KOM Player - A Platform for Experimental VoD Research

Michael Zink[1], Carsten Griwodz[1], Ralf Steinmetz[1,2]

[1]KOM - Industrial Process and System Communications
Darmstadt University of Technology
Merckstrasse 25
64283 Darmstadt, Germany
0049-6151-166151

[2]IPSI, German National Research Center for Information Technology
Dolivostrasse 15
64293 Darmstadt, Germany
0049-6151-869869

{michael.zink,carsten.griwodz,ralf.steinmetz}@kom.tudarmstadt.de

Abstract -- **In contrast to audio which is often streamed as complete music titles or even as a life feed from a radio station, video in today's Internet is almost only available as small clips and pre-generated programs. Although some of the problems concerning AV streaming are reasonably solved right now, some work in fields like wide-area distribution systems need further investigation to make applications like "True Video-on-Demand" work. Our research and the one of many others is focused on problems that have to be solved to make application like VoD work in the Internet. It is mainly concerned with wide area distribution. In this paper we presents a platform for experimental VoD research which is thought to support researchers working on VoD and wide-area distribution for audio and video content. This platform offers researchers the possibility to implement their ideas without building a complete streaming environment and in addition allows the combination of different implementations. After motivating the development of our platform we present the design of our platform, give an overview of the actual implementation and the existing components that we have already built. Finally example scenarios for the use of our platform in research are given.**

Keywords: VoD, Wide Area Distribution, RTSP, RTP

I. Introduction

The use of the Internet for an increasing number of multimedia applications has lead to an increase of the amount of audio and video (AV) traffic in the Internet. The technological basis for this development was laid by improvements in the infrastructure of the "last mile" (ADSL, cable modems).

The most favored technology for the transmission of AV content is the streaming technology which allows a client to start listening or watching content immediately after the request. The strong interest in AV streaming in the Internet caused the availability of many commercial applications. Unfortunately these applications can not compete with the quality of standard TV, especially in comparison to VoD scenarios. Consequently, several researcher are looking for ways to overcome the gap between standard TV and VoD. Recent examples are research in the efficient distribution of AV content, in the adaptation to network conditions including the TCP-friendliness of AV traffic, and in encoding techniques like layered video.

It is a common way to prove new ideas by analytical results and by simulation, but these approaches may hide complexity or shadow incorrect assumptions and are not always sufficient to prove applicability in real-world scenarios. [GBW97] for example has shown that an often used model for user requests in VoD systems does not model real user behavior. [MR98] has expressed doubts about typical web traffic modeling on the basis of anonymized web cache traces. Researchers would profit from implementations of their ideas, either for an evaluation of the implementation complexity, for an identification of the limits of the technique, or for an experimental deployment. Results from operating real-world systems, even on a small scale, provide input for further investigations using analysis and simulation. The KOM player documented in this paper is intended as a step towards such an experimental system for distributed VoD systems.

Many research results in AV streaming have been integrated into commercial products in recent years. Examples are AV en- and decoders (MPEG-x, H.2xx) and transport protocols (RTSP, RTP/RTCP). There is still a lot of research going on, like layered coding or reliable multicast. In order to show the applicability of these new techniques in AV streaming applications, they must be integrated into existing applications or completely new applications must be build.

Our research focus is mainly in AV distribution systems such as [VI96, HCS98, ZGJS00, FJL+97]. Conversations with other researchers in this field have shown that there is a need for a standards-compliant, free and open experimental AV streaming platform. Vendors focus on expanding their key technologies in proprietary building blocks while they keep their systems interoperable in all other blocks, they can rarely provide researchers with entirely open systems. Researchers, on the other hand, may want to share and combine their implementations with research groups in the same or in complementary areas which makes interoperability with standards inevitable.

The hope for interaction led us to redirect some of our time into the development of an AV streaming platform and to make it freely available.

II. Design

In this chapter the major design goals for an AV streaming environment are given. We decided to build our system based on IETF standards in order to achieve interoperability with streaming applications deployed in the Internet. Since free software developments did not include many appropriate building blocks when we started, most of our system is written from scratch. To make it acceptable by other researchers, we apply the following goals:

- Reusable from the technical as well as the legal point of view, modular with well-defined interfaces
- Interoperable with other standard compliant tools
- Integratable with existing code

A. *Reusability*

When trying to build an AV streaming application one realizes that some functionality is needed in different parts of the application. E.g., in the case of a video server and client, protocols like RTP/RTCP, RTSP and SDP are needed in both parts. Therefore it is necessary to implement these protocols in a way that they can be easily integrated into applications. In order to do this a well defined and documented API for each module is needed.

It is necessary to support different decoders, and several video servers that stream diverse encoding formats (e.g. H.263, MPEG-1, QuickTime). It is highly unlikely that the APIs of third party software are identical, leading to an adaptation effort whenever a new library is integrated. A generic wrapper is used to hide this differences from other parts of the system.

While such abstractions are typical for streaming applications, a generic structure like that of the JMF [DeC99] is rarely found. Existing approaches implement either hard-coded sequences, or they consider frameworks that allow the specification of an end-to-end behavior for complex multimedia systems. In the latter kind of systems, functionality is described at the level of cooperating distributed components ([Kae97, EN96]). It is typical for such frameworks to consider networking as a component that is also under the control of the framework. In an environment that ensures interoperability by specifying protocols (such as the RTSP streaming environment), we prefer a local approach. The control of the framework extends only over a single machine and RTSP is used explicitly for communication. The currently implemented components, called stream handlers, work at a granularity similar to the components of the JMF and do not provide an abstraction from the network. The stream handlers are modular media processing units that can be connected dynamically by a controlling entity to form a set of modules, which process data units sequentially. The sequence of data units is called the stream, the modules are the stream handlers.

Due to the interaction of RTP and RTCP, and the possibility of receiving data from several sources at a single port, a directed, non-cyclic graph of stream handlers is an appropriate streaming model. In case of RTP, a stream handler infrastructure seems to be appropriate only if dynamic reconfiguration is supported by the stream handlers as well as the controlling framework. A packet that arrives at an RTP Source Stream Handler (a stream handler that reads UDP packets from a network interface and interprets them as RTP packets) from an unexpected sender must be handled in an application-defined way: it may be appropriate to discard the packet, to assign it to a default path, or to create an additional stream for special processing.

Dynamic reconfiguration must also be supported to handle user interaction with a proxy cache of a VoD system if that cache acts also as a reflector. The client receives data from origin server through the proxy cache, which writes RTP packets from the origin server to disk as well as forwards them to the client. If the client pauses and the application decides to continue the caching operation, the trunk of the stream handler graph that forwards data to the client must be cut, while the trunk that stores data on disk must be maintained. If the client resumes viewing, the application must create a new stream handler graph, which retrieves the data from the cache.

B. *Interoperability*

We decided to support the protocols that are used in most AV streaming applications: RTP/RTCP [SCFJ96], RTSP [SRL98] and SDP [HJ98]. The decision was also due to the fact that these protocols are standardized by the IETF.

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. The companion 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 distinguish the packet streams. Each participant in an RTP session periodically sends an RTCP packet to all other participants in the session.

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. Usually RTSP does not typically deliver the media streams itself but controls streams that are being carried by some transport protocol like RTP.

SDP was originally intended as a complement for the session announcement protocol SAP to communicate conference addresses and tool-specific information over the MBone. Alternatives such as HTML postings or E-mail distribution of session descriptions were taken into account as well. With this primary goal in mind, SDP does not support negotiation of any of session information, but is just used for dissemination.

C. Integration

The design of our system allows the integration of already existing systems and tools. One example is the use of IBM's VideoCharger[1] as video server for our client. In this particular case we use mainly the data pump of the VideoCharger which is controlled by our RTSP implementation.

One major element in an AV streaming environment are audio and video decoders. Given the variety of encoding formats (H.263, MPEG-1,...) that are used by the applications interoperability can only be reached, if these formats are also supported by our architecture. Since we are not able to build en- and decoders for all these formats, and there are already C and C++ libraries that support some formats, we decided to make use of third party en- and decoders fairly simple. So far we have integrated three different libraries: MpegTV, smpeg and mpeglib. All three of them support MPEG-1 and mpeglib supports also MPEG-1 audio layer 3 (MP3).

III. Implementation

The implementation consists of three applications that are sufficient for building an experimental streaming media distribution system: client, server and proxy-cache.

1. http://www-4.ibm.com/software/data/videocharger/

When we started our implementation, there was only one implementation of RTSP available in open source[2]. We found out that this implementation preceded the RFC [SRL98] and was not easily updated and reused. After some unsatisfactory experiences in adapting existing RTP/RTCP implementations for our goals, we decided to integrate our own implementation [GJZ99] into the system.

We checked whether JMF [DeC99] fulfils our needs but MPEG-1 decoders e.g. are only available for Solaris and Windows. RTP was integrated in some open source projects like vic, but a closer look at this implementations showed us that RTP is highly intergrated and was therefore not usable as a starting point for our own implementation. In the MASH project a scalable multimedia architecture for distributed multimedia collaboration in heterogeneous environments [M+97] was developed. Streaming in MASH is realized by the MBone videoconferencing tools (vic, vat,...) and therefore bears the same problems a described for vic.

We are also aware of a stand-alone RTP library [Lie00], but this project started after we decided to implement our own RTP. The “Darwin” project [Dar00] was published by Apple after the start of our work as well. It is concerned exclusively with the server side and supports only the QuickTime file format [Tow99].

A. Overview

The implementation of the KOM-Player platform aims at the development of a research prototype in the area of wide-area distribution systems for streaming media in the Internet. The existing code base considers mainly the distribution of CBR MPEG-1 system streams, which was our initial target format because it combines hardware- and OS-independent playback capability with an appropriate quality. Since this encoding format does not support the scalability of encoding formats that can now and in the conceivable future be deployed in the Internet on a wide scale, more flexible encodings are considered as well in our research. This is reflected in ongoing implementation work.

During the development we realized that the integration of various independently developed data pumps, GUI and decoder modules with the RTSP code results in an abundance of intermediate states (e.g. RTP connection to video server established, decoder not yet initialized). To coordinate the state transitions, we have put an automaton (finite state machine) at the core of the session management (Figure 1). All events are controlled by the automaton to assure

---

[2]. http://www.realnetworks.com/devzone/library/rtsp/index.html

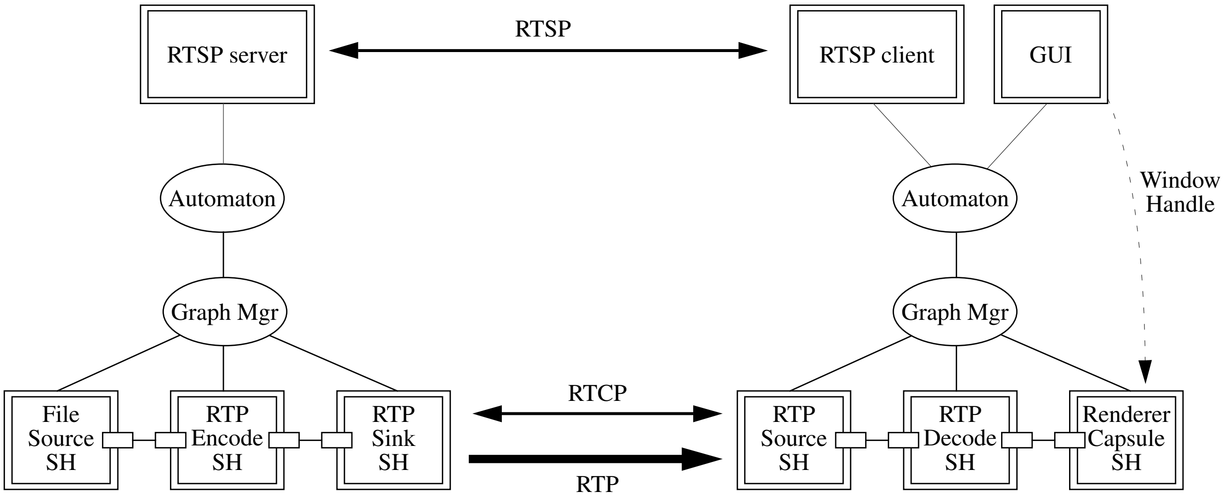

Figure 1: Client-server configuration overview

that only allowed actions will be executed. To enforce completeness of the automaton, we built a tool that generates automata in C++ from an automata language.

B. Server

The server implementation is multi-threaded, where the main purpose of the separation into threads was to achieve a better modularity. It uses only TCP for reliable delivery of RTSP packets [SRL98], RTSP over UDP is not supported. Like most other RTSP implementations, ours does currently not support deferred play requests.

Several data dumps are controlled concurrently by one RTSP server process. Two data pumps are implemented at this time. One relays commands to a low level API of the IBM VideoCharger server, the other is hand-coded and described in this paper. Our data dump uses UDP for the transmission of RTP packets over IP unicast and multicast. It supports MPEG-1 system CBR, MPEG-1 audio layer 3 (CBR, VBR under development) and raw H.261 data files.

An overview of the implementation is illustrated in Figure 2. Although it shows only a part of the implementation it gives an overview of the classes that are involved in the administration of one RTSP session. The implementation is separated into the 3 parts: RTSP session administration, RTSP communication and parsing and RTP streaming.

Each active RTSP session is represented by an RTSPServerSession object. The object inherits KOMServer class which implements the automaton for the server. The state machine is driven by the client's RTSP messages. The automaton controls the RTP data pump and answers the RTSP messages.

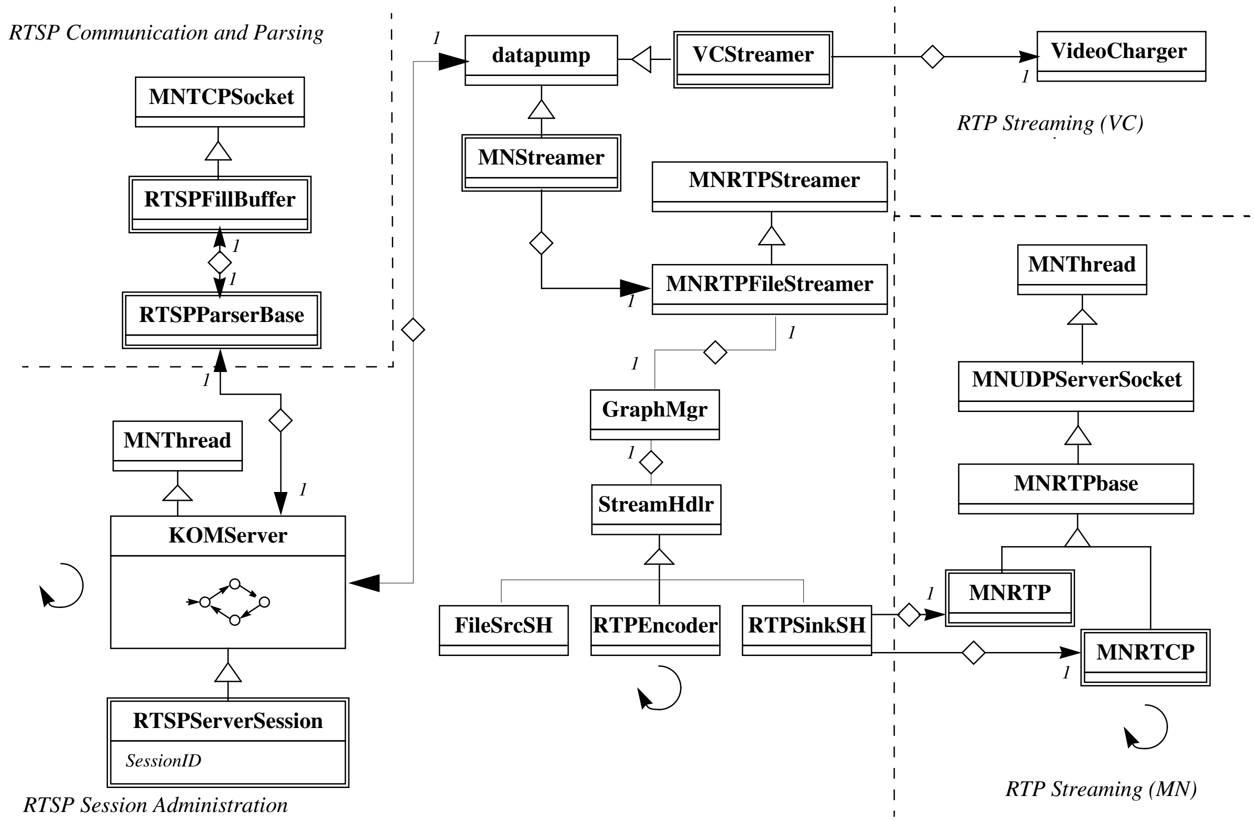

Figure 2: Server

Currently each RTSPServerSession runs in a thread. The RTSPServerSession object is associated with an RTSPFillBuffer object (which is a TCP socket). The RTSPFillBuffer object receives data from a TCP socket and parses the RTSP message. Each message object is delivered to its session for processing. The reception and parsing process is initiated by an object that monitors the RTSP communications.

The RTP streaming part performs the data transfer from a file to the network. It is separated into two threads, RTP and RTCP. It is managed by the RTSP part that initiates the streaming, changes the present streaming position within the stream, halts the streaming temporarily and destroys it. The RTP thread 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. The RTCP thread prepares and emits sender reports to a separate UDP port and receives receiver reports.

C. Client

The client of the KOM-Player platform uses third party C++ tools or libraries for the GUI and the decoder implementations. It is designed to cooperate with other streaming servers beside ours. This required interoperability testing with other RTSP and RTP/RTCP implementations. Taking part in an RTSP interoperability test [Fre00] with other streaming servers and clients helped us to enhance our own implementation. We tested the client with several servers and realized that the video format is currently the limiting factor.

Figure 3 shows the most important classes of the client. Four functional parts can be distinguished: RTSP session administration, RTSP communication, RTP streaming and content decoding. Since some of these parts are identical to the one used by the server, only the parts exclusively used in the client are described here. The comparison of the two class diagrams (Figure 2 and Figure 3) shows the reusable classes.

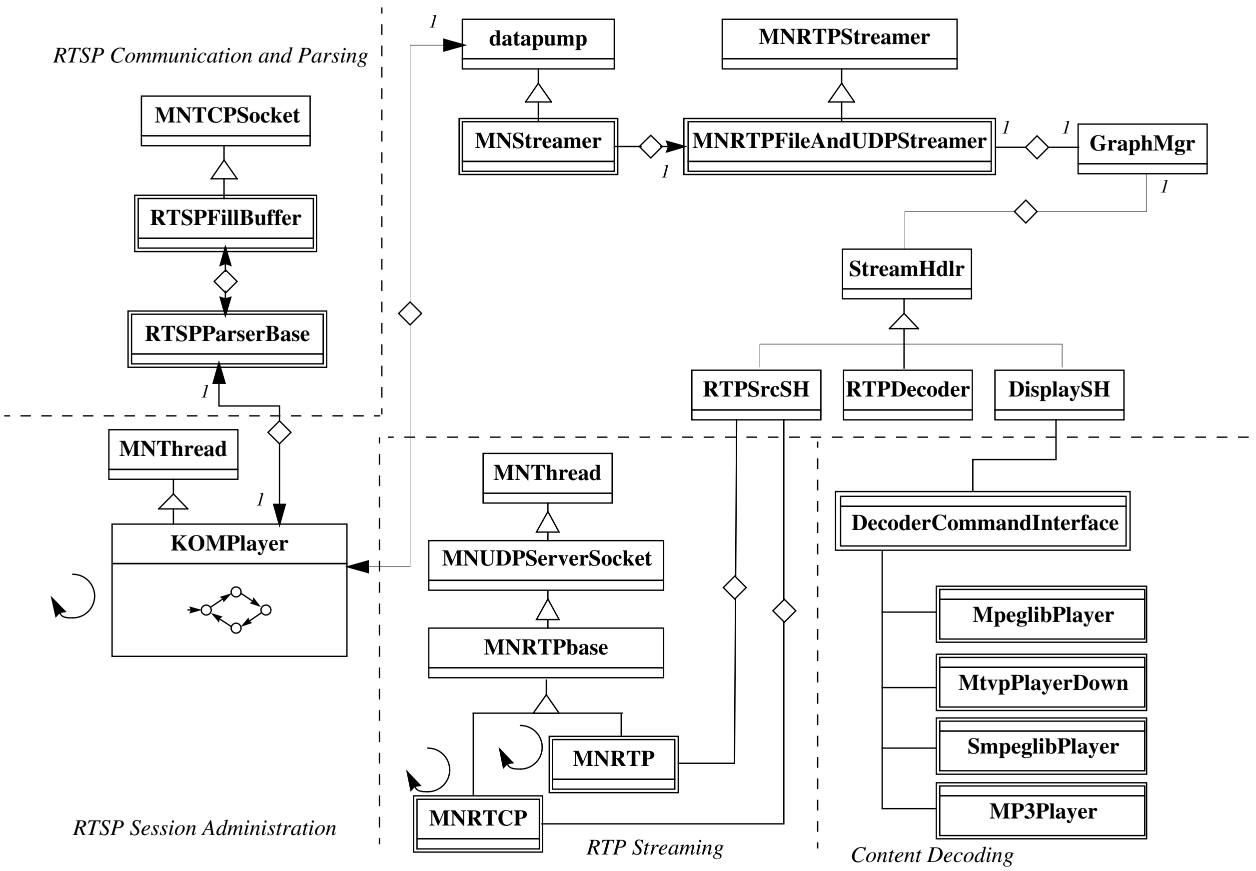

Figure 3: Client

Classes in this part are only used in the client since a visual presentation at the server is not necessary. In combination with the automaton and an abstract interface between the automaton and the decoder we can integrate several available decoders. The generic API makes it simple to integrate decoders into our client and we expect support for other decoders (e.g. MPEG-2, H.261) too.

The player is tested on Linux, its current GUI is based on the Qt library[3]. To work as a Netscape plug-in, the plugger tool[4] is used at this time.

D. Proxy Cache

Since one of our research topics is on caching for multimedia streams we also designed and are currently building a proxy cache for those streams. A detailed design can be found in [Gud00]. Figure 4 shows the class diagram of the proxy cache. Parts taken from server and client are not described again.

The proxy is not an RTSP proxy as understood in the RFC [SRL98]. Rather, it is an RTSP/RTP *proxy cache* that stores content in addition to handling RTSP requests.

Our caching approaches consider the typical commercial request to communicate with a stream's origin server for every RTSP session. RTSP messages from different RTSP sessions are multiplexed onto one connection between an origin server and a proxy. RTSP SessionIDs are the keys to de-multiplex sessions. 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.

A proxy server maintains an instance of the RTSPProxySession for each active RTSP session. Since the proxy cache acts as a client (of the server) and as a server (for the client) its automaton is a combination of the server and client automaton.

The most significant changes compared to server and client occur in the RTP streaming part of the proxy cache. This is caused by the fact that the proxy cache needs client and server functionality. The client functionality is different in a way that a stream can be

- stored on the local disc
- forwarded to the client and
- the combination of both

The server functionality means to forward an incoming or to send a cached stream to the requesting client. This functionality is shown in the RTP streaming part of Figure 5 and explained in more detail in [Gud00].

3. http://www.trolltech.com
4. http://www.hubbe.net/~hubbe/plugger.html

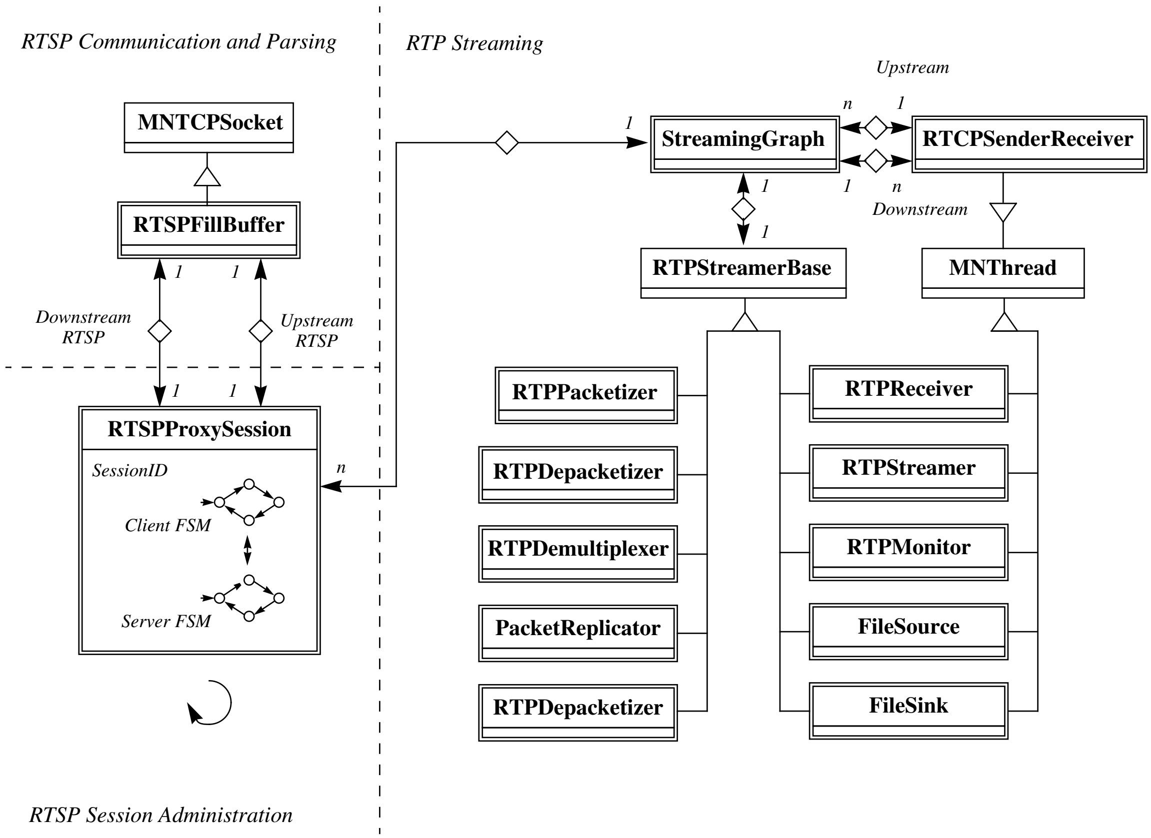

Figure 4: Proxy cache

E. RTSP/SDP Implementation

Since RTSP is an ASCII-based protocol we use a generated parser[5] to parse the RTSP messages. The parser is at the core of the RTSP implementation. In combination with a factory class that maps RTSP messages to their related classes, functional variations are easily implemented. This is useful in experimental environments were new functionality and modifications are tested. By using the existing RTSP implementation we were able to create RTSP classes for the proxy cache quickly that implement different behavior. SDP is also an ASCII-based protocol and implemented with an additional parser for SDP messages. Functional adaptation works as with the RTSP parser.

IV. Conclusions and Future Work

Most available commercial products today have a well defined and documented API that makes it convenient to integrate functionality of those products in other applications. Unfortunately APIs are often not sufficient for research.

5. using bison++, an extension of GNU bison: http://sunsite.bilkent.edu.tr/pub/languages/c++/tools/flex++bison++/

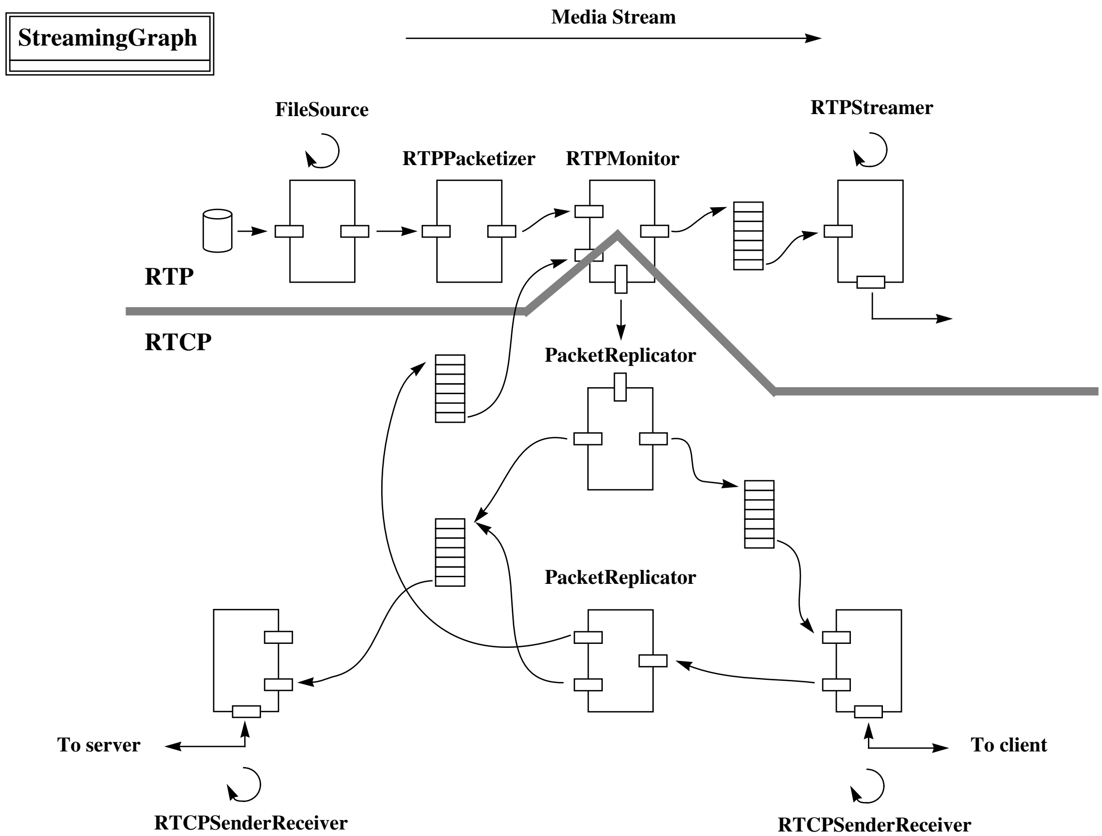

Figure 5: Streaming graph for the proxy cache

If e.g. it is assumed that a new en-/decoder technology should be tested in an application, the codecs must be integrated into the application. For distribution systems research, networking protocols or storage mechanisms must be replaced.

With our ongoing work on wide-area distribution systems for AV content it became clear that we had to build our own AV streaming platform to perform further investigations. We want to give some example scenarios for research that exploits our platform.

A. Caching

Multimedia streaming makes much greater demands on the transmission network than traditional data transmission because of the greater volume of the data and because of the real-time properties. Multimedia streaming requires a lot of memory space in hosts and high bandwidth in links. Large-scale video-on-demand can only be provided with 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. In addition all caches need proxy functionality to communicate with the origin server. The development of client-independent caching systems, and cooperative caching for high-volume content are important research issues.

B. Patching, Gleaning

The basic approach in Patching [HCS98] is the use of multicast for the delivery of a video streams to requesting clients. Clients that request the same video shortly after the start of its transmission start to store the multicast transmission in the local cache immediately. The server sends unicast streams (patch streams) to the clients containing the missing initial portion of the video, until the cached portion is reached. Then, the clients use their caches as cyclic buffers. Gleaning is an extension of Patching in a way that cache servers are deployed as proxy caches, i.e. clients will always connect to their proxy server to access data on the origin server. If the client requests a stream and this stream is already being delivered to a cache server or client, and the sender decides that the patching window for this stream is still open, it orders the cache server to join that multicast stream. Additionally, it starts the transmission of a patch stream to the proxy cache. The proxy cache has to set aside sufficient buffer space for the cyclic buffer to hold the length of the patch stream, even if it does not cache the movie; the stream is delivered as a unicast stream to the client. It is important to evaluate the implementation complexity to select the relevant optimization options.

C. Adaptive streaming

UDP is usually used as transport protocol for AV streaming. In contrast to TCP, UDP does not include any mechanisms like flow control and retransmission. Adaptation is also mandatory to keep UDP based streaming TCP-friendly and therefore avoid that those streams are blocked by backbone operators. Some commercial products provide adaptations to the actual network condition. We want to investigate how a caching support for adaptive streaming could be realized. Our focus is on caches that support caching for adaptive streams transporting different formats (MPEG-1, QuickTime, layered video) [Gud00]. We intend to integrate the adaptation functions into the proxy cache of the KOM-Player platform.

D. Reliable multicast

When caches are used in applications like VoD it must be ensured that the content on a cache is an exact copy of the original. In order to achieve this requirement, a transmission to the caches can be made independently from transmissions to clients, or the transmission to the client is extended to allow a reliable transmission to the cache by additional repair mechanisms.

The second case can be realized by extending RTP to remember lost packets and initiate retransmissions that are ignored by the client. We implemented a first version in our RTP stack and are right now integrating it in a proxy cache to test it.

We hope that our implementation can help other researchers in creating experimental platforms to test and demonstrate their new ideas. The KOM-Player platform[6] may be used under the conditions of the GNU public license but the core elements are also available for other licensing.

6. Available at: http://kom.e-technik.tu-darmstadt.de/kom-player

V. REFERENCES

[Dar00] Darwin Streaming Server Manual, 2000. http://www.publicsource.apple.com/projects/streaming/StreamingServerHelp/.

[DeC99] Linden DeCarmo. *Core Java Media Framework*. Prentice Hall, Upper Saddle River, New Jersey, USA, 1999.

[EN96] F. Eliassen and J. Nicol. Supporting Interoperation of Continuous Media Objects. *Theory and Practice of Object Systems: Special Issue on Distributed Object Management*, 2(2):95–117, 1996.

[FJL+97] S. Floyd, V. Jacobson, C. Liu, S. McCanne, and L. Zhang. A Reliable Multicast Framework for Light-weight Sessions and Application Level Framing. *Transactions on Networking*, 5(6):784–803, 1997.

[Fre00] Ron Frederick. RTSP Interoperability Bakeoff. In *Proceedings of the forty-eighth Internet Engineering Task Force*, August 2000.

[GBW97] Carsten Griwodz, Michael Bär, and Lars C. Wolf. Long-term Movie Popularity in Video-on-Demand Systems. In *Proceedings of ACM Multimedia'97*, pages 340–357, November 1997.

[GJZ99] Carsten Griwodz, Alex Jonas, and Michael Zink. Affordable Infrastructure for Stream Playback in the Internet. Technical Report TR-KOM-1999-07, Darmstadt University of Technology, December 1999. Avalaible at ftp://ftp.kom.e-technik.tu-darmstadt.de/pub/TR/TR-KOM-1999-07.ps.gz.

[Gud00] Gunnar Gudmundsson. Design of a Multiformat Capable Cache for Video Streaming. Diplomarbeit. Fachbereich Elektrotechnik und Informationstechnik, Darmstadt University of Technology, September 2000.

[HCS98] Kien A. Hua, Ying Cai, and Simon Sheu. Patching: A Multicast Technique for True Video-on-Demand Services. In *Proceedings of the ACM Multimedia Conference 1998, Bristol, England*, pages 191–200, September 1998.

[HJ98] Mark Handley and Van Jacobson. RFC 2327 - SDP: Session Description Protocol. Standards Track RFC, April 1998.

[Kae97] Thomas Kaeppner. *Entwicklung verteilter Multimedia-Applikationen*. Vieweg Verlag, 1997.

[Lie00] Jori Liesenborgs. *Voice over IP in networked virtual environments*. PhD thesis, Limburgs Universitair

Centrum, Limburg, Belgium, May 2000.

[MR98] Ian Marshall and Chris Roadknight. Linking Cache Performance to User Behaviour. In *Proceedings of the 3rd Int'l WWW Caching Workshop, Manchester, England*, June 1998.

[M+97] Steve McCanne et al. Toward a Common Infrastructure for Multimedia-Networking Middleware. In *7th Intl. Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV '97)*, May 1997.

[SCFJ96] Henning Schulzrinne, Stephen L. Casner, Ron Frederick, and Van Jacobson. RFC 1889 - RTP: A Transport Protocol for Real-Time Applications. Standards Track RFC, January 1996.

[SRL98] H. Schulzrinne, A. Rao, and R. Lanphier. RFC 2326 - Real Time Streaming Protocol (RTSP). Standards Track RFC, April 1998.

[Tow99] George Towner. *Discovering Quicktime*. Morgan Kaufmann Publishers, San Francisco, California, USA, 1999.

[VI96] S. Viswanathan and T. Imielinski. Metropolitan Area Video-on-Demand Service using Pyramid Broadcasting. *Multimedia System*, 4(4):197–208, 1996.

[ZGJS00] Michael Zink, Carsten Griwodz, Alex Jonas, and Ralf Steinmetz. LC-RTP (Loss Collection RTP): Reliability for Video Caching in the Internet. In *Proceedings of the Seventh International Conference on Parallel and Distributed Systems: Workshops*, pages 281–286, July 2000.