A Proxy Architecture for Collaborative Media Streaming

Verena Kahmann
kahmann@tm.uni-karlsruhe.de

Lars Wolf
Lars.Wolf@uni-karlsruhe.de

University of Karlsruhe, Institute of Telematics
Zirkel 2
Karlsruhe, Germany

ABSTRACT

Streaming media from the Internet is a successful application for end-users. With the upcoming success of mobile devices and home networking environments, cooperation among users will become more important in the future. To achieve such cooperation, explicit middleware standards have been defined. However, many of them have been built for specific networking technologies, and interoperability is hard to obtain. We propose a new concept for cooperation exemplary for collaborative media streaming by using IETF multimedia session control protocols together with a proxy architecture. This concept is more scalable on any networking environment and provides for easy interoperability.

1. INTRODUCTION

Media streaming from Internet media servers has become an important application recently. In contrast to downloading a media file as a whole, streaming offers flexibility in the sense that live streams as well as recorded files can be transmitted to the client. Additionally, the streams of a movie can be controlled as an aggregate or as single streams. The Real-Time Streaming Protocol (RTSP) [7] of the IETF multimedia control architecture provides the means for controlling the streaming session. Its functionality is comparable to a VCR.

Other common multimedia applications are video conferences where several users are able to share a common channel. The IETF has defined the Session Initiation Protocol (SIP) [2] for call or conference setup. Other common architectures are H.323 or Mbone conferencing tools. Media can be streamed to such a video conference also. For live streams, this may be achieved by subscribing to a common multicast group, e.g. by specifying a multicast session address in the invitation. However, for stored files, streaming is mostly done using unicast. Nonetheless, a user may want to share the actual content of his / her streaming session with other users in the conference, which we call *collaborative media streaming*. In this case, combining media streaming with conferencing is not trivial, because both synchronization and flexibility should be preserved.

Combining RTSP and SIP, it is possible to invite users to an existing streaming session and pass the URL of the stored movie to the users, i.e. to perform a *push* of the stream to a new user. In the other direction, i.e. to actively get or *pull* the stream, it is also possible to join such a streaming conference retrieving the URL of the stored movie using the SIP conferencing service. The push to or the pull from another user is what we generally call *application extension*. The application of the client then can do its own streaming session setup and control the stream with RTSP signaling (PAUSE, PLAY etc.). Overall, this results in several streams of the same content to different users. Hence, a synchronization between these media streams is needed to protect the collaborative nature of the scenario. However, such a synchronization is not covered by the RTSP standard.

Therefore, we define a proxy architecture to enable both synchronization and flexibility on collaborative streaming sessions. By using a proxy, existing media servers in the Internet need not be changed. In this architecture, we define the concept of association to bind two or more clients of a streaming session to a relation, which is more loosely coupled than a multicast group. For example, each user may jump freely in the movie, or can open sub-associations with other users.

Synchronization in our architecture is achieved by handling associations and calculating timing information for any client joining an association. We introduce a proposal to map session invitation and streaming control requests to association requests. We do this exemplary with IETF multimedia control protocols like SIP and RTSP. As we use IETF application layer protocols we are not dependent on special hardware or specialized middleware standards like HAVi, but such standards can inter-operate with our concept if they have the Internet protocol stack implemented.

An important point in our proxy concept is to keep signaling and data paths separate. The advantage is that streaming media is independent of the channel used for the invitation. The breakdown of the connection between users does not necessarily include the breakdown of the streaming session. Therefore the concept is suitable for networks with partial disconnectivity. Another advantage is that participants can choose their own streams, e.g. for audio streams with dif-

ferent languages or different levels of quality of service.

In the rest of this paper, we first introduce the architecture of a streaming proxy serving as a session management entity in section 2. Then we propose our concept of association in section 3 before considering session setup and mapping of control functionality to association requests in the following subsections. We give an overview of related work in section 4 before we finally conclude the paper.

2. STREAMING PROXY

The possible scenarios we consider for our architecture include getting a movie from another user or showing a movie to another user. In both cases, the other user must be located and in the case of getting a movie, even the application running on the other user's screen must be located. Moreover, the second user may wish to get the stream synchronized with the first client.

Therefore, we propose to use a proxy acting as a mediator between clients and media servers. Thus, existing media servers need not be rewritten with the ability for application extension. Since the proxy only influences the signaling path, it may be co-located with a media server or gateway, and even with a client.

The building blocks needed to deploy the application extension functionality are shown in figure 1. Note the separation of signaling and data path, since the proxy is only involved in the signaling path.

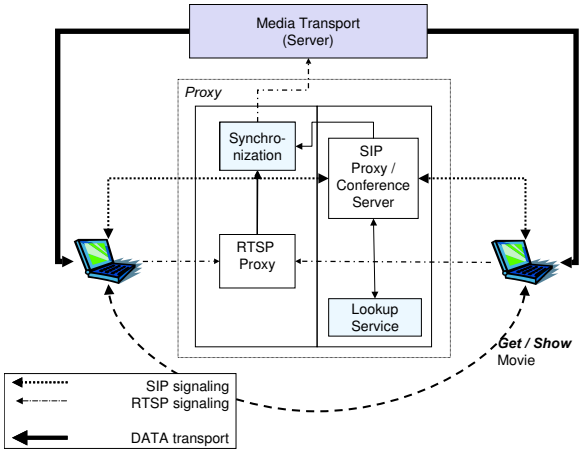

Figure 1: Streaming Proxy Building Blocks

The proxy functionality for the signaling protocols SIP and RTSP acts mostly as virtual client/server in each case. To enable collaborative streaming, the corresponding requests are passed from the SIP or RTSP proxy entities to the synchronization block (see section 3.1 for an example). This block processes the requests according to the functionality for implementing the association concept as shown in section 3 and sends them to a media server handling the actual media transport as done by existing RTSP servers, if necessary. Additionally, it could also give feedback to the clients for smoothing out buffer differences (confer section 3.2).

Proxies for SIP and RTSP are defined in the corresponding IETF documents [2, 7]. For a self-invitation, which is needed to pull an existing stream, a conferencing server should be used instead of a proxy. SIP conferencing is currently under consideration by the corresponding working group of the IETF.

3. ASSOCIATION CONCEPT

We have introduced the concept of association in [5]. An association is, in the easiest case, a structure containing the handle to the application, e. g. a streaming URL, and a list of client addresses. If synchronization of the clients should be obtained, the point of time and the position in the movie where streaming has been started must be stored. This is illustrated in figure 2. Clients can apply control commands like PAUSE or jumping to other points in the stream on their own, thus opening their own sub-association represented by a recursive structure with starting time, starting position and another client list in the figure. On leaving the sub-association, clients may easily return to the original timeline or to the position they left the main branch. In the latter case, this position has to be saved in the association structure.

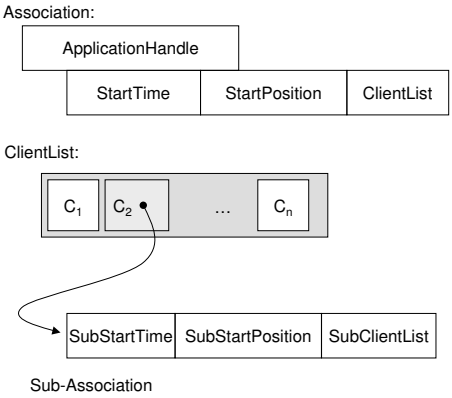

Figure 2: Association Structure

Other objects like access control lists can be easily added to the association, thus allowing the first client to grant or reduce permissions for looking up information on the application. Also, elements that should be controlled by a single person can be protected for a certain period on behalf of the association.

The data flow is separate from the association. Therefore, every client can take the desired streams by means of RTSP session setup independently from other clients, which is reasonable because clients or users apply their own capabilities or preferences, respectively. Nonetheless it is possible to request the same set of streams another client is currently receiving, e. g. by exporting that information to the lookup service and retrieving it on application lookup.

We show the architecture, namely the interaction of SIP and RTSP proxy building blocks with the association manager in figure 3. The *Association Mapper* maps SIP and RTSP protocol requests to the corresponding association methods which are given to the *Association Manager*. This building block manages the association list structure and returns timing information to the proxy building blocks.

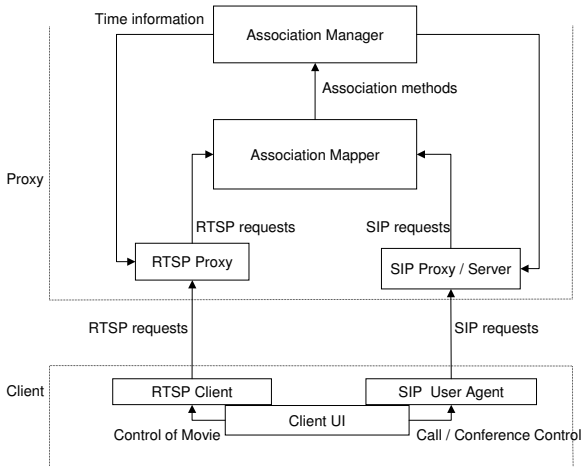

Figure 3: Architecture of Association Management

3.1 Association Methods

To obtain a defined interface and to enable re-usability of existing RTSP or SIP proxy functionality, we have defined a set of *association methods*. Each RTSP or SIP protocol request is mapped to an association method by the Association Mapper. These methods are used by the Association Manager to manipulate the association structure itself. Thus, they define an interface between the control functions applied by the client(s) and the logical relation between each of them.

In our case, an association will be initialized when the first client starts a streaming session with an RTSP SETUP request. This request will be mapped to `buildAssociation()`. Information on the association may also be sent to the service needed for application lookup.

The second client will be invited to the session in case of a push service or do a self-invitation in case of a pull. This SIP INVITE will cause a `bindAssociation()` method. We have invented this method to enable a relation between RTSP and SIP requests, e. g. to tie the following RTSP SETUP to the suitable association. It may also be reasonable to propagate information on the client to the Association Manager with this method or to return actual information on the association to the client vice versa.

Recall that for the demanded flexibility the clients must do their own streaming session setup. The RTSP SETUP by any following client will not require a `buildAssociation()` action. Even the information on single streams (like audio, video etc.) is not necessarily needed, but again may be stored for further lookup.

RTSP PLAY requests are mapped on the `joinAssociation()` method. In any RTSP PLAY request, a range parameter can be set to indicate the position in the movie from which streaming is to be started. If no range parameter is set, streaming will start from the beginning of the movie. The Association Manager will process the `joinAssociation` in the following way: If no binding to the association exists, the position and actual time are added to the association. Otherwise, the timing information, i. e. the position from where the stream will be played, has to be calculated for the client issueing the request, e.g. by using the RTSP GET_PARAMETER request for the time that has already passed. This timing information will be returned to the RTSP proxy in order to forward the RTSP PLAY request with the range parameter set to that position to the media server.

An example of a push service with the interaction of the proxy building blocks is shown in figure 4. In this example, the session setup of a first client C1 and the invitation with the following session setup for the second client C2 are illustrated. The corresponding association methods and their effect on the association structure are also shown. In the figure, the timeline for RTSP and SIP proxy building blocks is coupled with the timeline for the Association Mapper. All RTSP requests are forwarded to the Server, however, for the sake of clarity, only the request containing the range parameter is shown. For the same reason, most responses are omitted also.

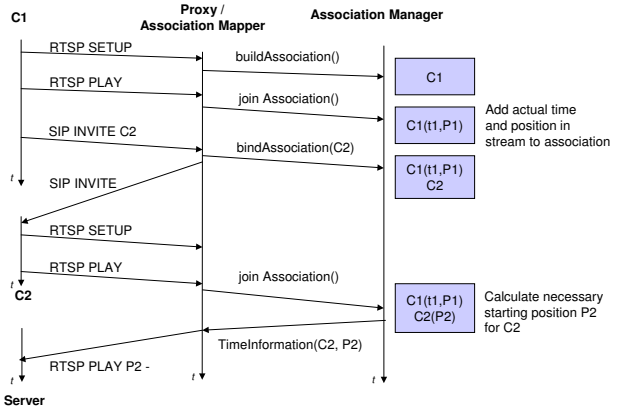

Figure 4: Association Setup for a Push Example

By default, applying any control functionality on the movie opens a sub-association, so this client will stream the content independently. To follow other clients into their sub-association or to get back to the original timeline, special commands have to be applied by the client application. Since RTSP or SIP do not handle that level of cooperation, we will implement additional functionality for the exchange of such messages. Besides several possibilities to manipulate the associations and the branches, helper functions like getting the list of clients in the association or in a certain sub-association can be implemented.

3.2 Synchronization Issues

We have kept signaling and data path separate for the reasons of flexible control. Another advantage is that data paths can be separate if distinct QoS should be applied or users are mobile with changing routes to them. Yet, this separation can be a disadvantage in some cases if synchronization is critical. If timing information is calculated by the association manager and passed to the media server, there is a delay between the synchronization point and the point when the new media streams to the second client are sent out. However, we apply the same mapping to each PLAY request, and if timing information can be calculated fast enough and transmission delay between proxy and server

will not change, e. g. if the proxy is co-located with the server, the time between joining an association and processing the PLAY request in the server will be the same for each client.

Another synchronization issue is caused by different play-out buffers in client applications. The playback in one client may start earlier or later dependent on buffer size. However, if these buffer sizes can be determined, the difference between them can be calculated and put into the timing information sent to the media server. Moreover, such information could be sent to clients to enable them to apply buffer-level control mechanisms. Proposals for implementing a protocol for synchronization between clients in that way can be found in [6].

Transcoding of the media streams for only a set of clients may cause additional delay potentially leading to poor synchronization. If transcoding is necessary, additional features have to be added, for example an artificial delay for other streams. However, to estimate which delay is introduced by transcoding may be difficult to do.

The approach that we follow means that modifications in the control functionality of current streaming clients and additional functionality of SIP and RTSP proxies (like the interaction with the association manager) will have to be implemented. However, synchronization with another streaming client can be achieved without changing SIP or RTSP protocol methods.

4. RELATED WORK

Up to now, several middleware approaches have been defined which allow the extension of applications. For example, in the home network environment, HAVi [3] defines a middleware layer which allows the extension of applications to different places. The drawback of these approaches is mostly the restrictedness on a special networking environment. If clients in different environments want to co-operate, additional mapping functionality must be deployed in both stacks. With our approach, every system having the IP suite implemented can run our application or be equipped with the proxy functionality. Thus, interoperability is easily provided.

In Internet environments, concepts of group communications have been implemented using multicast. Streaming multimedia content using multicast has been proposed e. g. in [4] or [8]. Sharing of streaming applications has not found much consideration yet. One approach is shown in [1] with a proposal for cooperative playback sessions. In that approach, the main concept is the conference control mechanism between clients and the propagation of control messages to other clients. Media streams are transported by using multicast.

However, there are some drawbacks implementing multicast for application extension and data transport. In today's Internet, multicast routing is typically not available, so it is necessary to provide for tunneling or the implementation of reflectors splitting multicast into unicast connections. Another disadvantage from the user point of view is that the user cannot use control functions like jumping to a different point in the movie. Our concept has the advantage of granting control to each user so that the clients need not necessarily follow but can apply all control commands themselves. Especially for learning systems this may be valuable, because users may want to repeat their lessons on their own without asking others.

5. CONCLUSION AND FUTURE WORK

In this paper, we presented an architecture for collaboration among streaming clients without specifying an explicit middleware layer. Control functionality like jumping to other points in the movie are mapped to methods of the new association concept which we proposed in the paper.

Future work will include implementation of our concepts in a streaming test bed by re-using existing SIP and RTSP components as far as possible. The connection of our concept with directory services or service location entities and the integration of content description is another issue we will handle. We will also consider efficient data transport mechanisms in RTP and have a close view on standardization efforts of SIP.

6. REFERENCES

[1] G. Fortino and L. Nigro. A cooperative playback system for on-demand multimedia sessions over Internet. In *Proc. of IEEE Conference on Multimedia and Expo*, New York, USA, Aug. 2000.

[2] M. Handley, H. Schulzrinne, E. Schooler, and J. Rosenberg. SIP: Session Initiation Protocol. RFC 2543, The Internet Engineering Task Force, March 1999.

[3] HAVi Specification 1.0. The HAVi Organization; http://www.havi.org/, January 2000.

[4] K. Jonas, J. Modeker, and M. Kretschmer. Get a KISS - Communication Infrastructure for Streaming Services in a Heterogeneous Environment. In *Proc. of ACM Multimedia Conference*, University of Bristol, UK, Sept. 1998.

[5] V. Kahmann and L. Wolf. Collaborative Media Streaming in an In-Home Network. In *Proc. of International Workshop on Smart Appliances and Wearable Computing (IWSAWC)*, Phoenix/Mesa, Arizona, Apr. 2001.

[6] K. Rothermel and T. Helbig. An Adaptive Stream Synchronization Protocol . In *Proc. of 5th International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV)*, Durham, New Hampshire, USA, Apr. 1995.

[7] H. Schulzrinne, A. Rao, and R. Lanphier. Real Time Streaming Protocol (RTSP). RFC 2326, The Internet Engineering Task Force, April 1998.

[8] S. Sen, D. Towsley, Z.-L. Zhang, and J. Dey. Optimal Multicast Smoothing of Streaming Video over an Internetwork. In *Proc. of IEEE INFOCOM*, New York, Mar. 1999.