

Fachbereich Elektrotechnik und Informationstechnik  
Institut für Datentechnik  
Fachgebiet Industrielle Prozeß- und Systemkommunikation  
Prof. Dr.-Ing. Ralf Steinmetz

## **Technische Universität Darmstadt**



### **Diplomarbeit**

# Design of a Multiformat Capable Cache for Video Streaming

von

Gunnar Gudmundsson  
September 2000

Betreuer: Dipl.-Ing. Michael Zink

Nr. KOM-D-0121

## **Ehrenwörtliche Erklärung**

Hiermit versichere ich, die vorliegende Diplomarbeit ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus den Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, den 7. September 2000

Gunnar Gudmundsson

# Acknowledgements

I would like to thank the supervisor, Dipl.-Ing. Michael Zink for many very successful discussions and a successful cooperation during my work on this project.

This work was supported by the D.A.A.D, the German Academic Exchange Service. I would like to thank the D.A.A.D for their support that made it possible for me to come to Germany and continue my studies.

I would gracefully like to acknowledge the help of Prof. Dr-Ing. Ralf Steinmetz for making it possible for me to work on this thesis at his institute.

I would like to thank my wife Gu nand my daughter Gu rúnMaría for all their support during my work on this project.

# Table of Contents

Ehrenwörtliche Erklärung .....	ii
Acknowledgements .....	iii
List of Figures .....	vi
1. Introduction .....	1
2. Real-Time Communication and Protocols in the Internet .....	2
2.1 Real-Time Communication .....	2
2.2 Internet Protocols for Real-Time Communications .....	4
2.2.1 RTP.....	4
2.2.2 RTSP - Real-Time Signaling Protocol.....	12
2.2.3 Loss Collection - LC-RTP and LC-RTCP .....	14
2.3 Caching approaches for media streaming .....	16
2.3.1 General .....	16
2.3.2 Video-on-Demand.....	18
2.3.3 Caching support in RTSP.....	20
2.3.4 Proposal for caching support in standards-based RTSP/RTP servers.....	22
3. KOM Implementation of RTSP/RTP .....	25
3.1 Class Diagram .....	25
3.2 Instance Drawing.....	26
4. A Design of an RTSP Capable Proxy.....	30
4.1 Design goals .....	31
4.2 Specification.....	32
4.2.1 Architecture Overview .....	32
4.2.2 A session with multicast distribution and packet recording.....	35
4.2.3 A session with streaming from cache and reflection.....	37
4.2.4 RTSP Administration .....	39
4.2.5 RTP Administration and Streaming .....	43
4.2.6 Cache Directory.....	49
4.2.7 Protocol Scenarios.....	49
5. Multiformat Support.....	51
5.1 Design goals .....	52
5.2 Specification.....	53
5.2.1 Architecture Overview .....	53
5.2.2 RTSP Administration .....	55
5.2.3 RTP Administration and Streaming .....	56
5.2.4 Caching Strategy .....	57
5.2.5 Protocol Scenarios.....	59
6. Summary.....	64
7. Further Work .....	67
Appendix A .....	68
Appendix B .....	70
Appendix C .....	74

Appendix D .....	79
Appendix E.....	85
References .....	94

# List of Figures

Fig. 1:	Real-time communication environment .....	3
Fig. 2:	An RTP packet header .....	8
Fig. 3:	Reliable data transmission with LC-RTP .....	16
Fig. 4:	Streaming with RTSP, LC-RTP and LC-RTCP .....	17
Fig. 5:	A caching hierarchy .....	19
Fig. 6:	Cut through caching.....	21
Fig. 7:	A Use Case for the KOM RTSP/RTP server implementation.....	28
Fig. 8:	A class diagram for the KOM RTSP/RTP server implementation.....	29
Fig. 9:	Multicast streaming and packet recording into a cache.....	36
Fig. 10:	Streaming from cache and reflecting .....	38
Fig. 11:	A class diagram for the RTSP/RTP proxy design .....	40
Fig. 12:	A datapath for streaming from a cache.....	47
Fig. 13:	A datapath for reflection and caching.....	48
Fig. 14:	Caching and formatswitching .....	52
Fig. 15:	A datapath for recording with formatswitching .....	58
Fig. 16:	RTSP scenario for streaming from origin server and recording.....	84
Fig. 17:	RTSP scenario for streaming from proxy cache.....	93

# 1. Introduction

Media streaming applications have become more popular on the Internet recently and will probably be installed on many end-systems in the near future. Such applications make much greater demands on the network and hosts than temporary web applications. They require much bandwidth and stable delay in the transmission and much storage space in hosts. One way of meeting these requirements is to store media content in a number of caches at different locations in the network and provide the clients with streaming from the caches. Caching increases the overall performance of the streaming in comparison to streaming from a central server because less network resources are applied (streams traverse fewer network hops) and the streaming effort is distributed among a number of hosts.

The network technology applied in the Internet today provides end users with best effort service and does not guarantee the provision of network service with specified quality during a communication session. The techniques for such provisions have been developed but they are not widely applied in the Internet at present. Consequently, a media streaming process experiences different quality in the data delivery dependent of the conditions in the network at each time. A streaming server can adapt its streaming to the conditions in the network. The server can send a media stream in a format requiring high bandwidth transmission and providing good quality as long as the client experiences good quality and no congestion in the network. The server switches to another format that requires lower bandwidth as soon as congestion is experienced. This approach to adapting media streaming to conditions in the network is called *formatswitching* and a server performing the procedure provides *multiformat support*.

The goal of this thesis is to design a real-time streaming proxy. The proxy must be able to store media content into a local cache and provide clients with media streaming from the cache upon request. The proxy should be able to switch between different encodings during streaming for the purpose of adapting to changing transmission conditions in the network. The design should be an extension to an existing real-time streaming server design that was developed at the institute.

The structure of the thesis is as follows: An overview of real-time communication and corresponding protocols in the Internet is given in Chapter 2 and an overview of the existing streaming server implementation is given in Chapter 3. The design of the real-time streaming proxy is separated into two chapters. The proxy functionality and necessary changes to the origin server are specified in Chapter 4. The multiformat support for the proxy and the origin server is specified in Chapter 5 as an extension to the design in Chapter 4.

## 2. Real-Time Communication and Protocols in the Internet

This section describes some issues regarding real-time communication in the Internet. In the first part the requirements of real-time data delivery to the communication network are introduced. The second part gives an introduction to some protocols for real-time communication that are used in the Internet and the chapter closes with descriptions of caching approaches that have been proposed for media streaming.

### 2.1 Real-Time Communication

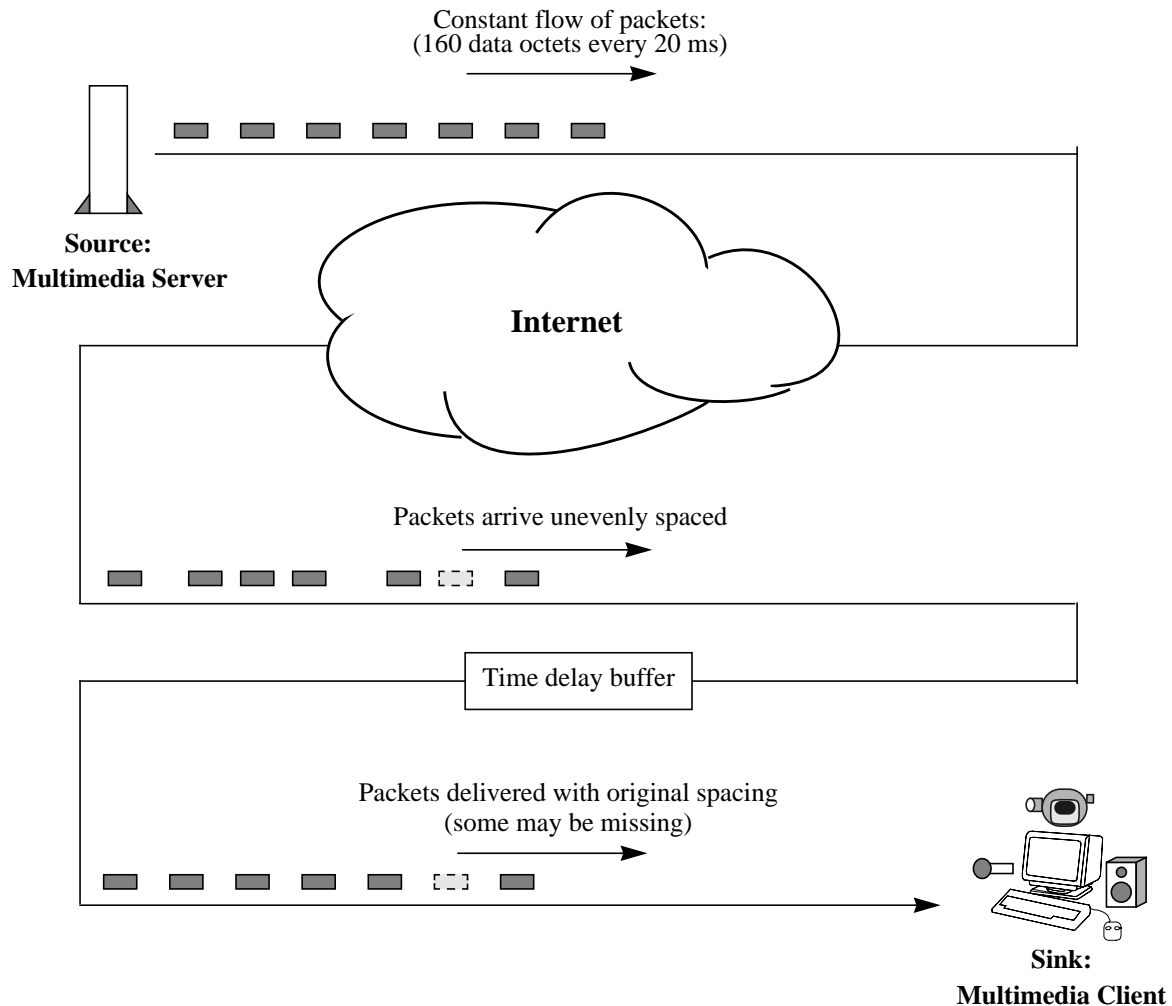
Most of the traditional internet applications including file transfer, electronic mail and world-wide web are non-real-time applications. The most important performance metric for such applications are throughput and delay. The applications typically require reliable data transfer between communicating hosts and rely on transport service that provides the data without losses and in correct order. The timing properties of the delivery are not very important for non real-time applications.

On the other hand are timing properties of the data delivery very important for real-time applications. Most of them require the network to deliver transferred data at a constant rate equal to the sending rate. Some real-time applications associate a timestamp with each block which is its deadline. A block is unusable after the deadline has expired. Following are examples of real-time applications: audio and video conferencing, telephony, command and control systems, real-time monitoring, remote medical diagnosis and video on demand [Sta98].

Fig. 1 illustrates a typical real-time communication environment. The server transmits a stream at a constant rate to the Internet. The packets are transmitted through the Internet that provides best effort service only and does not guarantee any quality of service for the transmission. The packets arrive unevenly spaced at the destination due to the varying delay introduced by the Internet. To compensate for this a buffering process buffers incoming packets and releases them at a constant rate to a decoding process. The buffer introduces an additional delay.

The buffer must be able to compensate for the maximum variation in delay that the packets experience which is called *delay jitter*. The delay jitter depends on the conditions in the Internet and determines the required size of the buffer.

Real-time applications can be categorized into two groups, soft and hard real-time applications. Soft real-time applications are tolerant for loss of some parts of the data that is transferred between applications but hard real-time applications are not tolerant for



**Fig. 1: Real-time communication environment [Sta98]**

any data losses [Sta98]. Audio and video conferencing application and video on demand are examples of soft real-time applications. Remote medical diagnosis and command and control systems in a power plant are examples of hard real-time applications.

Media streaming is the process of providing a client with audio and/or video content (e.g. movies or TV and radio programmes) from a server.

The transport level protocol TCP [RFC793] is widely used by applications in the Internet. It provides reliable data transfer over IP networks and works well with non real-time applications but is not suited for real-time applications. The following properties disqualify it for use as the transport protocol for such applications [Sta98]:

- In real-time communication it is necessary to associate timing information within transferred data. TCP is not able to provide this service.
- TCP is not suitable for multicast communication because it is limited to set up a connection between two end points.

- TCP provides a reliable data transfer over unreliable network. It requests retransmission of lost packets and restores the original packet sequence. This service introduces delay into the data delivery and can violate the timing requirements of the real-time communication between sender and receiver.

Another widely used transport protocol in the Internet, UDP [RFC768] is well suitable to multicast communication and does not introduce delay due to retransmission of lost packets but it is not capable of associating timing information within transferred data.

A real-time transport protocol is described in the next section. The protocol implements mechanism for associating timing information within packets and is well suited for soft real-time applications. The protocol is not a fully functional transport protocol but is typically run over UDP.

## 2.2 Internet Protocols for Real-Time Communications

The following sections describe a protocol framework that is typically used for real-time communication in the Internet.

### 2.2.1 RTP

RTP, the real-time transport protocol framework, provides end-to-end delivery services for data with real-time characteristics. These services are suitable for various distributed applications that transmit real-time data, such as interactive audio and video.

In the following the properties of the protocol framework are stated [RFC1889]:

- RTP can be used to carry various formats of real-time data. The protocol provides a **payload type identification** that makes it possible for the receiver to recognize the format of the data being carried by the protocol.
- RTP does not guarantee delivery or prevent out-of-order delivery nor does it assume that the underlying network is reliable and deliver packets in sequence. The protocol marks each transmitted packet with a **sequence number** that enables the receiver to reconstruct the correct packet order. The protocol provides a **delivery monitoring** mechanism that is able to monitor the quality of the data delivery at the receiver's end and report the results to the sender. The monitoring provides the sender with a feedback and the sender can adapt the characteristics of its delivery to the results from the monitoring.
- RTP does not provide any mechanism to ensure timely delivery and neither does it provide quality-of-service guarantees nor does it address resource reservation issues. The protocol leaves it to the underlying network to fulfill the requirements that the application makes to the transmission.

- RTP provides a **timestamping** mechanism that delivers a timestamp within the data transmission from the sender to the receiver. The timestamp corresponds to the generation instant of the corresponding data and makes it possible for the receiver to perform scheduling and synchronization. The timestamp makes it possible for the receiver to measure variation in the transmission delay.
- Additionally to the contemporary point-to-point data transfer using unicast transmission, the RTP supports data transfer to multiple destinations using multicast distribution if provided by the underlying network.
- RTP does not provide multiplexing/demultiplexing of data packets itself but relies on the corresponding capabilities of underlying networks.

RTP itself does not implement a fully functional real-time transport protocol. A cooperation with another transport protocol is needed. Although RTP can be used in cooperation with different underlying transport protocols it is typically run on top of UDP. UDP is message oriented and provides multiplexing/demultiplexing and checksum services. It introduces much less overhead than TCP because it lacks the error control and flow control that TCP provides and can be used in multicast distribution. It is well suited as an underlying protocol for RTP. Together RTP and UDP form the functionality of a real-time transport protocol well suited for soft real-time applications.

Typical applications using RTP for delivering real-time data include multiparticipant interactive multimedia conferences, Internet telephony sessions and media streaming sessions from a mediaserver.

The RTP defines two types of RTP relays, mixers and translators [RFC1889]:

- A **mixer** receives data packets from one or more sources, possibly changes the data format, combines the data into one new packet and forwards it. The mixer makes timing adjustments among the received streams and generates its own timing and a new synchronization identifier (see Section 2.2.1.2 and Section 2.2.1.3) for the resulting stream. A mixer can be applied at the border between a high-speed network and a low-speed network. A group of streams is received from the high-speed network, mixed into one stream and transmitted into the low-speed network. The mixer enables the users who are connected to the lower-speed network to take part in the audio conference and does not affect the transmission quality of the communication between the participants connected to the high-speed network.
- A **translator** receives data packets and forwards them without mixing or changing the synchronization of the stream. A translator might modify the payload data of a packet but the synchronization source identifier is not altered. A translator can be used to convert encodings of data streams or replicate from multicast to unicast. A translator can also be used as an application-level filter in a firewall.

The functionality of the protocol framework is separated into two protocols, each having its own purposes:

- **RTP Data Transport Protocol - RTP** carries data that has real-time characteristics.
- **RTP Control Transport Protocol - RTCP** monitors the quality of the data delivery and manages information on the active streams in the on-going communication session.

### 2.2.1.1 Protocol Architecture

RTP is indeed not an independent protocol itself, rather is it best viewed as a framework that applications use directly to implement a single protocol. Without the application-specific information, RTP is not a full protocol and there is a strong coupling between the functionality of RTP and the functionality of the application [Sta98].

Most distributed real-time applications are more tolerant to some losses in the data and receiving some data out of sequence than to delay in the transmission. RTP takes this into account and does neither recover data that is lost in the transmission nor restore the packet sequence. It is up to the application to take care of these issues if necessary. RTP does not provide the application with a reliable data delivery but it performs calculation of the quality of the service and provides the application with the results with its companion protocol RTCP. RTP run on top of UDP/IP does introduce much less delay into the transmission than TCP/IP would do.

For each encoding to be carried by the protocol, it is necessary to specify an application specific profile which specifies how the applications apply RTP for carrying the specific encoding [RFC1889].

### 2.2.1.2 RTP Data Transfer Protocol

In this section the term RTP refers to the RTP data transfer protocol.

The purpose of the RTP Data Transfer Protocol is to carry data that has real-time properties.

The protocol supports data transfer among a number of participants in a session. An RTP session is a logical association among two or more RTP entities that are communicating with the protocol. The RTP session from the viewpoint of a particular participant can be identified by network addresses and a pair of port numbers [Sta98]:

- **Participant IP addresses:** If the participants communicate via IP multicast the multicast group defines the set of participants and this is the group's multicast IP address. If the participants communicate via IP unicast this is a set of unicast IP addresses.
- **RTP port number:** All participants in a session use the same destination port address for RTP transfers.
- **RTCP port number:** All participants in a session use the same destination port address for RTCP transfers.

Packets of the underlying transport protocol (e.g UDP) that encapsulate the RTP packets contains these information in the address fields. These fields are necessary for the correct routing in the network and delivery to the correct process (multiplexing/demultiplexing) in the transport protocol.

The management of the participation of the entities in the session is not provided by the protocol and must be provided by another protocol when needed.

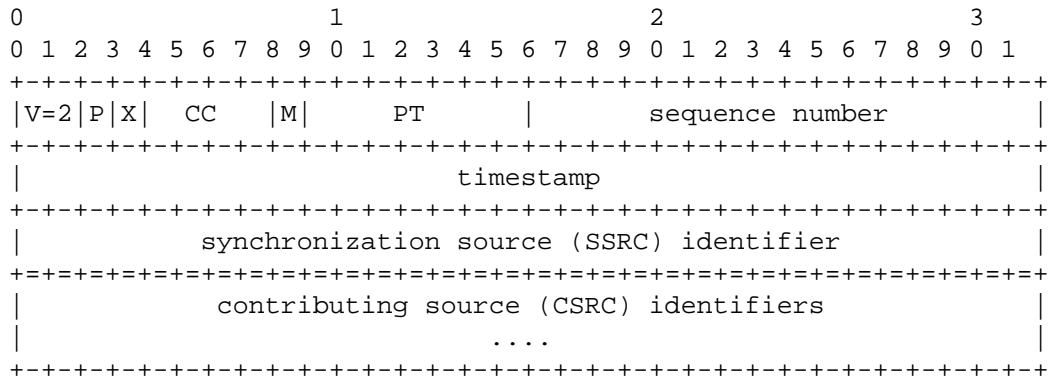
A typical application for the RTP is a distributed video conferencing session where video and audio streams must be transferred among the participants. Transmission of videosignals requires high-speed network access and video-capable hosts. Some users might not be able to send or receive videosignals but might nevertheless want to take part in the conference and send and receive audio only. Typically the transmission of audio and video signals is separated into two separate RTP sessions to support the needs of such users. The users not able to handle the videosignal only take part in the audio session but the others take part in both sessions.

### **RTP Packet Format**

RTP defines a single packet format and each RTP packet contains a fixed header and payload data. The fixed header is very compact and RTP adds a small overhead to the application's data. Real-time applications normally need a lot of bandwidth and send data in small messages to reduce the delay. A small header is therefore very important for the performance of the protocol. An RTP packet may also contain an extension header that follows the fixed header. The purpose of the extension header is to give the application the chance to provide the receiver with additional service that is specific to the application and is not provided by RTP. The format of the extension header is defined in [RFC1889].

Fig. 2 illustrates the format of an RTP packet. The fields that are important for this thesis are described shortly in the following [RFC1889]:

**payload type (PT):** This field identifies the format of the RTP payload and determines its interpretation of the application. The codes for various formats have been assigned.



**Fig. 2: An RTP packet header [RFC1889]**

**sequence number:** The sender increments the sequence number by one for each sent RTP packet that it sends. This gives the receiver the chance to detect if packets were lost during transmission and to restore the packets sequence.

**timestamp:** The timestamp corresponds to the moment when the first octet of the payload in the RTP data packet was sampled. It is derived from a clock that is incremented monotonically and linearly in time and it gives the receiver the chance to regenerate timing associations between packets and between related streams that were transmitted through separated RTP sessions. The timestamp also makes it possible for a receiver to measure packet arrival jitter.

**Synchronization source (SSRC):** This is an identification of a source of a stream of RTP packets. SSRC is chosen randomly by a sender but must be unique within an RTP session. Each sender must be able to detect and resolve a possible collision that occurs if the senders happen to choose the same SSRC. An example of a synchronization source is the sender process of a stream of RTP packets that were derived from a microphone or a videocamera. All packets from a synchronization source form part of the same timing and sequence number space so a receiver uses the identifier to group packets for playback. The SSRC identifier is also used for grouping packets for delivery monitoring, performed by receivers or network management tools.

**Contributing source (CSRC):** A synchronization source identification of source of a stream of RTP packets that has contributed to the combined stream produced by an RTP mixer. The mixer inserts a list of SSRC identifiers of the sources that contributed to the generation of the RTP packet.

### 2.2.1.3 RTP Control Protocol

The RTP data transfer protocol explained in previous section transfers user data among all participants in a session. A separate control protocol (RTCP) provides feedback to the RTP sources in the RTP session and to all participants in the session as well. The same underlying transport service is used for both protocols (usually UDP) but a different port

is used to separate the packet streams. Each participant in an RTP session periodically sends an RTCP packet to all other participants in the session. The RTCP protocol performs four functions [RFC1889]:

- **Quality of service and congestion control:** RTCP provides feedback on the quality of data distribution. The RTCP packets are sent to the multicast session that means that all members in the session can monitor how well other members are performing and receiving. Receivers estimate data rates and quality of transmitted data stream from sender reports. Senders are informed on the quality each receiver is experiencing from receiver reports. The receiver report informs on the fraction of lost packets and the jitter the receiver is experiencing. The sender might use the feedbacks to adapt the data rate of its transmitted streams to the rates the network can easily carry. A network manager monitoring an RTP session can locate a network problem from information from receiver reports.
- **Identification:** RTCP carries a persistent identifier for an RTP source, called CNAME. The CNAME provides more information on the source than the SSRC does and is more reliable because the SSRC might change if a conflict is detected or if a process must be restarted. Receivers use the CNAME to associate multiple data streams from a given participant and for related RTP sessions. This association can be used to synchronize audio and video.
- **Session size estimation and scaling:** The first two functions require that each participant sends RTCP packets periodically. In order for the RTP to scale up to a large number of participants the rate of the RTCP transmission must be controlled. Each participant receives packets from all other participants through multicast transmission, observes the number of participants in the session and controls its rate accordingly.
- **Session control:** RTCP provides primitive session control services for loosely controlled sessions where no parameter negotiations or membership control is needed. For more advanced session control services an additional session control protocol is needed.

One might argue that transferring a feedback in a multicast way produces too much packets and is overwhelming. It might be sufficient if a participant limits its feedback transmission to the corresponding senders. The idea to transfer the feedback to all participants is twofold:

- to give each member of an RTP session the chance to monitor the quality of the delivery that other members are experiencing for comparing to the one it is experiencing itself.

- to be able to build a networking management process that joins an RTP session, does not send any RTP packets to the session but monitors the quality of the delivery that all members in the session are experiencing.

The following five packet types have been defined for RTCP:

- Sender Report (SR)
- Receiver Report (RR)
- Source Description (SDES)
- Goodbye (BYE)
- Application specific

In the following some fields and features of the packet type that are important for this thesis are stated.

### **Sender Report**

Receivers send reception quality feedback to senders and other participants in the RTP session with multicast transmission. The feedback is provided using a Sender Report or a Receiver Report depending on whether the receiver also is in a sender role in the session or not. Sender Reports and Receiver Reports contain Synchronization Source Identifier (SSRC) identifying the source of the packet. The sender report contains one **sender information block** and a number of **receiver report blocks**. Each sender information block contains [RFC1889]:

**NTP Timestamp:** The absolute time when the report is sended and can be used by the sender in combination with timestamps returned in receiver reports to estimate round-trip time to receivers.

**RTP Timestamp:** A timestamp generated from the same timer as used in the RTP packets. This timestamp lets receivers place the report correctly in the appropriate time sequence with the received RTP packets from the same source.

**Sender's Packet Count:** Total number of RTP data packets transmitted by this sender so far in this session.

**Sender's Octet Count:** Total number of RTP payload octets transmitted by this sender so far in this session.

The last three fields lets the receiver calculate the average data rate of the sender and in conjunction with the received RTP packets the receiver calculates the fraction of lost data.

Each receiver report block provides a reception quality feedback for a stream from a particular sender. The block contains [RFC1889]:

**SSRC:** The identification of the source that this report block refers to.

**Fraction lost:** The fraction of RTP data packets from the SSRC that were lost since the previous receiver report block was sent.

**Cumulative number of packets lost:** Total number of RTP data packets from SSRC lost during this RTP session.

**Interarrival jitter:** An estimate of the jitter that RTP data packets from SSRC experience.

**Last SR timestamp:** The absolute timestamp from last SR packet received from the sender.

**Delay since last SR:** The delay between receipt of the last SR packet from the particular sender and the transmission of this block.

The delay jitter is defined as the maximum variation in delay experienced by the RTP packets [Sta98]. The sender of the receiver report block calculates an estimation of the jitter with a statistical exponential average. For the calculation it uses the timestamps from the received RTP packets and the times of the packets' arrivals.

The last two fields define the observation timerange for the Fraction lost field.

The fields of the reception report block are valuable for senders, receivers and network managers to monitor situations in the network. Packet loss values indicate persistent congestion and jitter indicates transient congestion in the network and can be a warning of increasing congestion developing in the network [Sta98].

## Receiver Report

The Receiver Reports include the SSRC identifier of the initiator of the packet and a set of reception report blocks but no sender report blocks.

A source provides a **Source Description packet SDES** to provide other participants with more description about itself. The packet contains various fields, for example **NAME** containing real user name of the source and **LOC** containing the geographic location of the source. A source provides the participants with a **Goodbye BYE packet** when leaving the session and the **Application-defined packet** is intended for experimental use for functions and features that are specific to a particular application.

With its optional extension header in RTP packets and the application specific RTCP packet the RTP framework gives engineers the possibility to experiment with new features for carrying various payload formats.

## 2.2.2 RTSP - Real-Time Signaling Protocol

RTSP is an application-level protocol that provides control over the delivery of real-time data. The protocol is typically applied for control over continuous time-synchronized streams of continuous media such as audio and video and acts as a "network remote control" for media servers. RTSP does not typically deliver the media streams itself but controls streams that are being carried by some transport protocol like RTP [RFC2326].

RTSP defines an RTSP session which is the complete process of streaming a movie from a media server to a client according to the requests initiated from the client, e.g. viewing of a movie. The process includes setting up a transport mechanism for the media streaming, delivering the stream from the server to the client and closing the connection and freeing resources. The server assigns each session a unique session identifier during the setup phase which identifies the session during its lifetime and corresponding messages. The RTSP session identifier is not to be confused with the session defined within the RTP protocol.

Clients and servers that communicate via RTSP may not necessarily maintain a persistent connection for the communication but may open and close many connections with a reliable transport protocol such as TCP to issue requests or communicate via a connection-less protocol such as UDP. A client may pipeline its requests to the server but the sender must process the requests and send resulting messages in the same order that the requests were received.

Each RTSP request-response pair must be assigned a sequence number specified by the CSeq field. For every RTSP request with a given CSeq, a corresponding response with the same CSeq exists. A request that is retransmitted must contain the same CSeq as the original one.

An RTSP server maintains a state for each active RTSP session. Each state is identified by the corresponding RTSP session identifier. Normally an RTSP client issues requests to a server but an RTSP server can also issue a request to a client. A server can for example request a client on its reception and decoding capabilities.

RTSP defines a presentation as a set of one or more streams presented to the clients as a complete media feed. A presentation description contains various information about media streams within a presentation including encodings and network addresses and is included within RTSP messages as a message body. Presentation descriptions are important for installing corresponding decoders and setting up network services. The format of the descriptions is not part of RTSP but typically the session description protocol, SDP is applied. For a description of SDP see [RFC2327].

RTSP provides eleven different methods for requesting delivery services or information on media streams. In the following each method is described shortly [RFC2326]:

**DESCRIBE:** This method that can be sent by a client to a server retrieves the description of a presentation or a media object from a server. The corresponding presentation or object is identified by URI in the message. The server responds with a description of the requested resource, typically in a SDP format. The request and response pair constitutes an initialization phase of RTSP but does not install a RTSP session in the server or affect its state.

**ANNOUNCE:** This method serves two purposes: When received by a client the method updates the description of an on-going session. A client receives an ANNOUNCE method when a server would like to change the encoding of an on-going media stream. When received by a sender a description of a session to be stored on the server is being posted to the server.

**GET\_PARAMETER:** This method that can be sent by a client or a server requests the value of a parameter of a presentation or a stream specified in the URI field of the method.

**OPTIONS:** This method that can be sent by a client or a server can be used to request information on the various capabilities of the communication partner.

**PAUSE:** This method that can be sent by a client to a server requests the server to interrupt the streaming of a particular presentation or a single stream within a specified session temporarily.

**PLAY:** This methods that can be sent by a client to a server requests the server to start a streaming of a particular presentation or a single stream to a session that has already been initialized.

**RECORD:** This method that can be sent by a client to a server includes a presentation description and requests a server to record a presentation specified within the description.

**REDIRECT:** This method that can be sent by a server informs the client that it must redirect its streaming requests to another server.

**SETUP:** This method that can be sent by a client to a server specifies a transport mechanism to be used for the streaming of the media specified within the URI field in the method. This method is issued in the initialization phase of a streaming process but a client might also issue a SETUP request to change the transport mechanism of a stream that is already playing. Upon reception of a SETUP method, a server creates a RTSP session state, assigns a new session identifier and inserts it to the response. All following requests and responses corresponding to the session must contain the identifier. The SETUP method might correspond to a stream locally stored on a media server or to a live stream.

**SET\_PARAMETER:** This method that can be sent by a client or a server requests to set the value of a parameter for the corresponding presentation or a single stream specified by the URI field in the message.

**TEARDOWN:** This method that is sent from a client to a server requests the server to stop streaming of the media specified within the URI field in the method and free corresponding resources. If the URI is a presentation the corresponding session identifier is deleted.

RTSP is typically used for the control of streaming of media from a media server. The scenario in Appendix A illustrates such an RTSP operation:

### **2.2.3 Loss Collection - LC-RTP and LC-RTCP**

The RTP protocol framework described in Section 2.2.1 does not provide reliable transport service but leaves it to the application to take care of retransmissions of lost data. Some streaming applications might be quite tolerable for some data loss up to a certain limit but others require a reliable transfer. A service provider operating a streaming cache would like to store the media streams in a complete form into its cache to be able to provide his customer with as good service as possible. The provider requires a reliable transfer of its streams from an origin server into its cache.

LC-RTP (Loss Collection Real-Time Transport Protocol) is an extension of the RTP protocol that was developed at the KOM Institute for reliability purposes. The protocol provides reliable transfer of content and maintains a conformity to the RTP in the way that standard RTP clients that are not capable of using the reliability advantages that LC-RTP provides are nevertheless able to receive its streams [Gri00].

The protocol's process can be split into two parts: transmission and retransmission. At first the movie or presentation is transferred with a regular RTP transmission from the beginning until the end and then lost packets are retransmitted upon request from the clients.

#### **Transmission**

This part works like a regular RTP transmission and lasts until the sender transmits a BYE message at the end of the movie/presentation. The sender streams a movie to a multicast address and the stream is received by the members of the multicast group. In addition to the regular real-time services carried by the RTP fields, the server inserts a byte count to each transmitted RTP packet that specifies the position of the payload in the stream relative to the begin [Gri00].

A standard RTP client receives the stream, prepares it for playback and/or stores it into a cache. An LC-RTP capable client performs the two following functions with the help of the byte counts in addition to the standard client:

- The client uses the byte counts to determine gaps in the stream caused by losses in the transmission. The client collects the information on the gaps into a loss-list for use in the retransmission phase.

- The client uses the byte counts to reserve correct amount of space in the file system for the lost data that is to be filled during the retransmission phase. This lets the client optimize and plan the storing of the data into the file system into its natural sequence. This enables the cache to stream more efficient.

The byte count is transferred within an extension header in each RTP packet and is RTP-conform. Standard RTP clients who are not able to interpret the byte count can receive the LC-RTP stream as well without any problems.

### **Retransmission**

After the transmission of the BYE message at the end of a movie or a presentations the sender enters the retransmission phase and collects loss-lists that are received from clients. After some time the data corresponding to the byte counts requested in the loss-lists is retransmitted to the multicast group. The collection-retransmission procedure is repeated until no more loss-lists are received or the maximum number of repeats has been reached. The server collects the requests before responding to be able to respond to many equal requests with one multicast message [Gri00].

An LC-RTP capable client enters the retransmission phase after the reception of the BYE message at the end of a movie or a presentation and after processing corresponding data. It sends the maintained loss-list to the sender and requests a set of the data referred to by the list. It uses a companion protocol to LC-RTP that is called LC-RTCP. The protocol is an extension of RTCP that provides transfer of loss-lists via application-specific packets. The client receives retransmitted data packets through the LC-RTP session and updates its loss-list. The retransmitted data is sent to a IP multicast address and a client might receive data packets it did not request. A client requests retransmission until its list is empty or the maximum number of retransmission procedure repeats has been reached. The functionality of LC-RTP is illustrated in Fig. 3.

One might argue that it is not necessary to apply a specific protocol for reliable real-time transport protocol for transport of content from an origin server into media caches. One might think that a normal reliable file transfer protocol in conjunction with TCP (e.g. ftp) could be used instead.

LC-RTP is well suited to the following scenario: A client requests a streaming procedure of a particular movie or a programme. The responsible origin server points the client to a multicast address and starts streaming the movie to the multicast group. The client joins the group and starts receiving the requested stream. A media cache follows the communication between the client and the server. The client determines that the requested stream is subject to be requested from other clients later and storing it into the cache might be valuable. The client joins the multicast group receives the stream and stores it into local storage. After the streaming has reached the end of the movie the cache enters the retransmission phase of the LC-RTP protocol and requests data from the origin server that were lost during the normal transmission with LC-RTCP and completes the storing. The

client does not enter the retransmission phase. The LC-RTP makes it possible to store a stream into cache parallel to serving a client. A transfer of the complete movie into the cache via ftp following streaming from the cache to the client is also a possibility but introduces an unacceptable delay for the client.

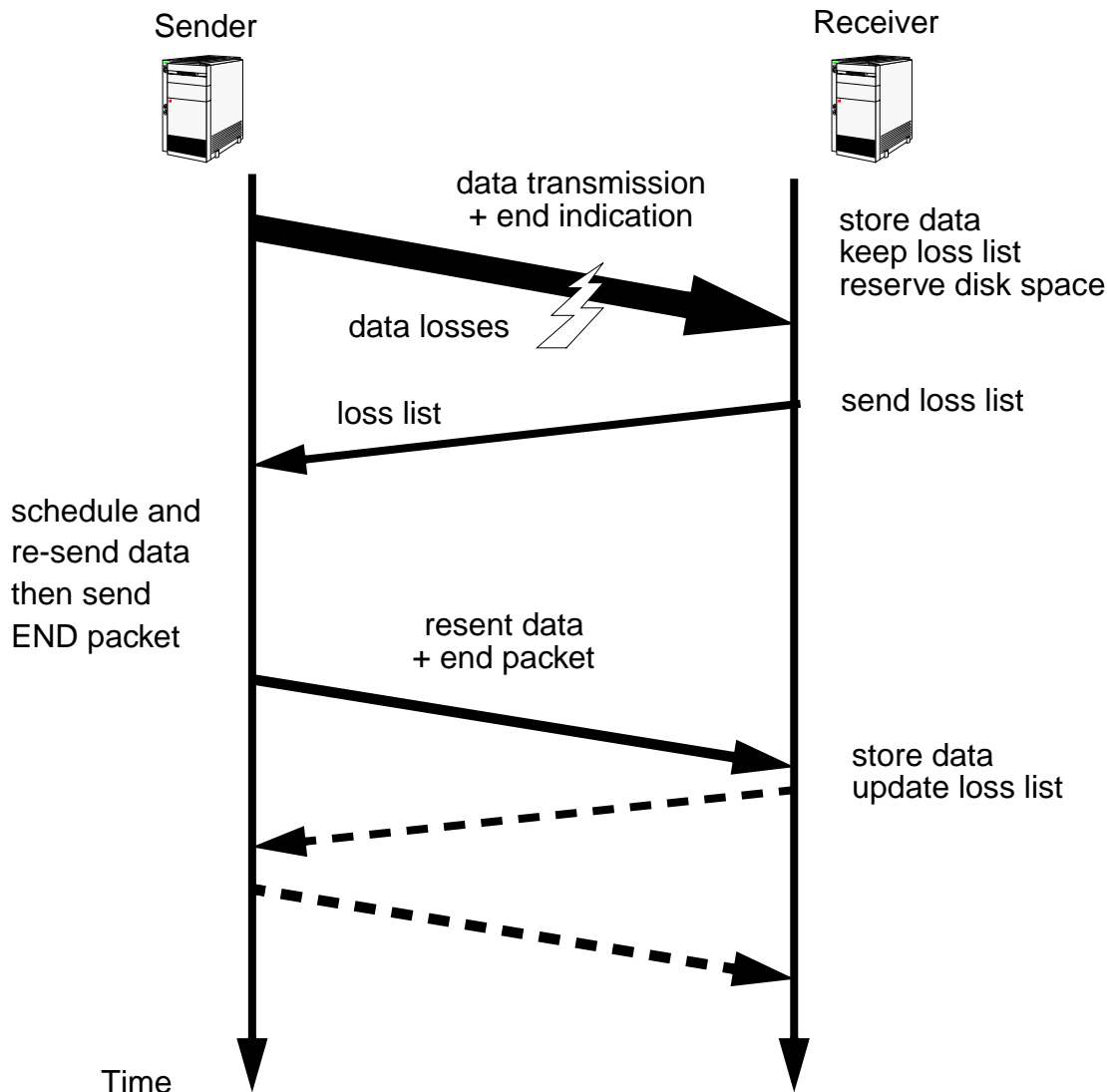
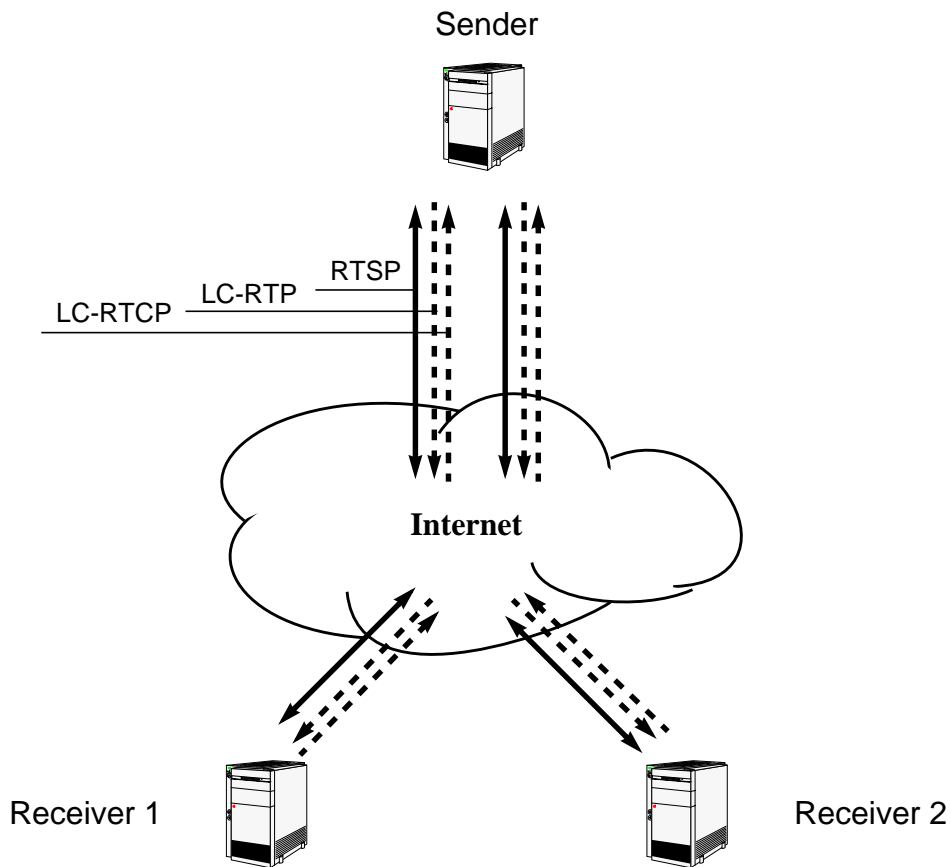


Fig. 3: Reliable data transmission with LC-RTP [Jon99]

## 2.3 Caching approaches for media streaming

### 2.3.1 General

Caching is widely applied in computer technology to increase performance. A random access cache memory used in computers is a high-speed memory with much shorter access time than the main memory. An application of a cache in computers can reduce the



**Fig. 4: Streaming with RTSP, LC-RTP and LC-RTCP**

average memory access time of the computer. Data that the processor accesses frequently is saved into the cache and accessed later from the cache directly and not from the main memory.

In distributed systems caches are used to save bandwidth and reduce response times. A good example is the world wide web. Before the introduction of the web, the usage of the internet was limited to the personal of universities and research institutions typically using it for e-mail and file transfer. After the web's introduction the number of users grew dramatically and the volume of data transferred by each user increased due to graphics and other multimedia contents. In a very short time the requirements to the transport system grew fast and the users experienced long response times and bad performance. The performance of the web was improved by increasing the line capacity of the network and introducing caches. Three types of caches are applied in the world wide web [Küf98]:

- **Hard disk cache at the client side:** Each client manages its own cache and uses it for its own requests and does not serve requests from other clients. Data frequently accessed by the client is retrieved directly from local disk and not over the network.

- **Memory cache at the server:** The server serves frequently requested pages directly from memory without accessing the data from hard disk but data must nevertheless be transferred from the server to the client.
- **Cache-Server:** A Cache-Server forwards request-response pairs between clients and origin servers and stores pairs into local storage as they pass by. Responses that are requested and exist in the cache are directly retrieved from the cache without forwarding the request to the origin server. Nevertheless might the cache server need to consult with the server concerning consistency before responding. All customers of a Cache-Server profit from the caching performed by the server and Cache-Servers are normally able to store much more data than a hard disc client-cache. Cache-Servers save network resources.

### 2.3.2 Video-on-Demand

The term ‘Video-on-Demand’ is often used with respect to the delivery of a movie to a consumer. It is used with varying meanings according to the degree of interactivity that is provided to the user. Some arts of video-on-demand are stated in the following [GBW97]:

Broadcast, or No-Video-on-Demand is used for television as we know it today because it does not provide the user with any interactivity. Pay-per-view service also belongs to this group because the user is not able to have influence on the program of the sending station.

Quasi-Video-on-Demand provides the user to subscribe to various channels where each channel covers particular interest field. The interactivity of the consumer is restricted to changing his subscription to channels.

Near-Video-on-Demand identifies the procedure when the same movie is transmitted over multiple channels with temporal offset. The user can not control the delivery of the content but can choose the point in time when he starts to watch at a granularity of minutes. The user can jump back and forth in the stream or make a short coffee break by changing between the channels but the granularity is limited to the offset between the channels.

True-Video-on-Demand provides the user to control delivery of the content in a way one does with video delivery from a VCR. The user can choose the starting time of the movie, can halt the delivery temporally and jump back and forth in the stream as he likes.

Video-on-demand applications and media streaming applications in general require much more resources from servers, clients and transmission networks than contemporary applications due to the volume of the data they handle. At the server side the applications set higher requirements on space in file system for storing and on processing power for feeding data into the network. They set higher requirements on bandwidth in links and bufferspace in nodes in the transport network and at the client side they require more processing power and memory for buffering, decoding and playing.

## Video-on-demand arrangements

A provision of a video-on-demand service can be realized with one central videosever within the environment of a company or an institution where there are a few users being served at a time.

In order to support a large scale video-on-demand service with a large number of concurrent streams the video-on-demand system must be arranged as a distributed system [GBW97]. A distributed video-on-demand system can be arranged as a hierarchy of video servers illustrated in Fig. 5. The root server of the hierarchy, called the *origin server* stores all content available to a consumer and each intermediate server stores a subset of the content and acts as a *proxy cache* server for the next lower layer. An end proxy cache receives all requests from *clients* within a particular area of the network. If a proxy cache receives a request for a content which is not contained within it, the request is forwarded to the next layer. A request for a content that is only contained within the origin server traverses the hierarchy from a leaf through the intermediate caches to the root. The streaming of the content along the reverse path follows. The performance of this arrangements depends on the size and location of the video servers but also on the location of the content in the hierarchy. A reasonable arrangement of the hierarchy regarding these issues saves costs, increases reliability and improves performance [GZL+00].

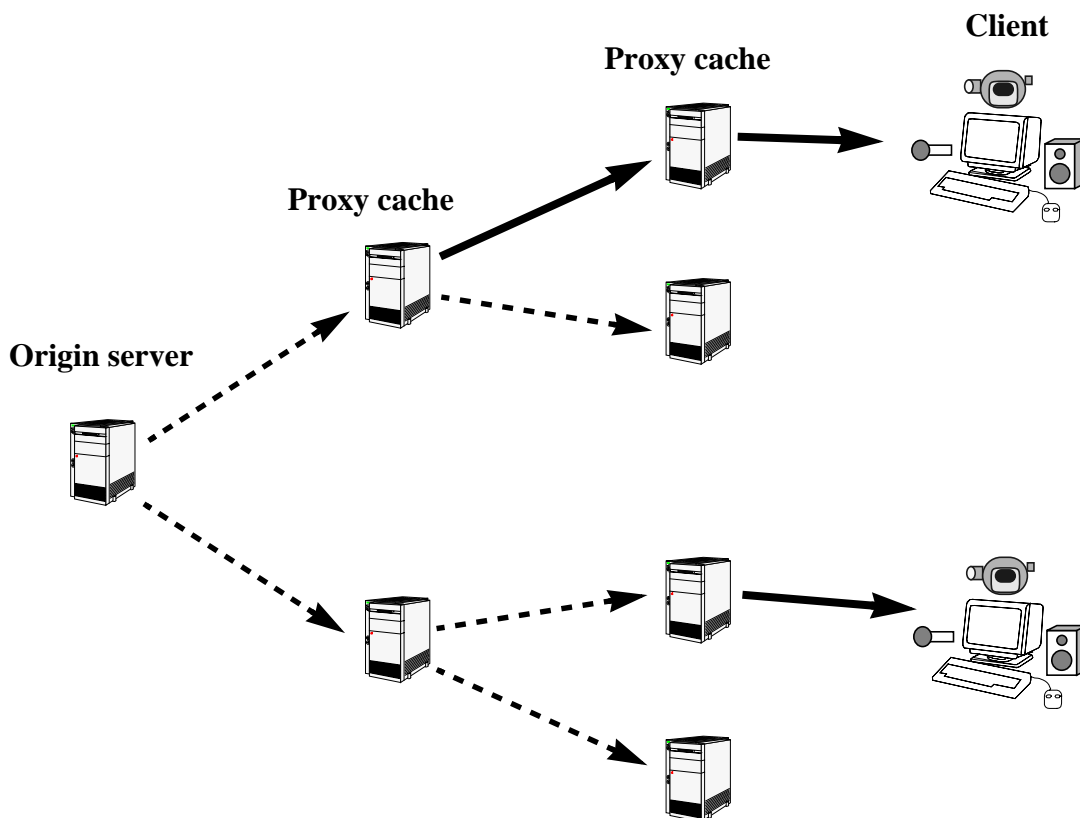


Fig. 5: A caching hierarchy

The popularity of a particular content provided by a video-on-demand system changes with time [GBW97] and in order to optimize the performance of the hierarchy the content of the caches must be updated from time to time.

A **caching strategy** is the chosen approach of a system for the decision whether an object should be kept in a certain cache or whether it should be removed [Gri00]. The decision of a caching strategy depends on various parameters including the object's size and hit rate and the cache's size and utilization. Various caching strategies that can be used for the management of streaming caches are described in [Küf98].

Two general approaches can be followed to transfer content between an origin server and a cache:

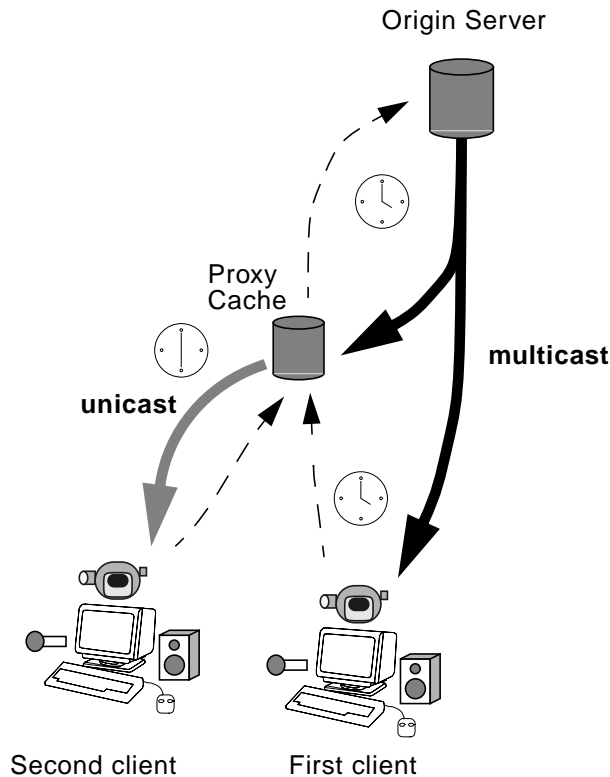
**File transfer:** A content file is transferred in a complete form from an origin server to a cache with a reliable file transfer protocol (e.g. ftp). The cache has the complete information stored on the file systems and must be able to parse the various file formats provided by the origin server and packetize the data into a stream [FGK+00].

**Cut through approach:** A cache records RTP packets to local storage as they pass by on the way from an origin server to the client. A loss collection process takes care of retransmission of lost packets and completes the caching process after the normal transmission. This procedure might be performed with the LC-RTP [Gri00] described in Section 2.2.3. The disadvantage of this approach is that information is lost due to packetization at the server and a cache might not be able to stream the packets. This approach is illustrated in Fig. 6.

### 2.3.3 Caching support in RTSP

A caching mechanism for RTSP based session is defined in [RFC2326]. It allows the caching of a session description that is sent within a response to a DESCRIBE request and included within an ANNOUNCE message. It also allows caching of continuous media data that is typically delivered by RTP.

On receiving a SETUP or PLAY request, the cache checks whether it has a copy of the requested continuous media content or its description. If a copy is found a DESCRIBE request is sent to the origin server and the Last-Modified Header included in the response is compared to the date of the local copy. If the copy is not up-to-date the transport parameters received within the SETUP message are modified appropriately and the message is forwarded to the origin server. The media data is streamed from the origin server to the client and the cache possibly records the stream into its local storage. During the streaming the cache forwards control commands such as PLAY and PAUSE directly to the origin server without modifications.



**Fig. 6: Cut through caching**

An RTSP-conformant cache is of the "cut-through" variety. It can not retrieve a whole resource from an origin server before serving the client but can only record a stream as it passes by. This prevents the introduction of additional latency.

An RTSP cache is required to behave as a regular media server to the client and as a regular client to the media server.

An RTSP cache must store the presentation/session description of its entries but typically eliminates all transport parameters from the description because they are not valuable for the media streaming from the cache to the client. An RTSP cache that is able to translate the content between encodings creates a session description for each encoding it can offer.

RTSP defines cache-directives that are contained within SETUP messages. The cache-directives specify the caching properties of the media stream referred to within the message. Following are descriptions of the cache-directives defined within RTSP:

**no-cache:** Indicates that caching the media stream is not allowed.

**public:** Indicates that the media stream is cacheable by any cache.

**private:** Indicates that the media stream is intended for a single user and may only be cached by a private cache.

**no-transform:** A cache might find it useful to perform a conversion of the data from the original format to another format to save storage space. This directive indicates that a data conversion is not allowed. This might be practical in applications like medical imaging, scientific data analysis and those using end-to-end authentication where a transformation would destroy the data.

**only-if-cached:** Indicates that a media stream should be delivered only if it already exists in the cache and not be delivered from an origin server.

**max-stale:** Indicates that the client is willing to accept a media stream that has exceeded its expiration time.

**min-fresh:** Indicates that the client wants a response that will still be fresh for at least specified number of seconds.

**must-revalidate:** Indicates that a cache must do an end-to-end revalidation of the entry every time it is requested.

An RTSP server can insert an **Expires header field** into a DESCRIBE response. The field indicates the date and time after which the description should be considered stale. A cache may not return a stale cache entry unless it is first validated with the origin server. The presence of an Expires field only specifies the timerange when the cache may feel free to return the entry without consulting the origin server. The field does not imply that the original resource will change or be destroyed before or after the specified point in time.

#### 2.3.4 Proposal for caching support in standards-based RTSP/RTP servers

In this section, the caching support in standards-based RTSP/RTP server that was proposed in [FGK+00] is shortly described.

The following issues concerning a media streaming cache are identified in [FGK+00]:

- **Transfer Loss:** In media streaming applications, media data is normally carried over UDP which does not provide lossless transport service so an additional method is required that provides lossless copying of media data from an origin server into a cache.
- **Transformation Loss:** An origin server transforms media data locally stored at the server into RTP packets, called packetization. This transformation typically causes some information loss and the RTP packets may not carry all of the information needed for originating the media stream. An additional method is required that provides the cache with the additional information required for originating the media stream.

- **Cache Coherency:** A cache must be able to check whether its replicated content is up-to-date
- **Access Accounting:** An origin server must be provided with access information to the media streams that are replicated in the caches.
- **Authorization:** An origin server must be able to control the access to the media content replicated at caches through authorization methods. It must be able to authorize caches to serve replicated content. This might happen on a per-viewing basis or on a periodic basis.
- **Copy Protection:** Replication of media content must be limited to caches that follow the origin server's rules concerning access accounting and authorization and actions must be taken that prevent the making unauthorized copies.

In order to solve the transfer loss and transmission loss issues the authors propose a so called **packet transfer** approach:

The original media file is not transferred from the origin server to the proxy. Instead the RTP packets generated from the original media are transferred over a loss-less protocol. In addition to the RTP packets some "meta" information present in the media file but not present in the RTP packets is also transferred via a separate meta channel. This solves the transformation loss issues and proxies have complete information about the media with this approach. Two types of messages are proposed for "meta" information:

**Sample/Frame Info:** This type of message describes the sample/frame contained in one RTP packet. Its "transmission time offset" field holds the transmission time of a RTP packet relative to its RTP timestamp (RTP timestamp holds the sampling instant). It can be used by caching servers to match the origin server's transmission schedule of the packets without having to analyze the content. It is primarily practical to make streaming of media content whose sampling and transmission order do not match (e.g. MPEG-1 and MPEG-2) more efficient.

**Stream Info:** This type of message describes the attributes of the stream. It is sent at the beginning of the stream prior to sending any RTP packets or other metachannel packets. It describes the stream's preference to transport service, real-time delivery or reliable delivery.

The authors propose to send RTP and RTCP packets in-band within the reliable RTSP stream and setup a separate reliable RTSP port for the meta channel.

The cache coherency issue is solved with the same approach as described in the RTSP caching support (see Section 2.3.3).

The advantage of the proposal is that due to the meta channel, a cache is able to stream a media content without having to analyze the content and know all possible formats. A disadvantage is that no methods for solving the access accounting, authorization and copy protection issues are described.

The authors propose that RTP, RTCP and RTSP packets should be carried over a loss-less connection. TCP does not suite well for real-time transport (see Section 2.1) but LC-RTP described in Section 2.2.3 is more suitable.

## 3. KOM Implementation of RTSP/RTP

In the framework of the project "Medianode", running at the KOM Institute, an RTSP/RTP based server and a client were implemented. The implementation gives the researcher at the Institute the chance to experience with modifications and extensions of the protocols. In this section some issues of the design of the server are shortly presented that are important for the considerations in the subsequent chapter.

The RTSP/RTP based server was developed with object oriented methods and programmed in C++. The implementation is multithreaded and the purpose of the separation into threads was to reach a better performance. It uses TCP for reliable delivery of RTSP packets and can support non-persistent as well as persistent connections as described in Section 2.2.2 and [RFC2326]. It uses UDP for carrying RTP packets and supports unicast and multicast streaming. Streaming of MPEG-1 video is supported and support for streaming of the MP3 and H.261 coding formats is under development.

### 3.1 Class Diagram

An overview of the implementation is illustrated in the class diagram on Fig. 8. Although the overview is coarse and shows only a part of the whole implementation it gives a good overview of the classes that are involved in the administration and streaming of one RTSP session. The implementation can be separated into the following parts according to functionality: RTSP Session Administration, RTSP Communication and Parsing and RTP Streaming. In the following each part along with corresponding classes is described shortly.

#### **RTSP Session Administration**

For each active RTSP session there exists an RTSPServerSession object which is the heart of the administration of the session. The object inherits from the KOMServer class (which is not instantiated) which implements a state machine for the server session similar to the one described in [RFC2326]. The state machine is affected by the received RTSP messages. The state transitions of the machine affect the RTP streaming objects and sends RTSP messages back to the client. The RTSPServerSession runs in a thread.

The RTSPServerSession object has an association to an RTSPFillBuffer object (which is a TCP socket) but the association is temporary if the client uses non-persistent RTSP connections.

## RTSP Communication and Parsing

The RTSPFillBuffer object receives data from a TCP socket and parses it into a corresponding message object. After the parsing an association is built with a RTSPServerSession object and the parsed object is delivered to the session which acts appropriately according to the content of the parsed object.

The reception and parsing process is initiated from a Selector object that monitors the RTSP communications and runs in a thread.

## RTP Streaming

This part performs the streaming from a file in the file system to the network. It is separated into two threads, RTP and RTCP which run concurrent to the RTSP part. It is managed by the RTSP part that initiates the streaming, can change the present streaming position within the stream, can halt the streaming temporarily and destroy it. The following functions are performed by each thread:

**RTP:** This part performs the delivery of the media content to the network. It includes: accessing and reading from a media file in the file system, preparing RTP packets (packetization) and scheduling and emitting the RTP packets to the network.

**RTCP:** This part prepares and emits sender reports to a separate UDP port and receives receiver reports. No further processing of the receiver reports is performed.

## 3.2 Instance Drawing

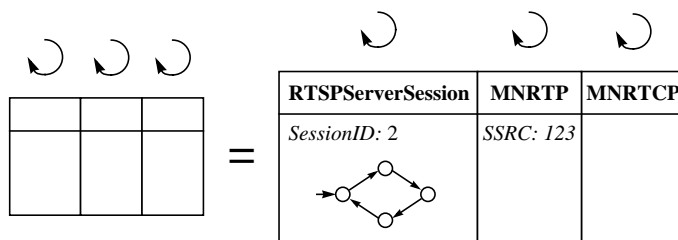
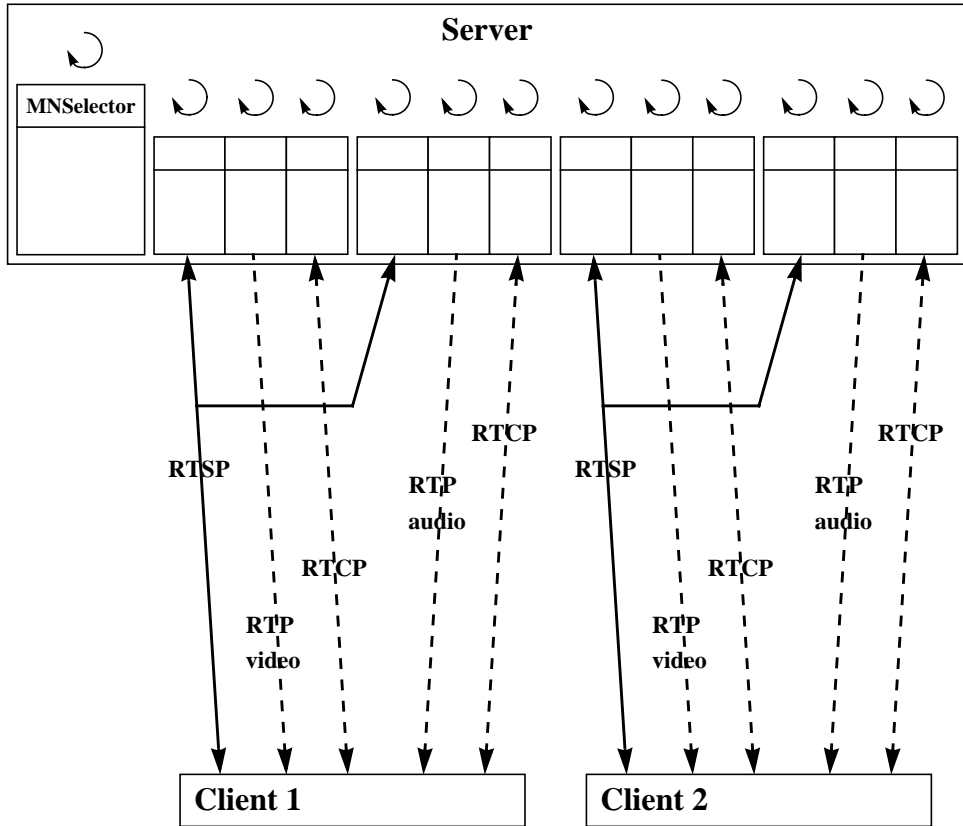
On Fig. 7 an instance of the RTSP/RTP based server together with corresponding objects is illustrated in a streaming session with two client instances. The server streams audio and video content to the clients via unicast transport.

In this scenario, the audio and video streaming for each client is separated into two RTSP sessions. The concurrency arrangement is shown with shaped arrows and shows that for each RTSP session there are three concurrent threads running. An additional selector thread monitors all RTSP ports, receives an RTSP message from a TCP socket and delivers it to the corresponding RTSP session instance, one at a time. Each line on the figure identifies a port-to-port communication between client and server. The solid lines identify TCP connections and the dotted ones identify UDP communication. Client and server typically communicate through one persistent TCP connection but the implementation also supports non-persistent RTSP connections.

Each RTSP Session is identified with a Session Identifier (Session ID) that is assigned by the server. The server uses the identifier to demultiplex the RTSP messages.

Each participant in an RTSP Session allocates an SSRC (see Section 2.2.1.3) to its RTP/RTCP instances.

The clients on the figure might be receiving the same stream synchronously, (i.e. a synchronous pay-per-view stream) but the present implementation would nevertheless instantiate two complete RTSP Session and RTP Streaming instances for this case.

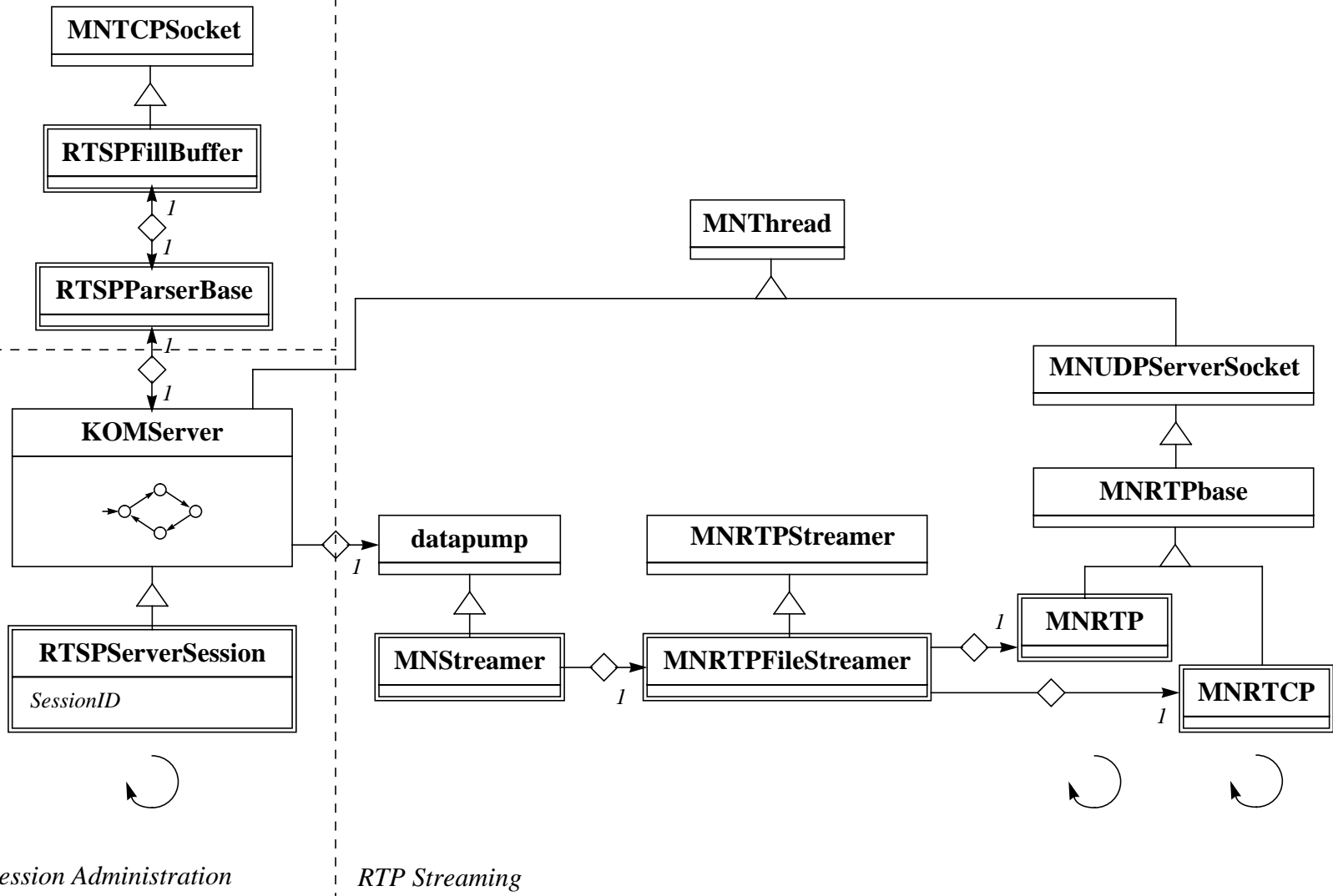


———— TCP Communication  
 - - - - - UDP Communication

**Fig. 7: A Use Case for the KOM RTSP/RTP server implementation**

*RTSP Communication and Parsing*

Fig. 8: A class diagram for the KOM RTSP/RTP server implementation



## 4. A Design of an RTSP Capable Proxy

Multimedia streaming makes much greater demands on the transmission network than traditional datatransmission because of greater volume of the data and real-time properties. Multimedia streaming requires a lot of memory space in hosts and nodes and high bandwidth in links. Efficient and fair scheduling is needed in the nodes in order to limit the variations in end-to-end delay in the transmission.

Large-scale video-on-demand can only be provided with the technology of distributed systems. Content can be stored in a number of caches that are located at various places in a network. A streaming request from a client is served from a cache instead of from a centrally located server. This saves network resources and provides the user with better quality and lower costs.

It is reasonable to build a proxy design that supports the technology that is already widely applied in media servers and clients. Otherwise it might not be applied because many end users already use a media streaming client and are not willing to migrate to a different technology. Clients and media servers that use RTP/RTCP and RTSP protocols are widely applied in the Internet. A good example is the streaming suite from RealNetworks.

Following are two possible arrangements for a proxy design:

- An origin server participates in each streaming session, independent of if the streaming is performed from the origin server or from a proxy. The origin server performs access accounting, authorization and controls the streaming. The proxy does not store the media in a complete form to its cache but only in a form to be able to stream under the administration of the origin server.
- A proxy is able to stream a media content from its cache to a client without any interventions of an origin server. A proxy must perform access accounting and authorization and have the media content stored in a complete form in its cache.

The second architecture is more fault tolerant in some ways because the proxy is able to provide media streaming although the origin server or the communication network between the proxy and the origin server is temporarily not in operation.

In the second architecture the proxy must store the complete media content and user authentication information to be able to stream without any interventions of the origin server. A proxy must account for access in a correct way and keep the authentication information secret. A proxy may not give an untrustworthy proxies a copy of the content. An origin server does not have a control over the proxy and relies on its trustiness.

It is likely that an origin server and a proxy are not operated by the same organization. A television company or a content production company might operate an origin server and a telecom operator or an internet service provider might operate proxies. If this is the case

the first architecture is more attractive from the origin server's point of view for security reasons. The origin server might like to take care of authorization and access accounting itself and keep authentication information local and not distribute it to other systems.

A design of an RTSP capable proxy is specified in the following sections. The design is based partly on the existing implementation (described in Section 3.) and specifies extensions that are necessary for the proxy support. In the beginning some design goals are presented and then changes to RTSP Administration together with a new stream handler architecture are described. The section closes with a presentation of the necessary changes to the protocol and two example scenarios are illustrated in Appendix B and Appendix C.

## 4.1 Design goals

The goals of the design of the proxy are stated in the following:

***Simplicity:*** The design should be as simple as possible because otherwise people are not willing to implement it and experiment with it.

***Real-Time Streaming:*** A proxy must be able to provide real-time streaming of data from local storage to the network with unicast or multicast transport. It must implement an RTP packetizer and a RTP packet streaming scheduler.

***Real-Time Recording:*** A proxy must be able to receive RTP packetstream through unicast or multicast transport and record it into a local storage into a complete state. The proxy must implement the LC-RTP protocol.

***Real-Time Reflection:*** A proxy must be able to receive RTP packetstream and stream it out to a number of ports. The real-time properties of the stream must be preserved. The proxy must be able to reflect between the following combination of transport methods: unicast to unicast, unicast to multicast, multicast to unicast and multicast to multicast. Reflection is a practical feature for the operation in a network where multicasting can not be provided in the complete topology or not at all. It is practical for a session with a number of receivers, a livestreaming session or a pay-per-view broadcasting.

***Authorization:*** A proxy receives streaming requests from clients and/or other proxies and possibly delivers the requestor with a real-time streaming from a local storage or by reflecting a stream received from another proxy or an origin server. A proxy is not allowed to start streaming, reflecting or recording before consulting with the origin server and getting a corresponding permission. A proxy is not allowed to grant another proxy a permission for streaming, reflecting or recording.

**Interface to a client:** A proxy implements the same interface to a client as a normal origin server does. A client does not recognize any difference in the RTSP, RTP or RTCP semantic between communicating with an origin server or a proxy. A client that communicates with a proxy directs all its requests to the proxy instead of directing them to the corresponding origin servers when the proxy is not applied.

**Existing implementation:** The proxy design should apply as much of the existing implementation as possible.

**Connections:** A proxy should be able to handle persistent and non-persistent connections for RTSP communication.

**Connections to origin servers:** A proxy can communicate with a number of origin servers. A connection to a particular origin server is installed on-demand upon receipt of a request for streaming of content that originated from the server.

**Caching hierarchy operation:** A proxy should be capable of operating in a caching hierarchy. It forwards all its RTSP requests to a fixed proxy that serves in the next layer in the hierarchy. The RTSP communication between the proxy and the origin server traverses the hierarchy.

**Caching of session descriptions:** A proxy should not respond to a DESCRIBE request with a cached session description but forward the request to the origin server and then the corresponding response back to the client. The complete RTSP communication dialogue runs between the client and the origin server with the help of proxies and it is essential that the DESCRIBE dialogue also do so. This is conformant to the multiformat support presented in Chapter 5 and to the work on patching support that is currently under development at the KOM institute.

## 4.2 Specification

A design of an RTSP based proxy is specified in this section. The section starts with an architecture overview. Two example sessions are then described and illustrated with figures to give a better overview of the design. Each part of the design is then described in more detail in subsequent sections and the section closes with specifications of necessary changes to the RTSP protocol. Example protocol scenarios are given in Appendix B and Appendix C.

### 4.2.1 Architecture Overview

In this section main properties of the RTSP based proxy design are stated. The architecture assumes that an origin server participates in each streaming session, independent of if streaming is performed from the origin server or from a proxy.

**Scenarios:** The following three main scenarios have been identified:

- Streaming from an origin server to a client
- Streaming from a proxy to a client
- Recording into a cache in the proxy

For the first scenario it is necessary to differentiate between two cases: Streaming directly from an origin server to a client or streaming through a proxy which reflects the media stream from an input port to a number of output ports. The reflection lets a proxy record a packet stream as it passes by and provide an application level point-to-multipoint distribution which is practical for livestreaming and pay-per-view applications. The reflection introduces additional end-to-end delay. The reflection might be carried out between two separated multicast sessions, from a unicast communication to a multicast session, from a multicast session to a unicast communication or from one unicast communication to another.

**RTSP Communication:** An origin server participates in every RTSP session which issues streaming of content that originated from the server. RTSP messages from different RTSP sessions are multiplexed to one reliable connection (TCP) between an origin server and a proxy. RTSP SessionIDs provide the demultiplexing capability and a single CSeq sequence is maintained for the connection. A proxy installs an RTSP connection to an origin server on-demand when a request for the particular origin server is received from a client. The connection is torn down when no more active RTSP sessions between the proxy and the origin server exist.

**RTCP Communication:** RTCP messages from different RTSP sessions are multiplexed to one reliable connection (TCP) between an origin server and a proxy. The SSRCs provide the demultiplexing capability. A proxy installs an RTCP connection to an origin server on-demand when an RTSP session is set up. The connection is torn down when no more active RTSP sessions between the proxy and the origin server exist. If a proxy communicates with a client with RTCP messages, it communicates with each session through a separate port (UDP). The following paragraphs describe the functionality of a proxy in the aspect of RTCP communication in four different scenarios:

**Direct Streaming:** An origin server provides a client with media streaming without the intervention of the proxy in the delivery of RTP packets. The proxy does not provide any forwarding of RTCP messages between the origin server and the client and vice versa. The partners communicate directly through unicast or multicast.

**Reflection:** A proxy reflecting a data stream forwards sender reports from the origin server to the client and receiver reports in the opposite direction.

***Streaming from a cache:*** A proxy that provides media streaming from a local cache, allocates a SSRC to the stream, informs the origin server on the allocation through RTSP and sends sender reports to the client and the origin server. The proxy receives receiver reports from the client stores them for acquisition purposes and forwards them to the origin server. The origin server does not send any RTCP messages in this scenario.

***Recording into a cache in the proxy:*** A proxy that records data stream into its local cache acts like any other media stream consumer in the aspect of RTCP communication. The proxy sends receiver reports to the origin server and allocates a SSRC in order to identify the reports.

**SessionID:** Origin servers allocate the SessionIDs for the RTSP communication between origin servers and proxy. A proxy allocates the SessionIDs for the RTSP communication between the proxy and the client. In most cases the SessionIDs remain the same for the two links but a collision is possible if a client communicates with two servers at the same time or if the proxy is operated in a caching hierarchy. The proxy must monitor the allocation and allocate a different SessionID in order to avoid identifier collisions.

**SSRC:** Each participant in a streaming session that either provides another participant with media streaming or consumes media content (active server or a client but not a passive origin server) allocates a SSRC in order to identify its RTP and RTCP packets. A SSRC collision is unlikely but possible. In order to resolve the collision within one session, an algorithm is proposed in [RFC1889]. A proxy that performs reflection of media streaming multiplexes RTCP messages from different sessions to one connection. The proxy must ensure the uniqueness of the SSRCs that are multiplexed to the connection. The proxy can either perform a SSRC conversion for the RTCP packets that it forwards or require the originator of the message to allocate a new SSRC. The second approach requires uniqueness among the SSRC of all active sessions.

**Administration:** The administration of the streaming in one RTSP session is separated into two parts: RTSP and RTP Administration. The RTSP Administration maintains the state of each RTSP session, receives and sends RTSP messages and acts on the streaming process. The RTP Administration manages a datapath that performs streaming, receipt and/or reflection of media content. It also manages reception and forwarding of RTCP packets. The two parts interact in the way that the RTSP part influences the streaming process and the RTP part notifies the RTSP part when particular events are recognized from the received stream. The RTSP Administration is identified with RTSP in Fig. 11. For a description of RTSP Administration see Section 4.2.4. The RTP Administration is identified with Streaming Graph in Fig. 11 and RTPStreamingGraph in Section 4.2.5. For a description of RTP Administration see Section 4.2.5.

**Cache Directory:** The cache directory maintains information on the content kept in the cache. Each entry in the directory contains information from the SDP session description and is indexed according to the session name. Each entry also contains the URI in order to give the proxy the chance to contact the origin server when the corresponding content is requested. For a description of the cache directory see Section 4.2.6.

**Caching Strategy:** This module decides if a media stream should be recorded into a local storage or not. Various issues can influence such a decision like popularity that depends on the age of the content [GBW97] and the price that the content provider demands. This module also decides if a content should be erased from a cache.

## 4.2.2 A session with multicast distribution and packet recording

Fig. 9 illustrates a streaming session with the participation of an origin server, a proxy and one client. The origin server streams media content requested from a client to a multicast address. The media content is streamed from the origin server because it did not exist in the cache of the proxy. The proxy records the packets of the session into the cache in the hope that another client might request the media in the near future.

The client directs all of its RTSP messages to the proxy and the proxy forwards RTSP requests and responses between the client and the server. The origin server sends its sender reports directly to a multicast address. The client also sends its receiver reports directly to a multicast address.

The proxy is a participant in the session due to the packet recording process. It communicates in a RTSP session with the origin server. It receives RTP and RTCP packets directly from the multicast session and sends its receiver reports to the session.

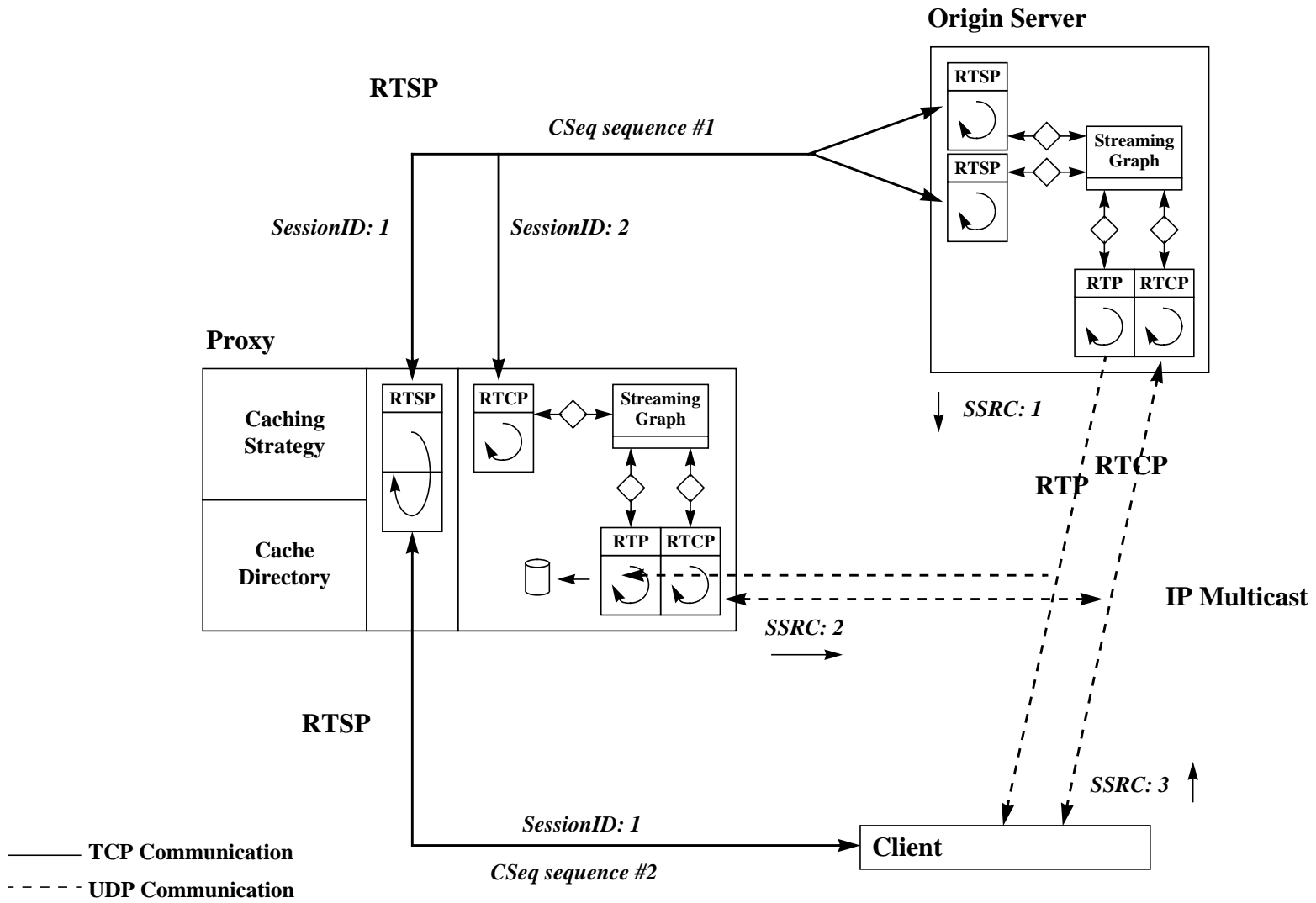
The client and the proxy communicate with the origin server in separate RTSP sessions with unequal SessionIDs. The proxy multiplexes the corresponding RTSP messages to one connection and uses SessionIDs to identify the messages.

The RTCP messages sent to the multicast address are limited to RTCP communication within the session. The algorithm described in [RFC1889] resolves the SSRC collision problem.

The proxy manages two RTSP administration instances (RTSP objects), one for client's participation in the session and one for its own. The object corresponding to the client does not have an association to any streaming instances because the proxy does not provide the client with streaming but only with controlling. The streaming graph managed by the proxy manages reception and storing of media content.

The origin server manages two RTSP administration instances (RTSP objects), one presenting the client's participation in the session and one presenting the proxy's. These objects both have associations with the streaming graph because the origin server

Fig. 9: Multicast streaming and packet recording into a cache



provides both participants with media streaming from the same datapath. The origin server receives receiver reports from the client and the proxy and can monitor how well the streaming process performs.

A difference between the RTSP administration object in the origin server and the proxy is noticeable. This difference is described in Section 4.2.4.

### **4.2.3 A session with streaming from cache and reflection**

Fig. 10 illustrates two independent streaming sessions. The client is participating in both sessions with two independent applications. The description of each session follows:

#### ***Session 1:***

In this session, the proxy provides the client process identified "Application 1" with media streaming from its local cache.

The client directs its RTSP and RTCP messages (receiver reports) to the proxy. The proxy forwards RTSP and RTCP messages between client and origin server in both directions. The proxy sends sender reports to the client and the origin server.

The proxy maintains an RTSP administration instances for the session and a RTP administration instance that performs streaming.

The origin server maintains an instance for the RTSP administration that might implement an access accounting procedure for the media streaming. The origin server also maintains an RTP administration instance that receives sender and receiver reports and monitors the performance of the streaming process.

#### ***Session 2:***

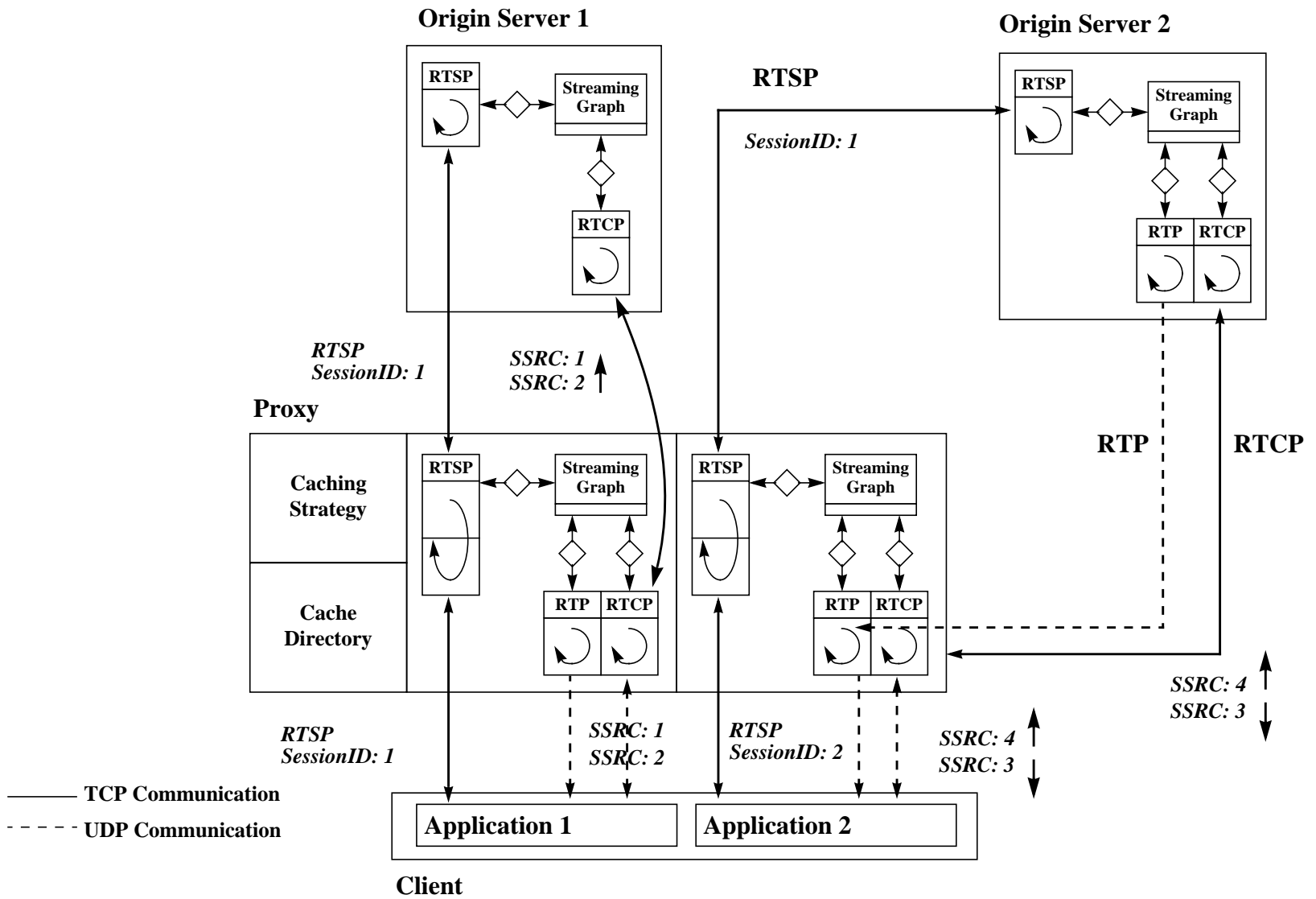
In this session, the proxy reflects a media stream from the origin server identified "Origin Server 2" to the client process identified "Application 2".

The client directs its RTSP and RTCP messages (receiver reports) to the proxy. The proxy forwards RTSP and RTCP messages between client and origin server in both directions. The origin server sends sender reports to the client and the origin server.

The proxy maintains an RTSP administration instances for the session and a RTP administration instance that performs the reflection.

The origin server maintains an instance for the RTSP administration that might implement an account accessing procedure for the media streaming. The origin server also maintains an RTP administration instance that performs streaming, receives receiver reports and monitors the performance of the streaming process.

Fig. 10: Streaming from cache and reflecting



The connections between the proxy and origin servers are installed on-demand when a request for the particular origin server is received from a client.

A difference between the RTSP administration object in the origin server and the proxy is noticeable. This difference is described in Section 4.2.4.

The proxy resolves collision in the SessionIDs allocated by the origin servers. It allocates the SessionID 2 to the client process identified with "Application 2".

#### **4.2.4 RTSP Administration**

The purpose of the RTSP Administration is to maintain a state of an RTSP Session, receive and emit RTSP messages and influence the process that performs the media streaming. Proxies implement RTSP Administration functionality for each active RTSP Session in an RTSPProxySession class shown in Fig. 11. Origin servers implement the functionality in the RTSPServerSession class which must be extended from the one discussed in Section 3. to support cooperation with a proxy as described below.

##### ***Origin Server***

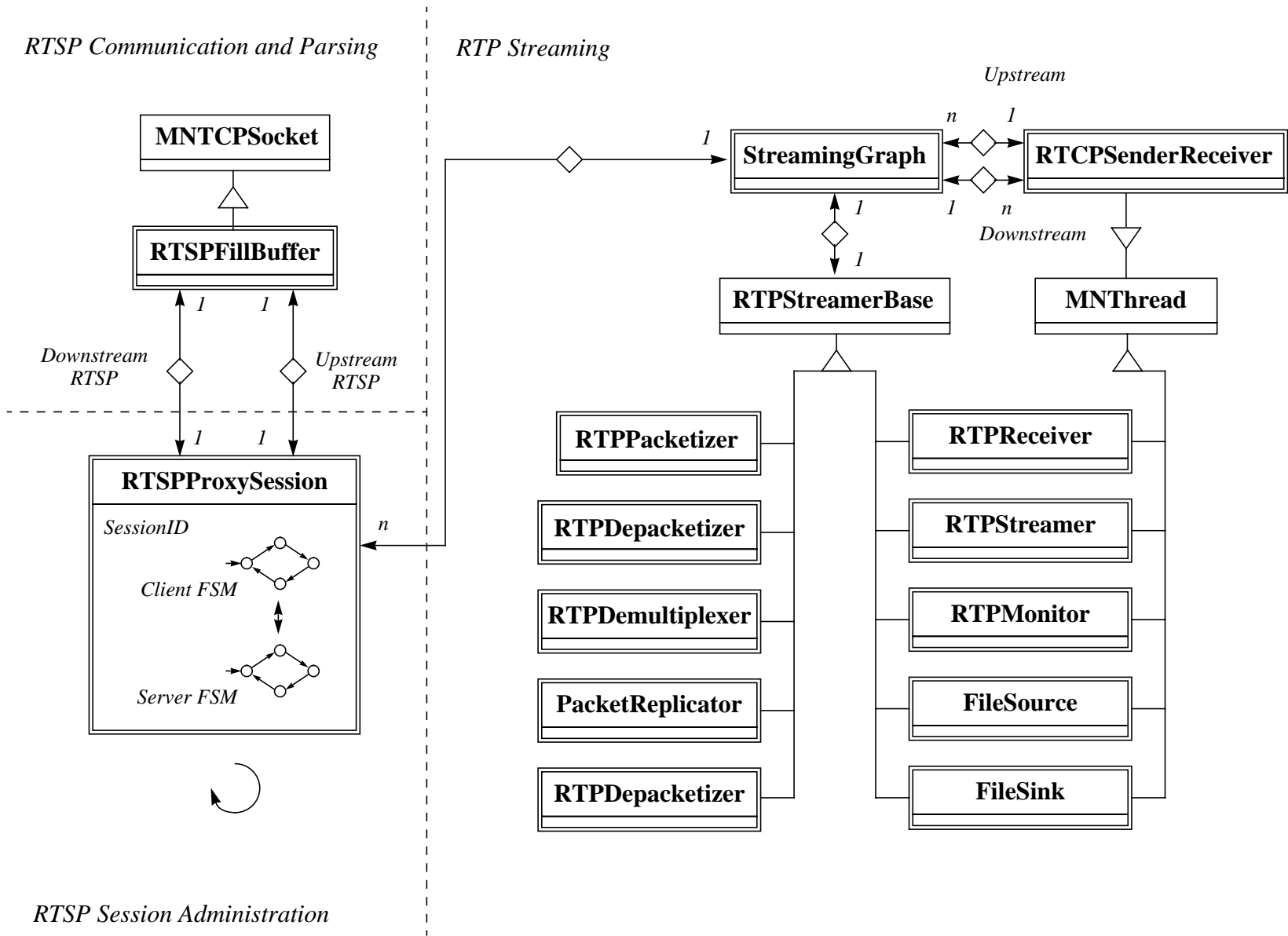
An origin server maintains an instance of the RTSPServerSession for each active RTSP session the server administrates. This applies to the three following cases identified in Section 4.2.1: streaming from an origin server to a client, streaming from a proxy to a client and recording into a cache in the proxy. These cases are illustrated in Fig. 9 and Fig. 10. It is important to recognize that in Fig. 9 the origin server maintains two instances, one corresponding to the streaming to a client and another corresponding to the recording performed by the proxy. An RTSPServerSession object has associations to at least one RTP Administration instance (Streaming Graph) which either performs streaming or monitors streaming quality through receiver and sender reports.

An RTSPServerSession object is associated with one Streaming graph if the RTSP session serves streaming of one medium and with more graphs if the session performs an aggregate control of a number of media or streams a container file.

An origin server monitors the streaming of content that originated from it at all times and keeps the state of each participation within the RTSPServerSession. Hence, the object is well suited to provide an access accounting process with necessary information to charge a client for watching content or to charge a proxy for recording.

The implementation of the RTSP Administration in the RTSP Server (discussed in Section 3.) must be extended to support the cooperation with the proxy. The following necessary changes have been identified:

Fig. 11: A class diagram for the RTSP/RTP proxy design



1. The state machine must be able to differentiate between the cases when the origin server performs media streaming and when a proxy performs streaming. In the first case the RTSP Administration is responsible for controlling the streaming, interacts closely with the Streaming Graph and monitors the quality of the data delivery. In the second case the origin server monitors the quality of the data delivery and the interaction with the Streaming Graph is not as close.
2. The state machine must interact with a Streaming Graph that has a different interface than the hard wired RTP implementation used in the present RTSP server. This might require an introduction of additional states into the state machine.
3. The parser and the state machine must take changes in the protocol introduced in Section 4.2.7 into account.

The close interaction with the Streaming graph probably requires some new waiting states in the state machine between the normal states because the RTSP and Streaming graph run concurrently.

It is possible to implement the functionality with two separate state machines, each implementing one of the cases identified in the first point or to cope with the both cases in one state machine and distinguish between the cases with a variable.

### ***Proxy***

A proxy server maintains an instance of the `RTSPProxySession` for each active RTSP session the proxy administrates. This applies to the three following cases identified in Section 4.2.1: streaming from an origin server to a client (with or without reflection), streaming from a proxy to a client and recording into a cache in the proxy. These cases are illustrated in Fig. 9 and Fig. 10. It is important to recognize that in Fig. 9 the origin proxy maintains two separate instances, one corresponding to providing the client with an RTSP signaling and another corresponding to the recording performed by the proxy.

An `RTSPProxySession` object possibly has associations to RTP Administration instances (Streaming Graph) which either perform streaming, reflection, recording and/or forward receiver and sender reports. An `RTSPProxySession` object does not have any association to a Streaming graph when only providing RTSP signaling (see Fig. 9). An `RTSPProxySession` is associated with one Streaming graph if the RTSP session administrates streaming, reflection or recording of one medium and with more graphs if the session performs an aggregate control of a number of media or streams a container file.

A proxy keeps the state of each process performing streaming, reflection or recording within the `RTSPProxySession`. Hence the object is well suited to provide an access accounting process with necessary information to charge a client or an origin server for the delivery of a media stream.

The implementation of the RTSP Administration for the proxy is different from the one for the origin server. Two different cases important for the implementation have been identified:

- **Streaming or Reflection:** Each RTSP session involved in a streaming or a reflection scenario must implement communication with a client as well as an origin server because the origin server takes a full part in the signaling scenario. The implementation may not simply forward messages between sockets but must maintain states for both communication dialogues and interact with a streaming process.
- **Recording:** Each RTSP session involved in a recording scenario must implement communication with an origin server only.

These two different cases are illustrated in Fig. 9 and Fig. 10 with different symbols identifying the RTSPProxySession instances.

**Streaming or reflection:** For the first case the RTSPProxySession implements a state machine that has both the functionality of the client and the server state machines described in [RFC2326]. The machine can be implemented as two machines (similar to the ones described in [RFC2326]) that interact closely. The client machine communicates with the origin server and the server machine communicates with the client. The machines interact through signals. Additional waiting states have to be added to the server machine (that communicates with the client). The server machine is situated in a waiting state when expecting a response from the origin server delivered from the client machine (that communicates with the origin server). The client machine might also have to implement a couple of response waiting states for the cases when an origin server is able to send a request to the client. Some additions might be necessary in the RTSP parser to cope with extensions in the protocol (see Section 4.2.7).

The machine must differentiate between the following cases: streaming from origin server and performing reflection, streaming from origin server and providing signaling only and streaming from cache.

**Recording:** For the second case the RTSPProxySession implements a state machine that has the functionality of the client machine described in [RFC2326]. Some waiting states might be added because of the interaction with the recording process and some additions might be necessary in the RTSP parser to cope with extensions in the protocol (see Section 4.2.7).

## Identifiers

The origin server allocates SessionIDs without taking the allocation of other servers into account. The RTSPProxySession must monitor the allocations perform a conversion if a collision occurs. The synchronization identifiers (SSRC) are discussed in Section 4.2.5.

## Routing of messages

The proxy and origin server communicate via a single TCP connection and multiplex the messages onto one message stream. Received messages are demultiplexed and handed out to the corresponding RTSPProxySession according to the SessionID in the message. The receipt of the message acts on the state machine which emits a new message or sends an error message back to the sender. A problem arises for the demultiplexing and routing of messages that do not contain any SessionID and/or do not belong to any RTSPProxySession because a session has not been installed and no RTSPProxySession object exists.

Special care is to take of the routing of the following messages:

*SETUP, DESCRIBE, GET\_PARAMETER*: Build a temporary state that keeps routing information through the proxy (a reference to the requesting client) and is indexed by outgoing CSeq and IP address (of origin server) and route the message to the URL specified in the message.

*DESCRIBE response* and *GET\_PARAMETER response*: Find the corresponding temporary state that was built when the request was received. Use the information from the state to route the response through the proxy.

*SETUP response*: Build a new RTSPProxySession instance that includes the information from the SETUP response and from the temporary state built when the corresponding SETUP message was received. Forward the message according to the routing information specified in the temporary state.

### 4.2.5 RTP Administration and Streaming

In order to implement the various streaming scenarios discussed in previous sections, a coarse design of a modular multithreaded RTP/RTCP based stream handler architecture was developed. The architecture provides RTP media streaming, reflecting, recording and RTCP packet generation, processing and forwarding. It can implement the streaming scenarios required in an origin server and a proxy. The architecture defines a datapath and a streaming graph:

**Datapath:** The datapath is a network of building blocks that together implement RTP streaming and RTCP control functionalities. Each block has well defined purposes and interfaces. The execution of the datapath is separated into threads to improve the performance and to prevent a blocking situation to occur in the datapath. A blocking situation might occur if a file descriptor blocks for a moment in accessing the file system. It is possible to implement a number of functionalities in a same datapath, e.g. reflecting and recording. It is also possible to serve a number of clients that receive the media stream synchronously with a single datapath. In this case a number of RTSP sessions take part in

a single datapath. This might be practical for providing application level media stream replication in a livestreaming or a pay-per-view session if the network does not support multicast.

**Streaming graph:** The purpose of a streaming graph is to build a datapath and maintain it. This includes, creating instances of the building blocks and connecting together into a graph. It also includes, doing necessary changes to the network as the participation of the RTSP session in the datapath changes. The streaming graph manages the datapath topology in a connection graph. A streaming graph interacts with `RTSPProxySession` or `RTSPServerSession` objects that correspond to the RTSP session participating in the datapath.

The purpose of each building block is described in the following:

**RTPReceiver:** The `RTPReceiver` block implements an UDP socket, receives packets from the socket and forwards them to another building block. It monitors the incoming packets and notifies the streaming graph if unexpected information are found in the RTP header, e.g. a new SSRC is found or a payload type identifier changes. The block does not send any RTP packets to the socket. The block is active, it runs a separate thread and executes functions of other passive blocks.

**RTPStreamer:** The `RTPStreamer` block implements an UDP socket, pulls RTP packets from a queue and forwards them to the socket. The block does not receive any RTP packets from the socket. It implements the real-time scheduling of the packets and runs a separate thread. The streaming graph must configure the scheduler according to the streaming format applied.

**RTPDepacketizer:** The `RTPDepacketizer` block filters RTP header from the RTP packet. It is a passive block, does not run a separate thread and is run from another thread.

**RTPPacketizer:** The `RTPPacketizer` block prepares RTP packets for transmission. The implementation varies among encoding schemes and must conform to the packetization specifications described in the payload type dependent profiles. The block runs in a conjunction with a `FileSource` block in one thread.

**RTPMonitor:** The `RTPMonitor` block provides packet statistics and RTCP functionality for RTP streaming. It maintains statistics for the RTP packets that have been transmitted, generates sender reports and collects receiver reports. This block might implement estimation of the performance of the streaming.

**Queue:** This block implements the packet queuing functionality for data transfer between threads, known as the producer/consumer problem. The queue must be protected from write hazards due to two simultaneous write attempts. A queue is in essence a passive object and does not run a separate thread. The queue is described here as a separate block but might as well be implemented within other blocks.

**PacketReplicator:** The PacketReplicator block receives packets and forwards them to a number of outputs. The number of the outputs and the connections to the input and output blocks are configured by the streaming graph at the beginning of the session or changed during its lifetime.

**FileSource:** The FileSource block accesses data from a file system. It maintains a file descriptor, reads data from the file system and forwards it to an output. The position of the file descriptor within the file can be configured with a byte count at the beginning. It is also possible to change the position with a new byte count during the session. This might be practical for providing retransmission of lost data (loss collection) as described in Section 2.2.3. The block runs a separate thread.

**FileSink:** The FileSink block writes data into the file system. It maintains a file descriptor, receives data from an input and writes it into the file system. The position of the file descriptor within the file can be configured with a byte count at the beginning. It is also possible to change the position with a new byte count during the session. This might be practical for the loss collection functionality as described in Section 2.2.3. The block runs a separate thread.

**RTCPSenderReceiver:** The RTCPSenderReceiver block implements a socket and receives and sends RTCP packets. The block does not have to implement real-time scheduling like the RTP streamer does. It simply pulls packets from a packet queue and delivers to a socket (sending) and receives packets from a socket and delivers to an output (reception). According to the description in Section 4.2.1 is RTCP communication between origin server and proxy supposed to be implemented with reliable connections (TCP) but the communication between client and proxy should be implemented with a UDP sockets (due to the requirement to keep the interface with the client unchanged). The RTSPSenderReceiver must be able to implement both variations. A RTCPSenderReceiver block runs a separate thread. An RTCPSenderReceiver block implemented within a proxy and taking care of communication between a proxy and an origin server has an association to a number of datapaths. The block performs a classification of sender report blocks (based on SSRCs) to the corresponding datapath. An RTCPSenderReceiver block implemented within an origin server has an association to a number of datapaths. The block performs a classification of receiver report blocks (based on SSRCs) to the corresponding datapath.

Two example datapaths are described and illustrated in the following:

#### *Streaming from cache*

The Fig. 12 illustrates a datapath corresponding to the session 1 described in Section 4.2.3. The datapath performs RTP streaming from a cache. It generates sender reports and sends to the client and the origin server. It also receives receiver reports from the client and forwards to the origin server. The figure is separated into two parts, one performing RTP streaming and the other performing RTCP message generation and forwarding. The

execution is separated into four threads. The first thread reads data from file system and prepares RTP packets and sender reports. The second thread performs scheduling and transmission of RTP packets. The third and fourth threads perform reception of receiver reports and generation of sender reports, replication and transmission.

### *Reflection and caching*

The Fig. 13 illustrates a datapath corresponding partly to the session 2 described in Section 4.2.3. In addition to the reflection described in Section 4.2.3 a recording procedure is also performed. The datapath performs reflection from an RTP stream and caching. It receives sender reports and forwards to the client and the packet monitoring block corresponding to the caching. The monitoring block generates receiver reports that are transmitted to the origin server. The datapath receives receiver reports from the client and forwards to the origin server. The execution is separated into five threads. The first thread receives RTP packets and forwards them into a queue for the reflection and a queue for the caching. The second thread performs scheduling and transmission of RTP packets. The third thread performs caching into the file system. The fourth and fifth threads perform generation and forwarding of receiver reports and forwarding of sender reports.

One might argue that an origin server is not able to estimate the performance of the streaming from the received receiver and sender reports correctly of the following reason:

The RTT estimation calculated by the origin server depends on the comparison of timestamp of a sender report and the reception time of a receiver report. The result of the estimation carried out with an existence of a proxy differs from the one if no proxy exists. The forwarding in the proxy introduces additional delay and if the streaming is performed by the proxy, sender reports are generated by the proxy but the reception timestamps of receiver reports are determined by the origin server. The estimation compares timestamps that originated from different clocks.

This issue does not cause any harm in the estimation of the streaming performance if changes in the RTT estimation are observed and not absolute values.

The performance of streaming is normally estimated from the packet loss rate and the introduction of the proxy does not change the results of the packet loss rate estimation because the estimation is carried out by the client and the results are forwarded within receiver reports from the client to an origin server without any changes.

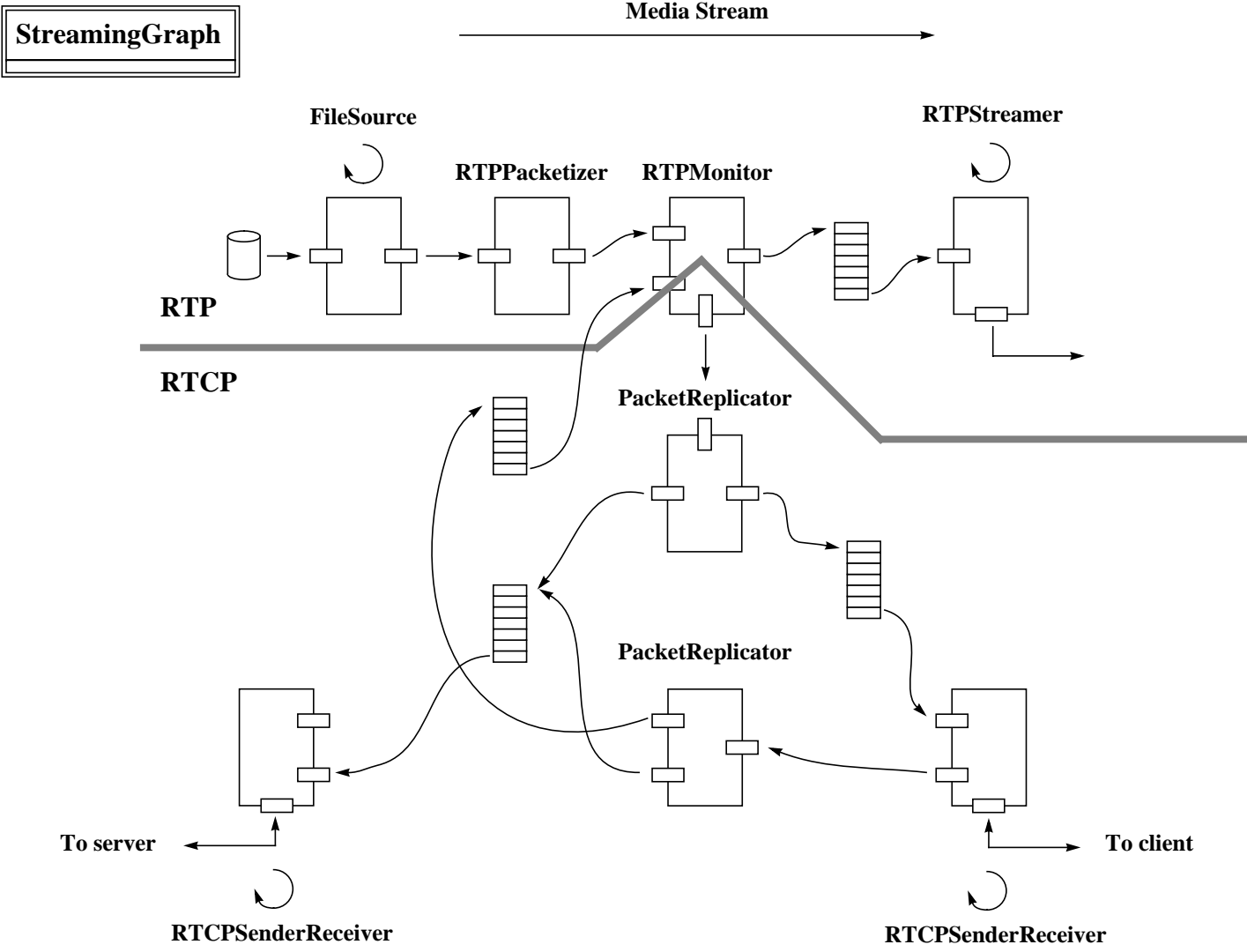


Fig. 12: A datapath for streaming from a cache

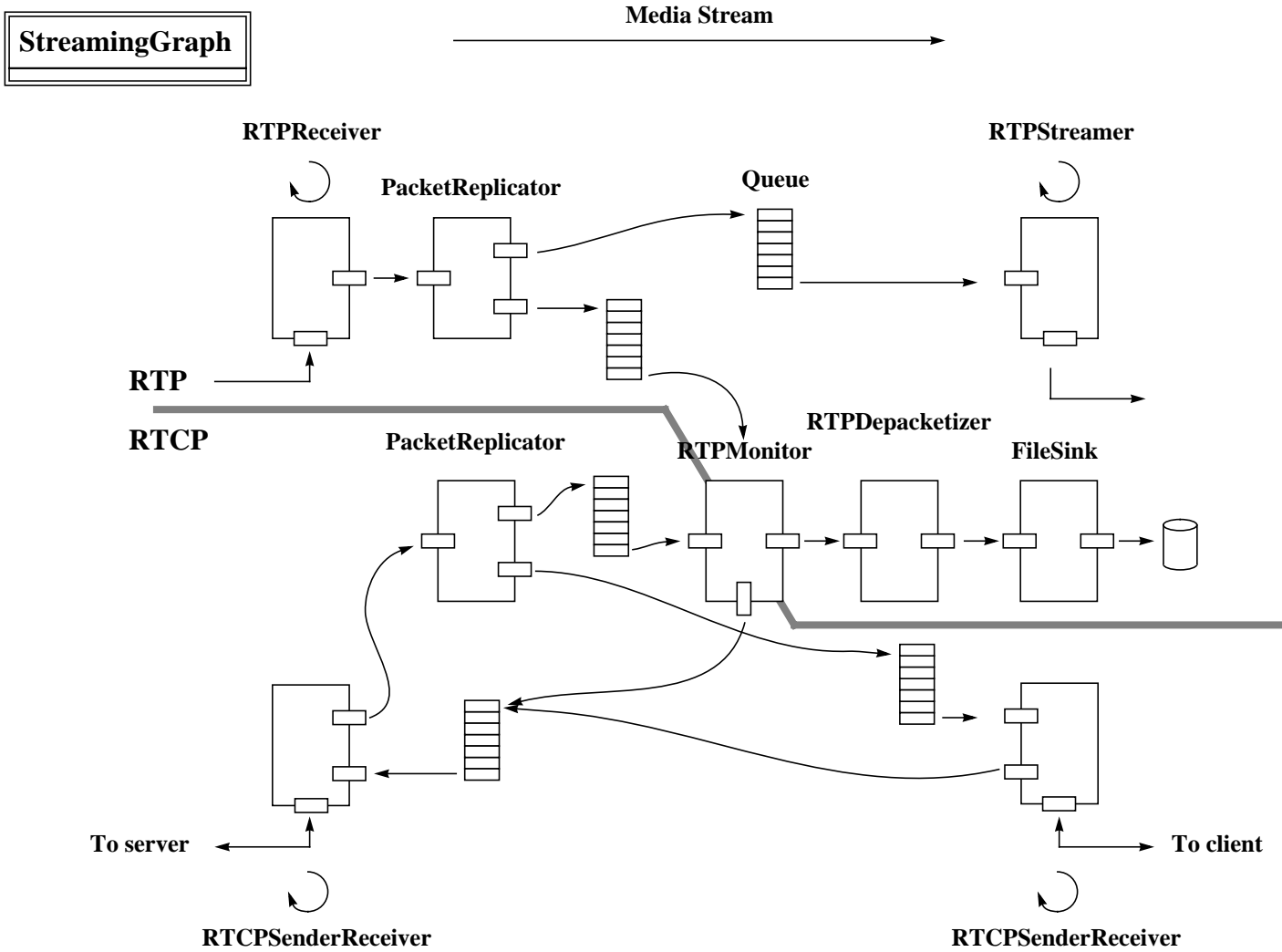


Fig. 13: A datapath for reflection and caching

## 4.2.6 Cache Directory

The cache dictionary stores an entry for every media content in the cache. Each entry includes information from the SDP session description and the URI of the original media. The cache dictionary is indexed after the session name indicated in the session description and the URI. Each entry stores the path and name of the streaming file stored in the file system of the cache.

The purpose of the session name indexing is to enable the proxy to serve requests that provide the name of a movie or programme and not URI.

Each entry must store the corresponding URI in addition to the information from the session description to enable the proxy to setup communication with the corresponding origin server. A proxy is able to store media content from and to setup connections to a number of origin servers.

## 4.2.7 Protocol Scenarios

This section describes necessary extensions to the RTSP protocol between the proxy and the origin server for the proxy functionality described in the previous sections.

**Streaming from a cache or streaming from an origin server:** In the setup phase of a session the signaling must differentiate between the following two cases: a proxy is able to stream from cache or origin server must stream the requested media. A new transport parameter called `proxy_streaming` differentiates between these cases. A proxy adds the text "*proxy\_streaming*" into the transport header of the SETUP message to inform the origin server that it is able to take care of the streaming. The origin server adds the text to the transport header in the response if it agrees that the proxy streams. If the text does not exist in one of the two messages the streaming is performed by the origin server.

**Streaming to a client or recording:** In the setup phase of a session the signaling must differentiate between the following two cases: a proxy requests a setup of a streaming session on behalf of a client or for recording purposes. A new transport parameter called `proxy_caching` differentiates between the cases. A proxy adds the text "*proxy\_caching*" into the transport header of the SETUP message to inform the origin server that it would like to setup the session for recording purposes. If an origin server is not willing to let a proxy record a session, it sends an error response back including with error number 401 and the text "Unauthorized".

**RTCP communication between origin server and proxy:** In a streaming session using unicast transport, RTCP messages between origin server and proxy are multiplexed over a single TCP connection or a single UDP port pair. In the setup phase, the origin server and the proxy must negotiate on a port pair for UDP communication or a port of a TCP server socket that generates RTCP connections. A proxy willing to setup a TCP

connection for the RTCP messages inserts the following text into the transport header of the SETUP message: "*proxy\_RTCP=TCP*" that informs the origin server that the proxy is willing to setup a TCP connection. The origin server puts the following text to the transport header in the response: "*proxy\_RTCP=TCP/port*", where port identifies the port number of the server socket. A proxy that would like to send RTCP messages via UDP inserts the following text to the transport header: "*proxy\_RTCP=UDP/proxy\_port*", where proxy\_port identifies the port number of the proxy. The origin server inserts corresponding text to the response: "*proxy\_RTCP=UDP/originserver\_port*".

### **Association between Session IDs and SSRCs**

Receiver Reports are multiplexed over a single connection between proxy and origin server. The origin server must be informed on the association between SessionID and the SSRC of each receiver to be able to associate the receiver reports to the corresponding sessions in order to monitor the performance of the streaming in each session. The proxy sends a SET\_PARAMETER message that contains the following text: "*SessionID-SSRC: <SessionID> = <SSRC>*". A proxy sends a new SET\_PARAMETER message as soon as it recognizes that a receiver has changed its SSRC due to collision resolution.

### **An origin server requests the content of a cache**

An origin server can send a DESCRIBE message to a proxy to request the content of the cache. If the proxy recognizes that the message originated from the server specified in the URI it responds with a set of SDPs from the cache directory that corresponds to the URI. The DESCRIBE message might specify a particular URI or not.

### **An origin server requests a deletion of a content in cache**

Normally the caching strategy in the proxy takes care of deletion of old content that is not more profitable for the proxy to keep in the cache. It might also be practical for the origin server to request a proxy to delete a media from the cache. An origin server sends a SET\_PARAMETER message with the following text for this purpose: DELETE <URI>. If the host part of the URI matches to origin server it deletes the content from the cache.

Two RTSP Protocol scenarios are described and illustrated in Appendix B and C. They include the necessary extensions for the proxy support.

## 5. Multiformat Support

A media streaming server with a multiformat support is able to switch from one encoding format to another within one streaming session. The purpose of the formatswitching is to adapt the media streaming to the conditions in the network and stream media content in a format that suites the condition at any time. This effort is due to the fact that the network technology applied in the Internet today provides end users with best effort service and does not guarantee the provision of network service with specified quality during a communication session. The conditions in transmission of a media stream can change during the session.

A proxy typically serves requests from clients located within one city, a quarter in a city or an organization. The conditions in the network path between proxy and client might change from time to time as well as in the global Internet if best effort network service is used. It is therefore reasonable to implement a formatswitching functionality into a proxy to enable the proxy to adapt media streaming when it streams from its cache.

It is likely that an origin server and a proxy are not operated by the same organization. A television company or a content production company might operate an origin server and a telecom operator or an internet service provider might operate proxies. The origin server must keep synchronization between the different formats to be able to perform formatswitching. If the proxy and the origin server are not operated by the same organization the origin server would like to participate in each streaming session to be able to perform access accounting and keep the content in a complete form locally. An origin server would like to keep the synchronization information between the different formats locally and not to distribute it to proxies to prevent an untrustworthy proxy from streaming and switching between formats without the participation of an origin server.

A multiformat support for the proxy design from the previous chapter is specified in this chapter. During the normal streaming phase the proxy records the packets of the stream as they pass by. At the end of the phase the proxy has recorded non-continuous ranges of different formats into its cache if the origin server performed formatswitching during the streaming phase. This is illustrated in the upper part of Fig. 14. The proxy maintains the boundaries of the recorded ranges whilst recording and uses this information to request a transmission of the missing ranges during the loss collection phase. This is illustrated in the lower part of Fig. 14. This enables the proxy to store media content in different formats into a local cache. The design specified in this chapter enables a proxy to provide media streaming from a local cache with formatswitching. The formatswitching is controlled by an origin server.

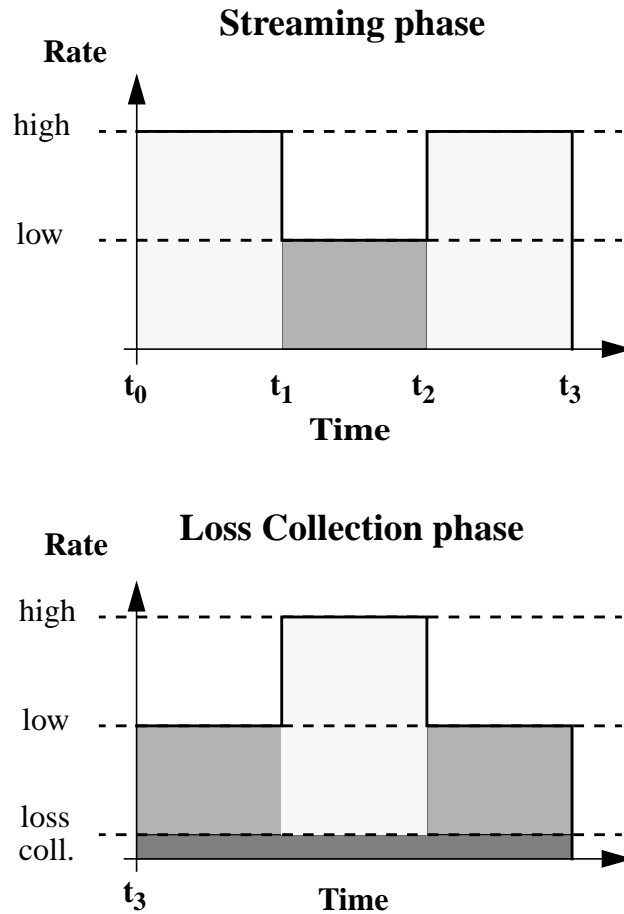


Fig. 14: Caching and formatswitching

## 5.1 Design goals

The goals of the design of the multiformat support are stated in the following:

**Existing proxy design:** The architecture should specify a multiformat support for the proxy design specified in Chapter 4.

**Simplicity:** The design should be as simple as possible because otherwise people are not willing to implement it and experiment with it.

**Interface to a client:** The multiformat proxy support may not cause a client to experience any changes to the protocols RTP, RTSP and SDP specified in [RFC1889], [RFC2326] and [RFC2327] respectively.

**Caching hierarchy operation:** A proxy implementing the multiformat support should be capable of operating in a caching hierarchy.

*Formatswitching decision:* An origin server decides to switch from one format to another. A proxy is only allowed to perform formatswitching following a corresponding request from an origin server.

## 5.2 Specification

A design of multiformat support for an RTSP based proxy is specified in this section. It is an extension to the architecture specified in Chapter 4. The section starts with a short overview. Necessary extensions to the RTSP Administration, RTP Administration and streaming are specified in Section 4.2.4 and Section 4.2.5. The chapter closes with specifications of necessary changes to the RTSP protocol to support the multiformat streaming and caching. Multiformat support extensions have been added to the example protocol scenarios specified in Appendix B and Appendix C. The results are given in Appendix D and Appendix E.

### 5.2.1 Architecture Overview

In this section the main properties of multiformat extensions to the RTSP based proxy design are stated. The architecture assumes that only an origin server knows the synchronization between different formats for a particular media content and is the only one that is able to determine the exact bytepositions within leaving and entering streams where a formatswitching can be carried out.

An origin server monitors the performance of the streaming for each RTSP session. It receives receiver reports from all clients participating in a session administrated by the origin server. The receiver reports provide the server with a feedback on the quality of the streaming that the initiator of the report experiences. The server also receives sender reports corresponding to the streaming process it administrates.

A formatswitching is to be considered in the following cases:

- If a receiver experiences small or none data loss rate for some time and low jitter it is reasonable to conclude that the network is able to provide the user with higher data rate. A media streaming process might under these conditions switch to a format that requires more network resources and provide the user with better quality.
- If a receiver experiences high data loss rate or high jitter it is reasonable to conclude that a congestion is building up in the network. A media streaming process must under these conditions switch to a format that requires less network resources to avoid congestion.

This enables the origin server to monitor the performance of the streaming and either perform a switching if it is streaming itself or to request a proxy to switch its streaming to another format.

**Scenarios:** The following applies to each of the scenarios identified in Section 4.2.1:

- *Streaming from an origin server to a client:* The origin server announces a format-switching event with an ANNOUNCE message, waits for responses and carries the formatswitching out in its datapath. This applies to all the cases where an origin server streams, independent of if a proxy performs a reflection or not.
- *Streaming from a proxy to a client:* The origin server requests a proxy to perform formatswitching with an ANNOUNCE message, the proxy forwards the request to its clients, waits for responses, carries the formatswitching out in its datapath if the responses were positive and sends a positive response back to the origin server.
- *Recording into a cache in the proxy:* A recording process behaves like any other client. It must be able to take care of different formats and report the origin server on the quality that it is experiencing. A recording process may not influence the origin server to switch from one format to another because it is recording packets as they pass by and lost packets are transmitted in the loss collection phase at the end of the session.

**SSRC:** A unique SSRC must be allocated to each format in a streaming session for the following reasons:

- *Sender reports:* An origin server maintains server statistics for each format separately. It sends the statistics in separate sender report blocks within sender reports. Each block is identified with a corresponding SSRC and enables the proxy or a client to associate the block to a corresponding format.
- *Receiver reports:* A client must collect performance statistics for each format separately. It sends the statistics in separate receiver report blocks within receiver reports. Each block is identified with a corresponding SSRC and enables the origin server to associate the block to a corresponding format.
- *Loss lists:* At the end of a recording session, the loss collection protocol requests retransmission of data from different formats within the single session. It builds a loss list that contains requests corresponding to different formats. It must identify each request entry with the corresponding SSRC.

**RTSP Communication:** The following RTSP signaling functionality is needed for the multiformat support. The message specifications are given in Section 4.2.7:

- Origin servers and proxies inform each other with a corresponding transport header that they are able to take care of formatswitching.

- An origin server announces or requests a formatswitching event and identifies the exact bytepositions in the leaving and entering streams. This is performed with the ANNOUNCE message and an additional attribute field.
- A proxy allocates SSRCs for available formats and informs an origin server on the formats and the allocation if the media streaming is performed by the proxy. An origin server performs the allocation and informs the proxy if the origin server performs the streaming.
- Origin servers and proxies identify each available format with a unique payload type. The payload types can be static as well as dynamically allocated (see [RFC2327] and Section 5.2.5).
- A DESCRIBE message with a formatswitching request header requests a proxy and an origin server to list the available formats for a particular media content.

**RTCP Communication:** An unique SSRC is allocated to each format in a streaming session for the reasons previously stated. An origin server maintains server statistics for each format separately. It sends the statistics in separate sender report blocks within sender reports. A client and a recording process in a proxy must collect performance statistics for each format separately. They send the statistics in separate receiver report blocks within receiver reports. The RTCP communication must not be extended apart from this issue.

**Cache Directory:** The cache directory in a proxy with multiformat support must be able to store, search and retrieve information for different formats of a single media content. It must be able to create an SDP session description containing information on the formats and must allocate unique filenames for the files storing the different format of media content.

**Caching Strategy:** In addition to the functionality described in Section 4.2.1, the caching strategy must decide what formats of the media content are to be recorded into a local cache (see Section 5.2.4).

## 5.2.2 RTSP Administration

In the initialization of a streaming session the RTSP Administration gets information on available formats from the cache directory if streaming is performed by a proxy or a similar directory module if streaming is performed by the origin server. The RTSP Administration allocates a unique SSRC to each format. The RTSP Administration sends a SET\_PARAMETER message to the communication partner (proxy to an origin server or vice versa) informing it about the allocation.

An origin server maintains a performance state for each RTSPServerSession object. The state stores the fraction lost field from the latest receiver report block received for the session. The corresponding Streaming Graph object triggers an update of the value everytime when a new receiver report block for the session is received. The value influence the formatswitching decision carried out by the Streaming Graph and is described in Section 5.2.3. One might argue that it is insufficient to make a formatswitching decision based on the latest fraction lost value reported and an estimation of the fraction lost based on the history of the reported value is necessary (e.g. an exponential averaging). The fact is that receiver reports are not frequently transmitted and one report covers pretty long discovering time and it is reasonable to conclude that it provides a stable estimation of the performance of the streaming.

### 5.2.3 RTP Administration and Streaming

This section describes changes that are necessary to the RTP Administration and Streaming part of a proxy and an origin server in order to support formatswitching.

**RTPMonitor:** The RTPMonitor block must be extended to take care of multiple formats. It must maintain statistics for each format for the following purposes:

- *Loss List:* The block must in the case of a recording scenario in a proxy determine the ranges for each format that are not received by the proxy, in order to be able to request the transmission of the ranges through a loss list during the loss collection phase.
- *Receiver Report:* The block must in the case of a recording scenario in a proxy keep a separate statistics state for each SSRC, in order to be able to provide the origin server with correct receiver reports for each SSRC.
- *Sender Report:* The block must in the case of a streaming scenario from an origin sever or a proxy keep a separate state for each SSRC, in oder to be able to provide correct sender reports for each SSRC.

**RTPDemultiplexer:** This is a new class that demultiplexes RTP packets based on SSRC in the header from one input to a number of output ports.

#### *Origin server*

The origin server copies the fraction lost fields reported from clients and makes a formatswitching decision based on the their values. An RTCPSEnderReceiver object knows the association between SSRCs and a datapath. As soon as a receiver report is received forwards the object the report to an input queue for the RTPMonitor. The RTPMonitor forwards the receiver report to the Streaming Graph. The Streaming Graph knows the association between SSRCs and SessionIDs and an update of the fraction lost field in the corresponding RTSPServerSession object follows. At this point in time the

Streaming Graph discovers the fraction lost fields of all RTSPServerSession objects taking part in the datapath (normally this is a single object but for a livestreaming session there exist a number of objects). The Streaming graph might decide to perform a formatswitching dependent of the values of the fields. If this is the case the Streaming Graph specifies the bytepositions within the leaving and the entering stream for the formatswitching and requests the RTSPServerSession objects to send an ANNOUNCE message. The logic behind the decision is a simple threshold if a single session participates in the session. The formatswitching is carried out when responses from clients have been received. RTSPServerSessions trigger the Streaming Graph when they receive a reponse and the Streaming Graph then performs the switching.

### *Proxy*

The following specification corresponds to a recording session within a proxy. An origin server sends a session description to the proxy in the beginning of the recording session that lists all the formats that can appear in the stream during the lifetime of the session (see Section 5.2.5). The Streaming Graph builds a network of building blocks that can take care of the recording. An example of a network is shown in Fig. 15. The figure corresponds to the streaming scenarios in Fig. 9. The network contains an instance of RTPDepacketizer and FileSink for each format that can appear. An instance of RTPDemultiplexer demultiplexes RTP packets from the RTPMonitor to the corresponding RTPDepacketizer - FileSink pair. The RTPMonitor maintains information on the received and missing information for each format. The datapath is able to receive RTP packets from any formats that are listed in the session description in the beginning and can collect missing packets for each format during the loss collection phase.

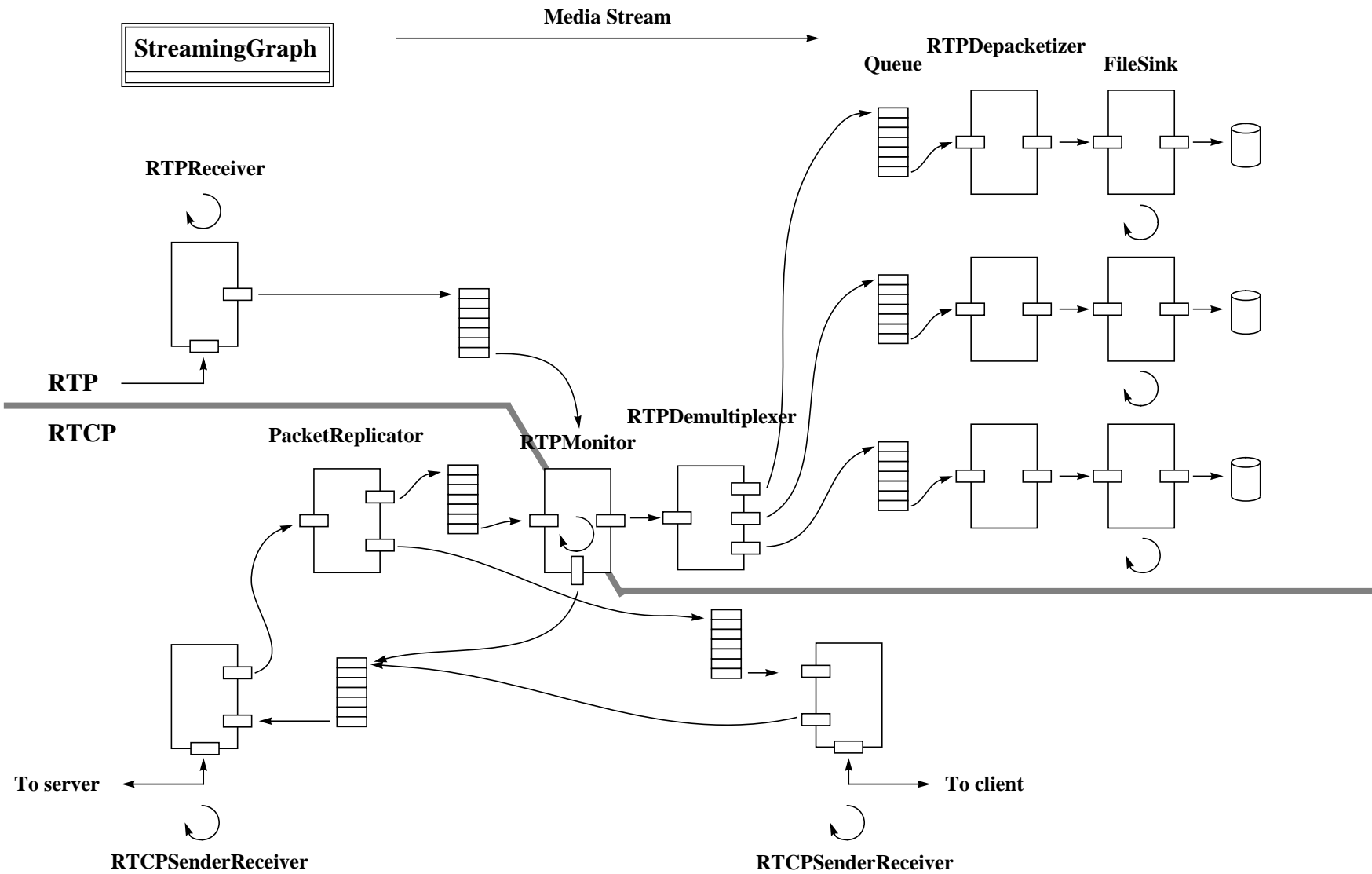
## 5.2.4 Caching Strategy

A caching strategy module in a proxy decides if a media stream should be recorded into a local cache from a session as described in Section 5.2.4. In addition to this functionality a caching strategy in a proxy with multiformat support must decide what formats of the session should be cached. Three different strategies have been identified:

**All available formats:** A proxy caches all the formats of the media content that are available from the origin server but not only the formats that appear during the streaming session. *Advantage:* A proxy that streams media from a cache streams the complete session from its cache and does not have to request an origin server to take over at a point in time when formatswitching is requested because appropriate format does not exist in cache. It provides its customers with a more reliable service. *Disadvantage:* This approach requires a lot of storage space for the cache.

**Formats that appear during the session:** A proxy caches only the formats of the media content that appear during the streaming session. *Advantage:* This approach requires less storage space than the previous one. *Disadvantage:* A proxy that streams media from a

Fig. 15: A datapath for recording with formatswitching (corresponds to Fig. 9)



cache might have to request an origin server to take over when formatswitching is requested because a format appropriate to the conditions in the network does not exist in the cache. If the route from the origin server is congested at the moment when the switching takes place the client might experience unreliable service.

**A set of chosen formats:** A proxy caches only the formats that suite the network service between the proxy and the client. A proxy is often located near end users in the network topology and the network can carry streaming with more bandwidth from a proxy to the client than from an origin server. A proxy might want to cache the formats that provide the best quality and require high bandwidth because other formats are never accessed from the cache. *Advantage:* This is the best strategy if the route from the proxy to the client is reliable. *Disadvantage:* The approach does not perform well if the chosen set of formats does not suite the network service any more.

The network technology implemented in the Internet today does only provide best effort service and no service with guaranteed quality of data delivery. A streaming process that relies partly on streaming from a cache in a proxy and partly on an origin server according to conditions in a network is unreliable. It is therefore reasonable to cache a particular media content in all available formats in order to provide a client reliable service.

Decisions like these are the problem of caching strategy module but it is important that the RTSP signaling between a proxy and an origin server supports the three varieties. The signaling presented in the next section supports all the three varieties.

### 5.2.5 Protocol Scenarios

This section describes necessary extensions to the RTSP protocol between the proxy and the origin server for the multiformat support described in the previous sections.

**SDP Session Description:** A proxy and an origin server insert additional entries into the session description to identify formats that they are able to stream. A unique payload type is used to identify the formats. The payload type might be dynamically allocated (see [RFC2327]). Following fields are inserted into the session description: *m= .. <fmt list>*, *a=fmtp* and *a=formatswitching*.

*m= .. <fmt list>*: A format list in the media field [RFC2327] can be used to identify available formats. The list contains the corresponding payload type codes separated with a space and can contain static as well as dynamic payload types. Dynamic payload types correspond to encoding types that have been parameterized with an attribute field *a=rtpmap* (see [RFC2327]). The first format in the list identifies the default format.

*a=fmtp*: This attribute field provides the association of a static or a dynamic payload type to an SSRC that has already been allocated for the streaming. This enables a proxy and an origin server to classify sender reports and receiver reports to a corresponding format and setup the necessary demultiplexing functionality in the datapath (see Section 5.2.3). The format of the field is as following:

*a=fmtp*:<payload type> *ssrc*=<SSRC>

*a=formatswitching*: This attribute is new and is contained within an ANNOUNCE message and requests a formatswitching. It identifies SSRCs of leaving and entering streams and the exact byte positions of the formatswitching within the both streams. The format of the field is as following:

*a=formatswitching*: <leaving SSRC> <leaving byte pos>

<entering SSRC> <entering byte pos>

Following examples show application of the attribute fields:

**Example 1:** A session description identifying available formats in a proxy or an origin server and corresponding SSRCs that have been allocated:

```
v=0
o=- 2890844526 2890842807 IN IP4 130.83.139.118
s=Mission Impossible 2
m=video 0 RTP/AVP 97 98 99
a=control:rtsp://lute/movies/mi2.en
a=rtpmap:97 MP1S
a=fmtp:97 rate=800
a=fmtp:97 ssrc=1234
a=rtpmap:98 MP1S
a=fmtp:98 rate=1300
a=fmtp:98 ssrc=2345
a=rtpmap:99 MP1S
a=fmtp:99 rate=1500
a=fmtp:99 ssrc=3456
```

**Example 2:** A session description requesting a formatswitching between two of the formats identified in the previous example:

```
v=0
o=- 2890844526 2890842807 IN IP4 130.83.139.118
s=Mission Impossible 2
m=video 0 RTP/AVP 97 98 99
a=control:rtsp://lute/movies/mi2.en
a=formatswitching:1234 12345678 2345 23456789
```

**Transport Request Header:** A proxy inserts the field *proxy\_formatswitching* into a transport header of an RTSP message to inform an origin server that it is able to perform formatswitching. The origin server inserts the text into the response if it supports formatswitching and is willing to setup an RTSP session with formatswitching functionality. The field is simply the text "proxy\_formatswitching".

**SET\_PARAMETER:** A transmission of an SET\_PARAMETER message regarding formatswitching has four purposes:

- A proxy sends a SET\_PARAMETER message to the origin server to describe the content of its cache for a particular media. It inserts an SDP session description into the message that contains all available formats. The origin server can estimate from the description if the corresponding stream request is to be served by the proxy or from the origin server. The message is sent before a SETUP message.
- A proxy sends a SET\_PARAMETER message to the origin server to inform it on allocated SSRCs. The message is sent after a session has been setup for streaming from a proxy. The SSRCs in the description enable the origin server to classify receiver reports to formats if streaming is performed by the proxy.
- An origin server sends an SET\_PARAMETER message to the proxy to describe the formats available for a particular media. It inserts an SDP session description into the message that contains all available formats.
- An origin server sends an SET\_PARAMETER message to the proxy to inform it on allocated SSRCs. The message is sent after a recording session has been setup. The SSRCs in the description enable the proxy to classify sender reports to formats.

**ANNOUNCE:** An origin server emits an ANNOUNCE message to announce a formatswitching or request a proxy to perform a formatswitching. The message contains an SDP session description for the media and the formatswitching attribute previously described. The attribute specifies the positions of the formatswitching within the leaving and entering streams and refers to the SSRC allocation identified in previously transmitted session description.

**DESCRIBE:** The content of a proxy and an origin server can be requested with a DESCRIBE message. If proxy\_formatswitching field is inserted into the message the receiver must response with a session description containing entries for all formats that it is able to stream if it supports formatswitching.

The protocol extensions are illustrated in two example protocol scenarios in Appendix D and Appendix E that extend the scenarios from Fig. 9 and Fig. 10 for formatswitching.

## Formatswitching during the setup phase

Consider the following protocol scenario describing a setup phase for an RTSP media streaming session. The scenario describes a setup phase that adds a receiver to a session that is already running with a number of users. C, P and S are abbreviations for a client, a proxy and an origin server respectively:

- (1) C --> P DESCRIBE
- (2) P --> S DESCRIBE
- (3) S --> P OK
- (4) P --> C OK
- (5) C --> P SETUP
- (6) P --> S DESCRIBE
- (7) S --> P OK
- (8) P --> S SETUP
- (9) S --> P OK
- (10) P --> C OK

The first four steps do neither cause any state creation within the origin server nor within the proxy. The DESCRIBE phase is thought for information retrieval and not for installation of communication entities. A problem occurs if the origin server decides to carry out a formatswitching between the timepoints (3) and (8) because the session description forwarded to the client in (3) does not match the entering format and the client is not a participant in the session yet when the origin server announces the event with an ANNOUNCE message. If the formatswitching occurs before (3) the session description transmitted in (3) is up-to-date and if it is carried out after (8) the origin server announces the event with an ANNOUNCE message.

**Solution 1:** Recognize the problem at the proxy and send an ANNOUNCE message from the proxy to the client between (9) and (10). *Advantage:* The proxy solves the problem without any necessary changes at the client or the server. *Disadvantage:* The proxy must create a temporary state from the DESCRIBE response.

**Solution 2:** Recognize the problem at the proxy and send a BAD REQUEST from the proxy to the client at (8) and ignore the rest of the scenario. The client will probably try the whole scenario again. *Advantage:* It depends on the client implementation if this approach works. *Disadvantage:* The proxy must create a temporary state from the DESCRIBE response.

**Solution 3:** Recognize the problem at the proxy and send an ANNOUNCE message from the proxy to the client after (10) that informs the client on formatswitching. *Disadvantage:* The proxy must create a temporary state from the DESCRIBE response.

**Solution 4:** The origin server installs a temporary state after receiving message (2) that disallows it to carry a formatswitching out within some seconds. *Advantage:* The problem never occurs. *Disadvantage:* The origin server must install a state after receiving a DESCRIBE request.

**Solution 5:** Assume that a transmission of a SETUP message follows directly from a receipt of a DESCRIBE response. Ignore the problem (without trying to recognize it) because the probability that it occurs is very low and hope that the client recognizes the format from the received datastream (from the payload type field in the RTP header) and gets along the problem. Hope that the client either switches to a new decoder or recognizes the problem and starts the scenario over again. *Advantage:* No changes needed. *Disadvantage:* Does not solve the problem in any way.

All solutions except for solution 5 require the proxy or the origin server to install a temporary state resulting from the DESCRIBE request. The purpose of the DESCRIBE message is to retrieve information and not build a state.

It is rather unlikely that a problem of this kind occurs. Solution 5 is probably the best one although it is not a solution. Experiments with an implementation will show if this approach is reasonable.

### **Serving new requests during recording**

Requests for a media content that arrive at a proxy when the media is being recorded into the proxy make a little problem. The proxy can not request the origin server to start a new media streaming to serve the request because that influences the quality of the transmission for the stream being recorded. This is at least the case if the network does not provide guaranteed quality of service. The proxy must serve the request from the cache in the same form it was recorded into the cache with the corresponding formatswitching and data losses (loss collection starts after the normal streaming process). During the recording a proxy classifies the received RTP packets according to SSRCs and writes them to corresponding files (see Fig. 15). The proxy must jump between the files to provide the client that makes a request during caching with continuous streaming. The proxy must maintain a list of formatswitching that took place during the recording and keep it until the loss collection has been carried out.

### **Identifying a format with a unique SSRC**

One might argue that a unique payload type can be used to identify each format. The problem is that a receiver report and a sender report only contains SSRC but no payload types. Statistics must be maintained for each format separately and it must be able to classify sender reports and receiver reports to corresponding format. In a session that contains media from different origin servers, payload types might not be unique any more. This is the case if different origin servers transmit media with the same encoding.

## 6. Summary

A design of a real-time streaming proxy is specified in Chapter 4. The design is partly based on the existing KOM implementation. A design of an origin server that supports a proxy arrangement is also given in Chapter 4.

The design can support the following scenarios:

- streaming from an origin server direct to a client
- streaming from an origin server to a proxy and reflecting the stream to a client
- streaming from a proxy to a client
- recording a session into a proxy

An origin server participates in every session (independent of if it performs the streaming or if a proxy does it) and controls the streaming. It monitors the signaling in the session and maintains a state for the participation of each client in a session and can perform access accounting.

The design is split into two parts: RTSP Administration and RTP Administration and streaming.

The RTSP Administration functionality in a proxy is represented with the `RTSPProxySession` object that manages one RTSP session and corresponds to the participation of one client in a streaming process. The object maintains two finite state machines, the server machine and the client machine that interact. The server machine maintains a state for the communication between proxy and a client and reacts on a receipt of a message from a client or a signal from the client machine. It can influence the RTP administration part, transmit messages and act on the client machine. The client machine maintains a state for the communication between the proxy and a server and reacts on a receipt of a message from an origin server or a signal from the server machine. It can influence the RTP administration part, transmit messages and act on the server machine.

The RTSP Administration functionality in an origin sever is represented with the `RTSPServerSession` object that manages one RTSP session and corresponds to the participation of one client in a streaming process. The object maintains a state machine, reacts on a receipt of a message and can influence the RTP administration part and transmit messages. This machine might also implement an access accounting functionality.

The RTP Administration and streaming part consists of a datapath and a streaming graph:

Media streaming is performed by a datapath. A datapath is a multithreaded modular system that is able to perform streaming. A datapath is able to stream to a number of clients that all receive the same media stream synchronously. This is practical for livestreaming or pay-per-view applications. Following are examples of streaming procedures performed by a datapath: streaming from a file system, receiving and writing a stream into a file system (recording) and receiving a stream and forwarding to a number of clients (reflection). The datapath performs streaming, monitors performance of the streaming experienced by receivers and sends and collects sender and receiver reports.

A streaming graph installs and controls a datapath and interacts with the RTSP Administration. In the beginning at a session, does the object create streaming building blocks, connects them and starts streaming. A streaming graph modifies the arrangement if necessary and tears it down at the end of the session. A streaming graph has associations to many RTSP sessions but each RTSP session only has an association to one streaming graph.

RTSP messages transmitted between a proxy and a particular origin server are multiplexed into a single TCP connection. This is done for scalability reasons. TCP provides reliable transmission. Unique RTSP SessionIDs enable the receiver to demultiplex the messages to corresponding session objects (RTSPProxySession or RTSPServerSession).

RTCP messages transmitted between a proxy and a particular origin server are multiplexed into a single TCP connection. This is done for scalability reasons. TCP provides reliable transmission. Unique SSRCs enable the receiver to demultiplex the messages to corresponding datapaths (RTPMonitor objects).

The following protocol extensions are necessary for the proxy support: A proxy inserts the text "proxy\_streaming" into a SETUP message to indicate that it is ready to perform the streaming from a cache. A proxy inserts the text "proxy\_caching" into a SETUP message to indicate that the purpose of the session is to record the media content into a cache.

Fig. 9 illustrates multicast streaming sessions from an origin server to a client and a recording process into a proxy's cache. Fig. 10 illustrates two sessions. In the first session a proxy is streaming from a cache to a client. In the second a proxy receives a streaming from an origin server and reflects it to a client. These sessions are discussed in aspect of object instances and protocol scenarios in Chapter 4 and Chapter 5. A complete protocol scenario example for the sessions in Fig. 9 is illustrated in Appendix B. A complete protocol scenario example for the sessions in Fig. 10 is illustrated in Appendix C.

A design of a multiformat proxy support is specified in Chapter 5. It enables a streaming process to switch between formats of the same media content in order to adapt the streaming to changing transmission conditions in the network. The design is an extension of the proxy design specified in Chapter 4.

An origin server monitors the quality of the streaming that each receiver experiences and performs a formatswitching (if it is performing the streaming itself) or requests the corresponding proxy to perform the formatswitching. The origin server keeps the synchronization information between various formats locally and does not distribute the information for copyright reasons.

The following protocol extensions are necessary for the multiformat support: A proxy inserts the text "proxy\_formatswitching" into a SETUP message to indicate that it supports formatswitching. Origin servers and proxies inform each other on available formats in their storage and identify all available formats within a single SDP description. This is supported by SDP and does not require any changes to the SDP protocol. An origin server sends an Announce message to request a proxy to jump between formats. The message contains an SDP description that identifies the two formats, called leaving and entering formats and specifies the two byte positions within the formats where the formatswitching is to take place. This requires a new attribute field in the SDP description (see Section 5.2.5).

The server (origin server or a proxy) performing the streaming allocates a unique SSRC to each available format before the streaming process begins. Origin servers and proxies maintain statistics for streaming or receiving of media streams for each format separately. A proxy informs the origin server on the SSRC allocation if the proxy performs streaming. The SSRCs enable the origin server to classify received receiver reports to the corresponding format. An origin server informs a proxy on the SSRC allocation if the origin server performs streaming. The SSRCs enable the proxy to classify received sender reports to the corresponding format. This does not require any new attribute field within the SDP protocol because the "a=fmtp" field can be used.

A datapath performing a recording of a media stream with formatswitching is illustrated in Fig. 15. A complete protocol scenario for the example in Fig. 9 together with formatswitching functionality and a formatswitching event is described in Appendix D and illustrated in Fig. 16 in Appendix D. A complete protocol scenario for the example in Fig. 10 together with formatswitching functionality and formatswitching events is described in Appendix E. A complete protocol scenario for streaming from proxy cache in Fig. 10 is illustrated in Fig. 17 in Appendix E.

## 7. Further Work

This chapter states two issues regarding further work on the project presented in this thesis:

### **Implementation**

The design presented in the previous chapters has not been implemented. An implementation should be built to research the field of media streaming in the Internet with the protocols presented in Section 2.2. Various scenarios can be tested including a datapath serving a single client and a datapath serving a number of clients with different scenarios. The design does not specify the aspects of the system in detail but gives an overview of the system and specifies the protocol scenarios in detail. Some work need to be done on other parts, for example: The specification of the closely related state machines of RTSPProxySession object, datapath management functionality of the Streaming graph object and interaction between RTSPProxySession object and Streaming Graph object (as well as interaction between RTSPServerSession object and Streaming Graph object ).

### **Formatswitching in aspect of encodings**

The design does not handle formatswitching in aspect of particular encoding formats. Various encoding formats should be observed to research if additional header information need to be generated for the entering format following a formatswitching. This might be needed in order for the client to be able to start decoding after the formatswitching. It is questionable if a proxy is able to generate these information or if an origin server must perform the generation. The latter case requires a transport mechanism for the header information from the origin server to the proxy. This issue is to be researched.

# Appendix A

## An example of an RTSP scenario

The following scenario illustrates an streaming from a multimedia content including audio and video from a media server to a client. The scenario includes a formatswitching operation performed by the server. The server informs the client on the formatswitching through a transmission of the Announce message.

*Assumption:* The client knows the URI of the requested presentation at the beginning of the scenario. The client might have got the information through from a website retrieved through an HTTP connection.

- (1) C->S: DESCRIBE rtsp://lute/cn/3 RTSP/1.0  
CSeq: 1
  
- (2) S->C: RTSP/1.0 200 OK  
CSeq: 1  
Content-Type: application/sdp  
Content-Length: 193  
  
v=0  
o=- 2890844526 2890842807 IN IP4 130.83.139.118  
s=Communication Networks Lecture 3  
m=audio 0 RTP/AVP 0  
a=control:rtsp://lute/cn/3/audio.en  
m=video 0 RTP/AVP 31  
a=control:rtsp://lute/cn/3/video.en
  
- (3) C->S: SETUP rtsp://lute/cn/3/audio.en RTSP/1.0  
CSeq: 2  
Transport: RTP/AVP/UDP;unicast;client\_port=3056-3057
  
- (4) S->C: RTSP/1.0 200 OK  
CSeq: 2  
Session: 12345678  
Transport: RTP/AVP/UDP;unicast;client\_port=3056-3057;  
server\_port=5000-5001
  
- (5) C->S: SETUP rtsp://lute/cn/3/video.en RTSP/1.0  
CSeq: 3  
Transport: RTP/AVP/UDP;unicast;client\_port=3058-3059
  
- (6) S->C: RTSP/1.0 200 OK  
CSeq: 3  
Session: 23456789  
Transport: RTP/AVP/UDP;unicast;client\_port=3058-3059;  
server\_port=5002-5003
  
- (7) C->S: PLAY rtsp://lute/cn/3/audio.en RTSP/1.0  
CSeq: 4  
Session: 12345678

(8) S->C: RTSP/1.0 200 OK  
CSeq: 4  
Session: 12345678

(9) C->S: PLAY rtsp://lute/cn/3/video.en RTSP/1.0  
CSeq: 5  
Session: 23456789

(10) S->C: RTSP/1.0 200 OK  
CSeq: 5  
Session: 23456789

...

(11) S->C: ANNOUNCE rtsp://lute/cn/3 RTSP/1.0 200 OK  
CSeq: 6  
Content-Type: application/sdp  
Content-Length: 193

v=0  
o=- 2890844526 2890842807 IN IP4 130.83.139.118  
s=Communication Networks Lecture 3  
m=audio 0 RTP/AVP 1  
a=control:rtsp://lute/cn/3/audio.en  
m=video 0 RTP/AVP 31  
a=control:rtsp://lute/cn/3/video.en

(12) C->S: RTSP/1.0 200 OK  
CSeq: 6

...

(13) C->S: TEARDOWN rtsp://lute/cn/3/audio.en RTSP/1.0  
CSeq: 7  
Session: 12345678

(14) S->C: RTSP/1.0 200 OK  
CSeq: 7

(15) C->S: TEARDOWN rtsp://lute/cn/3/video.en RTSP/1.0  
CSeq: 8  
Session: 23456789

(16) S->C: RTSP/1.0 200 OK  
CSeq: 8

# Appendix B

## Streaming and recording RTSP scenario with the proxy extensions

The following RTSP protocol scenario illustrates streaming of an MPEG-1 file and corresponds to the RTSP session illustrated in Fig. 9. C, P and S are abbreviations for client, proxy and origin server respectively.

*Assumption:* The client knows the URI of the requested presentation at the beginning of the scenario. The client might have got the information through from a website retrieved through an HTTP connection.

- (1) C->P: DESCRIBE rtsp://lute/movies/mi2 RTSP/1.0  
CSeq: 1
- (2) P->S: DESCRIBE rtsp://lute/movies/mi2 RTSP/1.0  
CSeq: 1
- (3) S->P: RTSP/1.0 200 OK  
CSeq: 1  
Content-Type: application/sdp  
Content-Length: 162  
  
v=0  
o=- 2890844526 2890842807 IN IP4 130.83.139.118  
s=Mission Impossible 2  
m=video 0 RTP/AVP 97  
a=control:rtsp://lute/movies/mi2.en  
a=rtpmap:97 MP1S  
a=fmtp:97 rate=1500
- (4) P->C: RTSP/1.0 200 OK  
CSeq: 1  
Content-Type: application/sdp  
Content-Length: 162  
  
v=0  
o=- 2890844526 2890842807 IN IP4 130.83.139.118  
s=Mission Impossible 2  
m=video 0 RTP/AVP 97  
a=control:rtsp://lute/movies/mi2.en  
a=rtpmap:97 MP1S  
a=fmtp:97 rate=1500
- (5) C->P: SETUP rtsp://lute/movies/mi2.en RTSP/1.0  
CSeq: 2  
Transport: RTP/AVP/UDP;multicast
- (6) P->S: DESCRIBE rtsp://lute/movies/mi2 RTSP/1.0  
CSeq: 2
- (7) S->P: RTSP/1.0 200 OK

```

CSeq: 2
Content-Type: application/sdp
Content-Length: 162

v=0
o=- 2890844526 2890842807 IN IP4 130.83.139.118
s=Mission Impossible 2
m=video 0 RTP/AVP 97
a=control:rtsp://lute/movies/mi2.en
a=rtpmap:97 MP1S
a=fmtp:97 rate=1500

(8) P->S: SETUP rtsp://lute/movies/mi2.en RTSP/1.0
      CSeq: 3
      Transport: RTP/AVP/UDP;multicast

(9) S->C: RTSP/1.0 200 OK
      CSeq: 3
      Transport: RTP/AVP/UDP;multicast;destination=224.2.0.1;
                port=3456-3457;ttl=16
      Session: 1

(10) P->C: RTSP/1.0 200 OK
      CSeq: 2
      Transport: RTP/AVP/UDP;multicast;destination=224.2.0.1;
                port=3456-3457;ttl=16
      Session: 1

(11) P->S: SETUP rtsp://lute/movies/mi2.en RTSP/1.0
      CSeq: 4
      Transport: RTP/AVP/UDP;multicast;proxy_caching

(12) S->P: RTSP/1.0 200 OK
      CSeq: 4
      Transport: RTP/AVP/UDP;multicast;destination=224.2.0.1;
                port=3456-3457;ttl=16
      Session: 2

(13) C->P: PLAY rtsp://lute/movies/mi2.en RTSP/1.0
      CSeq: 3
      Session: 1

(14) P->S: PLAY rtsp://lute/movies/mi2.en RTSP/1.0
      CSeq: 5
      Session: 1

(15) S->P: RTSP/1.0 200 OK
      CSeq: 5
      Session: 1

(16) P->C: RTSP/1.0 200 OK
      CSeq: 3
      Session: 1

(17) P->S: PLAY rtsp://lute/movies/mi2.en RTSP/1.0
      CSeq: 6
      Session: 2

```

```

(18) S->P: RTSP/1.0 200 OK
        CSeq: 6
        Session: 2

(19) P->S: SET_PARAMETER rtsp://lute/movies/mi2.en RTSP/1.0
        CSeq: 7
        Content-length: 22
        Content-type: text/parameters

        SessionID-SSRC: 1=4567

(20) S->P: RTSP/1.0 200 OK
        CSeq: 7

(21) P->S: SET_PARAMETER rtsp://lute/movies/mi2.en RTSP/1.0
        CSeq: 8
        Content-length: 22
        Content-type: text/parameters

        SessionID-SSRC: 2=5678

(22) S->P: RTSP/1.0 200 OK
        CSeq: 8
.....

(23) C->P: TEARDOWN rtsp://lute/movies/mi2.en RTSP/1.0
        CSeq: 4
        Session: 1

(24) P->S: TEARDOWN rtsp://lute/movies/mi2.en RTSP/1.0
        CSeq: 9
        Session: 1

(25) S->P: RTSP/1.0 200 OK
        CSeq: 9

(26) P->C: RTSP/1.0 200 OK
        CSeq: 4
...

(27) P->S: TEARDOWN rtsp://lute/movies/mi2.en RTSP/1.0
        CSeq: 10
        Session: 2

(28) S->P: RTSP/1.0 200 OK
        CSeq: 10

```

The scenario belongs to two streaming sessions from an origin server to a multicast address. In session 1, a client receives a stream from the multicast address. The following signals belong to session 1: (1) - (10), (13) - (16), (23) - (26). In session 2, the proxy records packets from the multicast address to local cache. The following signals belong to the session 2: (11) - (12), (17) - (22), (27) - (28).

**Session 1:** In the beginning the proxy forwards a DESCRIBE request and a response between a client and an origin server. The proxy does not cache the response. In (5) the client requests a setup of a session, proxy requests a session description from the origin server (because it does not cache session descriptions in general) and searches the corresponding entry in its cache. The proxy does not find a corresponding entry in the cache and forwards the setup message to the origin server and requests a port of a server to build a TCP connection for RTCP communication. The origin server answers positive and the proxy forwards the positive response to the client. The proxy decides to record the session into a cache and sends a setup request to the origin server (see Session 2). The streaming follows. As soon as the datapath of the proxy recognizes the SSRC of the receiver within the first receiver report, the proxy informs the origin server on the association between the SessionID and SSRC (19). In (23) the client asks to end the streaming and the session is torn down.

**Session 2:** In (11), the proxy decides to record the session into a proxy and sends a setup request to the origin server (see Session 2) and the streaming and recording follows. The proxy informs the origin server on the association between the SessionID and SSRC for the recording session (21). In (27) the proxy has completed loss collections and asks to end the streaming and the session is torn down.

# Appendix C

## Streaming and reflection RTSP scenario with the proxy extensions

The following RTSP protocol scenario illustrates streaming of an MPEG-1 file and corresponds to the RTSP session illustrated in Fig. 10. C, P and S are abbreviations for client, proxy and origin server respectively.

*Assumption:* The client knows the URI of the requested presentation at the beginning of the scenario. The client might have got the information through from a website retrieved through an HTTP connection.

- (1) C->P: DESCRIBE rtsp://lute/movies/mi2 RTSP/1.0  
CSeq: 1
- (2) P->S: DESCRIBE rtsp://lute/movies/mi2 RTSP/1.0  
CSeq: 1
- (3) S->P: RTSP/1.0 200 OK  
CSeq: 1  
Content-Type: application/sdp  
Content-Length: 162  
  
v=0  
o=- 2890844526 2890842807 IN IP4 130.83.139.118  
s=Mission Impossible 2  
m=video 0 RTP/AVP 97  
a=control:rtsp://lute/movies/mi2.en  
a=rtpmap:97 MP1S  
a=fmtp:97 rate=1500
- (4) P->C: RTSP/1.0 200 OK  
CSeq: 1  
Content-Type: application/sdp  
Content-Length: 162  
  
v=0  
o=- 2890844526 2890842807 IN IP4 130.83.139.118  
s=Mission Impossible 2  
m=video 0 RTP/AVP 97  
a=control:rtsp://lute/movies/mi2.en  
a=rtpmap:97 MP1S  
a=fmtp:97 rate=1500
- (5) C->P: SETUP rtsp://lute/movies/mi2.en RTSP/1.0  
CSeq: 2  
Transport: RTP/AVP/UDP;unicast;client\_port=3056-3057
- (6) P->S: DESCRIBE rtsp://lute/movies/mi2 RTSP/1.0  
CSeq: 2
- (7) S->P: RTSP/1.0 200 OK  
CSeq: 2

Content-Type: application/sdp  
Content-Length: 162

v=0  
o=- 2890844526 2890842807 IN IP4 130.83.139.118  
s=Mission Impossible 2  
m=video 0 RTP/AVP 97  
a=control:rtsp://lute/movies/mi2.en  
a=rtpmap:97 MP1S  
a=fmtp:97 rate=1500

- (8) P->S: SETUP rtsp://lute/movies/mi2.en RTSP/1.0  
CSeq: 3  
Transport: RTP/AVP;proxy\_streaming;proxy\_RTCP=TCP
- (9) S->P: RTSP/1.0 200 OK  
CSeq: 3  
Transport: RTP/AVP;proxy\_streaming;proxy\_RTCP=TCP/7071  
Session: 1
- (10) P->C: RTSP/1.0 200 OK  
CSeq: 2  
Transport: RTP/AVP/UDP;unicast;client\_port=3056-3057;  
server\_port=5000-5001  
Session: 1
- (11) C->P: PLAY rtsp://lute/movies/mi2.en RTSP/1.0  
CSeq: 3  
Session: 1
- (12) P->S: PLAY rtsp://lute/movies/mi2.en RTSP/1.0  
CSeq: 4  
Session: 1
- (13) S->P: RTSP/1.0 200 OK  
CSeq: 4
- (14) P->C: RTSP/1.0 200 OK  
CSeq: 3  
...
- (15) P->S: SET\_PARAMETER rtsp://lute/movies/mi2.en RTSP/1.0  
CSeq: 5  
Content-length: 22  
Content-type: text/parameters  
  
SessionID-SSRC: 1=4567
- (16) S->P: RTSP/1.0 200 OK  
CSeq: 5  
...
- (17) C->P: DESCRIBE rtsp://marimba/movies/sc RTSP/1.0  
CSeq: 1
- (18) P->S: DESCRIBE rtsp://marimba/movies/sc RTSP/1.0  
CSeq: 1

(19) S->P: RTSP/1.0 200 OK  
 CSeq: 1  
 Content-Type: application/sdp  
 Content-Length: 157

v=0  
 o=- 2890844526 2890842807 IN IP4 130.83.139.118  
 s=Space Cowboys  
 m=video 0 RTP/AVP 97  
 a=control:rtsp://marimba/movies/sc.en  
 a=rtpmap:97 MP1S  
 a=fmtp:97 rate=1500

(20) S->P: RTSP/1.0 200 OK  
 CSeq: 1  
 Content-Type: application/sdp  
 Content-Length: 157

v=0  
 o=- 2890844526 2890842807 IN IP4 130.83.139.118  
 s=Space Cowboys  
 m=video 0 RTP/AVP 97  
 a=control:rtsp://marimba/movies/sc.en  
 a=rtpmap:97 MP1S  
 a=fmtp:97 rate=1500

(21) C->P: SETUP rtsp://marimba/movies/sc.en RTSP/1.0  
 CSeq: 2  
 Transport: RTP/AVP/UDP;unicast;client\_port=3058-3059

(22) P->S: DESCRIBE rtsp://marimba/movies/sc RTSP/1.0  
 CSeq: 2

(23) S->P: RTSP/1.0 200 OK  
 CSeq: 2  
 Content-Type: application/sdp  
 Content-Length: 157

v=0  
 o=- 2890844526 2890842807 IN IP4 130.83.139.118  
 s=Space Cowboys  
 m=video 0 RTP/AVP 97  
 a=control:rtsp://marimba/movies/sc.en  
 a=rtpmap:97 MP1S  
 a=fmtp:97 rate=1500

(24) P->S: SETUP rtsp://marimba/movies/sc.en RTSP/1.0  
 CSeq: 3  
 Transport: RTP/AVP;unicast;client\_port=3060;proxy\_RTCP=TCP

(25) S->P: RTSP/1.0 200 OK  
 CSeq: 3  
 Transport: RTP/AVP/UDP;unicast;client\_port=3060;  
           proxy\_RTCP=TCP/7071;server\_port=5000  
 Session: 1

(26) P->C: RTSP/1.0 200 OK  
 CSeq: 2

```

Transport: RTP/AVP/UDP;unicast;client_port=3058-3059;
server_port=3062-3063
Session: 2

(27) C->P: PLAY rtsp://marimba/movies/sc.en RTSP/1.0
CSeq: 3
Session: 2

(28) P->S: PLAY rtsp://marimba/movies/sc.en RTSP/1.0
CSeq: 4
Session: 1

(29) S->P: RTSP/1.0 200 OK
CSeq: 4

(30) P->C: RTSP/1.0 200 OK
CSeq: 3
...

(31) P->S: SET_PARAMETER rtsp://marimba/movies/sc.en RTSP/1.0
CSeq: 5
Content-length: 22
Content-type: text/parameters

SessionID-SSRC: 1=4567

(32) S->P: RTSP/1.0 200 OK
CSeq: 5
....

(33) C->P: TEARDOWN rtsp://lute/movies/mi2.en RTSP/1.0
CSeq: 4
Session: 1

(34) P->S: TEARDOWN rtsp://lute/movies/mi2.en RTSP/1.0
CSeq: 6
Session: 1

(35) S->P: RTSP/1.0 200 OK
CSeq: 6

(36) P->C: RTSP/1.0 200 OK
CSeq: 4
...

(37) C->S: TEARDOWN rtsp://marimba/movies/sc.en RTSP/1.0
CSeq: 4
Session: 2

(38) P->S: TEARDOWN rtsp://marimba/movies/sc.en RTSP/1.0
CSeq: 6
Session: 1

(39) S->P: RTSP/1.0 200 OK
CSeq: 6

(40) P->S: RTSP/1.0 200 OK
CSeq: 4

```

The scenario belongs to two streaming sessions. The following signals belong to session 1, where a proxy streams from a cache: (1) - (16), (33) - (36). The following signals belong to the session 2, where a proxy receives a stream from an origin server and reflects it to a client: (17) - (32), (37) - (40).

**Session 1:** In the beginning the proxy forwards a DESCRIBE request and a response between a client and an origin server. The proxy does not cache the response. In (5) the client requests a setup of a session, proxy requests a session description from the origin server (because it does not cache session descriptions in general) and searches the corresponding entry in its cache. The proxy finds an entry in the cache and forwards the setup message with corresponding transport parameter (proxy\_streaming) to the origin server and requests a port of a server to build a TCP connection for RTCP communication. The origin server answers positive and the streaming begins. As soon as the datapath of the proxy recognizes the SSRC of the receiver within the first receiver report, the proxy informs the origin server on the association between the SessionID and SSRC (15). In (33) the client asks to end the streaming and the session is torn down.

**Session 2:** In the beginning the proxy forwards a DESCRIBE request and a response between a client and an origin server. The proxy does not cache the response. In (21) the client requests a setup of a session, proxy requests a session description from the origin server (because it does not cache session descriptions in general) and searches the corresponding entry in its cache. The proxy does not find an entry in the cache and forwards the setup message to the origin server and requests a port of a server to build a TCP connection for RTCP communication. The origin server answers positive, the proxy recognizes a conflict in the SessionID allocations and solves the problem and the reflection begins. As soon as the datapath of the proxy recognizes the SSRC of the receiver within the first receiver report, the proxy informs the origin server on the association between the SessionID and SSRC (31). In (37) the client asks to end the streaming and the session is torn down.

# Appendix D

## Streaming and recording RTSP scenario with the formatswitching extensions

The following RTSP protocol scenario illustrates streaming of an MPEG-1 file and corresponds to the RTSP session illustrated in Fig. 9. C, P and S are abbreviations for client, proxy and origin server respectively. The scenario is also illustrated in Fig. 16.

*Assumption:* The client knows the URI of the requested presentation at the beginning of the scenario. The client might have got the information through from a website retrieved through an HTTP connection.

- (1) C->P: DESCRIBE rtsp://lute/movies/mi2 RTSP/1.0  
CSeq: 1
- (2) P->S: DESCRIBE rtsp://lute/movies/mi2 RTSP/1.0  
CSeq: 1
- (3) S->P: RTSP/1.0 200 OK  
CSeq: 1  
Content-Type: application/sdp  
Content-Length: 237  
  
v=0  
o=- 2890844526 2890842807 IN IP4 130.83.139.118  
s=Mission Impossible 2  
m=video 0 RTP/AVP 97 98 99  
a=control:rtsp://lute/movies/mi2.en  
a=rtpmap:97 MP1S  
a=fmtp:97 rate=800  
a=rtpmap:98 MP1S  
a=fmtp:98 rate=1300  
a=rtpmap:99 MP1S  
a=fmtp:99 rate=1500
- (4) P->C: RTSP/1.0 200 OK  
CSeq: 1  
Content-Type: application/sdp  
Content-Length: 237  
  
v=0  
o=- 2890844526 2890842807 IN IP4 130.83.139.118  
s=Mission Impossible 2  
m=video 0 RTP/AVP 97 98 99  
a=control:rtsp://lute/movies/mi2.en  
a=rtpmap:97 MP1S  
a=fmtp:97 rate=800  
a=rtpmap:98 MP1S  
a=fmtp:98 rate=1300  
a=rtpmap:99 MP1S  
a=fmtp:99 rate=1500
- (5) C->P: SETUP rtsp://lute/movies/mi2.en RTSP/1.0

```

        CSeq: 2
        Transport: RTP/AVP/UDP;multicast

(6) P->S: DESCRIBE rtsp://lute/movies/mi2 RTSP/1.0
        CSeq: 2

(7) S->P: RTSP/1.0 200 OK
        CSeq: 2
        Content-Type: application/sdp
        Content-Length: 237

        v=0
        o=- 2890844526 2890842807 IN IP4 130.83.139.118
        s=Mission Impossible 2
        m=video 0 RTP/AVP 97 98 99
        a=control:rtsp://lute/movies/mi2.en
        a=rtpmap:97 MP1S
        a=fmtp:97 rate=800
        a=rtpmap:98 MP1S
        a=fmtp:98 rate=1300
        a=rtpmap:99 MP1S
        a=fmtp:99 rate=1500

(8) P->S: SETUP rtsp://lute/movies/mi2.en RTSP/1.0
        CSeq: 3
        Transport: RTP/AVP/UDP;multicast

(9) S->P: RTSP/1.0 200 OK
        CSeq: 3
        Transport: RTP/AVP/UDP;multicast;destination=224.2.0.1;
                port=3456-3457;ttl=16
        Session: 1

(10) P->C: RTSP/1.0 200 OK
        CSeq: 2
        Transport: RTP/AVP/UDP;multicast;destination=224.2.0.1;
                port=3456-3457;ttl=16
        Session: 1

(11) P->S: SETUP rtsp://lute/movies/mi2.en RTSP/1.0
        CSeq: 4
        Transport: RTP/AVP/UDP;multicast;
                proxy_caching;proxy_formatswitching

(12) S->P: RTSP/1.0 200 OK
        CSeq: 4
        Transport: RTP/AVP/UDP;multicast;destination=224.2.0.1;
                port=3456-3457;ttl=16;
                proxy_formatswitching
        Session: 2

(13) S->P: SET_PARAMETER rtsp://lute/movies/mi2.en RTSP/1.0
        CSeq: 5
        Session: 2
        Content-length: 294
        Content-type: application/sdp

        v=0

```

```

o=- 2890844526 2890842807 IN IP4 130.83.139.118
s=Mission Impossible 2
m=video 0 RTP/AVP 97 98 99
a=control:rtsp://lute/movies/mi2.en
a=rtpmap:97 MP1S
a=fmtp:97 rate=800
a=fmtp:97 ssrc=1234
a=rtpmap:98 MP1S
a=fmtp:98 rate=1300
a=fmtp:98 ssrc=2345
a=rtpmap:99 MP1S
a=fmtp:99 rate=1500
a=fmtp:99 ssrc=3456

(14) P->S: RTSP/1.0 200 OK
      CSeq: 5

(15) C->P: PLAY rtsp://lute/movies/mi2.en RTSP/1.0
      CSeq: 3
      Session: 1

(16) P->S: PLAY rtsp://lute/movies/mi2.en RTSP/1.0
      CSeq: 6
      Session: 1

(17) S->P: RTSP/1.0 200 OK
      CSeq: 6

(18) P->C: RTSP/1.0 200 OK
      CSeq: 3

(19) P->S: PLAY rtsp://lute/movies/mi2.en RTSP/1.0
      CSeq: 7
      Session: 2

(20) S->P: RTSP/1.0 200 OK
      CSeq: 7
...

(21) P->S: SET_PARAMETER rtsp://lute/movies/mi2.en RTSP/1.0
      CSeq: 8
      Content-length: 22
      Content-type: text/parameters

      SessionID-SSRC: 1=4567

(22) S->P: RTSP/1.0 200 OK
      CSeq: 8
...

(23) P->S: SET_PARAMETER rtsp://lute/movies/mi2.en RTSP/1.0
      CSeq: 9
      Content-length: 22
      Content-type: text/parameters

      SessionID-SSRC: 2=5678

(24) S->P: RTSP/1.0 200 OK

```

```

CSeq: 9
....
(25) S->P: ANNOUNCE rtsp://lute/movies/mi2.en RTSP/1.0
CSeq: 10
Session: 1
Content-Type: application/sdp
Content-Length: 168

v=0
o=- 2890844526 2890842807 IN IP4 130.83.139.118
s=Mission Impossible 2
m=video 0 RTP/AVP 98
a=control:rtsp://lute/movies/mi2.en
a=rtpmap:98 MP1S
a=fmtp:98 rate=1300

(26) P->C: ANNOUNCE rtsp://lute/movies/mi2.en RTSP/1.0
CSeq: 4
Session: 1
Content-Type: application/sdp
Content-Length: 168

v=0
o=- 2890844526 2890842807 IN IP4 130.83.139.118
s=Mission Impossible 2
m=video 0 RTP/AVP 98
a=control:rtsp://lute/movies/mi2.en
a=rtpmap:98 MP1S
a=fmtp:98 rate=1300

(27) C->P: RTSP/1.0 200 OK
CSeq: 4

(28) P->S: RTSP/1.0 200 OK
CSeq: 10

(29) S->P: ANNOUNCE rtsp://lute/movies/mi2.en RTSP/1.0
CSeq: 11
Session: 2
Content-Type: application/sdp
Content-Length: 168

v=0
o=- 2890844526 2890842807 IN IP4 130.83.139.118
s=Mission Impossible 2
m=video 0 RTP/AVP 98
a=control:rtsp://lute/movies/mi2.en
a=rtpmap:98 MP1S
a=fmtp:98 rate=1300

(30) P->S: RTSP/1.0 200 OK
CSeq: 11
....
(31) C->P: TEARDOWN rtsp://lute/movies/mi2.en RTSP/1.0
CSeq: 5
Session: 1

```

```
(32) P->S: TEARDOWN rtsp://lute/movies/mi2.en RTSP/1.0
      CSeq: 12
      Session: 1

(33) S->P: RTSP/1.0 200 OK
      CSeq: 12

(34) P->C: RTSP/1.0 200 OK
      CSeq: 5
...

(35) P->S: TEARDOWN rtsp://lute/movies/mi2.en RTSP/1.0
      CSeq: 13
      Session: 2

(36) S->P: RTSP/1.0 200 OK
      CSeq: 13
```

The scenario belongs to two streaming sessions from an origin server to a multicast address. In session 1, a client receives a stream from the multicast address. The following signals belong to session 1: (1) - (10), (15) - (18), (31) - (34). In session 2, the proxy records packets from the multicast address to local cache. The following signals belong to the session 2: (11) - (14), (19) - (30), (35) - (36).

**Session 1:** In the beginning, the proxy forwards a DESCRIBE request and a response between a client and an origin server. The proxy does not cache the response. The response identifies all formats of the media content that are available at the origin server. In (5) the client requests a setup of a session, proxy requests a session description from the origin server (because it does not cache session descriptions in general) and searches the corresponding entry in its cache. The proxy does not find a corresponding entry in the cache and forwards the setup message to the origin server and requests a port of a server to build a TCP connection for RTCP communication. The origin server answers positive and the proxy forwards the positive response to the client. The proxy decides to record the session into a cache and sends a setup request to the origin server (see Session 2). The streaming follows. As soon as the datapath of the proxy recognizes the SSRC of the receiver within the first receiver report, the proxy informs the origin server on the association between the SessionID and SSRC (21). In (31) the client asks to end the streaming and the session is torn down.

**Session 2:** In (11), the proxy decides to record the session into a proxy and sends a setup request to the origin server (see Session 2) and the streaming and recording follows. The origin server informs the proxy on the SSRC allocation (13). The proxy informs the origin server on the association between the SessionID and SSRC for the recording session (23). In (35) the proxy has completed the loss collection phase and makes a request to end the streaming and the session is torn down.

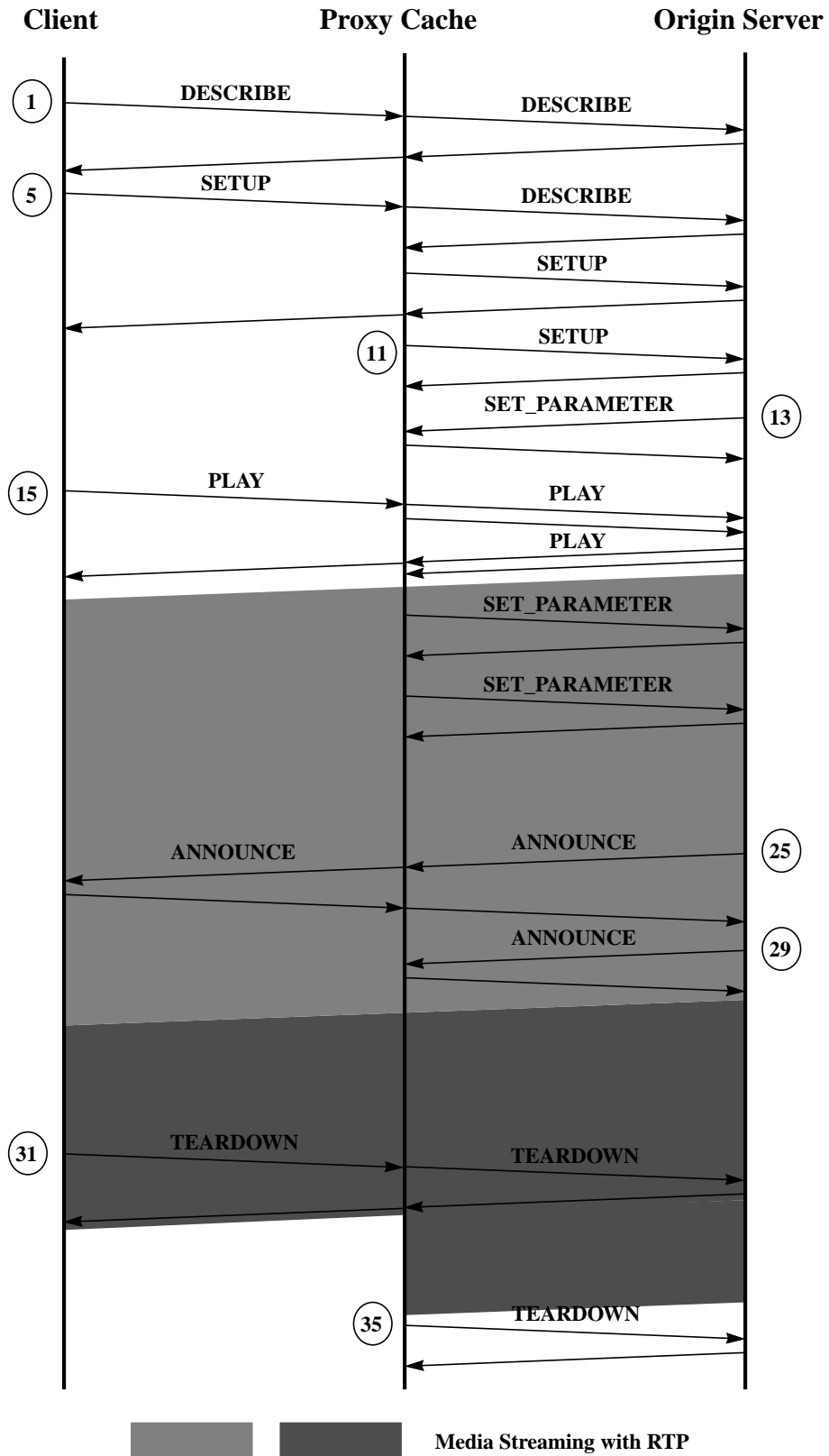


Fig. 16: RTSP scenario for streaming from origin server and recording

# Appendix E

## Streaming and reflection RTSP scenario with the formatswitching extensions

The following RTSP protocol scenario illustrates streaming of an MPEG-1 file and corresponds to the RTSP session illustrated in Fig. 10. C, P and S are abbreviations for client, proxy and origin server respectively. The RTSP protocol scenario for session 1 is illustrated in Fig. 17.

*Assumption:* The client knows the URI of the requested presentation at the beginning of the scenario. The client might have got the information through from a website retrieved through an HTTP connection.

```
(1) C->P: DESCRIBE rtsp://lute/movies/mi2 RTSP/1.0
      CSeq: 1

(2) P->S: DESCRIBE rtsp://lute/movies/mi2 RTSP/1.0
      CSeq: 1

(3) S->P: RTSP/1.0 200 OK
      CSeq: 1
      Content-Type: application/sdp
      Content-Length: 237

      v=0
      o=- 2890844526 2890842807 IN IP4 130.83.139.118
      s=Mission Impossible 2
      m=video 0 RTP/AVP 97 98 99
      a=control:rtsp://lute/movies/mi2.en
      a=rtpmap:97 MP1S
      a=fmtp:97 rate=800
      a=rtpmap:98 MP1S
      a=fmtp:98 rate=1300
      a=rtpmap:99 MP1S
      a=fmtp:99 rate=1500

(4) P->C: RTSP/1.0 200 OK
      CSeq: 1
      Content-Type: application/sdp
      Content-Length: 237

      v=0
      o=- 2890844526 2890842807 IN IP4 130.83.139.118
      s=Mission Impossible 2
      m=video 0 RTP/AVP 97 98 99
      a=control:rtsp://lute/movies/mi2.en
      a=rtpmap:97 MP1S
      a=fmtp:97 rate=800
      a=rtpmap:98 MP1S
      a=fmtp:98 rate=1300
      a=rtpmap:99 MP1S
      a=fmtp:99 rate=1500
```

(5) C->P: SETUP rtsp://lute/movies/mi2.en RTSP/1.0  
 CSeq: 2  
 Transport: RTP/AVP/UDP;unicast;client\_port=3056-3057

(6) P->S: DESCRIBE rtsp://lute/movies/mi2 RTSP/1.0  
 CSeq: 2

(7) S->P: RTSP/1.0 200 OK  
 CSeq: 2  
 Content-Type: application/sdp  
 Content-Length: 237

v=0  
 o=- 2890844526 2890842807 IN IP4 130.83.139.118  
 s=Mission Impossible 2  
 m=video 0 RTP/AVP 97 98 99  
 a=control:rtsp://lute/movies/mi2.en  
 a=rtpmap:97 MP1S  
 a=fmtp:97 rate=800  
 a=rtpmap:98 MP1S  
 a=fmtp:98 rate=1300  
 a=rtpmap:99 MP1S  
 a=fmtp:99 rate=1500

(8) P->S: SET\_PARAMETER rtsp://lute/movies/mi2.en RTSP/1.0  
 CSeq: 3  
 Content-length: 237  
 Content-type: application/sdp

v=0  
 o=- 2890844526 2890842807 IN IP4 130.83.139.118  
 s=Mission Impossible 2  
 m=video 0 RTP/AVP 97 98 99  
 a=control:rtsp://lute/movies/mi2.en  
 a=rtpmap:97 MP1S  
 a=fmtp:97 rate=800  
 a=rtpmap:98 MP1S  
 a=fmtp:98 rate=1300  
 a=rtpmap:99 MP1S  
 a=fmtp:99 rate=1500

(9) S->P: RTSP/1.0 200 OK  
 CSeq: 3

(10) P->S: SETUP rtsp://lute/movies/mi2.en RTSP/1.0  
 CSeq: 4  
 Transport: RTP/AVP;proxy\_streaming;  
 proxy\_RTCP=TCP;proxy\_formatswitching

(11) S->P: RTSP/1.0 200 OK  
 CSeq: 4  
 Transport: RTP/AVP;proxy\_streaming;  
 proxy\_RTCP=TCP/7071; proxy\_formatswitching  
 Session: 1

(12) P->C: RTSP/1.0 200 OK  
 CSeq: 2  
 Transport: RTP/AVP/UDP;unicast;client\_port=3056-3057;

```

server_port=5000-5001
Session: 1

(13) P->S: SET_PARAMETER rtsp://lute/movies/mi2.en RTSP/1.0
CSeq: 5
Session: 1
Content-length: 294
Content-type: application/sdp

v=0
o=- 2890844526 2890842807 IN IP4 130.83.139.118
s=Mission Impossible 2
m=video 0 RTP/AVP 97 98 99
a=control:rtsp://lute/movies/mi2.en
a=rtpmap:97 MP1S
a=fmtp:97 rate=800
a=fmtp:97 ssrc=1234
a=rtpmap:98 MP1S
a=fmtp:98 rate=1300
a=fmtp:98 ssrc=2345
a=rtpmap:99 MP1S
a=fmtp:99 rate=1500
a=fmtp:99 ssrc=3456

(14) S->P: RTSP/1.0 200 OK
CSeq: 5

(15) C->P: PLAY rtsp://lute/movies/mi2.en RTSP/1.0
CSeq: 3
Session: 1

(16) P->S: PLAY rtsp://lute/movies/mi2.en RTSP/1.0
CSeq: 6
Session: 1

(17) S->P: RTSP/1.0 200 OK
CSeq: 6

(18) P->C: RTSP/1.0 200 OK
CSeq: 3
...

(19) P->S: SET_PARAMETER rtsp://lute/movies/mi2.en RTSP/1.0
CSeq: 7
Content-length: 22
Content-type: text/parameters

SessionID-SSRC: 1=4567

(20) S->P: RTSP/1.0 200 OK
CSeq: 7
....

(21) S->P: ANNOUNCE rtsp://lute/movies/mi2.en RTSP/1.0
CSeq: 8
Session: 1
Content-Type: application/sdp
Content-Length: 240

```

```

v=0
o=- 2890844526 2890842807 IN IP4 130.83.139.118
s=Mission Impossible 2
m=video 0 RTP/AVP 97 98
a=control:rtsp://lute/movies/mi2.en
a=rtpmap:97 MP1S
a=fmtp:97 rate=800
a=rtpmap:98 MP1S
a=fmtp:98 rate=1300
a=formatswitching 1234 12354678 2345 23456789

(22) P->C: ANNOUNCE rtsp://lute/movies/mi2.en RTSP/1.0
CSeq: 4
Session: 1
Content-Type: application/sdp
Content-Length: 240

v=0
o=- 2890844526 2890842807 IN IP4 130.83.139.118
s=Mission Impossible 2
m=video 0 RTP/AVP 98
a=control:rtsp://lute/movies/mi2.en
a=rtpmap:98 MP1S
a=fmtp:98 rate=1300

(23) C->P: RTSP/1.0 200 OK
CSeq: 4

(24) P->S: RTSP/1.0 200 OK
CSeq: 8
....

(25) C->P: DESCRIBE rtsp://marimba/movies/sc RTSP/1.0
CSeq: 1

(26) P->S: DESCRIBE rtsp://marimba/movies/sc RTSP/1.0
CSeq: 1

(27) S->P: RTSP/1.0 200 OK
CSeq: 1
Content-Type: application/sdp
Content-Length: 232

v=0
o=- 2890844526 2890842807 IN IP4 130.83.139.118
s=Space Cowboys
m=video 0 RTP/AVP 97 98 99
a=control:rtsp://marimba/movies/sc.en
a=rtpmap:97 MP1S
a=fmtp:97 rate=800
a=rtpmap:98 MP1S
a=fmtp:98 rate=1300
a=rtpmap:99 MP1S
a=fmtp:99 rate=1500

(28) P->C: RTSP/1.0 200 OK
CSeq: 1

```

```

Content-Type: application/sdp
Content-Length: 232
v=0
o=- 2890844526 2890842807 IN IP4 130.83.139.118
s=Space Cowboys
m=video 0 RTP/AVP 97 98 99
a=control:rtsp://marimba/movies/sc.en
a=rtpmap:97 MP1S
a=fmtp:97 rate=800
a=rtpmap:98 MP1S
a=fmtp:98 rate=1300
a=rtpmap:99 MP1S
a=fmtp:99 rate=1500

(29) C->P: SETUP rtsp://marimba/movies/sc.en RTSP/1.0
CSeq: 2
Transport: RTP/AVP/UDP;unicast;client_port=3058-3059

(30) P->S: DESCRIBE rtsp://marimba/movies/sc RTSP/1.0
CSeq: 2

(31) S->P: RTSP/1.0 200 OK
CSeq: 2
Content-Type: application/sdp
Content-Length: 232

v=0
o=- 2890844526 2890842807 IN IP4 130.83.139.118
s=Space Cowboys
m=video 0 RTP/AVP 97 98 99
a=control:rtsp://marimba/movies/sc.en
a=rtpmap:97 MP1S
a=fmtp:97 rate=800
a=rtpmap:98 MP1S
a=fmtp:98 rate=1300
a=rtpmap:99 MP1S
a=fmtp:99 rate=1500

(32) P->S: SETUP rtsp://marimba/movies/sc.en RTSP/1.0
CSeq: 3
Transport: RTP/AVP;unicast;client_port=3060;proxy_RTCP=TCP

(33) S->P: RTSP/1.0 200 OK
CSeq: 3
Transport: RTP/AVP/UDP;unicast;client_port=3060;
proxy_RTCP=TCP/7071;server_port=5000
Session: 1

(34) P->C: RTSP/1.0 200 OK
CSeq: 2
Transport: RTP/AVP/UDP;unicast;client_port=3058-3059;
server_port=3062-3063
Session: 2

(35) C->P: PLAY rtsp://marimba/movies/sc.en RTSP/1.0
CSeq: 3
Session: 2

```

(36) P->S: PLAY rtsp://marimba/movies/sc.en RTSP/1.0  
 CSeq: 4  
 Session: 1

(37) S->P: RTSP/1.0 200 OK  
 CSeq: 4

(38) P->C: RTSP/1.0 200 OK  
 CSeq: 3

...

(39) P->S: SET\_PARAMETER rtsp://marimba/movies/sc.en RTSP/1.0  
 CSeq: 5  
 Content-length: 22  
 Content-type: text/parameters

SessionID-SSRC: 1=5678

(40) S->P: RTSP/1.0 200 OK  
 CSeq: 5

...

(41) S->P: ANNOUNCE rtsp://marimba/movies/sc.en RTSP/1.0  
 CSeq: 6  
 Session: 1  
 Content-Type: application/sdp  
 Content-Length: 157

v=0  
 o=- 2890844526 2890842807 IN IP4 130.83.139.118  
 s=Space Cowboys  
 m=video 0 RTP/AVP 98  
 a=control:rtsp://marimba/movies/sc.en  
 a=rtpmap:98 MP1S  
 a=fmtp:98 rate=1300

(42) P->C: ANNOUNCE rtsp://marimba/movies/sc.en RTSP/1.0  
 CSeq: 4  
 Session: 2  
 Content-Type: application/sdp  
 Content-Length: 157

v=0  
 o=- 2890844526 2890842807 IN IP4 130.83.139.118  
 s=Space Cowboys  
 m=video 0 RTP/AVP 98  
 a=control:rtsp://marimba/movies/sc.en  
 a=rtpmap:98 MP1S  
 a=fmtp:98 rate=1300

(43) C->P: RTSP/1.0 200 OK  
 CSeq: 4

(44) P->S: RTSP/1.0 200 OK  
 CSeq: 6

....

(45) C->P: TEARDOWN rtsp://lute/movies/mi2.en RTSP/1.0

```

        CSeq: 5
        Session: 1

(46) P->S: TEARDOWN rtsp://lute/movies/mi2.en RTSP/1.0
        CSeq: 9
        Session: 1

(47) S->P: RTSP/1.0 200 OK
        CSeq: 9

(48) P->C: RTSP/1.0 200 OK
        CSeq: 5
...

(49) C->P: TEARDOWN rtsp://marimba/movies/sc.en RTSP/1.0
        CSeq: 5
        Session: 2

(50) P->S: TEARDOWN rtsp://marimba/movies/sc.en RTSP/1.0
        CSeq: 7
        Session: 1

(51) S->P: RTSP/1.0 200 OK
        CSeq: 7

(52) P->C: RTSP/1.0 200 OK
        CSeq: 5

```

The scenario belongs to two streaming sessions. The following signals belong to session 1, where a proxy streams from a cache: (1) - (24), (45) - (48). The following signals belong to the session 2, where a proxy receives a stream from an origin server and reflects it to a client: (25) - (44), (49) - (52).

**Session 1:** In the beginning the proxy forwards a DESCRIBE request and a response between a client and an origin server. The proxy does not cache the response. The response identifies all formats of the media content that are available at the origin server. In (5) the client requests a setup of a session, proxy requests a session description from the origin server (because it does not cache session descriptions in general) and searches the corresponding entry in its cache. The proxy finds an entry in the cache and sends a description that identifies the formats that are available in the cache to the origin server (8). The proxy then forwards the setup message to the origin server with corresponding transport parameter (proxy\_streaming) to the origin server and requests a port of a server to build a TCP connection for RTCP communication (10). The origin server answers positive and the streaming from the cache begins. The proxy informs the origin server on the allocated SSRCs for each format with a SDP description in (13). As soon as the datapath of the proxy recognizes the SSRC of the receiver within the first receiver report, the proxy informs the origin server on the association between the SessionID and SSRC (19). The origin server does not recognize any problems from the receiver reports received from the client and decides to switch to a format that requires more bandwidth

(21) and the proxy is able to perform the switching in (23). In (45) the client asks to end the streaming and the session is torn down. The protocol scenario is also illustrated in Fig. 17.

**Session 2:** In the beginning the proxy forwards a DESCRIBE request and a response between a client and an origin server. The proxy does not cache the response. The response identifies all formats of the media content that are available at the origin server. In (25) the client requests a setup of a session, proxy requests a session description from the origin server (because it does not cache session descriptions in general) and searches the corresponding entry in its cache. The proxy does not find an entry in the cache and forwards the setup message to the origin server and requests a port of a server to build a TCP connection for RTCP communication. The origin server answers positive, the proxy recognizes a conflict in the SessionID allocations and solves the problem and the reflection begins. As soon as the datapath of the proxy recognizes the SSRC of the receiver within the first receiver report, the proxy informs the origin server on the association between the SessionID and SSRC (39). The origin server does not recognize any problems from the receiver reports received from the client and decides to switch to a format that requires more bandwidth (41). The origin server is able to perform the switching after (44). In (49) the client asks to end the streaming and the session is torn down.

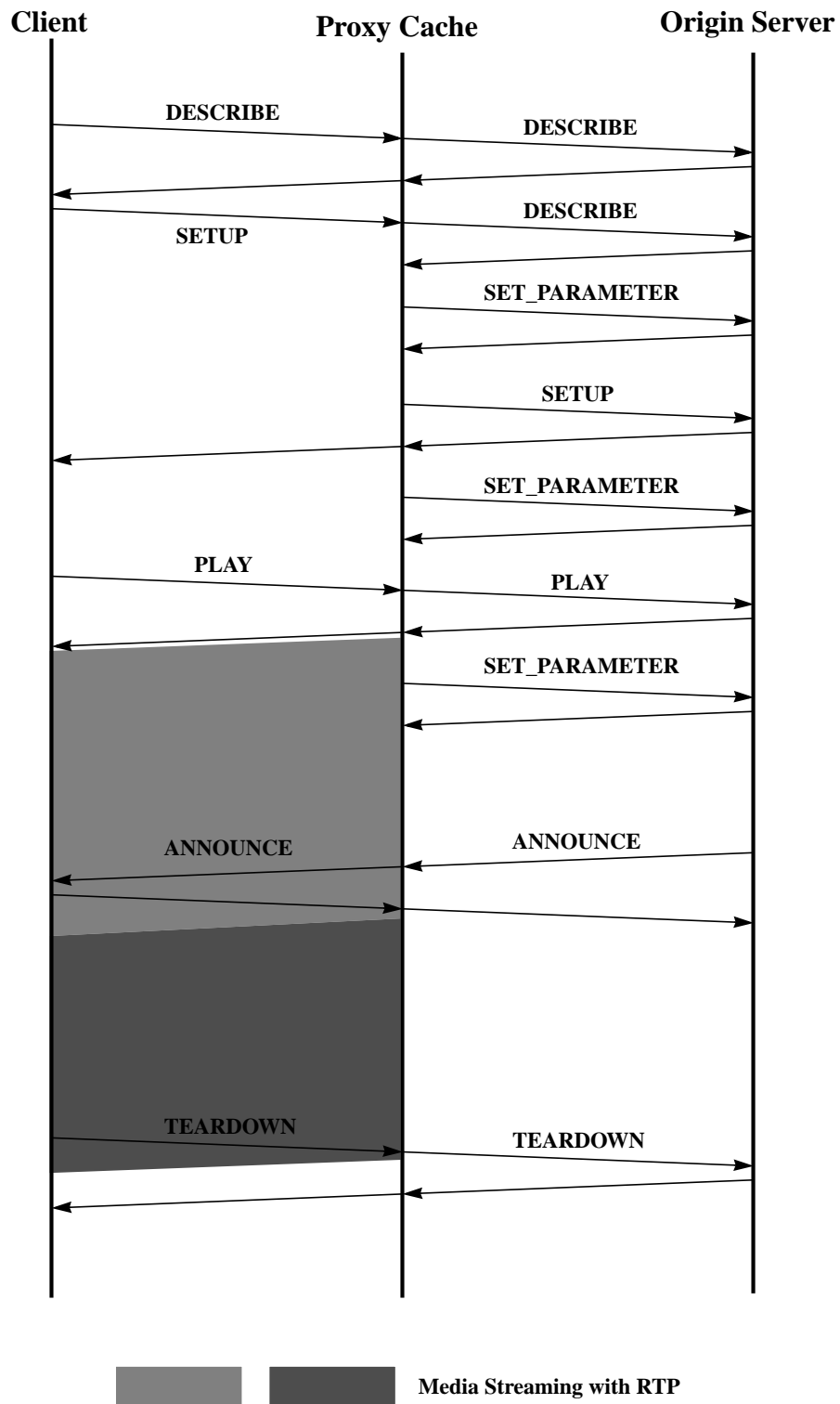


Fig. 17: RTSP scenario for streaming from proxy cache

# References

- [FGK+00] Frederik R., Geagan J., Kellner M. and Periyannan A. "Caching Support in Standards-based RTSP/RTP Servers", Internet draft, IETF, March 2000.
- [GBW97] Carsten Griwodz, Michael Bär, and Lars C. Wolf. "Long-term Movie Popularity in Video-on-Demand Systems". In *Proc. of ACM Multimedia'97*, pages 349-357, November 1997.
- [Gri00] Carsten Griwodz. "Wide-area True Video-on-Demand by a Decentralized Cache-based Distribution Infrastructure", Dissertation, Fachbereich Informatik, Technische Universität Darmstadt, June 2000.
- [GZL+00] Carsten Griwodz, Michael Zink, Michael Liepert, Giwon On, and Ralf Steinmetz. "Multicast for Savings in Cache-based Video Distribution". In *Multimedia Computing and Networking 2000*. SPIE, January 2000.
- [Jon99] Alex Jonas. "Integration von Loss-Collection-RTP (LCRTP) in eine VoD Architektur", Diplomarbeit KOM-D-0093, Technische Universität Darmstadt, November 1999.
- [Küf98] Christine Küfner. "Möglichkeiten für den Einsatz von Lastverteilungsstrategien verteilter Systeme in der Videoverteilung", Studienarbeit KOM-S-0022, Technische Universität Darmstadt, September 1998.
- [RFC793] Postel, J. "Transmission Control Protocol", RFC793, IETF, September 1981.
- [RFC768] Postel, J. "User Datagram Protocol", RFC768, IETF, August 1980.
- [RFC1889] H. Schulzrinne, S. Casner, R. Frederick and V. Jacobson. "RTP: A Transport Protocol for Real-Time Applications", RFC1889, IETF, January 1996.
- [RFC2326] H. Schulzrinne, A. Rao and R. Lanphier. "Real Time Streaming Protocol (RTSP)", RFC2326, IETF, April 1998.
- [RFC2327] M. Handley and V. Jacobson. "SDP: Session Description Protocol", RFC2327, April 1998.
- [Sta98] William Stallings. *High-Speed Networks, TCP/IP and ATM Design Principles*. Prentice Hall, Upper Saddle River, New Jersey, USA, 1998.