CACHING OF STREAMING MULTIMEDIA ON THE INTERNET

CS736 PROJECT REPORT

Anurag Gupta
Kiran Chitluri

{anurag, kiran}@cs.wisc.edu

Computer Sciences Department
University of Wisconsin, Madison
Madison, WI – 53706

December 21, 1998

ABSTRACT

Real Time Streaming Multimedia is growing at an increasing rate on the Internet. Caching the multimedia files as they are routed through company firewalls and proxies is the ideal way to incur big savings on Internet traffic and enable high-speed delivery of media content to the end user. Currently *Inktomi Corporation* has developed the only commercial implementation of a Streaming Multimedia Caching Proxy. By studying the **Real Time Streaming Protocol** (*RTSP*) used for control of real time continuous media and the **Real Time Protocol** (*RTP*) used for media data delivery and using a Reference Implementation of a player and server supporting streaming media, we have developed a free academic implementation of a Proxy capable of caching streaming media, for the research community.

1. INTRODUCTION

Multimedia on the Internet is growing at a dramatic rate with the number of media URL's tripling over the last nine months. Bandwidth improvements in the near future will only aid in making multimedia more popular on the Internet. Today, Streaming Media is the most popular way to transmit multimedia over the Internet because it allows the user to experience the media as it is being received instead of forcing the user to wait for the entire file to be downloaded before it can start playing it. Since these media files change rarely, they lend themselves well to caching. By servicing requests for multimedia files from cached copies at Proxies, organizations can save big on expensive Internet Traffic. In addition, high-speed delivery of the media content is possible since the media files are being brought closer to the end user.

RealNetwork Inc. is one of the largest companies that support streaming continuous multimedia. Most Streaming Media Servers today are ones provided by RealSystems and over 85% of streaming media on the Internet are in Real formats **[1]**. Streaming multimedia has only recently been publicly available. In April 1995 Real Audio introduced Streaming Audio and shortly after in August 1995 Streamworks introduced Internet streaming video. Currently, Inktomi Corporation's Traffic Server is the *only* commercial implementation of a multimedia caching proxy.

We describe the protocol specifications for Streaming Multimedia essential in the design and implementation of an intermediate proxy capable of caching multimedia files. We also discuss the implementation of our application level proxy designed to interact with a reference implementation of the Real Time Streaming Protocol (RTSP) client and server downloaded from the RealNetworks Web site.

This report is organized as follows. In **Section 2**, we explain the protocols used for the transmission of streaming multimedia. **Section 3** discusses the different types of proxies. In **Section 4**, we discuss the details of the reference implementation of the client and server and how they exchange control messages and data. In **Section 5**, we describe the implementation of our application level proxy and our caching method. We also discuss the changes we made to the reference implementation of the client. **Section 6** presents the results and in **Section 7**, we discuss possible future work.

2. PROTOCOL STUDY

A new set of protocols have recently been proposed to support the timely delivery of Real Time Streaming Multimedia. The three major protocols that we studied include the Real Time Streaming Protocol (RTSP), Real Time Protocol (RTP), and Real Time Control Protocol (RTCP).

RTSP is an application-level multimedia presentation 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, using the Transmission Control Protocol (TCP) or the User Data Protocol (UDP). All known RTSP servers to date are TCP-based, though the specification has provisions for a UDP-based version.

Real Time Transport Protocol (RTP) is a UDP packet format for multimedia data streams responsible for the actual data transmission. RTP contains a set of conventions that provide end-to-end network transport functions suitable for applications streaming real-time data, such as audio, or video over multicast or unicast network services. RTSP and H.323 (used for Internet Telephony) use RTP for data transmission.

Real Time Control Protocol (RTCP) is a control protocol that is a part of RTP. RTCP helps with lip synchronization, QOS management, and packet loss recovery. RTCP packets are multicast periodically to the same multicast group as data packets.

2.1 REAL TIME STREAMING PROTOCOL (RTSP)

The Real Time Streaming Protocol (RTSP) (RFC 2326) is an Internet Engineering Task Force (IETF) Proposed Standard for the control of streaming media on the Internet. It was jointly submitted to the IETF in

October 1996 by RealNetworks and Netscape Communications Corp. and was published as an IETF Proposed Standard in April 1998.

RTSP is a control protocol that initiates and directs delivery of streaming multimedia from media servers. RTSP does not deliver data (though the RTSP connection may be used to tunnel RTP traffic for ease of use with firewalls and other network devices). RTP and RTSP will likely be used together in many systems, but either protocol can be used without the other. A RTSP session is not dependent on a Transport Level connection such as a TCP connection. During a RTSP session, a client can open multiple reliable transport connections to the server or alternatively it may use a connectionless protocol such as UDP.

A common question asked is why not use HTTP for streaming media off web servers. There are several reasons. First of all, HTTP uses TCP, which enforces reliability instead of timeliness needed in a streaming multimedia. TCP does not support multicast, which is essential in large-scale multimedia delivery over the Internet. RTSP may interact with HTTP on the initial servicing of a client request off a web page, but unlike HTTP, RTSP Requests always contain the absolute URL. In addition, HTTP is not well suited for time based seeking of files for delivery. Finally, RTSP server needs to maintain state in almost all cases versus the stateless nature of HTTP. RTSP introduces new methods in order for the server to maintain this state information.

RTSP is a text based protocol and uses the ISO 10646 character set in UTF-8 encoding (RFC 2279). RTSP messages can be carried over any lower-layer transport protocol that is 8-bit clean. The requests contains methods (method to be performed on a resource), the object the method is operating upon and the parameters to further describe the method. **[2]**

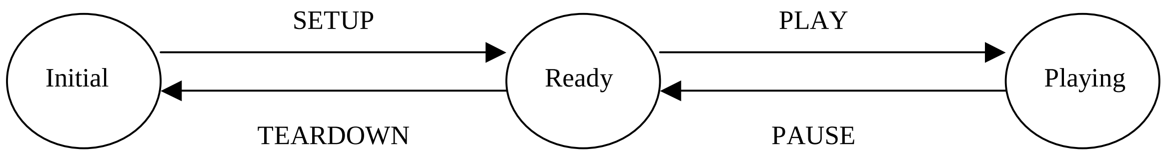
2.1.1 RTSP METHODS

The RTSP methods as defined in the protocol are:

OPTIONS	*Get available methods*
SETUP	*Establish transport*
ANNOUNCE	*Change description of media objects*
DESCRIBE	*Get (low-level) description of media objects*
PLAY	*Start playback, reposition*
RECORD	*Start recording*
REDIRECT	*Redirect client to new server*
PAUSE	*Halt delivery, but keep state and server resources allocated*
SET_PARAMETER	*Device or encoding controls*
TEARDOWN	*Removes state*

2.1.2 RTSP State Transitions

Fig. 1 below describes the state transitions using the RTSP defined methods.



SETUP	*Start an RTSP session and resource allocation for a stream*
PLAY and **RECORD**	*Start data transmission on the stream*
PAUSE	*Temporarily halt a stream without freeing server resources*
TEARDOWN	*Free resources associated with stream and end of a session*

Fig. 1

RTSP is multi-server capable, meaning that each media stream in a multi-stream session can reside on separate servers for the same presentation. The client can establish concurrent control sessions with the individual servers. Thus, RTSP is extensible, secure, transport-independent, and usually sends data out-of-band in a different protocol, RTP in our case.

2.2 REAL TIME PROTOCOL (RTP)

RTP is a Transport Protocol for real time applications that provides semantics for end-to-end delivery services of real time data. These delivery services include payload type identification, sequence numbering, timestamping and delivery monitoring. Applications typically run RTP on top of UDP to make use of its multiplexing and checksum services. However, RTP may be used with other suitable transport or network protocols. It can support data transfer to multiple destinations using multicast distribution if provided by the underlying network.

RTP itself does not provide any mechanism to ensure timely delivery of data or provide other quality-of-service guarantees, but relies on lower layer services to do so. It does not guarantee delivery or prevent out-of-order delivery nor does it assume that the underlying network is reliable and delivers packets in sequence. Data can be sent UDP unicast, multicast UDP or interleaved with the RTSP control stream. **[3]**

2.3 Real Time Control Protocol (RTCP)

The RTP Control Protocol (RTCP) is based on the periodic transmission of control packets to all the participants in the session, using the same distribution mechanism as the data packets. The multiplexing of data and control packets is handled by the underlying protocol. It provides many services to maintain a high quality of data transmission:

- **QoS monitoring and congestion control**

  RTCP packets contain information for quality-of-service monitoring. Since they are multicast, all session members can survey how the other participants are faring. Applications that have recently sent audio or video data generate a sender report. It contains information useful for inter-media synchronization as well as cumulative counters for packets and bytes sent. These allow receivers to estimate the actual data rate. Session members issue receiver reports for all video and audio sources they have heard from. The information in the reports include packet loss, highest sequence number received and inter-arrival jitter. **[3]**

- **Inter-media synchronization**

  RTCP sender reports contain the real time (wall clock) and a corresponding RTCP timestamp. These two values allow for lip-syncing of audio and video.

- **Identification**

  RTP data identify their origin through a random 32-bit identifier. RTCP messages contain an SDES (source description) packet containing the canonical name, a globally unique identifier of the session participant.

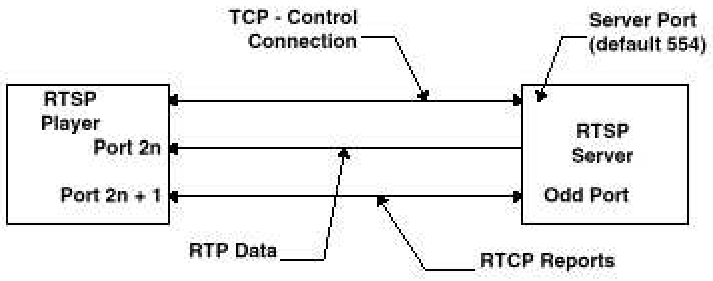
- **Session Size estimation and scaling**

  RTCP packets are sent periodically by each session member. The desire for the latest control information must be balanced with the desire to limit control traffic to a small percentage of the total traffic.

2.4 Example

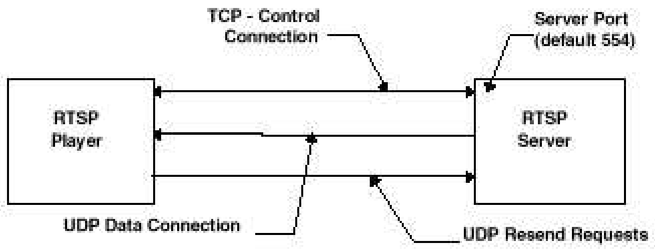
RealNetworks ***G2 RealServer*** uses Two Packet formats for sending media data to an RTSP client:

a) Standard Real Time Transport Protocol: RTP



This protocol uses RTCP for data flow control and for ensuring that the quality-of-service is maintained.

b) RealNetwork's Real Data Transport: RDT



Instead of using RTCP, RDT uses a second simplex UDP path from client to the server to request for lost UDP media packets.

Thus, Streaming Media can be delivered using the above described three protocols. **Fig. 2** describes the general communication between the client and server using these protocols.

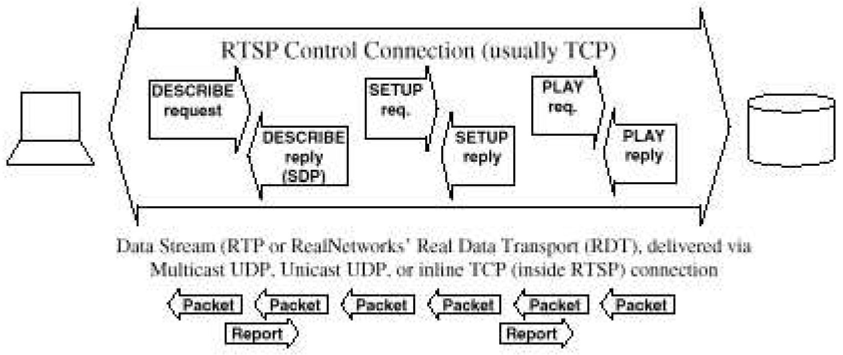

RTSP Control Connection (usually TCP)
DESCRIBE request
DESCRIBE reply (SDP)
SETUP req.
SETUP reply
PLAY req.
PLAY reply
Data Stream (RTP or RealNetworks' Real Data Transport (RDT), delivered via Multicast UDP, Unicast UDP, or inline TCP (inside RTSP) connection
Packet
Report

Fig. 2 RTSP – "The Internet VCR Remote Control Protocol "

3. PROXY

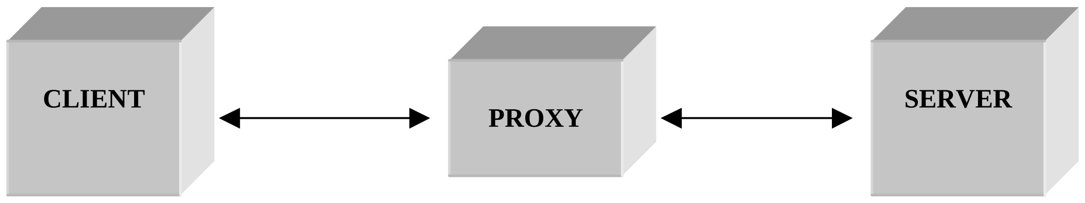
A proxy server is a server that acts as an intermediary between an end user and the Internet. The end user is typically part of an enterprise. By directing all traffic from the end user to the Internet through the proxy, the enterprise can ensure security, administrative control, and caching service. A proxy server is associated with or part of a gateway server that separates the enterprise network from the outside network and part of a firewall server that protects the enterprise network from outside intrusion.

A proxy server receives a request for an Internet service from a user. If it passes filtering requirements, the proxy server looks in its cache to see if it can service the request. If it finds the requested resource in its cache, it returns it to the user without needing to forward the request to the Internet. If the resource is not in the cache, the proxy server, acting as a client on behalf of the user, requests the resource from the server out on the Internet. When the resource is returned, the proxy server relates it to the original request and forwards it on to the user. An advantage of a proxy server is that its cache can serve all users. If one or more Internet sites are frequently requested, these are likely to be in the proxy's cache, which will improve user response time.

Proxies usually are of two types:

1. Application Level Proxy

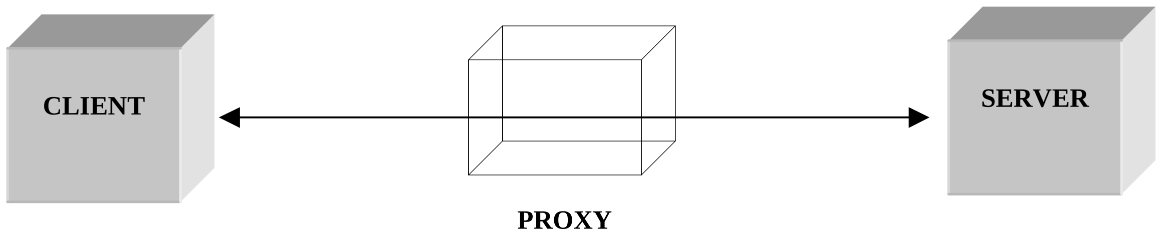
An Application Level Proxy relies on the application having knowledge of the proxy. The application is aware that it is connecting to the proxy and hence, must be aware of the proxy protocol. But it does not need to know the low-level protocol details.



Client has hostname and port number of Proxy

2. Transparent Proxy

The Transparent Proxy operates by monitoring the network traffic and only letting through those connections that match certain protocols. The applications are not aware that their requests are going through the proxy. Hence, the client's need not be modified or configured.



Client has no knowledge of Proxy

4. RTSP REFERENCE IMPLEMENTATION

We downloaded the RTSP Reference Implementation of the client and server from the **RealNetworks** Web site. This implementation, as the name suggests, is the only a reference implementation. It does not implement many of the features of the RTSP protocol and is basically a strip down version of RealNetworks Real Player and Real Server. Specifically, it has the following limitations:

- Supports a single stream
- No RTCP support
- Supports a limited type of WAV files only
- Does not support video
- Supports a limited number of audio payload types

So, our goal was not to improve upon any of these features or add any other feature to the given reference implementation of the client and server. Given these limitations and constraints, we wanted to design and implement an application level proxy that was capable of caching the streaming media. So, our next step was to study the reference implementation and see how control messages are exchanged between the client and server and how data is transferred between the server and the client.

The next section describes the RTSP control messages that are exchanged between the client and server in the reference implementation.

RTSP Handshaking

The Reference Implementation's implementation of the RTSP handshaking is presented in **Fig. 3**.

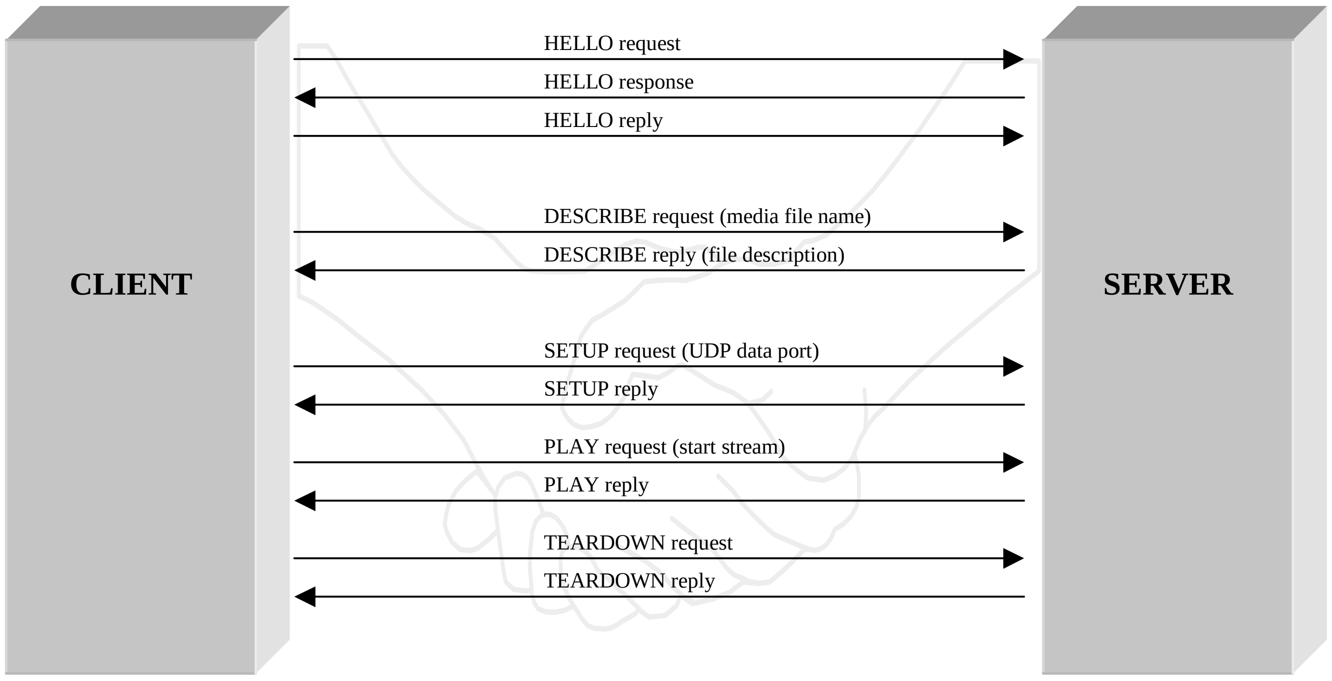

CLIENT
SERVER
HELLO request
HELLO response
HELLO reply
DESCRIBE request (media file name)
DESCRIBE reply (file description)
SETUP request (UDP data port)
SETUP reply
PLAY request (start stream)
PLAY reply
TEARDOWN request
TEARDOWN reply

Fig. 3 RTSP Message Exchange

Description of RTSP Dialog

1. *HELLO*

   The Client first sends a HELLO Request to the server. The server responds with a HELLO Response. Upon receiving the HELLO Response, the client sends a HELLO Reply back to the server.

   In the reference implementation, the RTSP control is done over a TCP connection, which is a reliable protocol, yet this three way initial HELLO messages are exchanged. This is because the RTSP protocol does not require that the control exchange be done over a reliable connection. It can be done over an unreliable connection like UDP also.

2. *DESCRIBE*

   Upon sending the HELLO reply, the client then builds a DESCRIBE Request in which it includes the media file name that it wants to play and sends it to the server. If the server has the requested file, it responds with a DESCRIBE reply which includes the file characteristics (payload type, etc). If the server does not have the specified file, then it sends an error message to the client

3. *SETUP*

   After receiving the DESCRIBE Reply, the client opens a UDP port for receiving the actual data packets and sends the UDP port number on which it will wait for the data packets from the server, as part of the SETUP Request. On receiving the SETUP Request, the server sends a SETUP reply.

4. *PLAY*

   After receiving the SETUP reply, the client sends a PLAY Request to the server, requesting the server to start the transmission of the data packets of the requested file. On receiving the PLAY Request, the server sends a PLAY Reply, opens a UDP port for sending the data packets and starts sending the data packets on the stream. After the client receives the PLAY Reply, it starts listening on the UDP port for the data.

5. *TEARDOWN*

   The client requests the end of the session by sending the TEARDOWN Request to the server. The server sends a TEARDOWN reply when it receives the TEARDOWN request.

DATA TRANSMISSION

The reference implementation transmits data between the server and client through a UDP connection. A one way UDP connection is established from the server to the client. The server pumps the RTP data packets onto the UDP connection to the client at a constant rate. The end of the media file is indicated by sending an empty RTP packet with the RTP header extension. The RTCP protocol is not implemented in the reference implementation. Therefore, no flow control and quality control is done. **Fig. 4** shows the RTSP and RTP communication between the client and server in the reference implementation.

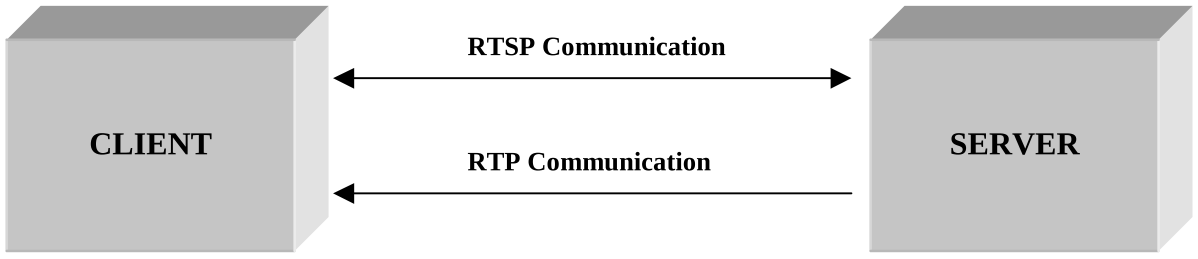


Fig. 4 RTSP and RTP Communication

5. PROXY IMPLEMENTATION

With the addition of an Application Level Proxy between the client and server, the control messages exchanged between the client and server have to be modified. We made the following changes in the control messages exchanged between the client and server to incorporate our proxy:

1. *Change in the HELLO Message*

   In the reference implementation, the HELLO request is sent by sending a message to the server from the client. We changed this message by making the client send the server URL and the media filename in the initial HELLO request.

   The proxy, on receiving the request from the client, parses the server URL and the filename from the message. It then checks to see if it has a cached copy of the file. If it has, it starts servicing the client request without making a connection with the server. If it does not have the requested media file, it establishes a connection to the server using the server URL parsed from the client message and forwards the HELLO request from the client to the server without the server URL and media filename. It then receives the server reply and passes it to the client. For the remainder of the RTSP communication, it simply acts as an intermediary for the delivery of the messages, taking the messages from the client and sending them to the server and vice versa.

2. *Change in the SETUP Message*

   In the reference implementation, the client sends the port number to the server on which it will receive the data packets in the SETUP message.

   When we introduce the proxy, it must open a UDP port to receive the data packets from the server and another data port to send the data packets to the client. Hence, the number of the UDP data port that it opens to receive the data packets has to be sent to the server so that the server can send the data packets to this port. Hence, we parse the data port number sent by the client in the SETUP request and replace it with the UDP data port number of the proxy. Similarly, in the SETUP reply from the server, we replace the port number (*the proxy UDP port*) with the port number originally sent by the client. The UDP port number sent by the client is used by the proxy to send the data packets to the client.

3. *Change in the PLAY Message*

   The port numbers are changed in the PLAY messages exchanged between the client and server.

4. *Change in the TEARDOWN Message*

   The port numbers are changed in the TEARDOWN messages exchanged between the client and server.

CACHING THE STREAMING DATA

When a media file is requested by the client, the proxy checks if it has a cached copy of it. If it doesn't, then it makes a connection with the server. The server then sends the data packets to the proxy after the initial RTSP handshake. The proxy, on receiving the data packets, passes them to the client and also caches the data packets.

We cache the media file on the file system on which the proxy is running. A new file is created with the server URL appended to the media filename, so that all cached files are unique. Subsequent data packets received from the server are written to this file without the RTP header.

Thus, when a client requests a media file that the proxy has cached, it reads the data from the file, constructs the RTP data packet and sends it to the client without making a connection to the server.

We have implemented this caching scheme only for simplicity. It is in no way the most efficient way of doing caching and several different good caching methods can be employed here. Our focus was not on implementing the most efficient caching technique, but rather to actually implement caching of the streaming media on the proxy.

Fig. 5 gives the overall design and implementation of the proxy added between the client and server.

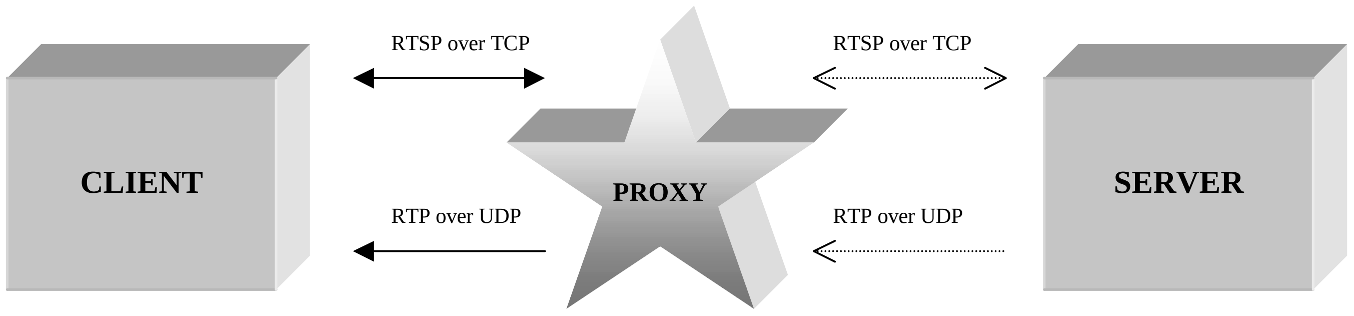


Fig. 5 Overall Design of Proxy

Fig. 5 shows how the communication takes place between the client, proxy and server. The client always opens a two-way TCP connection for the RTSP message exchange and a one way UDP connection for receiving the data packets from the proxy. The dotted lines between the proxy and the server show that a two-way TCP connection for RTSP message exchange and a one way UDP connection for data is established between the proxy and the server on a cache miss, that is when the proxy does not have the media file which is requested by the client. In this case, the proxy will contact the server to get the requested media file.

6. RESULTS

We have been able to successfully design and implement the proxy that caches streaming media. We tested the proxy with a sample audio file. The audio file, as cached by the proxy, had the same content when the proxy serviced the client request indicating that the proxy cached the correct data and there was no degradation in the quality of the audio at the client.

7. FUTURE WORK

The work that can de done in the future can be broken up into three parts:

1. *Improving the Proxy for the given Reference Implementation*

   The current proxy does not support all types of audio payload types that the reference implementation does. It does not support the dynamic types. This could not be done, not because it was not possible but rather due to time constraints. So, the next step would be to modify the proxy to that it can support the dynamic types too.

2. *Improving the Reference Implementation*

   There is tremendous scope in improving the Reference Implementation of the client and server. The following changes can be made to it:

   - *Adding RTCP control for data flow and quality control*
   - *Supporting multiple streams*
   - *Adding support for video files*

3. *General Issues*

    - *Caching Scheme*

      Various efficient caching schemes can be incorporated into the proxy to make it efficient and studies can be done to see the performance of the proxy on the incorporation of those schemes

    - *Real Players and Servers*

      The proxy can be plugged in between real commercial players and servers to study how they communicate and the messages that they exchange. The purpose of this study will be to see if we can develop a general proxy that can be plugged in between any real streaming player and server

8. ACKNOWLEDGMENTS

We would like to thank **Prof. Pei Cao** for her help, support and advice throughout the time span of the project. We would also like to thank **RealNetworks Inc.** for making the Reference Implementation freely available. And last, but not the least, we would also like to thank **Kevin Beach** for his help during the initial stages of the project.

9. REFERENCES

1. RealNetworks Web Site: http://www.real.com

2. Internet Draft: The Real Time Streaming Protocol (RTSP)

3. Internet Draft: The RTP: A Transport Protocol for Real-Time Applications

4. Web Site: http://www.cs.columbia.edu/~hgs/rtsp