

Towards Interoperable Multimedia Streaming Systems ¹

Ellen(Xiaolan) Zhang, Don Towsley, Jack Wileden
Computer Science Department, University of Massachusetts at Amherst
{ellenz,towsley,jack}@cs.umass.edu

Abstract

Most existing multimedia streaming systems have been designed such that servers and clients from competing systems cannot directly communicate with each other. We refer to this as the multimedia streaming interoperability problem. In this paper, we propose a solution to this problem based on the usage of a translation proxy. We conducted a study designed to determine the signalling and data transport details of RealSystem and QuickTime. Based on this study, we developed a prototype translation proxy for the AMPS system, RealSystem and QuickTime. Experiments with this prototype verify the feasibility of this translation proxy approach. The paper concludes with a description of an open architecture for a translation proxy that enables adding supports for new clients and servers.

I. INTRODUCTION

There exist numerous servers and clients in the multimedia streaming domain, such as RealNetwork's RealServer and RealPlayer, Apple's QuickTime server and player, and Microsoft's Windows Media server and player. Unfortunately, different servers and clients are rarely able to communicate with one another due to differences in control signalling and data transport; we refer to this as the *multimedia streaming interoperability problem*. As a result, a user has to install client software for each kind of streaming server that he or she connects to; content providers often set up a number of servers so that users with different client software can view the video; research labs experimenting with a server that implements novel transmission schemes are required to build full-functioning clients from scratch; and different proxies need to be deployed to provide services to different streaming systems.

The focus of this paper is the interoperability problem. We propose a solution to this problem based on the deployment of a translation proxy between the client and server to handle differences in signalling and data transport between the two platforms. To the best of our knowledge, our work is the first systematic study of this problem. We developed and experimented with a prototype proxy that successfully enables an AMPS server [1] to stream a MPEG-1 video to a RealPlayer and a QuickTime player. In our study, we identified and solved the rate matching problem that comes up when streaming to a QuickTime player (Section IV-A.2). Based on our study, we propose an open architecture for a translation proxy that will support additional clients and servers, and will allow the integration of translation service with other proxy services (Section IV-C).

An alternative approach is to develop a plugin for the client so that it can function as a client to a different brand server. In our previous work [12], we explored this approach by developing a plugin for RealPlayer which interfaces with a AMPS server and provides a virtual file interface for the RealPlayer to access the video served by the AMPS server. This works when a RealMedia format stream is served. However,

¹This research was supported in part by National Science Foundation under awards EIA-0080119, ANI-9973092, ANI-9977635. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the funding agencies.

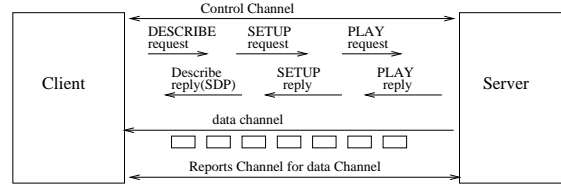


Fig. 1. Multimedia Streaming Overview

for an MPEG-1 stream, our experiments showed that the RealPlayer tries to read the ending part of the stream to obtain certain information before playing out the stream. As a result, the whole MPEG-1 stream has to be received and buffered before playback can begin, and thus the plugin approach doesn't work for this case. Burst.com [2], a company that provides a streaming content distribution network solution, has developed plugins for Windows Media Player and QuickTime player to access their proprietary protocol based streaming servers. Compared to a translation proxy approach, the plugin approach provides the client a lower latency on the server-client path, but has strong dependency on the API provided by the client.

This paper is organized as follows. Section II provides an overview of a multimedia streaming system and the interoperability problem, and discusses the general requirements for a translation proxy. Section III presents our preliminary investigation of the interoperability problem. In Section IV, the translation prototype proxy study is presented and the design of the translation proxy is studied. Section V summarizes this paper and discusses possible future work.

II. OVERVIEW OF MULTIMEDIA STREAMING SYSTEM AND TRANSLATION PROXY

In this section, we first describe a generic multimedia streaming platform. We then discuss how the control and data channels are mapped to network connections. Last we point out some of the differences that exist between actual platforms.

A. Components of a generic streaming system and their roles

Figure 1 shows typical communications between a streaming server and client. Two kinds of information are exchanged between the server and client during a multimedia streaming session: control/signalling information and multimedia data. We describe each in turn.

A.1 Control/Signalling Channel

Below we describe a typical signalling sequence between a client and a server:

1. The client first queries the server about a particular presentation and the server returns a description of the presentation.
2. The client then requests that the server set up a session for streaming the presentation, negotiating with the server about the transport options such as the data transport channel and delivery bandwidth. The server returns an acknowledgement that includes the transport option that will be used to send the data.
3. After the session is established, the client can request that the server initiate playback. The server returns an acknowledgement, and begins to send the data in a continuous manner. At any point in time, the client can request that the server pause the transmission, restart the transmission from a specified point, or tear down the session.

The signalling protocols used in the control channel include the standard RTSP (RealTime Streaming Protocol) [11] and some proprietary protocols such as PNA used by older versions of RealSystem [9], and

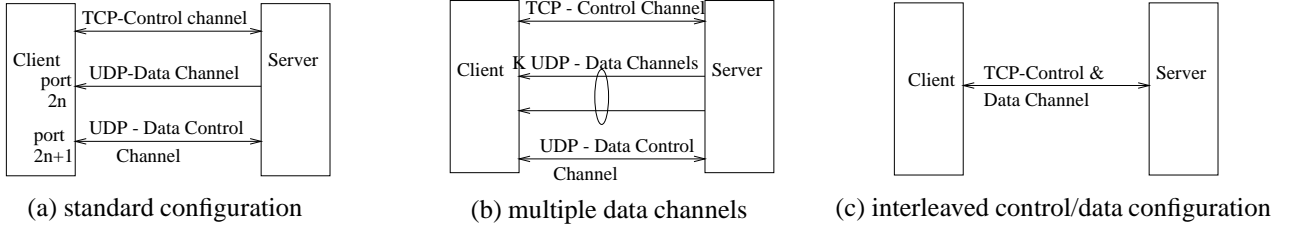


Fig. 2. Three different mappings

MMS (Microsoft Media Server protocol) used by Microsoft Windows Media [8]. RTSP is an application-level protocol that provides VCR-like control over the delivery of multimedia data. The standard protocol used by the server to describe a presentation to a client is SDP (Session Description Protocol) [4].

A.2 Data Channel

Streaming real time multimedia data over a best effort network requires a higher level protocol. This protocol needs to provide a mechanism that allows a client to reconstruct temporal relations among the data packets and detect packet loss to compensate for jitter and loss. The sender should also monitor the transmission and reception of the data and adapt the sending rate to respond to long term bandwidth constraints.

The standard protocol in this domain is RTP (Realtime Transport Protocol) [10]. RTP provides timestamps, sequence numbers, payload type identification, source identification, and other mechanisms. How a particular type of payload data should be carried in RTP is defined in a *payload format specification*. For example, RFC2250 [5] specifies the transport of a MPEG-1/MPEG-2 stream in RTP. Other examples of data protocols include RDT (Real Data Transport) and PNA used in RealSystem [9], and MMS used in Windows Media [8].

The client and server use a duplex data control channel to exchange timely information on the transmission and reception status of the data. They can take actions based on the reported loss and jitter. For example, if the client continues to report a high loss rate, the server can switch to a lower bit rate to transmit the presentation. RTP defines an auxiliary control protocol, RTCP (RealTime Transport Control Protocol), for this data control channel.

B. Mapping channels to network transports

There are different mappings of the control and data channels to actual network connections, as shown in Figure 2. UDP transport can be over either a unicast or a multicast address. In the current Internet, the network connections of clients are heterogeneous. Some clients sit behind a firewall that blocks UDP traffic or only allow TCP communications on certain ports, and some have access to multicast while others don't. In order to reach as many clients as possible, actual commercial streaming systems often support several streaming protocols and transport mechanisms, more specifically, streaming in TCP 2(c) and HTTP are widely supported. For example, in RealPlayer, a user can specify what protocol to use, or can let the player automatically select the best available transport mechanism. The MMS protocol used by Windows Media automatically looks for the best transport, unless the user specifies explicitly the type of transport to use in the URL: mmsu for UDP transport, mmst for TCP transport, http for HTTP streaming.

C. The Interoperability Problem and Translation Proxy

Currently, streaming servers and clients from different organizations cannot interoperate directly due to differences in the protocols used in the signalling and data channels. Server/client solutions are often tightly

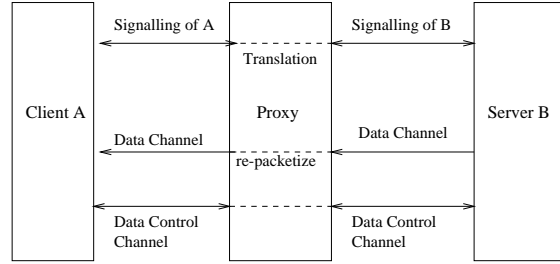


Fig. 3. Multimedia Streaming Translation Proxy

coupled and typically use different signalling protocols. Even when the same signalling protocol is used, they are typically incompatible (due to different extensions or incompliant usage). In the data path, the same difficulties are experienced. Different data transport protocols may be used by different platforms. When the same protocol is used, implementation details can still differ. Even though there is a standard specification for how to packetize a specific media type into RTP, the actual systems comply to the standard to different degrees.

A natural solution to this interoperability problem is to deploy a translation proxy between the server and client as shown in Figure 3. The translation proxy would accept requests made by client A, parse them and generate corresponding requests for server B. Upon receiving a response from server B, the proxy parses the response and generates a response to client A. On the data path, the proxy receives data packets from server B, repacketizes them according to client A, and then streams the packets to client A. For the data control channel, the proxy exchanges control information with Client A and Server B.

To work properly, the translation proxy needs to act both as a server for client A and as a client for server B. In the next section, we describe experiments that explore the interoperability problem among several streaming platforms. Based on the understanding of the platforms obtained from these experiments, we implemented a prototype and studied the translation proxy design problem.

III. A PRELIMINARY INVESTIGATION OF THE INTEROPERABILITY PROBLEM

In order to understand the feasibility of a translation proxy and to correctly design such a proxy, we performed a study to determine the differences among the following platforms:

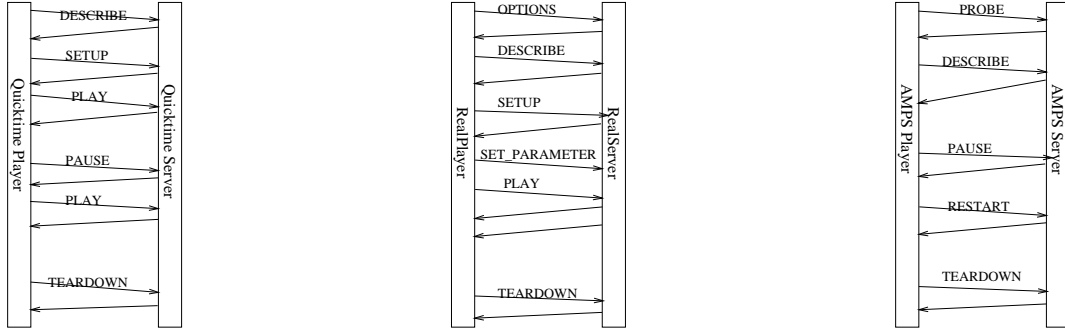
- AMPS server and client [1] developed at Dept. of Computer Science at Univ. of Massachusetts, Amherst
- QuickTime player and Darwin Streaming Server (DSS) from Apple Inc [7]
- RealPlayer and RealServer from RealNetworks Inc. [9]

All of these platforms support the standard protocols RTSP, RTP/RTCP and SDP to varying degrees. We focus on the streaming of MPEG-1 system stream because it's a standard codec supported by all three platforms.

In Section III-A, we describe the methods used to investigate the protocol implementation details. In Section III-B, we provide some background information about the MPEG-1 system stream. In the remainder of the section, we highlight the main features of the three platforms. We assume the reader is familiar with RTSP, RTP/RTCP and SDP (otherwise please refer to [4, 10, 11]).

A. Methods for analyzing the platforms

While we have full access to information regarding the AMPS testbed, there is little documentation available on low level details concerning QuickTime and RealSystem. To understand RTSP signalling, RTP



(a) QuickTime RTSP signalling

(b) Real RTSP signalling

(c) AMPS RTSP signalling

Fig. 4. RTSP signalling of three platforms

packetization and transmission, and RTCP usage of these two platforms, we first collected RTSP, RTP/RTCP traces exchanged between the two different client-server pairs. For example, we launched a QuickTime player on a local machine and opened a RTSP URI pointing to a DSS server, while at the same time running *windump* to capture the RTSP and RTP/RTCP packets flowing between the server and client. Next we analyzed the traces. In the case of the RTSP and SDP traces, we inferred the meaning of the extension headers and then tried to confirm these inferences through further experimentation. In the case of RTP/RTCP traces, we wrote a script to parse the traces and compare the RTP packet traces with the original MPEG-1 file to understand the RTP packetization. We also referred to the source code of the Darwin Streaming Server.

B. MPEG-1 System Stream

MPEG-1 system coding [3] defines how to multiplex multiple MPEG-1 audio and video streams (elementary streams) into a single system stream, using timestamps to provide system-level functions such as synchronization of audio and video, and random access into the stream. Basically, a MPEG-1 system is made up of a series of *packs*. Each pack is made up of a pack header followed by the system header (required for the first pack, optional for the remaining packs) and multiplexed packets from the audio and video streams. In the pack header, the SCR (*System Clock Reference*) field specifies the time when the stream bytes of the pack should enter the decoder. The packet header of each packet contains two timestamp fields specifying the time when the packet should be decoded and presented respectively. All these timestamps are based on a single 90kHz system clock.

C. Main Features of QuickTime

Figure 4.(a) shows a typical RTSP signalling sequence between a QuickTime player and server. It's an instance of the signalling sequence presented in Section II-A.1. QuickTime's usage of RTSP and SDP conforms to the standard. However, a number of extension headers are introduced. As these headers are not critical for the signalling, we omit them here.

Given an MPEG-1 system stream, the QuickTime server divides the stream into chunks of equal size (1438 bytes) and encapsulates them into RTP packets. The exception is that a pack header in the stream will always start a new packet. The timestamp of the RTP header is synchronized to the SCR of the pack header. The payload type field of the packet is set to 96, which is dynamically mapped to MPEG-1 system stream by SDP.

RTCP packets are exchanged between the QuickTime player and QuickTime server. RTCP is used for several purposes. First, the *sender reports* sent by the server are used to maintain inter-stream synchroniza-

tion. Secondly, the QuickTime player uses RTCP to send reception statistics so that the server can perform quality adaptation as needed. It's not clear whether any quality adaptation for an MPEG-1 movie is performed. Third, the *Receiver Report* packets from QuickTime player also serve as keep-alive messages to the server.

D. Features of RealSystem

Figure 4(b) shows the RTSP signalling between RealPlayer and RealServer. RealSystem supports streaming in both the proprietary RDT and the standard RTP. When RealPlayer initiates the streaming of a presentation with a *SETUP* request, it lists the kinds of data protocols it supports in order of preference.

A Challenge/Response mechanism is used in RealSystem. When RealPlayer connects to a server, the first RTSP message (*OPTION* request) has a *ClientChallenge* header, which RealServer answers with a *RealChallenge* header in the response. In the *SETUP* request sent next by RealPlayer, there is another challenge/response exchange. For older versions of RealPlayer, if the server fails to answer the second challenge correctly, RealPlayer reports an "Invalid server" error and tears down the connection. For newer versions of RealPlayer, a server's failure to answer the challenges doesn't cause the client's closing down of the session.

When adaptive streaming is supported for the presentation, RealPlayer could use a *SET_PARAMETER* request to set the delivery bandwidth based on its network connection. RealSystem uses the *SET_PARAMETER* message with a *Ping: Pong* header as a heart-beat message that probes the liveness of the server (the client) and announces its own liveness.

RealSystem extends SDP by introducing a number of attribute fields. For example, *WindowWidth* is used to specify the width of the rendering window, and *Preroll* is used to specify the amount of data (in terms of milliseconds) the client should buffer before starting playback.

RealSystem packetizes a MPEG-1 system stream by chopping the stream into equal size (1440 bytes) chunks and encapsulates them in RTP packets. The payload type header field is set to 101, which is mapped to an MPEG-1 system stream in the description of the presentation. An eight-byte header extension is used. The meaning of the extension header is unknown.

According to the traces we gathered, RealServer periodically sends a *Sender Report* to the client. RealPlayer doesn't send any RTCP packets to the server.

E. AMPS testbed

The AMPS testbed is a prototype research system which handles hybrid multicast/unicast transmission schemes. In our study, it used periodic broadcast and patching [1]. Both schemes assume that the client can simultaneously receive from multiple channels (see Figure 2(b)). For each incoming client request, the server calculates a transmission schedule and sends back the corresponding reception schedule to the client. The calculation of the transmission schedule needs to take into account several factors including the one way delay between client and server, client's bandwidth capacity etc.

Figure 4(c) illustrates the signalling between the current AMPS server and AMPS client. The *PROBE* request is introduced to measure the round trip time between the client and the server. In the *DESCRIBE* request, the one way delay calculated from the round trip time is sent in the *OneWayDelay* header. The server calculates the reception schedule for the client, and sends the reception schedule as part of the description of the presentation in response to the *DESCRIBE* request using SDP. The client extracts the reception schedule from the response and begins to listen on the channels accordingly.

The AMPS system adopts the packetization scheme for MPEG-1 video stream as specified in RFC

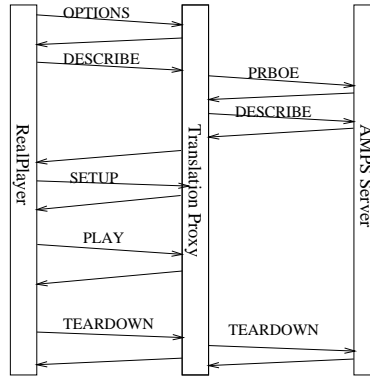


Fig. 5. RTSP Control Signalling between RealPlayer, Translation Proxy and AMPS Server

2250 [5]. In this scheme, RTP extension headers indicating the temporal dependency of the payload are introduced, so that the client can perform certain loss recovery and resynchronization. RTCP is not used in the AMPS testbed.

E.1 Summary

It is clear from our study that, for the three platforms described above, we cannot directly connect one of the clients to a server from a different system, due to the differences in RTSP signalling, RTP packetization and streaming.

IV. EVOLVING A PROXY-BASED SOLUTION

In this section, we explore the design issues associated with a translation proxy. In Section IV-A, we present our prototype proxy study. In Section IV-B we study the design requirements. In Section IV-C we overview the overall design of the AMPS proxy. In Section IV-D a design for the control plane is presented with an emphasis on control translation.

A. Translation Proxy Prototype

We based our translation proxy prototype on a proxy from Apple Corp. [6] that performs simple receiving and forwarding between a QuickTime server and a QuickTime player. Basically, for every RTSP request received from the client, the proxy extracts some information and/or modifies the request, and then forwards it to the server. Similarly, for every RTSP response received from the server, the proxy extracts useful information, modifies the response and then forwards it to the client, and starts tasks to receive and forward RTP packets and RTCP packets.

We focused on enabling streaming from an AMPS server to a QuickTime Player and RealPlayer because supporting other cases would be straightforward once we resolved these two cases. The following functionalities were added to the proxy: translation of the RTSP messages, a network reception module for the AMPS server, and repacketization and transmission of the MPEG-1 system stream. For these two cases, RTCP support in the proxy is not required. Below, we focus on some of the challenges that we encountered in the prototype study.

A.1 Control Translation

We illustrate the translation of control messages by examining the signalling required for a RealPlayer to connect to an AMPS server through the proxy (see Figure 5). The translation of RTSP messages between

the AMPS server and QuickTime player is similar. The proxy emulates a RealServer or a DSS when communicating with a RealPlayer or a QuickTime player. Although both RealSystem and QuickTime introduce some RTSP extension headers and SDP fields whose meanings we haven't been able to determine, our experiment showed that these extensions are optional and omitting/ignoring them doesn't affect the streaming of an MPEG-1 data to these two players.

For older versions of RealPlayer, the proxy needs to respond correctly to the challenges sent by the RealPlayer, otherwise the RealPlayer won't continue sending request to the proxy. It turns out this problem can be solved by making use of the free basic version of RealServer. We installed such a RealServer on the same machine as the proxy. When a RealPlayer of old version (the version of the player can be identified by the *User-Agent* header in the request sent by the client) sends a request with a challenge header to the proxy, the proxy sends the request to the RealServer and extracts the challenge header from the response returned by the RealServer. In this way, the proxy obtains a correct response to the challenge.

It is worth mentioning that by examining the *User-Agent* header in the first request sent by the client, the proxy can identify whether it's a RealPlayer or a QuickTime player. It can also identify the server type by simply forwarding the first request to the server, and examining the *Server* header in the response returned from the server.

A.2 Network Reception, Data Packetization and Transmission

On the data path, the translation proxy creates a thread to receive RTP packets from the AMPS server according to the reception schedule. On receiving a RTP packet, the thread takes the payload of the packet and stages it into a buffer based on the offset specified in the packet header. Another thread is started to transmit the data to the client. This thread periodically calculates the amount of data that needs to be sent, re-packetizes the data using Real or QuickTime's scheme, and then sends the packets to the client. Packetization of the MPEG-1 data requires scanning the buffer for the pack header. Our experiment showed that the packetization process can be performed in real time at the proxy.

An unexpected problem arises when we connect a QuickTime player to the AMPS server through the proxy. Streaming the data at the same rate from the proxy to a RealPlayer and to a QuickTime player results in smooth playback at the RealPlayer, but causes playback problems with the QuickTime player. A careful study of this problem revealed that QuickTime player is very sensitive to the sending rate. Sending the video faster or slower than the playout rate causes a playout problem with QuickTime player. One possible explanation is that the receiving buffer used by QuickTime player is very small so that it is easy to either underflow or overflow the buffer when the sending rate is too low or too high. Consequently, in the proxy, the transmission of RTP packets is carefully scheduled such that the sending rate equals the playout rate.

A.3 Conclusions from the Prototype Study

We have tested the prototype proxy in a local network environment. The video plays out smoothly on the client side, and the user perceived quality is comparable to that of a Darwin Streaming Server streaming to a QuickTime Player.

Through the prototype study, we confirmed the feasibility of constructing a translation proxy. We also found out that because the architecture of the prototype proxy is not modular, it would be difficult to extend to handle a variety of clients and servers. In the next section, we study the design criteria for a translation proxy.

B. Design Criteria for a Translation Proxy

The desirable features required of a software architecture for a translation proxy include:

- *Compatibility with AMPS Proxy Project:* Concurrent to this interoperability study, there is a project focusing on the development of a multimedia streaming proxy as part of the AMPS project. The goal of the project is to develop a research platform on which services such as caching, transcoding etc. can be conveniently deployed and tested. Translation in the control and data paths is one such service. Thus consistency with this general-purpose proxy project is an important design criterion.
- *Modularity:* Modularity will simplify adding support for new clients/servers or new codecs. Given a set of APIs and an SDK, interested parties can develop modules to handle translation in the control path and the data path.
- *Extendibility:* Our translation proxy needs to interact with systems for which we have limited knowledge and that are possibly changing over time. Consequently, the design of the system should make it as easy as possible to accommodate changes in control signalling of related systems.
- *Efficiency:* This is a crucial requirement for a proxy that handles multimedia streaming which has realtime constraints.
- *Robustness and Security:* The proxy needs to be robust to different kinds of failure conditions, overload conditions, and malicious users.

C. AMPS Testbed Proxy Design Overview

To provide for the flexible development and deployment of various services, the AMPS proxy has adopted a layered architecture as shown in Figure 6. The control plane provides proxy control functionalities: handling control signalling with clients and servers, providing a GUI to an end-user to monitor and control the proxy, and monitoring feedback from clients and servers to initiate flow control when necessary. The lowest level, the data plane, deals with the manipulation of the streaming media and provides actual proxy services to the upper layer. Modules that handle the reception, various transformations and network transmission of multimedia data reside in this layer. The service plane lies in the middle level. It includes resource managers for disk, memory and network, utilities such as a thread pool, and more importantly, a scheduler. The scheduler schedules data plane modules to provide the services requested by the control plane, using global knowledge among all ongoing services. The control plane communicates with the scheduler using a well-defined API.

In the control plane, there are a number of server control modules and client control modules. Each server control module provides functionalities required for the proxy to act as a server to certain type of client. These include parsing and handling the requests from the client, sending out the responses, etc. The Scheduler is called to set up the data plane services required to service the request. For example, when connecting an AMPS server with a RealPlayer, the AMPS-Real Translation Module is scheduled to repacketize the data using Real's scheme. Each client control module provides the functionalities required by the proxy to communicate with a certain type of server.

The Listener acts as an event dispatcher. It monitors a set of TCP ports for connections made by the clients. When a client connection arrives, the Listener dispatches a thread that runs the corresponding server control module to handle the client. Another function of the Listener is to monitor incoming requests for those client sessions that have only occasional signalling going on. As the proxy needs to handle multiple client sessions simultaneously, the use of the Listener enables the control plane to handle control signalling of all sessions in an efficient manner.

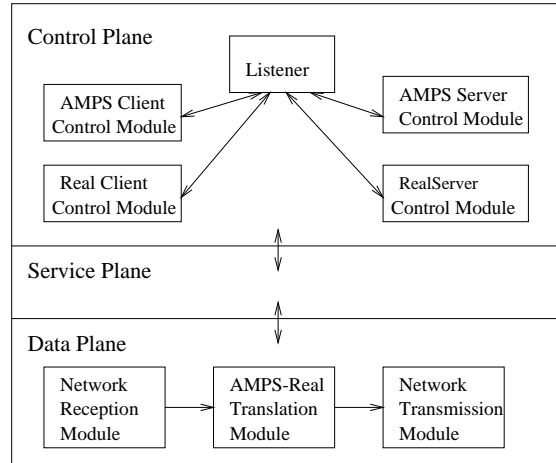


Fig. 6. AMPS Proxy Architecture

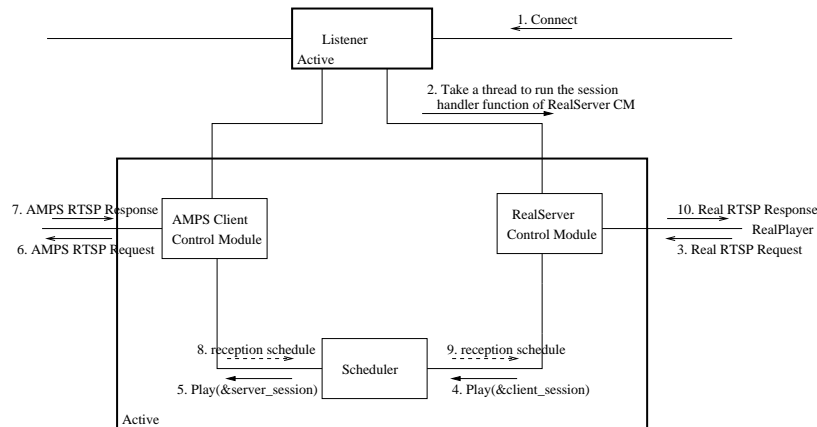


Fig. 7. Interactions among modules to serve a session

D. Control Module Interactions

Figure 7 shows the interactions among the control plane modules when a RealPlayer connects to an AMPS server through the proxy. Given a request from RealPlayer, the RealServer Control Module maps it to a call to the Scheduler, translates the returned result to a Real RTSP response, and sends the response to the client. The Scheduler, while scheduling data plane modules to stream the requested presentation, finds it needs to request the presentation from the origin server. The scheduler calls the AMPS Client Control Module to send the request in AMPS RTSP format to the server.

V. SUMMARY AND FUTURE WORK

To provide interoperability between different multimedia streaming systems is a complex and changing problem, as new transmission schemes and codecs are being developed. We hope this problem can be finally solved when a suite of standards are widely supported by various streaming systems in a compatible way. In our work, we have explored the idea of using a translation proxy to solve this problem. Our prototype study mainly handles the case where standard protocols are used, whereas the software architecture we proposed is general and open such that support for proprietary streaming protocols and HTTP streaming can be added.

The design and implementation of the proxy is an ongoing project. Several design problems have still to

be addressed, such as the design of the translation module for handling data path translation. After implementing the proxy, we plan to perform experiments to evaluate our design in terms of the ease of adding new client/server support in the proxy and the throughput of the control plane. Currently, the translation proxy has been deployed as a stand-alone application. In the future, we would like to investigate the deployment flexibility of the proxy, for example, we would like to study the possibility of deploying the proxy as a plugin for the client browser which could be dynamically downloaded and launched when a translation service is needed. We would also like to compare the plugin approach (which we described in the introduction) with the translation proxy approach in terms of performance and the amount of development effort required.

VI. ACKNOWLEDGEMENT

The authors would like to thank their colleagues in the AMPS project group, particularly Michael K. Bradshaw, who has provided many valuable suggestions on the design of the control plane and the writing of this paper.

REFERENCES

- [1] Michael K. Bradshaw, Bing Wang, Subhabrata Sen, Lixin Gao, Jim Kurose, Prashant Shenoy, and Don Towsley. Periodic broadcast and patching services - implementation, measurement, and analysis in an internet streaming video testbed. In *Proceedings of ACM Multimedia System 2001*, 2001.
- [2] Burst.com. Burstware delivery system. <http://www.burst.com>.
- [3] MPEG committee. Iso/iec international standard 11172, coding of moving pictures and associated audio for digital storage media up to about 1.5 mbits/s, 11 1993.
- [4] M. Handley and V. Jacobson. SDP: Session description protocol, April 1998.
- [5] D. Hoffman, G. Fernando, V. Goyal, and M. Civanlar. RTP payload format for MPEG1/MPEG2 Video, January 1998.
- [6] Apple Inc. Darwin streaming server. <http://www.publicsource.apple.com/projects/streaming/>.
- [7] Apple Inc. Quicktime. <http://www.apple.com/quicktime>.
- [8] Microsoft Inc. Windows media technology. <http://msdn.microsoft.com>.
- [9] RealNetworks Inc. Realsystem streaming platform. <http://www.realnetworks.com>.
- [10] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: A transport protocol for real-time applications, rfc 1889, January 1996.
- [11] H. Schulzrinne, A. Rao, and R. Lanphier. Real time streaming protocol (RTSP), rfc 2326, April 1998.
- [12] Ellen(Xiaolan) Zhang. Realsystem plugin for amps server. 2000.