // if it is a new hour

         // appends data from the last hour to the related date file
         String dateFileName = new String(prefix + "Date" + dateRef + ".txt" );
         concatenate( dateFileName , prefix + "LastHour.txt" );
         if( hourlast.exists() ){
            hourlast.delete();
         }

         // renames the last hour file as the previous hour file
         hour.renameTo(hourlast);
         hourRange = hourRangeNow;

         // if it is a new date save the actual date in the
         // statisticsLastDateRef file
         if( ! dateRef.equals( dateNow ) ){
            dateRef = dateNow;
            saveDateRef(dateRef);
         }
      }
   }

   // if there is not a new five minutes block just append the last report
   // to the existing last five minutes file
   concatenate( prefix + "LastFiveMinutes.txt", prefix + "LastReport.txt" );

}
```

```
/**
 * Utility to concatenate two files.
 *
 * @param file1 the first file ( file1 <- file1 + file2 ). If file1 does not
 * exits this method will copy file2 to file1.
 * <P>
 * @param file2 the file to be appended to file1.
 */
public static void concatenate( String file1, String file2 ){

   try{
     FileOutputStream output = new FileOutputStream( file1, true );
     FileInputStream input = new FileInputStream( file2 );
     int mybyte;
     while( (mybyte = input.read()) != -1 ){
        output.write(mybyte);
     }

     output.close();
     input.close();

   }
   catch( FileNotFoundException e){
      System.out.println("File not found " + e.getMessage());
   }
   catch( IOException e ){
      System.out.println("IOException : " + e.getMessage());
   }
}


/**
 * This method saves in a file a date in a string format. It is
 * used for save the date of the last report.
 * The file name is statisticsLastDateRef.txt.
 *
 * @param dater a string representing a date as DD-MM-YYYY
 */
public void saveDateRef( String dater ){
   try{
      DataOutputStream output = new DataOutputStream(
                          new FileOutputStream( prefix + "LastDateRef.txt") );

      output.writeUTF( dater );
      output.close();
   }
   catch( FileNotFoundException e){
      System.out.println("File not found " + e.getMessage());
   }

   catch( IOException e ){
      System.out.println("IOException : " + e.getMessage());
   }
}
```

```
   /**
    * This method retrives a date in a string format from the file
    * statisticsLastDateRef.txt.
    *
    * @return string representing a date as DD-MM-YYYY
    */
   public String retrieveDateRef(){

      String result = null;

      try{
         DataInputStream input = new DataInputStream(
                                 new FileInputStream( prefix + "LastDateRef.txt") );

         result = input.readUTF();
         input.close();
      }
      catch( FileNotFoundException e){}

      catch( IOException e ){
         System.out.println("IOException : " + e.getMessage());
      }
      return result;

   }


} // end of class FileCatalog
```

```
package org.web3d.vrtp.rtp;

import java.awt.*;
import java.awt.event.*;
import java.io.*;

/**
 * A Dialog to display and modify the program preferences.
 * <P>
 *
 * @author  Francisco Afonso (afonso@cs.nps.navy.mil)
 * @version 1.0
 */
public class ModifyPreferences extends Dialog {

    protected Checkbox recordBox, partBox, playBox;
    protected Label intervalLabel, intervalPresLabel, endLabel, viewerLabel;
    protected TextField intervalText, intervalPresText, viewerText;
    protected Choice endChoice;
    private   String [] strEndsIn = { "1 hour", "2 hours", "4 hours",
                "8 hours", "12 hours", "24 hours", "2 days", "1 week"};

    /**
     * Constructor.
     * <P>
     * @param parent the parent frame
     */
    public ModifyPreferences(Frame parent){
        super( parent, "Preferences" , true );
        setSize( 260 , 280 );
        setLayout(null);
        setBackground(Color.lightGray);

        addWindowListener( new CloseWindow() );

        partBox = new Checkbox("Send RTCP packets", true);
        partBox.setBounds(10,30,140,25);
        partBox.setBackground(Color.lightGray);
        add(partBox);

        playBox = new Checkbox("Play incoming media", true);
        playBox.setBounds(10,60,160,25);
        add(playBox);

        recordBox = new Checkbox("Record statistics", true);
        recordBox.setBounds(10,90,140,25);
        add(recordBox);

        intervalLabel = new Label( "Record Interval(sec)");
        intervalLabel.setBounds( 10, 120, 120, 25 );
        add(intervalLabel);

        intervalText = new TextField("30.0");
        intervalText.setBounds(130,120,40,25);
        add(intervalText);

        intervalPresLabel = new Label( "Presentation Interval(sec)");
```

```
        intervalPresLabel.setBounds( 10, 150, 140, 25 );
        add(intervalPresLabel);

        intervalPresText = new TextField("5");
        intervalPresText.setBounds(160,150,40,25);
        add(intervalPresText);

        endLabel = new Label( "Monitoring Period");
        endLabel.setBounds( 10, 180, 100, 25 );
        add(endLabel);

        endChoice = new Choice();
        endChoice.setBounds(120, 180, 80, 25);
        add(endChoice);
        for(int ii=0; ii < strEndsIn.length; ++ii){
           endChoice.add(strEndsIn[ii]);
        }
        endChoice.select(0);

        viewerLabel = new Label( "External viewer:");
        viewerLabel.setBounds( 10, 210, 120, 25 );
        add(viewerLabel);

        viewerText = new TextField("c:/Program Files/accessories/wordpad.exe");
        viewerText.setBounds(20,240,230,20);
        viewerText.setFont( new Font( null , Font.PLAIN , 10 ) );
        add(viewerText);

        loadPreferences();

    }

   /**
    * Enables the GUI input elements
    */
    public void enableInput(){
        recordBox.setEnabled(true);
        partBox.setEnabled(true);
        playBox.setEnabled(true);
        intervalText.setEnabled(true);
        intervalPresText.setEnabled(true);
        endChoice.setEnabled(true);
        viewerText.setEnabled(true);
    }

   /**
    * Disables the GUI input elements
    */
     public void disableInput(){
        recordBox.setEnabled(false);
        partBox.setEnabled(false);
        playBox.setEnabled(false);
        intervalText.setEnabled(false);
        intervalPresText.setEnabled(false);
        endChoice.setEnabled(false);
        viewerText.setEnabled(false);
    }
```

```
    /**
     * Saves the preferences in the file preferences.txt
     */
     public void savePreferences(){
       try{
          DataOutputStream output = new DataOutputStream(
                                       new FileOutputStream("preferences.txt") );

          output.writeBoolean( partBox.getState());
          output.writeBoolean( playBox.getState());
          output.writeBoolean( recordBox.getState());
          output.writeUTF(intervalText.getText());
          output.writeUTF(intervalPresText.getText());
          output.writeInt(endChoice.getSelectedIndex());
          output.writeUTF(viewerText.getText());

          output.close();
       }
       catch( FileNotFoundException e){
          System.out.println("File not found " + e.getMessage());
       }

       catch( IOException e ){
          System.out.println("IOException : " + e.getMessage());
       }
    }

    /**
     * Loads the preferences from the file preferences.txt
     */
    private void loadPreferences(){

       try{
          DataInputStream input = new DataInputStream(
                                       new FileInputStream( "preferences.txt") );

          partBox.setState(input.readBoolean());
          playBox.setState(input.readBoolean());
          recordBox.setState(input.readBoolean());
          intervalText.setText(input.readUTF());
          intervalPresText.setText(input.readUTF());
          endChoice.select(input.readInt());
          viewerText.setText(input.readUTF());

          input.close();
       }
       catch( FileNotFoundException e){}

       catch( IOException e ){
          System.out.println("IOException : " + e.getMessage());
       }

    }

} // end of class ModifyPreferences
```

```
package org.web3d.vrtp.rtp;

import javax.media.rtp.*;
import javax.media.rtp.event.*;
import javax.media.rtp.rtcp.*;

import java.io.*;
import java.sql.*;
import java.util.*;

/**
 * A class used by RtpMonitorManager objects to write the statistics
 * periodically to disk. It is created as a separate thread that waits for
 * a fixed period of time after writing data to disk.
 * <P>
 * This class writes data to the file called "statisticsLastReport.txt", that
 * contains only the last single report. The FileCatalog class is actually
 * responsible for transferring the data to the file
 * "statisticsLastFiveMinutes.txt" and other files.
 *
 * @author  Francisco Afonso (afonso@cs.nps.navy.mil)
 * @version 1.0
 */
public class RecordTask implements Runnable {

    SessionManager mgr;
    RtpMonitorManager myMon;
    Thread thread;
    boolean flgRun;
    PrintStream output;
    String dir;
    String prefix;
    String filename;
    Calendar timeNow;
    FileCatalog cat;
    int recordParam;

   /**
    * Constructor. It creates the session directory where the stats files will be
    * written. Also it creates a FileCatalog object that will organize the data in
    * several files (five minutes, hour and day).
    *
    *
    * @param mon the RtpMonitorManager that manages the monitoring session.
    * <P>
    * @param recInterval the interval between statistic samples, in seconds.
    */
    public RecordTask(RtpMonitorManager mon, double recInterval) {

        // saves parameters as internal variables
        myMon = mon;
        recordParam = (int) (recInterval * 1000);

        // gets the session manager associated with the monitoring session
        mgr = myMon.getSessionManager();
```

```
    // gets session address and port
    String session = myMon.getMediaLocator().getSessionAddress();
    String port =
       (new Integer(myMon.getMediaLocator().getSessionPort())).toString();

    // creates a new subdirectory for saving session statistics
    dir = new String( "./session" + session.replace('.','-') + "port" +port );
    File newdir = new File(dir);
    newdir.mkdir();

    // creates file statisticsLastReport.txt
    prefix = new String( dir + "/statistics" );
    filename = new String ( prefix + "LastReport.txt" );

    // creates a FileCatalog object
    cat = new FileCatalog( prefix );

    // executes as a thread
    thread = new Thread(this,"Record thread");
    thread.setDaemon(true);
    thread.start();

 }


/**
 * Resets a flag that is checked each time the thread associated with this
 * object is awaken after the wait command, causing the thread to end.
 *
 */
 public void exit(){
    flgRun = false;
 }

/**
 * Starts executing the recording and waiting until the recording interval
 * is over, in a loop, until the "exit" method is called.
 *
 */
 public void run(){

    flgRun = true;

    // runs continuously until stopped
    while(flgRun){

        try {

          // opens the last report file
          output = new PrintStream( new FileOutputStream( filename, false) );

          // gets actual time
          timeNow = Calendar.getInstance();

          // records statistics on file
          recordGlobalStats();
          recordStreamStats();
```

```
            recordFeedbacks();

            // calls the FileCatalog object to manage the data between the
            // several files
            cat.update(timeNow);

          }
          catch (FileNotFoundException e){
             System.out.println( "In RecordTask : " + e.getMessage() );
          }
          finally {
           output.close();
          }
          try{
             Thread.sleep(recordParam);
          }
          catch (InterruptedException e){}

      }
  }


/**
 * Records global statistics ( lines starting with D1 )
 *
 */
 private void recordGlobalStats(){

   GlobalReceptionStats stats = mgr.getGlobalReceptionStats();

   output.print( "D1 ");
   output.print( (new Time(timeNow.getTime().getTime()).toString() ) );

   output.print(' ');
   output.print(mgr.getAllParticipants().size() );
   output.print(' ');
   output.print(mgr.getRemoteParticipants().size() );
   output.print(' ');
   output.print(mgr.getActiveParticipants().size() );
   output.print(' ');
   output.print(stats.getBytesRecd() );
   output.print(' ');
   output.print(stats.getPacketsRecd() );
   output.print(' ');
   output.print(stats.getRTCPRecd() );
   output.print(' ');
   output.print(stats.getSRRecd() );
   output.print(' ');
   output.print(stats.getBadRTPkts() );
   output.print(' ');
   output.print(stats.getBadRTCPPkts() );
   output.print(' ');
   output.print(stats.getMalformedSR() );
   output.print(' ');
   output.print(stats.getMalformedRR() );
   output.print(' ');
   output.print(stats.getMalformedSDES() );
```

```
    output.print(' ');
    output.print(stats.getMalformedBye() );
    output.print(' ');
    output.print(stats.getLocalColls() );
    output.print(' ');
    output.print(stats.getRemoteColls() );
    output.print(' ');
    output.print(stats.getPacketsLooped() );
    output.print(' ');
    output.print(stats.getTransmitFailed() );
    output.print(' ');
    output.println(stats.getUnknownTypes() );

  }

 /**
  * Records stream statistics ( lines starting with D2 )
  *
  */
  private void recordStreamStats(){

    ReceiveStream stream;
    ReceptionStats stats;
    Participant part;
    long SSRC;
    String CNAME;

    Vector aux = mgr.getReceiveStreams();
    for(int ii = 0; ii< aux.size(); ++ii){
       stream = (ReceiveStream) aux.elementAt(ii);
       part = stream.getParticipant();
       SSRC = RtpUtil.correctSSRC(stream.getSSRC());
       if(part != null){
          CNAME = stream.getParticipant().getCNAME();
       }
       else {
          CNAME = new String("Unknown Participant");
       }
       stats = stream.getSourceReceptionStats();

       output.print( "D2 ");
       output.print( CNAME );
       output.print(' ');
       output.print(SSRC);
       output.print(' ');
       output.print(stats.getPDUlost() );
       output.print(' ');
       output.print(stats.getPDUProcessed() );
       output.print(' ');
       output.print(stats.getPDUMisOrd() );
       output.print(' ');
       output.print(stats.getPDUInvalid() );
       output.print(' ');
       output.println(stats.getPDUDuplicate() );
    }
  }
```

```
    /**
     * Records feedback statistics ( lines starting with D3 )
     *
     */
    private void recordFeedbacks(){

         Participant part;
         Vector reports, feedbacks;
         Report rep;
         Feedback feedbk;
         String CNAME;
         long fromSSRC, aboutSSRC;
         double fraction;
         long packetsLost, jitter;

         Vector aux = mgr.getAllParticipants();

         for(int ii = 0; ii< aux.size(); ++ii){
             part = (Participant)aux.elementAt(ii);
             CNAME = part.getCNAME();
             reports = part.getReports();
             for( int jj=0; jj< reports.size(); ++jj){
                 rep = (Report) reports.elementAt(jj);
                 fromSSRC = RtpUtil.correctSSRC( rep.getSSRC() );
                 feedbacks = rep.getFeedbackReports();
                 for( int kk=0; kk < feedbacks.size(); ++kk){
                    feedbk = (Feedback) feedbacks.elementAt(kk);
                    aboutSSRC = RtpUtil.correctSSRC( feedbk.getSSRC() );
                    fraction = (feedbk.getFractionLost())/256.0;
                    packetsLost = feedbk.getNumLost();
                    jitter = feedbk.getJitter();

                    output.print( "D3 ");
                    output.print( CNAME );
                    output.print(' ');
                    output.print( fromSSRC );
                    output.print(' ');
                    output.print( aboutSSRC );
                    output.print(' ');
                    output.print( fraction );
                    output.print(' ');
                    output.print( packetsLost );
                    output.print(' ');
                    output.println( jitter );

                 }
             }
         }

         return;

    }

} // end of class RecordTask
```

```
package org.web3d.vrtp.rtp;

import javax.media.*;
import javax.media.rtp.*;
import java.lang.*;
import java.net.*;

/**
 * A class that represents necessary information to define
 * an RTP session, as address, port and TTL.
 *
 * <P>
 * @author  Francisco Afonso (afonso@cs.nps.navy.mil)
 * @version 1.0
 */
public class RtpMediaLocator extends MediaLocator{


   /**
    * Defines the value of TTL if not provided.
    */
    public static final int TTL_UNDEFINED = 1;

    private String address = "";

    private int port;
    private int ttl = TTL_UNDEFINED;

    /**
     * Constructor.
     * @param locatorString Describes the session. It should have the
     * following format:
          rtp://address:port[/ttl] , where:
     * <P>
     *   address -> multicast address of the rtp session
     * <P>
     *   port    -> port number
     * <P>
     *   ttl (optional) -> time-to-live
     */
    public RtpMediaLocator(String locatorString) throws MalformedURLException
    {
        super( locatorString);

        parseLocator( locatorString);
    }

    // this method parses the locator string to get the various session data
    // as address, port and ttl
    private  void parseLocator(String locatorString)
                              throws MalformedURLException{

        String remainder = getRemainder();

        int colonIndex = remainder.indexOf(":");
        int slashIndex = remainder.indexOf("/",2);
```

```
    // gets the address
    if (colonIndex != -1)
        address = remainder.substring(2, colonIndex);
    else {
        throw new MalformedURLException(
          "RTP MediaLocator is Invalid. Must be of form rtp://addr:port/ttl");
    }

    // tests if the address is valid
    try{
        InetAddress Iaddr = InetAddress.getByName(address);
    }
    catch (UnknownHostException e){throw new MalformedURLException(
             "Valid RTP Session Address must be given");
    }

    // gets the port
    String portstr = "";
    if (slashIndex == -1)
        portstr = remainder.substring(colonIndex +1,
                                      remainder.length());
    else
        portstr = remainder.substring(colonIndex +1,
                                      slashIndex);

    // tests if the port is an integer
    try{
        Integer Iport = Integer.valueOf(portstr);
        port = Iport.intValue();
    }catch (NumberFormatException e){
        throw new MalformedURLException(
               "RTP MediaLocator Port must be a valid integer");
    }

    // gets the ttl
    if (slashIndex != -1){

        String ttlstr = remainder.substring(slashIndex+1,
                                            remainder.length());
        try{
            Integer Ittl = Integer.valueOf(ttlstr);
            ttl = Ittl.intValue();
        }catch (NumberFormatException e){}

    }
}


/** Returns the RTP Session address
 *@return String form of the RTPSession address
 */
public String getSessionAddress(){
    return address;
}
```

```
    /**
     * Returns the RTP session port.
     * @return RTP session port
     */
    public int getSessionPort(){
        return port;
    }

    /**
     * Returns the session Time-to-live.
     * @return time-to-live(TTL)
     */
    public int getTTL(){
        return ttl;
    }

} // end of class RtpMediaLocator
```

```
package org.web3d.vrtp.rtp;

import javax.media.rtp.*;
import javax.media.rtp.event.*;
import javax.media.rtp.rtcp.*;

import java.awt.*;
import java.awt.event.*;
import java.net.*;
import java.io.*;
import java.util.*;

/**
 * The RtpMonitor Application.
 * <P>
 * This class is a frame that implements the RtpMonitor GUI. <P>
 * If any command line argument is passed the RtpMonitorCommandLine class
 * is called instead.
 *
 * @author  Francisco Afonso (afonso@cs.nps.navy.mil)
 * @version 1.0
 */
public class RtpMonitor extends Frame
             implements ActionListener, ItemListener {

   TextField sessionText, sessionNameText;
   TextArea activeArea,passiveArea,feedbkArea;
   Choice streamChoice;
   Button start,stop,changeStream;
   Label sessionLabel, sessionNameLabel, streamLabel, feedbkLabel;
   Label endLabel;
   Label globalStatLabel, activeLabel, passiveLabel, feedbkFieldsLabel;
   Label [] gLab;
   TextField [] gText;
   String [] gField = {"Total Bytes", "Total Packets", "RTCP Packets",
                       "SR Packets", "Bad RTP Packets", "Bad RTCP Packets",
                       "Bad SR Packets","Bad RR Packets", "Bad SDES Packets",
                       "Bad BYE Packets", "Local Collisions",
                       "Remote Collisions", "Looped Packets",
                       "Failed Transmission", "Unknown Type" };
   Label [] rLab;
   TextField [] rText;
   String [] rField = {"Lost PDUs", "Processed PDUs", "MisOrdered PDUs",
                       "Invalid PDUs", "Duplicate PDUs" };

   int [] endsInHours = { 1, 2, 4, 8, 12, 24, 48, 168 };
   String locator;
   boolean flgPart, flgPlay, flgRecord;
   boolean flgActive = false;
   int presInterval;
   double recInterval;
   RtpMonitorManager monMgr;
   SessionManager mymgr;
   DisplayTask dispTask;
   long SSRCtoShow;
   Hashtable streamTable;
   boolean flgUpdateStreams;
```

```
Date endDate;
MenuBar bar;
Menu preferences, bookmarkMenu, filesMenu, helpMenu;
MenuItem viewPref, modifyPref, selectBm, addBm, deleteBm, aboutItem;
MenuItem last5Item, previous5Item, lastHourItem, previousHourItem, headerItem;
MenuItem exitItem;
ModifyPreferences modPrefDialog;
SelectBookmark    selBmDialog;
DeleteBookmark    delBmDialog;
AddBookmark       addBmDialog;
About             aboutDialog;


 /**
  * Constructor. It is called by main() if no command line argument is passed.
  * <P>
  * The constructor initializes the GUI components.
  */
public RtpMonitor()
{
   super ( "RtpMonitor");
   setSize(780,530);
   setLayout( null );

   preferences = new Menu("Preferences");

   viewPref = new MenuItem("View");
   viewPref.addActionListener(this);
   modifyPref = new MenuItem("Modify");
   modifyPref.addActionListener(this);

   preferences.add( viewPref );
   preferences.addSeparator();
   preferences.add( modifyPref );

   bookmarkMenu = new Menu("Bookmarks");

   addBm = new MenuItem("Add");
   addBm.setEnabled(false);
   deleteBm = new MenuItem("Delete");
   selectBm = new MenuItem("Select");
   selectBm.addActionListener(this);
   deleteBm.addActionListener(this);
   addBm.addActionListener(this);

   bookmarkMenu.add( selectBm );
   bookmarkMenu.addSeparator();
   bookmarkMenu.add( addBm );
   bookmarkMenu.addSeparator();
   bookmarkMenu.add( deleteBm );

   filesMenu = new Menu("File");

   last5Item = new MenuItem("Last five minutes");
   last5Item.addActionListener(this);
   previous5Item = new MenuItem("Previous five minutes");
   previous5Item.addActionListener(this);
```

```
lastHourItem = new MenuItem("Last hour");
lastHourItem.addActionListener(this);
previousHourItem = new MenuItem("Previous hour");
previousHourItem.addActionListener(this);
headerItem = new MenuItem("Header");
headerItem.addActionListener(this);
exitItem = new MenuItem("Exit");
exitItem.addActionListener(this);


filesMenu.add( last5Item );
filesMenu.add( previous5Item );
filesMenu.add( lastHourItem );
filesMenu.add( previousHourItem );
filesMenu.addSeparator();
filesMenu.add( headerItem );
filesMenu.addSeparator();
filesMenu.add( exitItem );

helpMenu = new Menu("Help");

aboutItem = new MenuItem("About");
aboutItem.addActionListener(this);
helpMenu.add( aboutItem);

bar = new MenuBar();
bar.add(filesMenu);
bar.add(bookmarkMenu);
bar.add(preferences);
bar.add(helpMenu);

setMenuBar( bar );

modPrefDialog = new ModifyPreferences(this);

sessionLabel = new Label( "Session Address");
sessionLabel.setBounds( 10, 60, 100, 25 );
add(sessionLabel);

sessionText = new TextField("rtp://___.___.___.___:_____/___");
sessionText.setBounds(110,60, 230, 25 );
add(sessionText);

sessionNameLabel = new Label( "Session Name");
sessionNameLabel.setBounds( 360, 60, 90, 25 );
add(sessionNameLabel);

sessionNameText = new TextField("");
sessionNameText.setBounds(450, 60, 180, 25 );
add(sessionNameText);

start = new Button("Start");
start.setBounds( 660, 60, 40, 25 );
start.addActionListener( this );
start.setEnabled(true);
add(start);
```

```
stop = new Button("Stop");
stop.setBounds( 720, 60, 40, 25 );
stop.addActionListener( this );
stop.setEnabled(false);
add(stop);

globalStatLabel = new Label( "Global Statistics");
globalStatLabel.setBounds( 160, 100, 100, 20 );
add(globalStatLabel);

gLab = new Label[15];
gText = new TextField[15];
int offsetx = 0;
int offsety = 0;
for( int ii=0; ii< 15; ii++){
   if(ii==8){
      offsetx= 220;
      offsety=-(ii*25);
   }
   gLab[ii] = new Label( gField[ii] );
   gLab[ii].setBounds( 10+offsetx, 130+(ii*25)+offsety, 115, 20 );
   add( gLab[ii]);
   gText[ii] = new TextField( "" );
   gText[ii].setBounds( 125+offsetx, 130+(ii*25)+offsety, 90, 20 );
   gText[ii].setEditable(false);
   add( gText[ii]);

 }

activeLabel = new Label( "Active Participants");
activeLabel.setBounds( 50, 340, 140, 20 );
add(activeLabel);

activeArea = new TextArea( "", 0, 0, TextArea.SCROLLBARS_BOTH );
activeArea.setBounds(10,360, 175, 180);
activeArea.setFont( new Font( "Courier" , Font.PLAIN , 12 ) );
activeArea.setEditable(false);
add(activeArea);

passiveLabel = new Label( "Passive Participants");
passiveLabel.setBounds( 230, 340, 140, 20 );
add(passiveLabel);

passiveArea = new TextArea( "", 0, 0, TextArea.SCROLLBARS_BOTH );
passiveArea.setBounds(195, 360, 175, 180);
passiveArea.setFont( new Font( "Courier" , Font.PLAIN , 12 ) );
passiveArea.setEditable(false);
add(passiveArea);

streamLabel = new Label( "Stream");
streamLabel.setBounds( 470, 120, 50, 25 );
add(streamLabel);

streamChoice = new Choice();
streamChoice.setBounds(520, 120, 250, 25);
streamChoice.setEnabled(false);
streamChoice.addItemListener(this);
```

```
  add(streamChoice);

  changeStream = new Button("Change");
  changeStream.setBounds( 720, 150, 50, 25 );
  changeStream.addActionListener( this );
  changeStream.setEnabled(false);
  add(changeStream);

  rLab = new Label[5];
  rText = new TextField[5];
  for( int ii=0; ii< 5; ii++){
     rLab[ii] = new Label( rField[ii] );
     rLab[ii].setBounds( 500, 170+(ii*25), 100, 20 );
     add( rLab[ii]);
     rText[ii] = new TextField( "" );
     rText[ii].setBounds( 605, 170+(ii*25), 90, 20 );
     rText[ii].setEditable(false);
     add( rText[ii]);

   }

  feedbkLabel = new Label( "Feedback Reports");
  feedbkLabel.setBounds( 510, 315, 100, 20 );
  add(feedbkLabel);

  feedbkFieldsLabel = new Label(
     "Username           Fraction Lost        Jitter         Packets Lost");
  feedbkFieldsLabel.setBounds( 420, 340, 350, 20 );
  add(feedbkFieldsLabel);


  feedbkArea = new TextArea( "", 0, 0, TextArea.SCROLLBARS_VERTICAL_ONLY );
  feedbkArea.setBounds(410,360, 360, 180);
  feedbkArea.setFont( new Font( "Courier" , Font.PLAIN , 12 ) );
  feedbkArea.setEditable(false);
  add(feedbkArea);

  setVisible(true);

  streamTable = new Hashtable();

  this.addWindowListener(new WindowAdapter(){
      public void windowClosing(WindowEvent e){
          System.exit(0);
      }
      });


}
```

```
 /**
  * Takes action when buttons are selected.
  *
  */
public void actionPerformed( ActionEvent e )
{
    if( e.getSource() == start){
       start.setEnabled(false);
       disableInputs();
       locator = sessionText.getText();
       recInterval = Double.parseDouble(
                              modPrefDialog.intervalText.getText());
       presInterval = Integer.parseInt(
                              modPrefDialog.intervalPresText.getText());
       flgPart = modPrefDialog.partBox.getState();
       flgPlay = modPrefDialog.playBox.getState();
       flgRecord = modPrefDialog.recordBox.getState();
       SSRCtoShow = 0;
       endDate = new Date( (new Date()).getTime()
          + endsInHours[modPrefDialog.endChoice.getSelectedIndex()]*3600000L);

       if( startSession() ){
          stop.setEnabled(true);
          addBm.setEnabled(true);
          changeStream.setEnabled(true);
          flgUpdateStreams = true;
       }
       else{
          start.setEnabled(true);
          enableInputs();
       }
    }

    if( e.getSource() == stop){
       flgActive = false;
       stop.setEnabled(false);
       addBm.setEnabled(false);
       monMgr.close();
       monMgr = null;
       mymgr = null;
       clearAllData();
       changeStream.setEnabled(false);
       start.setEnabled(true);
       enableInputs();
    }

    if( e.getSource() == changeStream){
       changeStream.setEnabled(false);
       flgUpdateStreams = false;
       clearStreamData();
       streamChoice.setEnabled(true);
    }
```

```
    if( e.getSource() == modifyPref){
        modPrefDialog.setTitle("Modify Preferences");
        modPrefDialog.enableInput();
        modPrefDialog.setVisible(true);
        modPrefDialog.savePreferences();
    }

    if( e.getSource() == viewPref){
        modPrefDialog.setTitle("View Preferences");
        modPrefDialog.disableInput();
        modPrefDialog.setVisible(true);
    }

    if( e.getSource()== selectBm){
        selBmDialog   = new SelectBookmark(this);
        selBmDialog.setVisible(true);
        selBmDialog = null;
    }

    if( e.getSource()== deleteBm){
        delBmDialog   = new DeleteBookmark(this);
        delBmDialog.setVisible(true);
        delBmDialog = null;
    }

    if( e.getSource()== addBm){
        addBmDialog   = new AddBookmark(this);
        addBmDialog.setVisible(true);
        addBmDialog = null;
    }

    if( e.getSource()== last5Item){
        runViewer( sessionText.getText() , "LastFiveMinutes.txt");
    }
    if( e.getSource()== previous5Item){
        runViewer( sessionText.getText() , "PreviousFiveMinutes.txt");
    }
    if( e.getSource()== lastHourItem){
        runViewer( sessionText.getText() , "LastHour.txt");
    }
    if( e.getSource()== previousHourItem){
        runViewer( sessionText.getText() , "PreviousHour.txt");
    }
    if( e.getSource()== headerItem){
        runViewer( sessionText.getText() , "Header.txt");
    }
    if( e.getSource()== aboutItem){
        aboutDialog   = new About(this);
        aboutDialog.setVisible(true);
        aboutDialog = null;
    }
    if( e.getSource()== exitItem){
      System.exit(0);
     }

}
```

```
/**
 * starts the monitoring session by creating a RtpMonitorManager object.
 */
private boolean startSession(){

   // tries to create the RtpMonitorManager object
   try{
      monMgr = new RtpMonitorManager(locator, flgPart, flgPlay,
                                     flgRecord, recInterval);
      mymgr = monMgr.getSessionManager();
   }
   catch( MalformedURLException e ){
      feedbkArea.setText(
               "MalformedURLException creating RtpMonitorManager:" + '\n');
      feedbkArea.append( e.getMessage() );
      return false;
   }
   catch(UnknownHostException e ){
      feedbkArea.setText(
               "UnknownHostException creating RtpMonitorManager:" + '\n');
      feedbkArea.append( e.getMessage() );
      return false;
   }
   catch( SessionManagerException e ){
      feedbkArea.setText(
              "SessionManagerException creating RtpMonitorManager:" + '\n');
      feedbkArea.append( e.getMessage() );
      return false;
   }
   catch( IOException  e ){
      feedbkArea.setText( "IOException creating RtpMonitorManager:" + '\n');
      feedbkArea.append( e.getMessage() );
      return false;
   }

   flgActive = true;

   // creates a DisplayTask object to update the statistics on screen
   dispTask = new DisplayTask( this, presInterval );

   return true;
}

/**
 * Updates the global statistics.
 */
 public void showGlobalStats(){

     GlobalReceptionStats stats = mymgr.getGlobalReceptionStats();

     gText[0].setText( new Integer(stats.getBytesRecd()).toString() );
     gText[1].setText( new Integer(stats.getPacketsRecd()).toString() );
     gText[2].setText( new Integer(stats.getRTCPRecd()).toString() );
     gText[3].setText( new Integer(stats.getSRRecd()).toString() );
```

```
    gText[4].setText( new Integer(stats.getBadRTPkts()).toString() );
    gText[5].setText( new Integer(stats.getBadRTCPPkts()).toString() );
    gText[6].setText( new Integer(stats.getMalformedSR()).toString() );
    gText[7].setText( new Integer(stats.getMalformedRR()).toString() );
    gText[8].setText( new Integer(stats.getMalformedSDES()).toString() );
    gText[9].setText( new Integer(stats.getMalformedBye()).toString() );
    gText[10].setText( new Integer(stats.getLocalColls()).toString() );
    gText[11].setText( new Integer(stats.getRemoteColls()).toString() );
    gText[12].setText( new Integer(stats.getPacketsLooped()).toString() );
    gText[13].setText( new Integer(stats.getTransmitFailed()).toString() );
    gText[14].setText( new Integer(stats.getUnknownTypes()).toString());

 }

/**
 * Updates the lists of active and passive participants.
 */
 public void showParticipants(){

    activeArea.setText("");
    Vector aux = mymgr.getActiveParticipants();

    Participant part;
    for(int ii = 0; ii< aux.size(); ++ii){
         part = (Participant)aux.elementAt(ii);
         activeArea.append( RtpUtil.getUsernameOrCNAME(part) + '\n');
    }
    passiveArea.setText("");
    aux = mymgr.getPassiveParticipants();

    for(int ii = 0; ii< aux.size(); ++ii){
         part = (Participant)aux.elementAt(ii);
         passiveArea.append( RtpUtil.getUsernameOrCNAME(part) + '\n');
    }

 }


/**
 * Updates the stream statistics.
 */
  public void showStreamStats(){

    if( !flgUpdateStreams){
      return;
    }

    ReceiveStream dispStream = null;
    ReceiveStream stream;
    ReceptionStats stats;
    Participant part;
    String display;
    streamChoice.removeAll();
    streamTable.clear();

    Vector aux = mymgr.getReceiveStreams();
    for(int ii = 0; ii< aux.size(); ++ii){
```

```
        stream = (ReceiveStream) aux.elementAt(ii);
        part = stream.getParticipant();
        if(part != null){
           display = new String( RtpUtil.getUsernameOrCNAME(part) + " / " +
                                 RtpUtil.correctSSRC(stream.getSSRC()) );
        }
        else{
           display = new String( "unknown_participant / " +
                                  RtpUtil.correctSSRC(stream.getSSRC()) );
        }

        streamChoice.add(display);
        streamTable.put( display, new Long( stream.getSSRC() ) );

        if( stream.getSSRC() == SSRCtoShow ){
           dispStream = stream;
           streamChoice.select(ii);

        }
     }


     if( SSRCtoShow == 0){
        if( aux.size() > 0){
           dispStream = (ReceiveStream) aux.elementAt(0);
           SSRCtoShow = dispStream.getSSRC();
        }
     }

     if( dispStream == null){
        for(int jj=0; jj<5; ++jj){
           rText[jj].setText("");
        }
        SSRCtoShow = 0;
        return;
      }

     stats = dispStream.getSourceReceptionStats();

     rText[0].setText( new Integer(stats.getPDUlost()).toString() );
     rText[1].setText( new Integer(stats.getPDUProcessed()).toString() );
     rText[2].setText( new Integer(stats.getPDUMisOrd()).toString() );
     rText[3].setText( new Integer(stats.getPDUInvalid()).toString() );
     rText[4].setText( new Integer(stats.getPDUDuplicate()).toString() );

  }
```

```
/**
 *  Updates the stream feedbacks.
 */
  public void showFeedbacks(){

     if( !flgUpdateStreams){
       return;
     }

     Participant part;
     Vector reports, feedbacks;
     Report rep;
     Feedback feedbk;

     feedbkArea.setText("");
     Vector aux = mymgr.getAllParticipants();
     for(int ii = 0; ii< aux.size(); ++ii){
         part = (Participant)aux.elementAt(ii);
         reports = part.getReports();

         for( int jj=0; jj< reports.size(); ++jj){
             rep = (Report) reports.elementAt(jj);
             feedbacks = rep.getFeedbackReports();
             for( int kk=0; kk < feedbacks.size(); ++kk){
                feedbk = (Feedback) feedbacks.elementAt(kk);
                if( feedbk.getSSRC() == SSRCtoShow ){
                   feedbkArea.append( fillBlanks(
                      RtpUtil.getUsernameOrCNAME(part) , 19 ) + "  " );
                   double fraction = (feedbk.getFractionLost())/256.0;
                   feedbkArea.append(
                      fillBlanks( String.valueOf(fraction), 6) + "    " );
                   feedbkArea.append(
                      fillBlanks(String.valueOf(feedbk.getJitter()),8) + " ");
                   feedbkArea.append(
                      fillBlanks( String.valueOf(feedbk.getNumLost()),10)
                                  + '\n' );

                }
             }
         }
     }
 }
```

```
 /**
  * Take action when selection boxes are used.
  *
  */
public void itemStateChanged( ItemEvent ie )
{

   if( ie.getSource() == streamChoice){

      SSRCtoShow = ((Long) streamTable.get( ie.getItem() ) ).longValue();

      flgUpdateStreams = true;
      changeStream.setEnabled(true);
      streamChoice.setEnabled(false);
   }

}



/**
 * Clears all stats info on screen.
 */
 private void clearAllData(){

   for(int jj=0; jj<15; ++jj){
      gText[jj].setText("");
   }
   for(int jj=0; jj<5; ++jj){
      rText[jj].setText("");
   }
   activeArea.setText("");
   passiveArea.setText("");
   feedbkArea.setText("");

   streamChoice.removeAll();
   streamTable.clear();

}

/**
 * Clears all stream related stats.
 */
 private void clearStreamData(){
   for(int jj=0; jj<5; ++jj){
      rText[jj].setText("");
   }
   feedbkArea.setText("");

}
```

```
/**
 * Disable user input components( used after a session is started )
 */
private void disableInputs(){
   sessionText.setEnabled(false);
   modifyPref.setEnabled(false);
   selectBm.setEnabled(false);
}

/**
 * Enable user input components( used after a session is stoped )
 */
private void enableInputs(){
   sessionText.setEnabled(true);
   modifyPref.setEnabled(true);
   selectBm.setEnabled(true);
}




/**
 * Returns the state of RtpMonitor. Also exits the program if the duration
 * is over.
 * @return true if the monitor is active
 */
public boolean isMonitoring(){
   if( endDate.compareTo(new Date()) < 0 ){
      System.exit(0);
   }

   return flgActive;

}


/**
 * Creates a string with a fixed size, starting by a given string and ending
 * with blank spaces.
 * @param strin the original string
 * @param size  the final size of the returning string
 * @return a string
 */
private String  fillBlanks( String strin, int size )
{
    StringBuffer spaces = new StringBuffer();
    for( int ii = 0; ii <= size; ++ii ){
        spaces.append( " ");
    }
    String newString = new String( strin + spaces );

    return newString.substring( 0, size);
}
```

```
   /**
    * Runs an external program specified in the preference menu
    * to display output files.
    * @param locator the session RtpMediaLocator
    * @param fileName the name of the file to be displayed
    */
   private void runViewer(String locator, String fileName){

      String prefix;

      Runtime r = Runtime.getRuntime();

      try{
         RtpMediaLocator rtpml = new RtpMediaLocator(locator);
         String session = rtpml.getSessionAddress();
         String port = (new Integer(rtpml.getSessionPort())).toString();
         String dir = new String( "./session"
                  + session.replace('.','-') + "port" + port );
         prefix = new String( dir + "/statistics" );
      }
      catch (MalformedURLException e) {
         System.out.println( e.getMessage() );
         return;
      }

      try{
         String prog = new String( modPrefDialog.viewerText.getText() + " " +
                                   prefix + fileName );
         Process p = r.exec(prog);
      }
      catch ( IOException e){
         System.err.println( e.getMessage() );
      }

   }


   /**
    * Method called upon executing class RtpMonitor. If there is no
    * argument an object of class RtpMonitor will be instantiated
    * and executed, otherwise the same will happen with an
    * RtpMonitorCommandLine object.
    */
   public static void main( String [] args )
   {
      int nArgs = args.length;

      if(args.length == 0){
         RtpMonitor myProg = new RtpMonitor();
      }
      else{
         RtpMonitorCommandLine myProg = new RtpMonitorCommandLine( args );
        myProg.run();
      }
   }

} // end of class RtpMonitor
```

```
package org.web3d.vrtp.rtp;

import javax.media.rtp.*;
import javax.media.rtp.event.*;
import javax.media.rtp.rtcp.*;

import java.io.*;
import java.net.*;
import java.util.*;

/**
 * A class used to start a RTP monitor from command line inputs.
 * <P>
 * This monitor does not have the option of playing streams.
 *
 * @author  Francisco Afonso (afonso@cs.nps.navy.mil)
 * @version 1.0
 */
public class RtpMonitorCommandLine {


    boolean willpart = false;
    boolean willplay = false;
    boolean willrecord = false;
    String locator = null;
    double interval = 30.0;
    int endsInHours = 168;
    Date endDate;

    RtpMonitorManager monMgr;


   /**
    * Method called upon executing class RtpMonitorCommandLine.
    */
    public static void main(String [] args){

      RtpMonitorCommandLine myProg = new RtpMonitorCommandLine( args );
      myProg.run();

    }

    /**
     * Constructor. It reads the command line arguments and sets
     * variables and flags.
     * The command line arguments have the following format:
     * <P>
     * java RtpMonitorCommandLine rtpLocator [options] <P>
     *    rtpLocator example: rtp://224.2.125.50:50328/127 <P>
     *   -part : monitor sends RTCP packets <P>
     *   -play : monitor play streams <P>
     *   -record : monitor records statistics <P>
     *   -i nnn : nnn defines the recording interval in seconds (default 30s) <P>
     *   -e ppp : ppp defines the monitoring duration in hours (default: 168
     *    hs)<P>
     *   -help  : displays command line format and aborts
     *
```

```
* <P>
* @param args an array of strings
*/
public RtpMonitorCommandLine(String [] args){

 int nArgs = args.length;

 if( nArgs == 0){
    System.out.println(
      "Format: java RtpMonitorCommandLine rtpLocator <options>");
    System.out.println(
      "  rtpLocator example: rtp://224.2.125.50:50328/127" );
    System.out.println(
      "  options: -part : monitor sends RTCP packets" );
    System.out.println(
      "           -play : monitor play streams");
    System.out.println(
      "           -record : monitor records statistics");
    System.out.println(
      "           -i nnn : nnn defines the recording interval in seconds
                 (default 30s)");
    System.out.println(
      "           -e ppp : ppp defines the monitoring duration in hours
                 (default: 168 hs)");
    System.out.println(
      "           -help  : displays command line format and aborts ");
    System.exit(0);
 }

 if( args[0].indexOf("-help") == 0 ) {
    System.out.println(
      "Format: java RtpMonitorCommandLine rtpLocator <options>");
    System.out.println(
      "  rtpLocator example: rtp://224.2.125.50:50328/127" );
    System.out.println(
      "  options: -part : monitor sends RTCP packets" );
    System.out.println(
      "           -play : monitor play streams");
    System.out.println(
      "           -record : monitor records statistics");
    System.out.println(
      "           -i nnn : nnn defines the recording interval in seconds
                  (default 30s)");
    System.out.println(
      "           -e ppp : ppp defines the monitoring duration in hours
                  (default: 168 hs)");
    System.out.println(
      "           -help  : displays command line format and aborts ");
    System.exit(0);
 }

 // parses the command line arguments to extract the options
 locator = args[0];
 int ii = 1;
 while( ii < nArgs ){
    if( args[ii].indexOf("-part") == 0 ) {
       willpart = true;
```

```
      }
      else if( args[ii].indexOf("-play") == 0 ) {
         willplay = true;
      }
      else if( args[ii].indexOf("-record") == 0 ) {
         willrecord = true;
      }
      else if( (args[ii].indexOf("-i") == 0) && (ii < nArgs -1) ) {
         interval = Double.parseDouble(args[ii+1]);
      }
      else if( (args[ii].indexOf("-e") == 0) && (ii < nArgs -1) ) {
         endsInHours = Integer.parseInt(args[ii+1]);
      }
      ii++;

   }

   endDate = new Date( (new Date()).getTime() + endsInHours*3600000L) ;

   // displays the selected options on the console
   System.out.println("locator  = " + locator );
   System.out.println("play = " + willplay );
   System.out.println("part = " + willpart );
   System.out.println("record  = " + willrecord );
   System.out.println("recording interval = " + interval + " seconds." );
   System.out.println("monitoring duration = " + endsInHours + " hours." );
 }


/**
 * Method that will create the RtpMonitorManager object and will exit the
 * program when the user defined duration is elapsed.
 */
 public void run(){

   // tries to create the RtpMonitorManager object
   try{
      monMgr = new RtpMonitorManager(locator, willpart, willplay,
                                     willrecord, interval);
   }

   catch( MalformedURLException e ){
      System.err.println(
           "MalformedRTPMRLException creating RTPMonitorManager: " );
      System.err.println( e.getMessage() );
      System.exit(0);
   }
   catch(UnknownHostException e ){
      System.err.println(
            "UnknownHostException creating RTPMonitorManager: " );
      System.err.println( e.getMessage() );
      System.exit(0);
   }
   catch( SessionManagerException e ){
      System.err.println(
            "RTPSessionManagerException creating RTPMonitorManager: " );
      System.err.println( e.getMessage() );
```

```
         System.exit(0);
      }
      catch( IOException  e ){
         System.err.println("IOException creating RTPMonitorManager: " );
         System.err.println( e.getMessage() );
         System.exit(0);
      }

      // runs until the user aborts the program (ctrl-C) or the
      // monitoring period is over
      while(true){

         // tests if the monitoring period is over
         if( endDate.compareTo(new Date()) < 0 ){
            System.exit(0);
         }

         try{
            Thread.sleep(50000);
         }
         catch (InterruptedException e){}
      }

   }

} // end of class RtpMonitorCommandLine
```

```
package org.web3d.vrtp.rtp;

import javax.media.rtp.*;
import javax.media.rtp.event.*;
import javax.media.rtp.rtcp.*;

// RTPSessionMgr class
import com.sun.media.rtp.*;
import java.io.*;
import java.net.*;
import java.util.*;
import javax.media.*;
import javax.media.protocol.*;

/**
 * A class that encapsulates all operations necessary to start a new
 * monitoring session, play its streams, and record statistical data.
 * <P>
 * This class does not display statistics on screen. That must be done
 * by another class, usually a frame, using data from a SessionManager
 * object ( see method getSessionManager ).
 *
 * @author  Francisco Afonso (afonso@cs.nps.navy.mil)
 * @version 1.0
 */
public class RtpMonitorManager implements ReceiveStreamListener {

   private RtpMediaLocator rtpml = null;
   private SessionManager mgr = null;
   private SessionAddress sessaddr = null;
   private boolean flgPart;
   private boolean flgPlay;
   private boolean flgRecord;
   private Hashtable windowlist;
   private RecordTask recTask;

   /**
    * @param locatorString the session description string
    * format, e.g. rtp://224.2.134.67:50980/127
    * <P>
    * @param willParticipate
    *        true if the Monitor will participate in the session,
    *             sending RTCP packets, false otherwise.
    * <P>
    * @param willPlayStreams true if the Monitor will play the
    *                        receiving streams, false otherwise.
    * <P>
    * @param willRecord true if the Monitor will record statistical
    *                   data about the session, false otherwise.
    * <P>
    * @param recordInterval the time between data recordings in seconds
    * @exception MalformedURLException
    *         if the locatorString argument does not conform to the syntax.
    * <P>
    * @exception UnknownHostException
    *            if InetAddress.getByName(session address) fails
    * <P>
```

```
 * @exception SessionManagerException
 *            Exception thrown when there is an error starting an
 *            RTPSessionManager
 * <P>
 * @exception IOException
 *            Exception thrown when there is an error in RTPSessionManager
 */
public RtpMonitorManager( String locatorString, boolean willParticipate,
                          boolean willPlayStreams, boolean willRecord ,
                          double recordInterval)
    throws MalformedURLException, UnknownHostException,
           SessionManagerException, IOException
{
   flgPart = willParticipate;
   flgPlay = willPlayStreams;
   flgRecord = willRecord;

   // creates the RtpMediaLocator object
   rtpml = new RtpMediaLocator(locatorString);

   // creates an empty RtpSessionManager object
   mgr = new RTPSessionMgr();

   // if the user select to play the streams, registers as a
   // listener for received streams
   if(flgPlay){
      mgr.addReceiveStreamListener(this);
      windowlist = new Hashtable();
   }

   // gets the InetAddress of the session
   InetAddress destaddr = InetAddress.getByName(rtpml.getSessionAddress());

   // gets the session port
   int port = rtpml.getSessionPort();

   // creates a SessionAddress objet to represent the session
   sessaddr = new SessionAddress ( destaddr, port, destaddr, port+1 );

   // call the method to generate a CNAME for the user
   String cname = mgr.generateCNAME();

   // gets the username from the system and adds it to
   // the string "/rtpMonitor". That will be the user name sent in
   // RTCP packets
   String username = null;
   try{
       username = System.getProperty("user.name")+ "/rtpMonitor";
   }
   catch(SecurityException e) {
       username = "RTPMonitor-user";
   }

   // creates the source description fields
   SourceDescription [] userdesclist = new SourceDescription[3];

   userdesclist[0] = new
```

```
        SourceDescription( SourceDescription.SOURCE_DESC_NAME,
                                 username, 1, false);
    userdesclist[1] = new
        SourceDescription( SourceDescription.SOURCE_DESC_CNAME,
                                 cname, 1, false);
    userdesclist[2] = new
        SourceDescription( SourceDescription.SOURCE_DESC_TOOL,
                                 "RTPMonitor v1.0" , 1, false);

    // generates a local address
    SessionAddress localaddr = new SessionAddress();

    // that is the fraction of RTCP bandwith compared to RTP
    double rtcpFraction = 0.05;

    // if the user has selected for no participation in the session,
    // sets the fraction above to zero ( no RTCP packets )
    if( ! flgPart )
       rtcpFraction = 0.0;

    // initiates the SessionManager object
    mgr.initSession( localaddr, userdesclist, rtcpFraction , 0.25 );

    int ttl = rtpml.getTTL();

    // starts the SessionManager object
    mgr.startSession( sessaddr, ttl, null);

    // if the user has selected for recording statistics, creates a
    // RecordTask objet to generate the reports periodically
    if(flgRecord){
       recTask = new RecordTask( this, recordInterval );
    }

}


/**
 * Method of classes that implement the ReceiveStreamListener
 * interface
 *
 */
public void update( ReceiveStreamEvent event){
    Player newPlayer = null;
    RtpPlayerWindow playerWindow = null;
    String cname = null;

    SessionManager source = (SessionManager) event.getSource();

    // if a new stream is received
    if( event instanceof NewReceiveStreamEvent){

      // gets the ReceiveStream object
      try{
        ReceiveStream stream =
                ((NewReceiveStreamEvent)event).getReceiveStream();
```

```
        // gets the Participant object associated with the stream
        Participant part = stream.getParticipant();

        // gets the participant canonical name
        if(part != null){
            cname = part.getCNAME();
        }

        // gets the stream DataSource associated with the stream
        DataSource dsource = stream.getDataSource();

        // creates a player to play the DataSource
        newPlayer = Manager.createPlayer(dsource);

        // if a player was created generates a player window
        if(newPlayer != null){
            playerWindow = new RtpPlayerWindow( newPlayer, cname);
            windowlist.put( stream, playerWindow);
        }

      }
      catch (Exception e){
          System.err.println(
                 "NewRecvStreamEvent exception " + e.getMessage() );
          return;
      }
    }

    // if the sender of a stream was identified
    if( event instanceof StreamMappedEvent){

        // gets the ReceiveStream object
        ReceiveStream stream =
                     ((StreamMappedEvent)event).getReceiveStream();

        // gets the Participant associated with the stream
        Participant part = stream.getParticipant();

        // retrieves the correct player window from the
        // hash table
        if(stream != null){
            playerWindow = (RtpPlayerWindow) windowlist.get(stream);
        }

        // change the title of the player window to include the
        // name of the sender
        if( (playerWindow != null) && (part != null)){
            playerWindow.Name(part.getCNAME());
        }
     }
}
```

```
   /**
    * Closes a monitor session, stops recording and closes player windows
    *
    */
   public void close(){

      if(flgRecord){
         recTask.exit();
      }

      if(flgPlay){

         Enumeration windows = windowlist.elements();
         while( windows.hasMoreElements()){
            RtpPlayerWindow currwindow =
                      (RtpPlayerWindow) windows.nextElement();
            if( currwindow != null){
               currwindow.killThePlayer();
            }
         }
      }

      mgr.closeSession(null);
      mgr = null;

   }


   /**
    * Returns the RtpMediaLocator associated with the RTP session
    *
    * @return the session media locator
    */
   public RtpMediaLocator   getMediaLocator(){ return rtpml;}

   /**
    * Returns the SessionManager object created by the monitor
    *
    * @return the SessionManager (RTPSessionMgr)
    */
   public SessionManager getSessionManager(){ return mgr;}

   /**
    * Returns the SessionAddress object associated with the RTP session
    *
    * @return the SessionAddress
    */
   public SessionAddress getSessionAddress(){ return sessaddr;}


} // end of class RtpMonitorManager
```

```
package org.web3d.vrtp.rtp;

/*
 * @(#)RTPPlayerWindow.java   1.7 98/03/28
 *
 * Copyright 1996-1998 by Sun Microsystems, Inc.,
 * 901 San Antonio Road, Palo Alto, California, 94303, U.S.A.
 * All rights reserved.
 *
 * This software is the confidential and proprietary information
 * of Sun Microsystems, Inc. ("Confidential Information").  You
 * shall not disclose such Confidential Information and shall use
 * it only in accordance with the terms of the license agreement
 * you entered into with Sun.
 */

import javax.media.Player;
import java.awt.*;
import com.sun.media.ui.*;

/**
 * This class is used to create a window for playing an audio/video
 * stream. It is a subclass of PlayerWindow, that added the
 * capacity of modifying the window name.
 * Both classes were developed by SUN. RTPPlayerWindow came with
 * JMF1.1 sample code and PlayerWindow is in the file JMF.jar.
 */
public class RtpPlayerWindow extends PlayerWindow {

    public RtpPlayerWindow(Player player, String title) {
      super(player);
      setTitle(title);
    }
    public void Name(String title){
      setTitle(title);
    }

} // end of class PlayerWindow
```

```
package org.web3d.vrtp.rtp;

import javax.media.rtp.*;
import javax.media.rtp.event.*;
import javax.media.rtp.rtcp.*;

import java.util.*;

/**
 * A class with some RTP utilities (static methods)
 *
 * @author  Francisco Afonso (afonso@cs.nps.navy.mil)
 * @version 1.0
 */

public class RtpUtil
{
   /**
    * Returns the participant username
    *
    * @param part the Participant object
    *
    * @return a string with the participant's username
    */
   public static String getUsername(Participant part){


      Vector sdeslist = part.getSourceDescription();
      if(sdeslist == null){
         return null;
      }

      SourceDescription des;
      for( int ii=0; ii < sdeslist.size(); ++ii){
         des = (SourceDescription) sdeslist.elementAt(ii);
         if( des.getType() == SourceDescription.SOURCE_DESC_NAME ){
            return des.getDescription();
         }
      }
      return null;

   }
```

```
   /**
    * Returns the participant's username or his CNAME,
    * if no username is known.
    *
    * @param part the Participant object
    *
    * @return a string with the participant's username or CNAME
    */
   public static String getUsernameOrCNAME(Participant part){
      String username = getUsername(part);
      if(username == null){
         return part.getCNAME();
      }

      else{
         return username;
      }
   }


   /**
    * Converts an number represented as a signed integer(32 bits)
    * to a long integer (64 bits). JMF methods return the SSRC as an
    * integer. As the SSRC is a 32 bits number, some are represented
    * in JMF as negative integers.
    * This convertion is necessary to present SSRCs as a positive integer.
    */
   public static long correctSSRC( long ssrc ){
      if(ssrc < 0){
         return (4294967296L + ssrc);
      }
      return ssrc;
   }


} // end of class RtpUtil
```

```
package org.web3d.vrtp.rtp;

import java.awt.*;
import java.awt.event.*;
import java.util.*;
import java.io.*;

/**
 * A Dialog to select a session bookmark. It displays a list choice
 * of session names.
 * <P>
 *
 * @author  Francisco Afonso (afonso@cs.nps.navy.mil)
 * @version 1.0
 */
public class SelectBookmark extends Dialog implements ItemListener {

    java.awt.List sessionNamesList;
    Vector sessionAddressVec;
    RtpMonitor theParent;

    /**
     * Constructor.
     * <P>
     * @param parent the parent frame
     */
    public SelectBookmark(Frame parent){
        super( parent, "Select Bookmark" , true );
        setSize( 250 , 200 );

        theParent = (RtpMonitor) parent;

        addWindowListener( new CloseWindow() );

        sessionNamesList = new java.awt.List( 5 , false);

        loadBookmarks();

        sessionNamesList.setBackground(Color.lightGray);
        sessionNamesList.addItemListener(this);

        add(sessionNamesList);

    }
```

```
     /**
      * Activated when a bookmark choice is made.
      * <P>
      * @param parent the parent frame
      */
     public void itemStateChanged( ItemEvent e){

        int index = sessionNamesList.getSelectedIndex();
        theParent.sessionText.setText(
            (String) sessionAddressVec.elementAt(index)  );
        theParent.sessionNameText.setText(
                           sessionNamesList.getItem(index) );
        setVisible(false);

     }


    /**
     * Loads the session bookmarks from file "bookmarks.txt"
     */
    private void loadBookmarks(){

       sessionAddressVec = new Vector();

       try{
          BufferedReader input =
                 new BufferedReader( new FileReader("bookmarks.txt") );
          String line;

          while( (line = input.readLine()) != null){
             int pos = line.lastIndexOf("rtp://");
             if (pos != -1){
                sessionAddressVec.addElement( line.substring(pos) );
                sessionNamesList.add( line.substring( 0 , pos ) );
             }
          }
          input.close();
       }
       catch ( FileNotFoundException e){
          System.out.println( "Select Bookmark: " + e.getMessage() );
       }
       catch ( IOException e){
          System.err.println(
                   "Exception reading bookmark: " + e.getMessage() );
       }

    }

}  // end of class SelectBookmark
```

APPENDIX E. COMPARISON RTP MIB VERSUS JMF STATISTICS

SessionTable

RTP MIB	JMF-based RTP Monitor Application	Comments
SessionIndex (Integer32) – an index of the conceptual row which is for SNMP purposes only and has no relation with any protocol value.	Not applicable.	-
SessionDomain (TDomain) – the transport layer protocol used for sending or receiving the stream of RTP packets in this session.	Not implemented.	It can be added. JMF uses UDP as the transport protocol. Extensibility to other protocols is possible, but not provided.
SessionRemAddr (Taddress) – the remote destination transport address on which the RTP data packet is sent and/or received.	Not implemented.	It can be added. In RTP Monitor the session transport address names the directory where the files are created and updated. e.g. session224.2.2.2port88888
SessionLocAddr (Taddress) – the local destination transport address on which the stream of data packet is being sent and/or received.	Not implemented.	It can be added.
SessionIfIndex (InterfaceIndex) – this value is set to the corresponding value from the Internet Standard MIB. This is the interface that the RTPStream is being sent to or received from.	Not applicable.	-
SessionSenderJoins (Counter32) – the number of senders that have been observed joined the session since SessionStartTime (see below).	Not implemented.	Not part of native JMF statistics. It can be derived in an JMF application by monitoring NewRecvStreamEvents.
SessionReceiverJoins(Counter32) - the number of receivers that have been observed joined the session since SessionStartTime (see below).	Not implemented.	Not part of native JMF statistics. It can be derived in an JMF application by searching for new participants in the list of participants managed by RTPSessionManager.

SessionByes (Counter32) – a count of RTCP BYE.	Not implemented.	Not part of native JMF statistics. It can be derived in an JMF application by monitoring ByeEvents.
SessionStartTime (TimeStamp) – the value of SysUpTime at the time that this row is created.	Not implemented.	That is the time when the Session Monitoring starts. It can be added to RTPMonitor.
SessionMonitor (ThuthValue) – set to true if sender or receivers in addition to the local RTP System are to be monitored.	Not applicable.	In RTP Monitor it is always true.
SessionRowStatus (RowStatus) – active when RTP/RTCP Messages are being sent or received by an RTPSystem. If this row is "notInService" it may be removed after 5 minutes.	Not implemented.	It can be implemented in RTP Monitor, but without removals.
Not provided by MIB.	Time (hh:mm:ss) – time of the report	Needed. Implicit in SNMP? **Bill Strahm comments:** "Times of the reports are in the sender/receiver report tables."
Not provided by MIB.	TotalParticipants (int) – the total number of participants attending the session.	Needed. Equals active + passive participants and also remote + local. **Bill Strahm comments: "**We argued about whether we should track Sender/Receiver Joins/Byes as a high watermark, a total count, or a current count. We decided on the counter. Depending on how you want to track "TotalParticipants" it very well may be rtpSessionSenderJoins + rtpSessionReceiverJoins".
Not provided by MIB.	RemoteParticipants(int) – the number of remote participants attending the session	Desirable. **Bill Strahm comments:** "Can simply be the count of receivers in the receiver table".

Not provided by MIB.	ActiveParticipants (int) – the number of active participants (senders) attending the session.	Needed. **Bill Strahm comments:** “See above”.
Not provided by MIB.	TotalBytesRecd (int) – the number of bytes received in the session, before any validation.	Needed for bandwidth calculations. **Bill Strahm comments:** “rtpRcvOctets in the receiver table”.
Not provided by MIB.	TotalPacketsRecd (int) – the total number of RTP and RTCP packets received in the session before any packet validation.	Needed for comparison of bandwidth versus packets per second performance. **Bill Strahm comments:** “rtpRcvPackets in the receiver table”.
Not provided by MIB.	RTCPPacketsRecd (int) – the total number of RTCP packets received in the session before any header validation.	Desirable. **Bill Strahm comments:** “Combination of SR/RR counts out of the sender/receiver table”.
Not provided by MIB.	SRPacketsRecd (int) – the total number of sender reports received in the session.	Desirable. **Bill Strahm comments:** “rtpSenderSR from the sender table.
Not provided by MIB.	BadRTPPackets (int) – the total number of RTP data packets that failed the RTP header validation check.	Desirable.
Not provided by MIB.	BadRTCPPackets (int) – the total number of RTCP packets that failed the RTCP header validation check.	Desirable.
Not provided by MIB.	MalformedSR (int) – the total number of invalid sender reports due to length inconsistency.	Desirable.
Not provided by MIB.	MalformedRR (int) – the total number of invalid receiver reports due to length inconsistency.	Desirable.
Not provided by MIB.	MalformedSDES (int) – the total number of invalid SDES packets due to length	Desirable.

	inconsistency.	
Not provided by MIB.	MalformedBYE (int) – the total number of invalid BYE packets due to length inconsistency.	Desirable.
Not provided by MIB.	LocalCollisions (int) – the total number of local collisions (SSRC collisions).	Desirable.
Not provided by MIB.	RemoteCollisions (int) – the total number of remote collisions (SSRC collisions).	Desirable.
Not provided by MIB.	PacketsLooped (int) – the total number of packets looped.	Desirable.
Not provided by MIB.	FailedTransmission (int) – the number of packets that failed to get transmitted.	Desirable.
Not provided by MIB.	UnknownRTCPType (int) – the number of individual RTCP packets types that were not implemented or not recognized.	Desirable.

SenderTable

RTP MIB	JMF-based RTP Monitor Application	Comments
SenderSSRC (Unsigned32) – the sender synchronization source identifier	SenderSSRC (long)	Slight type mismatch, but workable.
SenderCNAME (DisplayString) – the canonical name of the sender	SenderCNAME (String)	-
SenderAddress (Taddress) – the unicast transport source address of the sender.	Not provide by JMF.	-
SenderPackets (Counter64) – count of RTP packets sent by this sender, or observed by an RTP Monitor, since SenderStartTime.	SenderPackets (int)	Called ProcessedPDU by JMF – number of valid packets received from the selected source. Possible type issue. It can overflow.
SenderOctets (Counter64) – count of RTP octets sent by this sender or observed by an RTP Monitor, since SenderStartTime.	Not provided by JMF.	JMF only has a corresponding session count.
SenderTool (DisplayString) – Name of the application program source of the stream.	Not implemented.	It can be read from JMF RTPSourceDescription object.
SRs (Counter32) – a counter of the number of RTCP Sender Reports that have been sent from this sender or observed if the RTP entity is a monitor, since SenderStartTime.	Not implemented.	Not part of native JMF statistics. It can be derived in an JMF application by monitoring RecvSenderReportEvents.
SenderSRTime (TimeStamp) – the value of SysUpTime at the time that the last SR was received from this sender, in the case of a monitor or receiving host, or sent by this sender, in case of a sending host.	Not implemented.	It can be read from JMF RTPSenderReport object.
SenderPT (integer 0..127) – static or dynamic payload type from the RTP Header.	Not provided by JMF.	-
SenderStartTime (TimeStamp) – the value of SysUpTime at the time this row was created.	Not implemented.	It is the time when the Monitor detected this source. It can be derived by the JMF application.

Not provided by MIB.	LostPDU (int) – the difference between the number of packets expected as determined by the RTP sequence number range and the count of packets actually received and validated.	Desirable. **Bill Strahm comments:** "RcvrLostPackets in the Receiver Table".
Not provided by MIB.	MisorderedPDU (int) – the total number of data packets that came in out of order as per the RTP sequence number.	Desirable. **Bill Strahm comments:** "Not available in the MIB. It would have to be in the RcvrTable.
Not provided by MIB.	InvalidPDU (int) – the total number of RTP data packets that have failed to be within an acceptable sequence number range for an established SSRC id.	Desirable. **Bill Strahm comments:** "Not available in the MIB. It would have to be in the RcvrTable.
Not provided by MIB.	DuplicatePDU (int) – the total number of RTP data packets that match the sequence number of another already in the input queue.	Desirable. **Bill Strahm comments:** "Not available in the MIB. It would have to be in the RcvrTable.

RcvrTable

RTP MIB	JMF-based RTP Monitor Application	Comments
RcvrSRCSSRC (Unsigned32) – the SSRC of the sender	RcvrSRCSSRC (long)	Type issue.
RcvrSSRC (Unsigned32) – the SSRC of the receiver	RcvrSSRC (long)	Type issue.
RcvrCNAME (DisplayString) – the canonical name of the receiver.	RcvrCNAME (String)	-
RcvrAddr (Taddress) – the unicast transport address of the receiver.	Not implemented.	It can be added.

RcvrRTT (Gauge32) – the round trip time measurement taken by the source of the RTP stream.	Not provided by JMF.	This value can only be calculated by senders after receiving RTP feedback about their streams. JMF does not provide this statistic. This capability was requested was requested in the jmf-interest mailing list.
RcvrLostPackets (Counter64) – a count of RTP packets lost as observed by this receiver.	RcvrLostPackets (long)	-
RcvrJitter (Gauge32) – an estimate of delay variation as observed by this receiver.	RcvrJitter (long)	Type issue.
RcvrTool (DisplayString) – the name of the application program source of the stream.	Not implemented.	It can be read from JMF RTPSourceDescription object.
RRs (Counter32) – a count of the number of RTCP Receiver Reports that have been sent from this receiver since RcvrStartTime.	Not implemented.	Not part of native JMF statistics. It can be derived in an JMF application by monitoring RecvSenderReportEvents.
RcvrRRTime (TimeStamp) – the value of SysUpTime at the last RTCP Receiver Report was received or sent (in case of the sender).	Not provided by JMF.	JMF does not provide the time a feedback is received or sent. This capability was requested was requested in the jmf-interest mailing list.
RcvrPT (Integer) – static or dynamic payload type from the RTP header.	Not provided by JMF.	-
RcvrPackets (Counter64) – count of RTP packets received by this RTP host since RcvrStartTime.	Not implemented.	Called ProcessedPDU by JMF – number of valid packets received from the selected source. Same as SenderPackets in the SenderTable.
RcvOctets (Counter64) – count of RTP octets received since RcvrStartTime.	Not implemented.	Not provided by JMF. . Same as SenderOctets in the SenderTable.
RcvStartTime (TimeStamp) – the value of SysUpTime at the time that this row is created.	Not implemented.	It can be added.

APPENDIX F. RTPHEADER JAVADOC

The RtpHeader Javadoc is available at:

http://www.web3d.org/WorkingGroups/vrtp/javadoc/dis-java-vrml/mil/navy/nps/dis/RtpHeader.html

APPENDIX G. RTPHEADER SOURCE CODE

```
package mil.navy.nps.dis;

import mil.navy.nps.util.*;
import java.io.*;

/**
 * This class encapsulates the header of the Real-time Transport Protocol (RTP)
 * when used to transfer DIS packets as a payload.
 *
 * @version 1.0
 * @author Francisco Afonso (afonso@cs.nps.navy.mil)
 *
 * <dt><b>References:</b>
 * <dd>RTP: (RFC1889) <a href="http://www.ietf.org/internet-drafts/draft-ietf-
 * avt-rtp-new-04.txt">
 * http://www.ietf.org/internet-drafts/draft-ietf-avt-rtp-new-04.txt</a>
 *
 */

public class RtpHeader extends PduElement
{
   // this SSRC will be used for all transmitted packets
   private static long mySSRC;

   // this variable contains the next sequence number of a transmitted packet
   private static int  nextSequenceNumber;

   static
   {

      // assigns a random integer to the SSRC
      mySSRC  = (long)( Math.random() * UnsignedInt.MAX_INT_VALUE );

      // assign a random integer to the first sequence number
      nextSequenceNumber = (int)(Math.random()*UnsignedShort.MAX_SHORT_VALUE );
   }


   /**
    * Identifies the version of RTP (2 bits). RFC1889 defines the actual
    * version as two(2).
    *
    */
   public static final int RTP_VERSION = 2;
```

```
/**
 * Padding is being performed at the DIS protocol level.
 * Therefore the padding bit is set to zero.
 *
 */
public static final int RTP_PADDING = 0;


/**
 * The extension bit defines if the normal header will be followed
 * by an extension header.
 * Not needed in this application, and so set to zero.
 *
 */
public static final int RTP_EXTENSION = 0;


/**
 * Contains the number of contributing source identifiers in this header.
 * This is used only by mixers. Set to zero.
 *
 */
public static final int RTP_CSRC_COUNT = 0;


/**
 * This bit is used as a marker by a specific profile or
 * application.
 * Not used so far. Set to zero.
 *
 */
public static final int RTP_MARKER = 0;


/**
 * The payload type number was set to 111.
 * It belongs to the dynamic assignment range (96-127).
 * Numbers in this range do not need to be registered. <p>
 * See: <a href="http://www.ietf.org/internet-drafts/draft-ietf-avt-profile-
 *  new-06.txt">
 * http://www.ietf.org/internet-drafts/draft-ietf-avt-profile-new-06.txt</a>
 *  - Session 3.
 *
 *
 */
public static final int RTP_PAYLOAD_TYPE_FOR_DIS = 111;
```

```
/**
 * Contains the size of the header in bytes (= 12).
 */
public static final int sizeOf = 12;

// the packet sequence number
private UnsignedShort sequenceNumber;

// the packet timestamp
private UnsignedInt   timestamp;

// the packet Sincronization Source Identifier (SSRC)
private UnsignedInt   SSRC;


/**
 * Constructor. An empty header is created.
 */
public RtpHeader()
{
   sequenceNumber = new UnsignedShort();
   timestamp = new UnsignedInt();
   SSRC      = new UnsignedInt();

   return;
}


/**
*  Returns the packet sequence number.
*  @return the sequence number as an unsigned short (16 bits)
*/
public UnsignedShort getSequenceNumber()
{
  return (UnsignedShort)sequenceNumber.clone();
}


/**
*  Returns the packet timestamp.
*  @return the timestamp as an unsigned int (32 bits)
*/
public UnsignedInt getTimestamp()
{
  return (UnsignedInt)timestamp.clone();
}


/**
*  Returns the packet Syncronization Source Identifier.
*  @return the SSRC as an unsigned int (32 bits)
*/
public UnsignedInt getSSRC()
{
   return (UnsignedInt)SSRC.clone();
}
```

```
/**
*  Sets the packet sequence number.
*  @param pSequenceNumber the sequence number as an unsigned short
*  (16 bits)
*/
public void setSequenceNumber(UnsignedShort pSequenceNumber)
{
   sequenceNumber = pSequenceNumber;
}


/**
*  Sets the packet timestamp.
*  @param pTimestamp the timestamp as an unsigned int (32 bits)
*/
public void setTimestamp(UnsignedInt pTimestamp)
{
   timestamp = pTimestamp;

}


/**
*  Sets the Syncronization Source Identifier.
*  @param pSSRC the SSRC as a unsigned int (32 bits)
*/
public void setSSRC(UnsignedInt pSSRC)
{
   SSRC = pSSRC;
}


/**
*  Increments the sequence number. The RtpHeader class mantains a static
*  variable with the next sequence number to be assigned to a packet.
*  This function increments this variable. If the sequence number will
*  exceed the 32 bits boundaries it is set to zero.
*/
private void incrementSequenceNumber()
{

   // if after the increment the sequence number gets longer than 16
      bits
   // than it should be set to zero
   ++nextSequenceNumber;
   if( nextSequenceNumber > UnsignedShort.MAX_SHORT_VALUE ){
         nextSequenceNumber = 0;
   }

   return;
}
```

```
/**
 *  Prepares the header for sending. Assigns the sequencial number from
 *  a static variable, takes the timestamp from the DIS pdu and sets the SSRC.
 *  @param pdu the DIS pdu that will be transmitted
 */
 public void prepareToSend( ProtocolDataUnit pdu )
 {

    // assigns a sequence number (the next sequence number kept by a
       static
    // variable)
    sequenceNumber = new UnsignedShort( nextSequenceNumber );

    // increments the next sequence number variable
    incrementSequenceNumber();

    // assigns as a timestamp the Dis-Java-Vrml timestamp
    timestamp = pdu.getTimestamp();

    // assigns the common SSRC
    SSRC = new UnsignedInt( mySSRC );

    return;
 }


 /**
 *  Returns the size of the header.
 *  @return the header size
 */
 public int length()
 {
    return RtpHeader.sizeOf;
 }

/**
 *  Serializes the header into a DataOutputStream.
 *  @param outputStream the stream that will receive the serialized  header.
 */
 public void  serialize(DataOutputStream outputStream)
 {
    // creates the first and second byte from the header
    UnsignedByte  firstByte = new UnsignedByte( (RTP_VERSION * 64) +
        (RTP_PADDING * 32) + (RTP_EXTENSION * 16) + RTP_CSRC_COUNT );
    UnsignedByte  secondByte = new UnsignedByte( (RTP_MARKER * 128 )
         + RTP_PAYLOAD_TYPE_FOR_DIS );

    // serializes
    firstByte.serialize(outputStream);
    secondByte.serialize(outputStream);
    sequenceNumber.serialize(outputStream);
    timestamp.serialize(outputStream);
    SSRC.serialize(outputStream);
    return;
 }
```

```
   /**
   *  Fills the header contents with data from a DataInputStream
   *  @param inputStream the stream which contains the header.
   */
   public void  deSerialize(DataInputStream inputStream)
   {
      UnsignedByte firstByte = new UnsignedByte(0);
      UnsignedByte secondByte = new UnsignedByte(0);

      // deserializes
      firstByte.deSerialize(inputStream);
      secondByte.deSerialize(inputStream);
      sequenceNumber.deSerialize(inputStream);
      timestamp.deSerialize(inputStream);
      SSRC.deSerialize(inputStream);

      return;
   }


   /**
   *  Makes deep copies of all the instance variables.
   *
   */
   public Object clone()
   {
      RtpHeader newHeader = (RtpHeader)super.clone();

      newHeader.setSequenceNumber(this.getSequenceNumber());
      newHeader.setTimestamp(this.getTimestamp());
      newHeader.setSSRC(this.getSSRC());

      return newHeader;
   }

   /**
   *  Prints internal values for debugging.
   *
   */
   public void printValues(int indentLevel, PrintStream printStream)
   {
      StringBuffer  buf =
            ProtocolDataUnit.getPaddingOfLength(indentLevel);

      printStream.println(buf + "sequenceNumber: " +
            sequenceNumber.intValue());
      printStream.println(buf + "timestamp: " + timestamp.longValue());
      printStream.println(buf + "SSRC: " + SSRC.longValue());

      return;
   }

} // end of class RtpHeader
```

LIST OF REFERENCES

Advanced Neworks and Services, "Internet2 - Building the Next Generation Internet," [http://www.advanced.org/surveyor/]. August 1997.

Agarwal, Deb, "Rtpmon Information," [http://www-itg.lbl.gov/mbone/rtpmon.tips.html]. June 1997.

Apple Computer, Inc., *Quicktime 4*, [http://www.apple.com/quicktime/]. August 1999.

Baugher, Mark and others, "Real-Time Transport Protocol Management Information Base", Internet-Draft draft-ietf-avt-rtp-mib-05.txt, Internet Engineering Task Force, 12 April 1999. Available at: http://www.ietf.org/internet-drafts/draft-ietf-avt-rtp-mib-05.txt.

Booch, G., Jacobson, I., and Rumbaugh, J., "UML Specification version 1.1." 1 September 1997. Available at: http://www.rational.com/uml/index.jtmpl.

Brutzman, D. P., "Virtual Reality Transfer Protocol (vrtp)," [http://www.web3d.org/WorkingGroups/vrtp/]. June 1999.

Case, J., McCloghrie, K., and others, "Introduction to Community-based SNMPv2," RFC 1901, SNMP Research, Inc, Cisco Systems, Inc., Dover Beach Consulting, Inc., International Network Services. January 1996.

David, B., "rtpmon: A Third-Party RTCP Monitor," [http://bmrc.berkeley.edu/~drbacher/projects/mm96-demo/index.html]. September 1996.

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

Gamma, E., and others, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley Pub Co, 1995.

Grand, M., *Patterns in Java*, John Willey & Sons, Inc., 1998.

Handley, M. and Perkins, C., "Guidelines for Writers of RTP Payload Format Specifications," Internet-Draft draft-ietf-avt-rtp-format-guidelines-03.txt, Internet Engineering Task Force, 24 June 1999. Available at: http://www.ietf.org/internet-drafts/draft-ietf-avt-rtp-format-guidelines-03.txt.

Internet Engineering Task Force, Request for Comments (RFC) 2327, *SDP: Session Description Protocol*, April 1998. Available at: http://ietf.cnri.va.us/rfc/rfc2327.txt.

Institute of Electrical Electronic Engineers (IEEE), *Standard for Distributed Interactive Simulation IEEE Std 1278.1*, 1995.
Macedonia, M. R. and Brutzman, D. P., "MBone Provides Audio and Video Across the Internet," IEEE Computer, vol. 27, no. 4, pp. 30-36, April 1994. Available at: ftp://taurus.cs.nps.navy.mil/pub/i3la/mbone.html

Makofske, D. and Almeroth, K., "MHealth: A Real-Time Multicast Tree Visualization and Monitoring Tool,"
[http://imj.ucsb.edu/mhealth/]. June 1999.

Rational Software Corp., "Rational Rose,"
[http://www.rational.com/products/rose/index.jtmpl]. August 1999.

Robinson, J. L. and Stewart, J.A., "MultiMON - an IPmulticast Monitor."
[http://www.merci.crc.doc.ca/mbone/MultiMON]. June 1998.

Sarac, K. and Almeroth, K., "SDR Session Monitoring Effort - Global Sessions,"
[http://imj.ucsb.edu/sdr-monitor/global/index.html]. September 99.

Schulzrinne, Casner and others, "RTP: A Transport Protocol for Real-Time Applications," Internet-Draft draft-ietf-avt-rtp-new-04.txt (RFC 1889), Internet Engineering Task Force , 25 June 1999. Available at:
http://www.ietf.org/internet-drafts/draft-ietf-avt-rtp-new-04.txt.

Schulzrinne, "RTP Profile for Audio and Video Conferences with Minimal Control," Internet-Draft draft-ietf-avt-profile-new-06.txt (RFC 1890), Internet Engineering Task Force, 25 June 1999. Available at:
http://www.ietf.org/internet-drafts/draft-ietf-avt-profile-new-06.txt.

Stallings, W., *Data and Computer Communications*, Fifth Edition, pp. 685-697, Prendice Hall, 1997.

Sun Microsystems, Inc., "Java Media Framework API,"
[http://www.javasoft.com/products/java-media/jmf/index.html]. July 1999.

Sun Microsystems, Inc., *Java Media Framework API Programmer's Guide v. 0.7*, 21 May 1999. Available at:
http://www.javasoft.com/products/java-media/jmf/2.0/jmf20-07-guide.pdf

Sun Microsystems, Inc., *Java Media Framework Early Access Specification v. 0.7 (Javadoc)*. May 1999. Available at:
http://www.javasoft.com/products/java-media/jmf/2.0/jmf20-07-apidocs/index.html

Sun Microsystems, Inc., "Java Native Interface,"
[http://java.sun.com/products//jdk/1.2/docs/guide/jni/index.html]. August 1999.

UCL Networked Multimedia Research Group, "Videoconferencing Tool,"

[http://www-mice.cs.ucl.ac.uk/multimedia/software/vic/]. June 1999.

UCL Networked Multimedia Research Group, *User Guide for VIC v2.8 Version 1 (DRAFT),* 29 September 1998.Available at: http://www-mice.cs.ucl.ac.uk/multimedia/software/.

UCL Networked Multimedia Research Group, “Robust-Audio Tool,” [http://www-mice.cs.ucl.ac.uk/multimedia/software/rat/]. June 1999.

UCL Networked Multimedia Research Group, “Session Directory,” [UCL Networked Multimedia Research Group]. August 1999.

Word Wide Web Consortium, “Extensible Markup Language,” [http://www.w3.org/XML/]. August 1999.

THIS PAGE LEFT INTENTIONALLY BLANK

INITIAL DISTRIBUTION LIST

No. Copies

1. Defense Technical Information Center ……....2
   8725 John J. Kingman Rd., STE 0944
   Ft. Belvoir, Virginia 22060-6218

2. Dudley Knox Library ……..2
   Naval Postgraduate School.
   411 Dyer Rd.
   Monterey, California, 93943-5101

3. Chair, Code CS ……..... 1
   Department of Computer Science
   Naval Postgraduate School
   Monterey, California, 93943-5121

4. Dr. Michael J. Zyda, Code CS/Zk ……..…...1
   Naval Postgraduate School
   Monterey, California, 93943-5121

5. Dr. James Eagle, Code UW…..……..…....1
   Naval Postgraduate School
   Monterey, California, 93943-5121

6. Dr. Don Brutzman, Code UW/Br ……..…..... 1
   Naval Postgraduate School
   Monterey, California, 93943-5121

7. Rex Buddenberg, Code SM/Bu ……..... 1
   Naval Postgraduate School
   Monterey, California, 93943-5121

8. Don McGregor, Code C3……..……..... 1
   Naval Postgraduate School
   Monterey, California, 93943-5121

9. Dr. Michael R. Macedonia 1
   Chief Scientist and Technical Director
   US Army STRICOM
   12350 Research Parkway
   Orlando, FL 32826-3276

10. Dr. J. Mark Pullen 1
Department of Computer Science and C3I Center
George Mason University
Fairfax, VA 22030

11. Jaron Lanier 1
Chief Scientist
Advanced Network & Services, Inc.
200 Business Park Drive
Armonk, NY 10504 USA

12. Bob Barton 1
Fraunhofer CRCG
321 South Main St.
Providence, RI 02903

13. Michael D. Myjak 1
Vice President R&D
The Virtual Workshop, Inc.
P.O. Box 98
Titusville, Florida 32781

14. Dr. Christophe Diot 1
Sprint ATL
1 Adrian Court
Burlingame, CA 94010

15. Dr. Jon Crowcroft 1
Department of Computer Science
University College London
Gower Street
London WC1E 6BT
United Kingdom

16. Kevin C. Almeroth 1
Computer Science Department
University of California
Santa Barbara, CA 93106

17. Ivan Wong 1
Java Media Framework Techical Lead
Sun Microsystems
901 San Antonio Road
Palo Alto, CA 94303

18. Bill Strahm 1
Intel Corporation
2111 N.E.25th Avenue
Hillsboro, Oregon 97124

19. Lawrence A. Rowe 1
Computer Science Division - EECS
University of California, Berkeley
Berkeley, CA 94720-1776

20. Instituto de Pesquisas da Marinha 1
Rua Ipiru, 2
Ilha do Governador
Rio de Janeiro – RJ – Brazil

21. Diretoria de Sistemas de Armas da Marinha 1
Rua Primeiro de Março, 118
Rio de Janeiro – RJ – Brazil
CEP 20010

22. Centro de Instrução Almirante Wandenkolk 1
Ilha das Enxadas
Rio de Janeiro – RJ – Brazil
CEP 20000

23. LCDR Francisco Carlos Afonso ……..... 3
Rua Marques de Valenca, 40/201
Tijuca
Rio de Janeiro – RJ – Brazil
CEP 20010