

**College Of Engineering**

CMPE-207

NETWORK PROGRAMMING & APPLICATIONS

Fall 2015

**PROJECT PROPOSAL REPORT**

**(VIDEO STREAMING)**

SUBMITTED TO-

DR. WEI XU

BY-

SID-010206209, NAME: ROOPESH REDDY ANREDDY

SID-010698688, NAME: DIPEEKA PATIL

SID-010740496, NAME: KAJAL CHAUHAN

SID-010698675, NAME: PARAG SWAMI

**CONTENTS**

1. **Application Overview 1**
2. **Usage Model 4**
3. **Defined Services 5**
4. **Application level network protocols 5**
5. **Concurrency 6**
6. **Performance and Scalability 7**
7. **Testing Plan 8**

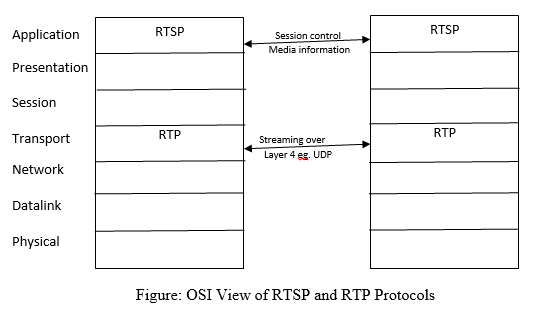
1. **APPLICATION OVERVIEW**

The Application which we are planning to build is a video streaming application. We plan to run this application in a client – server model. We will be implementing a streaming video server and client that communicate using the Real Time Streaming Protocol (RTSP) and send data using the Real Time Transport Protocol (RTP)

We plan to implement this application by using Java socket programming on both the client and server. The server can handle multiple connections, and we plan on achieving this by making the server concurrent using the multithreading concepts.

**Streaming Protocol**

Streaming protocols are deployed for the transmission of real-time streams. Two representatives, RTSP and RTP, have been chosen for the purpose of this work. Both protocols are that fundamental for the transmission of real-time streams, that all major streaming server products support the combination of the two. The RTSP suite makes it possible to deliver media and transport information in a streaming client/server architecture. It provides the means required to setup sessions for video on-demand and video control operations. RTP defines a standardized packet format. This format is a model for audio and video delivery over the network. It solves the media frame packing, the timestamp and the synchronization issues by adding a special RTP header to the packed media data. The format is suitable for both unicast and multicast transport. Figure below shows the RTSP and RTP in the OSI seven layers model.

****

**RFC 2326 - The Real Time Streaming Protocol (RTSP)**

The Real Time Streaming Protocol (RTSP) was developed by the IETF and published in 1998 as RFC 2326. RTSP is a protocol which allows a client to remotely control a streaming media server. The client controls the server by issuing remote-control-like commands such as "play" and "pause". The main function of RTSP is to establish and control one or more time-synchronized streams. RTSP is an application layer protocol. It does not transmit the streams by itself but helps low-level streaming protocols, such as RTP, to establish connections between the streaming partners. Furthermore, it provides a method to transmit media object description.

RTSP must provide states which indicate the status of the streaming process. In RTSP there are two states, session-less and in session. In session means that the client requested the setup of stream transmission, thus requiring an RTSP session for further stream control. An RTSP session identifier is transmitted from the server to the client. Only the session identifier allows the client to issue control commands on the session’s streams.

RTSP has an HTTP-similar syntax making it extensible and adaptable. RTSP comprises commands called methods such as SETUP, PLAY, PAUSE, etc. which indicate the requested action. There are essential methods which define how to create and leave an RTSP session. In addition, an RTSP session strictly defines its RTSP method flow. The Session Description Protocol describes multimedia content. It adds a means of multimedia information exchange, session announcement or session invitation to the RTSP suite.

**RFC 1889 - The Real Time Transport Protocol (RTP)**

The Real Time Transport Protocol (RTP) defines a model for transmission of real-time media. It was developed by the Audio-Video Transport Working Group of the IETF and published in 1996 as RFC 1889. The model describes packaging methods for data with real-time characteristics, such as interactive audio and video. Moreover, the RTP suite defines a control protocol. The RTP Control Protocol (RTCP) allows monitoring of the media data delivery. It adds minimal control and identification functionality to the RTP stream.

RTP neither guarantees timely delivery nor a continuous stream. On this matter it entirely depends on lower layer services. However, a Sequence Number Field is included in the RTP packet header in order to reconstruct the sender’s packet sequence. But this does not prevent the lower network layers from sending out-of-sequence packets. In 2.2 Multimedia frameworks 10 addition, it does not interfere into the transmitting logic of upper layers. The stream processing queues at the server often deliver media object parts out-of-sequence in order to force the clients to prebuffer media presentation. Another important part of the RTP packet header is the Payload Type. It identifies the data format of the RTP payload. In the RTP Profile for Audio and Video Conferences with Minimal Control RFC 1890 the payload types for audio and video RTP streams have been specified. Furthermore, a Timestamp field in the RTP header field reflects the sampling instant of the first octet in the RTP data packet.

RTCP was defined to add minimal control to the RTP. This control is obtained by transmitting periodic control packets to all participants of a streaming session. The control packets are distributed using the same mechanism as the RTP data packets. RTCP adds four functions to RTP transmissions:

1. It provides feedback about the quality of the data distribution (diagnoses network problems).

2. It carries a persistent transport-level identifier called CNAME.

3. It helps to adjust the RTP packet sending rate.

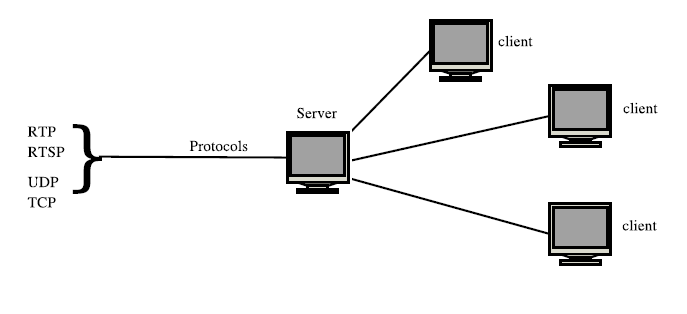
4. It conveys minimal session information: for example session’s participant identification.

The functions 1-3 are mandatory for an RTP transmission in an IP multicast environment. Additionally, they are recommended for all environments. For UDP and similar protocols, the RTP stream uses an even port number and the corresponding RTCP stream uses the next higher (odd) port number.

The RFC 1889 only defines the standard properties of the RTP suite. Several other protocols extend this basic concept with new ideas. Aforementioned RFC 1890 both lists the assigned payload types for media codecs and provides guidelines of correct media encoding for RTP transmissions. In addition, most assigned payload types are specified in an RFC specification which describes how to adjust the original RTP packet model to the special requirements.

1. **USAGE MODEL**

We plan on building a GUI for the end user, which can be run on a client machine. The request for the video to be streamed is sent from the client machine. The stream can be then viewed using the interface.



We plan on building a simple video player which is easy to operate and contains buttons such as setup, play, pause, forward and reverse. Using these buttons, the user can watch the video which is being streamed by the server.

1. **DEFINED SERVICES**

The basic service provided by this application is to stream videos of various formats such as MPEG, FLV etc. The users can be able to view the videos being streamed using the GUI where the user can access functions such as play, pause, setup, etc. by just a simple click of buttons.

1. **APPLICATION-LEVEL NETWORK PROTOCOLS**

We plan on getting this to work by using some application layer protocols such as RTP (Real Time Protocol) and RTSP (Real Time Streaming Protocol). Both these protocols operate in conjunction and are used to establish and control media sessions between end points. These protocols operate over UDP protocol, which is a connectionless service, which is usually preferred in a streaming application. The reason for choosing UDP over a connection oriented protocol such as TCP is due to the less overhead in maintaining the information of the packets.

1. **CONCURRENCY**

As this project is being implementing in Java, it has built-in support for concurrent programming. This enables you to have multiple flows of control, represented by multiple threads within a single program (process). A thread is a single sequential flow of control within a process. Each thread has a separate execution path, with its own beginning, program flow, current point of execution, and end. They are represented by Thread objects in Java.

Multithreading allows a program to perform multiple tasks concurrently. Although threads give the appearance of running concurrently, in a single- processor system the interpreter is switching between the threads and running them one at a time. Multiprocessor systems may actually run multiple threads concurrently. The threads all run in the same memory space, i.e., they can all access the same memory and call methods as if they were in a normal single threaded process. Rather than processing the incoming requests in the same thread that accepts the client connection, the connection is handed off to a worker thread that processes the request. That way the thread listening for incoming requests spends as much time as possible in the accept() call. That way the risk is minimized for clients being denied access to the server because the listening thread is not inside the accept()call.

The advantages of a multithreaded server compared to a single threaded server are summed up below:

1. Less time is spent outside the accept() call.
2. Long running client requests do not block the whole server

The more time the thread calling accept() spends inside this method call, the more responsive the server will be. Only when the listening thread is inside the accept() call can clients connect to the server. Otherwise the clients just get an error.

This way video streaming server can handle multiple request coming from different client. The server will create different thread for each incoming request and start communicating by sending the requested data packets to the client.

1. **PERFORMANCE AND SCALABILITY**

We plan to implement optimization and traffic control mechanisms in both server and client. The RTCP protocol is built on top of UDP protocol, which minimizes the overhead, when compared to a connection oriented service protocol. We plan on implementing a congestion controller class on the server side. In addition to congestion control we also plan to implement adaptive compression for the video streams in order to optimize the size of video frames which are being sent at the time of high traffic.

On the client side we plan on implementing a frame synchronizer, which can solve the problems of organizing the frames in a correct order and preventing the discrepancies where the frames arrive in different orders or the same frames arrive repeatedly.

1. **TESTING PLAN**

Test plan in this project basically focuses on unit testing already provided thorough black box testing, extensive coverage of source code, and testing of the interface.

1. Unit Testing
2. Black Box Testing
3. Extensive code coverage
4. Integration Testing

The interfaces that will be tested:

* 1. User Interface from where user can START PLAY or STOP streaming of video and audio.

The external interfaces that will be tested:

* 1. Local PCs
  2. Remote PCs.

The most critical performance measures to test are:

* 1. Test for the correctness of the standard output to see if they are the expected values.
  2. Test the method to see if it is sending back correct session information.
  3. Simulates traffic using RTP (Real Time Protocol) – Will test network connections using RTP over a UDP connection and analyzes the data streams for multimedia performance.
  4. Test network connection with Real Time Streaming Protocol.
  5. Test to see if the data flow rate has indeed been reduced by the server congestion algorithm.
  6. Test for sound quality and picture quality.