The Architecture of an Integrated RTSP, RTP and SDP Library

Florin Lohan, Irek Defée
Tampere University of Technology, Finland
Tel: +358 3 3653845
E-Mail: {Florin.Lohan, Irek.Defee}@tut.fi

Marius Vlad
Politehnica University of Bucharest, Romania
Tel: +40 93 240569
E-Mail: vladm@ulise.cs.pub.ro

ABSTRACT

We describe architecture and implementation issues of a software library for transport and control of multimedia content over IP networks. The library uses RTSP and SDP as control protocols and RTP for the content transport. All these three protocols are tightly integrated, which simplifies very much the use of the library. Its architecture allows an easy extension of the RTSP and SDP protocols for interoperability with other implementations. RTP profiles can also be added very easy. The library's API is very flexible, allowing easy mapping to other protocols or to simpler interface. The library was already successfully used in several applications.

I. INTRODUCTION

In recent years much work was dedicated by people all over the world for designing protocols for remote content retrieval. There are two aspects related to this, the content transportation from the server to the client, and the content control, that transmits commands issued by the client to the server. It became important that well-known protocols are used, so that many clients and servers can interoperate with each other.

IETF successfully tried to standardize these issues for the case when the underlying network is IP based. Thus, two standardization groups were created, the AVT (Audio Video Transport) group [1], which deals with content transport protocols, and the MMUSIC (Multiparty Multimedia Session Control) group [2], which deals with protocols related to media control.

The result of the AVT group's standardisation effort includes the RTP (Real Time Protocol) [3]. It provides specifications for media transportation from the server to client using data packets. RTP was created to use UDP as IP transport protocol, but it can also work with TCP. RTP may be accompanied by RTCP (Real Time Control Protocol) that sends feedback information from the client back to server.

The MMUSIC group's standardisation effort includes the RTSP (Real Time Streaming Protocol) [4] and SDP (Session Description Protocol) [5]. RTSP is used for sending commands (such as open stream, PLAY, PAUSE) from the client to the server, and the server's response back to the client. The SDP protocol is used inside RTSP, as data attachment to a command (DESCRIBE) in order to send to the client initialisation information related to the desired multimedia stream. This information should contain the number and transport method for each individual media streams and the compression method (standard). The RTSP was designed to rely on TCP as IP transport protocol, but it can also work over UDP. It was also design to use RTP as transport protocol.

Due to this strong relation between these three protocols (RTSP, SDP, RTP) we decided to implement them together in the same library. Thus we can better exploit the interdependencies between these protocols, and create an easier to use interface to the package.

There are several implementations of these protocols available [6], [7], but the novelty of this paper comes from the approach of implementing all three in the same library and also from easy protocols extensibility, which is necessary in several cases for interoperability with 3rd party applications [8], [9], [10]. Extending the protocols is done inside the client application, without modifying the library.

The paper is organized as follows: we are describing the protocols in section II, we present architecture and implementation issues for each protocol in section III, we present implemented extensions and applications in section IV and then we conclude the paper.

II. RTSP, SDP AND RTP PRESENTATION

The Real Time Streaming Protocol (RTSP) [4] is an application-level protocol for control over the delivery of data with real-time properties. RTSP provides an extensible framework to enable controlled, on-demand delivery of real-time data, such as audio and video. Sources of data can include both live data feeds and stored clips. This protocol is intended to control multiple data delivery sessions, provide a means for choosing delivery channels such as UDP, multicast UDP and TCP, and provide a means for choosing delivery mechanisms based upon RTP.

RTSP consists of text messages exchange between the client and the server. The client sends request messages, and the server answers to each request (similar to HTTP).

The protocol defines the set of methods (commands) a client can send to a server. Each method has several fields for communicating additional information. The most important methods are: DESCRIBE, SETUP, PLAY, PAUSE and TEARDOWN.

The DESCRIBE method is intended for retrieving initialisation information from the server about the media session that is going to be played. This information is stored as SDP description and contains global information about the session (information about the author, URL, bandwidth information, timing, encryption information) and it also contains information about

individual media streams (number, type, connection and transport information, other information).
After receiving the session initialisation information, the client can request the individual media streams that compose the media session. For example, if the client wants to play a movie, the movie will typically consist of two streams, an audio stream and a video stream.
Requesting individual media streams is performed with the SETUP method. The most important information field is the Transport field that contains information about the data transport possibilities of the client. The answer from the server also contains a Transport field that actually chooses one of the transport methods proposed by the client.
Example:

```
C->S SETUP rtsp://video.ex.com/video RTSP/1.0
     CSeq: 3
     Transport: RTP/AVP;multicast;ttl=127;mode=
         "PLAY",RTP/AVP;unicast;client_port=
         3456-3457;mode="PLAY"
S->C RTSP/1.0 200 OK
     CSeq: 3
     Session: 23456789
     Transport:RTP/AVP/UDP;unicast;client_port=
         3456-3457;server_port=5002-5003
```

In the above example, the client requests a media URL and advertises to the server two methods of data transport. The first method is the multicasting of the data using the RTP protocol, Audio-Video Profile. The second method the unicast streaming, where the server sends RTP packets to client's 3456 UDP port (the 3457 port is for RTCP use). In the server's response we see that it chose the second method and it also provided its port information (useful for RTCP).
After the media data stream is set up, the client has to send the PLAY command to the server in order that the server starts sending the data. Additional fields may specify the playing position and speed.
The PAUSE method interrupts the stream transmission.
When the client wants to end the media session it sends the TEARDOWN command to the control URL of the session. Individual media streams can also be TEARDOWNED.
The Real Time protocol (RTP) provides end-to-end network transport functions suitable for applications transmitting real-time data, such as audio, video or simulation data, over multicast or unicast network services. RTP does not address resource reservation and does not guarantee quality-of-service for real-time services. The data transport is augmented by a control protocol (RTCP) to allow monitoring of the data delivery in a manner scalable to large multicast networks, and to provide minimal control and identification functionality. RTP and RTCP are designed to be independent of the underlying transport and network layers.
Data sent over RTP is fragmented and transmitted in packets, similar to the UDP case. RTP also adds its header to the packet. The RTP header contains several fields that specify some additional information. The most important fields are:

- The payload type – specifies what kind of data the RTP packet is carrying. For different data types there are defined RTP profiles that specify additional information to be carried out in the RTP packet, as extension fields (header extension).
- The sequence number – specifies the number of the RTP packet, so that the receiver can identify the order of the packets and also packet loss, by counting the missing packets.
- The timestamp – provides timing information necessary at the receiver for the decoding and/or displaying process.

There are also RFC standards that define RTP profiles for different types of media that are to be carried over RTP.

III. THE DESIGN AND IMPLEMENTATION OF THE LIBRARY

The general architecture of the library is shown in Fig. 1.

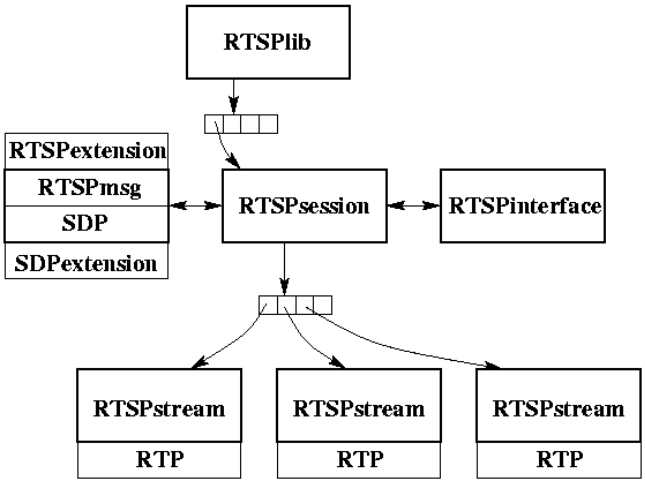

Figure 1. The architecture of the library

A. General Design Issues

The library was developed in C++ and C as a software package. It compiles on both WIN32 and Unix platforms (only tested on Linux and Solaris) due to an accompanying compatibility library that implements some operating system specific primitives (networking, threads and thread synchronization related functions).
The library features a central engine, the RTSPlib class, having only one instance per process (running program). This class generates RTSP sessions (when requested, by calling CreateRTSPclient, on the client side) or automatically, on the server side, when a new client connection arrives. A session instance (RTSPsession) handles all the RTSP messages that are using its connection. The RTSPsession instance has several other classes associated with it. The most important of these is the RTSPmsg.
This class handles all the RTSP messages. A big data structure is defined inside this class that can keep data

about all the RTSP commands and their associated fields.
The RTSP client starts requesting a command by filling information into this data structure (a data structure is pre-filled first and passed to the client). The RTSPmsg class then creates the RTSP text message based on this structure. The text message is then send to the server, which parses it and fills the same structure on the server side. This data structure is then passed to the server application.

B. Extending RTSP

The RTSP protocol provides provisions to extend the protocol by creating extension fields that accompany a message. Our library allows easy extensions by providing a virtual extension class (RTSPextension). If the application designer wants to extend the RTSP protocol, he or she only needs to derive this class.
The subclass also has a data structure and two functions, one for creating the additional text fields from the data structure and the second one for parsing the text fields into the structure.
When the client RTSPmsg instance creates an RTSP text message from its data structure, it also checks for an associated valid RTSPextension instance. If this exists, it will call its function for creating additional fields, and after this it will send the text message to the server. Thus, the client application has to fill with information two data structures: the data structure of the RTSPmsg class and the data structure for its subclass. On the server side, if the function parsing the RTSP text message finds unknown fields, it checks the existence of an RTSPextension based instance. If it finds it, it will call its parsing function to parse the unknown fields. Then both data structures (belonging to RTSPmsg and its subclass) will become available to the server application.

C. SDP Issues and Extensions

The SDP protocol is implemented in a similar manner. There is a class that is based on a data structure that can contain information about any SDP description. This class has functions for creating the text SDP message from this data structure and for parsing the SDP text message into the data structure. The SDP class can be extended in a similar manner as the RTSPmsg class, the extension superclass is called SDPextension.
The application server (eventually) fills the SDP data structure with information when answering to a DESCRIBE command. The SDP class then creates the SDP text message and attaches it to the RTSP answer to the DESCRIBE command. The client receives then the RTSP DESCRIBE response, and will detect the SDP message attachment. The SDP message will be detached and sent for parsing to the SDP class. This class will parse it and will fill its data structure, which is going to be available to the client application, together with the RTSP response message data structure. Eventual extension data structures (for RTSP and SDP) will also be available to the client.

For parsing RTSP or SDP extensions one can reuse the parsing functions used by RTSP, which are also exported by the library.
The RTSPsession instance generates streams (the RTSPstream class) when requested by a SETUP command and destroys them when receiving a TEARDOWN command. The stream instances are created both on client and on server. A stream instance keeps its own RTSP state and transport parameters.
The RTSPstream instance handles the data transport between the server and client. Each instance of this class also has associated the RTP class that corresponds to the transport parameters negotiated during the SETUP command.

D. RTP Issues

The RTP class has two important functions, one for data encapsulation into RTP packets, and the second one for data retrieval from received packets. The RTP class can send or receive data from the network by itself, or it can allow the RTSPsession to do these operations.
The library's interface with the application has two aspects: command handling and data handling. For this, two classes are used, the RTSPsession that contains functions for sending commands, and another class (RTSPinterface) that contains callback functions.

E. The API of the Library

The library's interface for data handling is realized through the RTSP session interface. When sending data, the corresponding stream is identified (e.g. by its corresponding URL) and the RTSPsession instance will then pass the data (and eventually other related information) to the corresponding RTSPstream instance. When receiving data, a callback function is called after the RTP packet is processed. The parameters of the callback function will identify the stream and will provide the data and the additional information contained in the RTP packet.
The library's interface to command handling is very powerful and allows application control to its deepest features. To each RTSP command correspond 4 functions, two direct and two callbacks. These functions have the suffixes _Req (Request), _Ind (Induced), _Rsp (Response) and _Cnf (Confirmation).
If the client application wants to send an RTSP command to the server, it will call the command's corresponding Request function. This function will process the command, will send it to the server and will return, without waiting for the answer from the server. On the server side, when an RTSP request arrives, the Induced function (which is a callback function) will be called. The server application has to provide command-handling mechanism in this callback function. From the Induced function or from other place (it also can be other thread) the server has to call the command's Response function that will send to the client the RTSP response to the client's request. On the client side, when an RTSP response is received, the corresponding Confirmation function is called (this is also a callback function). In this function the client may check if the command was

successful or it can see the eventual error. The Confirmation function can also be left void (unimplemented in the subclass of RTSPextension).
So, to use the library in a server application, the _Ind callbacks of the used RTSP commands have to be redefined (the standard requires some commands to be implemented and also recommends that most commands are implemented). If the server does not expect to receive data or to send commands, then the receiving data callback and the _Cnf callbacks need not to be redefined.
On the client side we suggest that the _Cnf functions to be redefined (at least for checking the return value of the RTSP command). If the client expects to receive data (media streams), then it should redefine the receiving data callback (this is true in most of the cases). If the client does not expect to receive commands it does not need to redefine the _Ind functions.

IV. IMPLEMENTED EXTENSIONS AND APPLICATIONS

We used this library in two major applications by now.

A. Complete System for the Transport of MPEG-4 Content over the Internet

The first application was developed together with Nokia Research Centre in Tampere, Finland and we aimed at designing and implemented a complete system for the transport of MPEG-4 content over the Internet [10].
The architecture of this system is presented in Fig. 2.

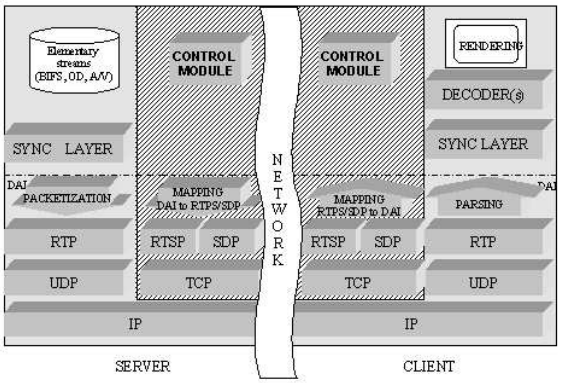

Figure 2. The architecture of a system for MPEG-4 transport over the internet

The end-to-end system contains a MPEG-4 server and a MPEG-4 client, connected via an IP network. The architecture implements the MPEG-4 delivery philosophy, based on the separation between the delivery layer and the application. The control part was implemented by mapping the DAI (DMIF Application Interface) primitives [11] to a set of RTSP commands (SDP was also used). This alternative was preferred to DMIF default signalling since it allows interoperability with other RTSP-based media servers [12]. Moreover, the usage of RTP, RTSP and SDP, well-established Internet protocols, makes future extensions straightforward (such as the addition of RTCP and RSVP, in order to deal with QoS and network resources). Using our library for this application involved creating RTSP extension fields and also mapping the library's interface to DAI primitives. We used an RTSP extension derived class to add several extension fields to RTSP. These fields were necessary for carrying DMIF specific information that we were not able to map to standard RTSP fields. The mapping onto DAI primitives was not a trivial thing and required careful study.
The implementation of the architecture was successfully demonstrated at the 50th MPEG meeting in Maui, Hawaii, in December 1999. At that time, this architecture was one of the first two examples of a complete system for the transport of MPEG-4 content over the Internet based on mapping onto IETF protocols.

B. Customized Network Library for a Client Interface with Oracle Video Server

This second application was developed for a project with Nokia Home Communications in Linköping, Sweden. Here we developed a complex networked MPEG-2 player, described in [9], that interoperates with Oracle Video Server [8].
The interoperability was a challenging task, since Oracle did not have a standard RTSP implementation in its product, but used its own extensions.
For this application we also had to have an RTSP extensions subclass in order to implement specific fields. We also had to extend the SDP protocol in order to get some more information about the movie that is streamed.
The Oracle Video Server is not using RTP but a proprietary transport protocol (OGF, similar with RTSP, however), described in the server documentation. We implemented this protocol so that our library to be capable of understanding what the server is sending.
The library also required mapping to a new interface, because the Oracle Video Server was expecting the RTSP commands in a certain non-standard order. The server was not using the DESCRIBE command, but expected a SETUP command for the URL of the MPEG-2 transport multiplexed stream, the answer to this SETUP command contains the SDP information that is usually sent as a response to a DESCRIBE command. If this SETUP command contained a special Oracle extension field, the server would start playing the stream immediately, without waiting for a PLAY command from the client.
The new mapping module hides from the application the complexity and hassle of interacting with the Oracle Video Server, but provides it with few very simple commands, like OpenStream, Play, Pause, EndStream, etc.

V. CONCLUSIONS

In this paper we presented the architecture and implementation issues of a library for multimedia delivery over Internet. The library uses well-established IETF protocols: RTSP, SDP and RTP. The novelty comes from the fact that all these interdependent protocols are implemented together in the same package,

having a common interface. The library also supports easy extension of the RTSP and SDP protocols, and also the addition of new RTP profiles and other RTP-like transport protocols. Two applications where this library was used were also briefly described. The usage of the library in these applications is relevant for the capabilities of the package.

As future work we are planning to make the integration of the three protocols involved even tighter. We are also planning to perform extensive testing and performance measurements. After achieving these goals we intend to release the source code of the library as GPL.

ACKNOWLEDGEMENTS

The authors are grateful to Mr. Roberto Castagno for his help during the development of this work.

REFERENCES

[1] IETF Audio/Video Transport Group, http://www.ietf.org/html.charters/avt-charter.html.

[2] IETF Multiparty Multimedia Session Control Group, http://www.ietf.org/html.charters/mmusic-charter.html.

[3] IETF Multiparty Multimedia Session Control Group, H. Schulzrinne et al. "RTP: A transport protocol for real-time applications", Internet draft, Internet Engineering Task Force (IETF), July 14, 2000.

[4] Schulzrinne et al. "Real time streaming protocol (RTSP)", Technical Report RFC 2326, Internet Engineering Task Force (IETF), Network Working Group, April 1998.

[5] M. Handley et al. "Session description protocol (SDP)", Technical Report RFC 2327, Internet Engineering Task Force (IETF), Network Working Group, April 1998.

[6] RTSP implementations list, http://www.cs.columbia.edu/~hgs/rtsp/implementations.html

[7] RTP information about implementations, http://www.cs.columbia.edu/~hgs/rtp

[8] Oracle Video Server, http://technet.oracle.com/products/oracle7/htdocs/media/info/xovsds3.htm

[9] Florin Lohan, Prakash Sastry, Irek Defée, "Broadband Network Set-Top Box System", CD-ROM Proceedings of PROMS 2000.

[10] Roberto Castagno, Serkan Kiranyaz, Florin Lohan, Irek Defée, "An Architecture Based on IETF Protocols for the Transport of MPEG-4 Content over the Internet", Proceedings of ICME 2000, vol. 3, pp. 1322-1325, August 2000.

[11] ISO/IEC 14496 –6:1999(E) Delivery Multimedia Integration Framework (DMIF).

[12] A. Basso et al. “Preliminary results in streaming MPEG-4 over IP with the MPEG-4/IETF-AVT payload format”, Technical report MPEG Systems Group, Melbourne, Australia, October 1999