California State University Los Angeles

Final Project: EE 442-01

Multimedia Networking

Term Project: A Simple On-line VCR

Team Members:

Babar H Baig **CIN 304xxxxxx**

Jaymin H Patel **CIN 304xxxxxx**

Table of Contents

Project Objective:

In this project our team was asked to create a video streaming software which will communicate between client and server using the Real- Time Streaming Protocol (RTSP) and send data using Real-Time Protocol (RTP). To complete this task we need to implement the RTP packetetization in the server.

We were provided a frame work from Dr. Zhao in form of Server.java, Client.java, RTPpacket.java and videostreaming.java. we were asked to only complete the missing codes for Client.java and RTPpacket.java.

Following are the attributes of each class.

- **Client**: This class implements the client and the user interface which you use to send RTSP commands and which is used to display the video.
- **Server**: This class implements the server which responds to the RTSP requests and streams back the video. The RTSP interaction is already implemented and the server calls routines in the RTPpacket class to packetize the video data. You do not need to modify this class.
- **RTPpacket**: This class is used to handle the RTP packets. This is the only class you need to work on. Please read the file carefully to understand the structure of this class.
- **VideoStream**: This class is used to read video data from the file on disk.

To use above frame work our team must complete, perform and test the following task.

The class you need to complete is RTPpacket which works together with the server to send out RTP packet. For this you will need to create the packet, set the fields in the packet header, and copy the payload (i.e., one video frame) into the packet. The basic tasks you need to complete are:

1. Set the RTP-version field (V). You must set this to 2.
2. Set padding (P), extension (X), number of contributing sources (CC), and marker (M) fields. *These are all set to zero in this lab*.
3. Set payload type field (PT). In this lab we use MJPEG and the type for that is 26.
4. Set the sequence number. The server gives this the sequence number as the Framenb argument to the constructor.
5. Set the timestamp. The server gives this number as the Time argument to the constructor.
6. Set the source identifier (SSRC). This field identifies the server. You can pick any integer value you like.

Because we have no other contributing sources (field CC == 0), the CSRC-field does not exist. The length of the packet header is therefore 12 bytes.

Real-Time Protocol (RTP):

Before we start our project we must need to understand the basics of RTP and how does it works. RTP is an internet protocol which basically designed to extensively transport and manage the real time communication and multimedia contents over the internet. It can either use the unicast or multicst network (depending upon the application).

The RTP was designed by the Internet Engineering Task Force (IETF) Request for Comments (RFC)1889 [1]. Its primarily goal was to transfer data from one end to another anywhere in the world.

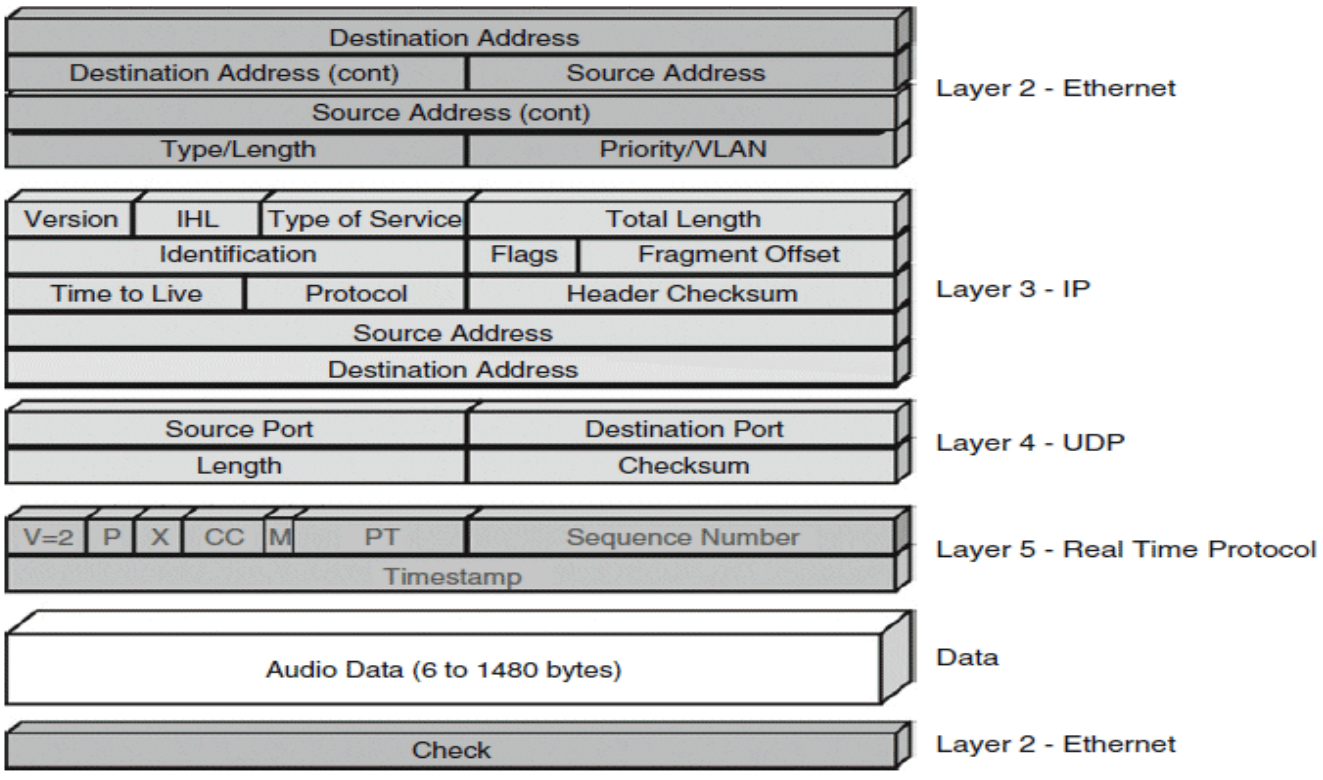
RTP only transfer data file from end to end, it does not guarantee and responsible for the quality of server (QoS). It only does its best effort to transport multimedia contents like audio and video from one end to another. The basic reason is that RTP has no acknowledge or confirmation mechanism. The quality of service and guarantee of delivery is been handle by Real-Time Control Protocol (RTCP).

RTP use RTCP to control and monitor the flow of data. This process allows receiver to monitor the transmission and notice if there is any packet loss. RTP is an application layer protocol works independently with Transport layer and Network layer protocols.

RTP Features:

There are many features related to RTP. Some most popular one are as follows.

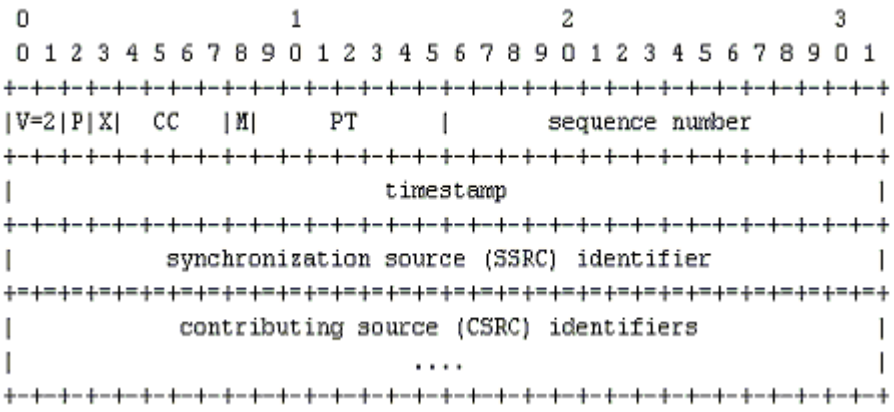
- RTP is a light weight and very efficient protocol.
- It typically integrated with the applications (VIC, VAT).
- Designed to work with reliable and fast point-to-point links.
- RTP run at the top to Datagram protocol (UDP), although it can also works with other transport protocol.



RTP at Layer 5 [2]

RTP Header Structure:

The RTP header structure describe it fuctionality. Each field has its own unique feature which interconnect with other components/ protocols and describe their relationship.



RTP header [3]

Padding (P): 1 bit

Padding could be set by application. In the event that it is one than it implies that the parcel contains one or more extra padding octets toward the end which are excluded in the payload. Cushioning (for encryption) last byte of the bundle contains padding.

Version (V): 2 bits

This part shows the version of the RTP. This should be 2 according to RFC 1889

CSRC count (CC): 4 bits

The CSRC count contains the number of CSRC identifiers that follow the fixed header. Contributing Source ID.

Extension (X): 1 bit

If the extension bit is set, the fixed header is followed by exactly one header extension.

Payload type (PT): 7 bits

This field identifies the format (encoding method; may change during session) of the RTP payload and determines its interpretation by the application. This field is not intended for multiplexing separate media.

Marker (M): 1 bit

Marker bit is used by specific applications to serve a purpose of its own. marker bit, indicate frame, and beginning of talksport. It is used to allow delay adjustment

Sequence number: 16 bits

The sequence number increments by one for each RTP data packet sent, it can be used by the receiver to detect packet loss and to restore packet sequence.

SSRC: 32 bits

The SSRC field recognizes the synchronization source. This identifier is chosen randomly, with the purpose to Synchronization Source ID. Each source picks at random. There are crash determination systems for recognizing/determining various source picking the same ID. At no two synchronization sources inside of the same RTP session will have the same SSRC identifier.

Timestamp: 32 bits

The timestamp demonstrates the inspecting of the first octet in the RTP information parcel. The testing must be gotten from a clock that additions and straightly so as to permit synch and jitter estimations. Clock recurrence may differs and reliant on the organization (common 8kHz sound, 90kHz video).

Real-Time Streaming Protocol (RTSP):

The real time streaming protocol (RTSP) belongs to the application layer. It is responsible to control the data with the real time basis. Mainly it deals to transfer real time audio and video between server and client. The multimedia contents which are transferred by the RTSP can be a real time data or a stored audio and video clips. RTSP aims to control multiple data transfer sessions and choosing different channels like (Datagram UDP) and TCP, there mechanism based of RTP. In other words we can say that RTSP is a server-client based protocol which transfer data over IP network.

RTSP is an application-level convention intended to work with lower-level conventions like RTP, RSVP to give a complete gushing administration over Internet. It gives intends to picking conveyance channels, (for example, UDP, multicast UDP and TCP), and conveyance systems based upon RTP. It works for substantial gathering of people multicast and single-viewer unicast.

In RTSP, every presentation and media stream is distinguished by a RTSP URL. The general presentation and the properties of the media are characterized in a presentation portrayal record, which may incorporate the encoding, dialect, RTSP URLs, destination location, port, and different parameters. The presentation portrayal document can be gotten by the customer utilizing HTTP, email or different means.

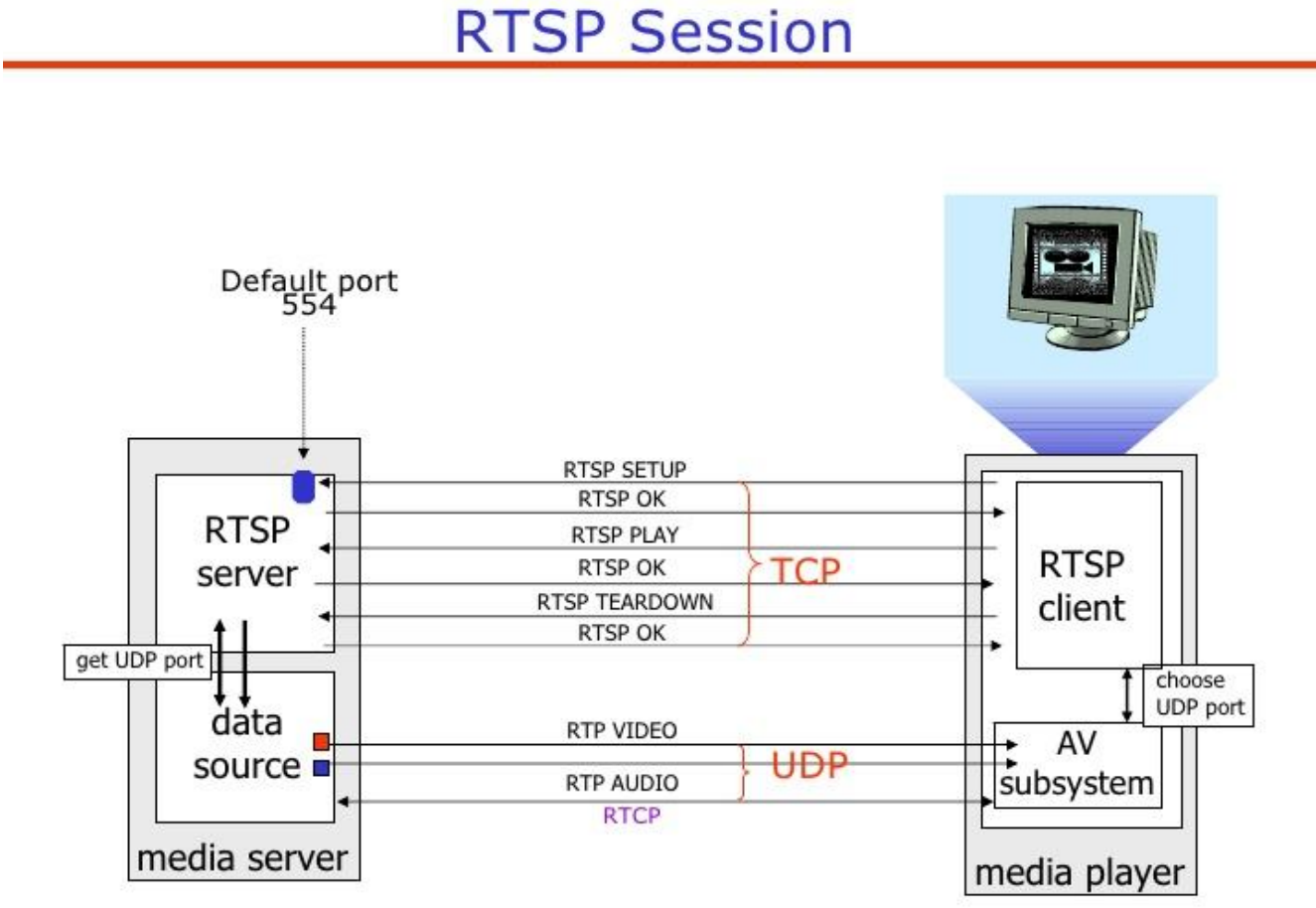
The properties of RTSP:

- Extendable (new routines and parametres are anything but difficult to include)
- Simple to parse (standard HTML or MIME parser can be utilized)
- Secure (HTTP confirmation systems, transport and system layer security components appropriate)
- Transport-autonomous (conventions, for example, UDP, RDP and TCP appropriate)
- Multi-server proficient (there can be media streams from diverse servers in one presentation)
- Control of recording devices (both playback and recording control conceivable)
- Division of stream control and gathering start (The main necessity is that the meeting start convention either gives or can be utilized to make an one of a kind meeting identifier)
- Intermediary and firewall cordial (convention ought to be promptly taken care of by both application and transport-layer firewalls)

- HTTP-accommodating (RTSP reuses HTTP ideas, where sensible)
- Suitable server control (i.e. servers ought not begin gushing to customers in a manner that customers can't stop the stream)
- Transport transaction (transport technique can be arranged just before spilling)
- Capacity negotiation(client must have an approach to see whether a portion of the essential elements are disabled in the server)

RTSP Operations:

RTSP controls a stream which may be sent by means of a different convention, free of the control channel. For instance, RTSP control might happen on a TCP association while the information streams by means of UDP. Consequently, information delivery proceeds with regardless of the possibility that no RTSP solicitations are gotten by the media server. a single media stream may be controlled by RTSP requests issued sequentially on different TCP connections. Hence, the server needs to keep up "session state" to have the capacity to relate RTSP asks for with a stream. The state moves are depicted in Section A. Numerous techniques in RTSP don't add to state. Then again, the taking after assume a focal part in characterizing the distribution and use of stream assets on the server: SETUP, PLAY, RECORD, PAUSE, and TEARDOWN.



RTSP Server and Client [4]

SETUP:

Initiate the communication with the server to allocate resources for a stream and start an RTSP session.

PLAY and RECORD:

This will initiate the data transmission from media server to the requested client.

PAUSE:

Temporarily halts a stream without freeing server resources. User can restart and play the session again from the same point where it pause.

TEARDOWN:

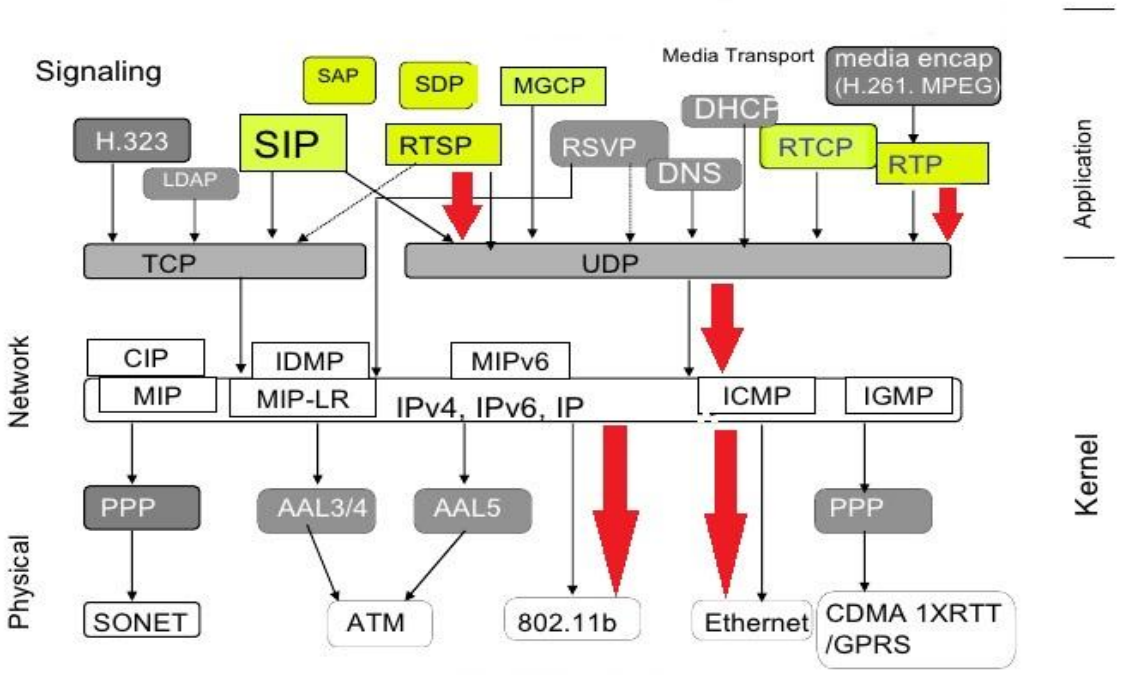
Frees resources associated with the stream. The RTSP session ceases to exist on the server.

RTSP methods that contribute to state use the Session header field to identify the RTSP session whose state is being manipulated. The server generates session identifiers in response to SETUP requests.

Setup, Play, Pause and Teardown buttons on RTSP client application.

Implementation:

To develop this system, we must need to understand the basic frame we used in our programming. The following diagram would show it clearly.



Our team worked on the provide framework to establish and demonstrate the working model of RTP using RTSP. Here RTP used RTSP to transfer audio and video file to the requested client. RTSP is acontrol convention that starting and coordinating conveyance of spilling mixed media from media servers, the "Web VCR remote control convention". RTSP does not convey information (however the RTSP association may be utilized to burrow RTP activity for usability with firewalls and other system devices). RTP and RTSP will probably be utilized together as a part of numerous frameworks, however either convention can be utilized without the other. The RTSP draft contains an area on the utilization of RTP with RTSP.

Here we used Datagram socket (UDP). Since the data volume of RTSP commands is likely to be low, the loss-induced congestion control of TCP is not particularly helpful, but may force a client or server into excessive back-off, with long delays. Session establishment is also faster, although this is not likely to be a major issue since RTSP requires its own setup.

RTSP Client Operations:

The Client send RTSP request to the server by utilizing the catches.

- Client sends SETUP. This charge is use to set up the session and transport parameters.
- Client sends PLAY. This begins the playback.
- Client may send PAUSE on the off chance that it needs to interruption amid playback.
- Client sends TEARDOWN. This ends the session and shuts the association

Our group errand was to execute the RTSP on the customer side. We finished the capacities that are called when the client taps on the catches in the client interface. For every catch in the interface there is a handler capacity in the code. You should execute the accompanying activities in every handler capacity. At the point when the customer begins, it additionally opens the RTSP attachment to the server. Utilize this attachment for sending all RTSP request.

Setup:

- We made an attachment for accepting RTP information and set the timeout on the attachment to 5 milliseconds.
- Send SETUP solicitation to server. We put the Transport header in which we determine the port for the RTP information attachment that we just made.
- Read answer from server and parse the Session header in the reaction to get ID

PLAY

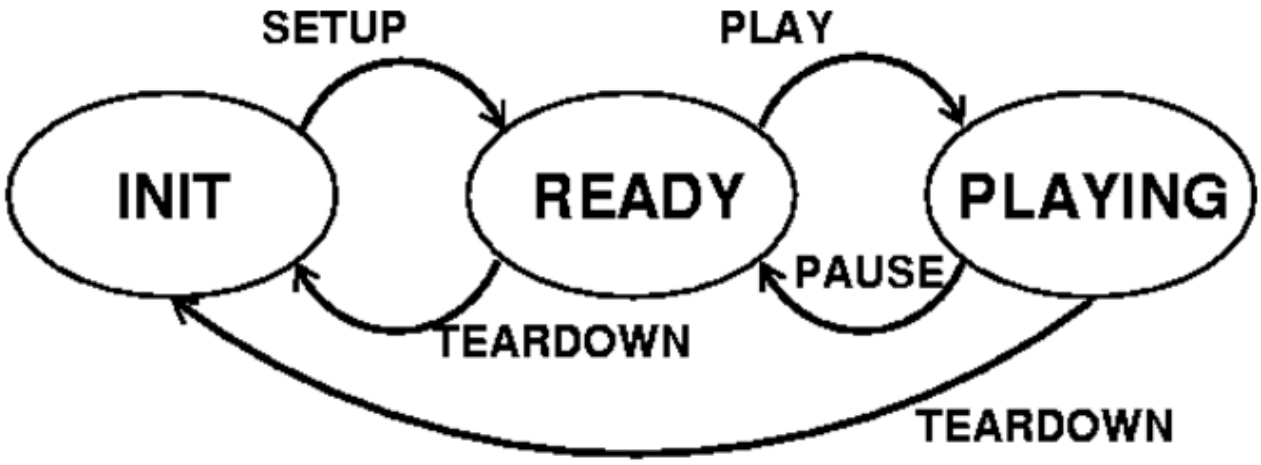
- Send PLAY requent. We put the Session header and utilize the session ID returned in the SETUP reaction.Listen server's reaction.

Stop

- Send PAUSE request. We put the Session header and utilize the session ID returned in the SETUP reaction. Listen server's reaction.

TEARDOWN

- Send TEARDOWN request. We put the Session header and utilize the session ID returned in the SETUP reaction. Listen server's reaction.



Client State Diagram

Complete coding at Client.java file:

We were given an incomplete java file for the client. So my team completed the missing part of the code. Following are the screenshots of complete code and explanation.

```
if (state == INIT)
    {
     //Init non-blocking RTPsocket that will be used to receive data
      try{
       //construct a new DatagramSocket to receive RTP packets from the server, on port RTP_RCV_PORT
       //RTPsocket = ...
                RTPsocket = new DatagramSocket(RTP_RCV_PORT);
              //   set TimeOut value of the socket to 5msec.


                      RTPsocket.setSoTimeout(5);
       //set TimeOut value of the socket to 5msec.
       //....
     }
```

When we start connection, we first check for state, if its initial then we have initiated the Datagram Socket. Each time when we create Datagram Socket object we wait for 5 second of time span.

```
      if (parse_server_response() != 200)
        System.out.println("Invalid Server Response");
      else
        {
          //change RTSP state and print new state
              state = READY;
          System.out.println("New RTSP state: READY");
          //System.out.println("New RTSP state: ....");
        }
    }//else if state != INIT then do nothing
```

We need to wait for the server response for about 200ms, if server does not responses, considers as invalid server response where as if it reacts then State will be changed to READY.

```
            {
//    change RTSP state and print out new state

                state=PLAYING;
                System.out.println("New RTSP state: PLAYING");

//    start the timer
                timer.start();
            }
        }//else if state != READY then do nothing
    }
}
```

Once RTSP state gets changed to PLAYING mode timer will get started.

```
if (state == PLAYING)
    {
      //increase RTSP sequence number
   RTSPSeqNb++;
      //........

      //Send PAUSE message to the server
      send_RTSP_request("PAUSE");

      //Wait for the response
     if (parse_server_response() != 200)
            System.out.println("Invalid Server Response");
      else
        {
          //change RTSP state and print out new state
          //........
              state = READY;
          //System.out.println("New RTSP state: ...");

          //stop the timer
          timer.stop();
        }
    }
```

If state is PLAYING, we increases the RTSPSeqNb++. When Pause occurse we stop the timer.

```
private void send_RTSP_request(String request_type)
{
        try{

                    RTSPBufferedWriter.write(request_type+" "+VideoFileName+" "+"RTSP/1.0"+CRLF);

                    RTSPBufferedWriter.write("CSeq: "+RTSPSeqNb+CRLF);

                    if(request_type.equals("SETUP")) {
                        RTSPBufferedWriter.write("Transport: RTP/UDP; client_port= "+RTP_RCV_PORT+CRLF);
                    }

                    else {
                        RTSPBufferedWriter.write("Session: "+RTSPid+CRLF);
                    }

                    RTSPBufferedWriter.flush();
                }
                catch(Exception ex)
                  {
                    System.out.println("Exception caught: "+ex);
                  }
```

In this part we have initiated the RTS Buffered Writer object to response for client request. With this object we pass few parameters such as Video File name, CRLF and Sequence number. If its first request then buffered writer needs Transport protocol name and client port number. Whereas if its historical request then it takes the session values. After serving to request buffered writer will get flushed.

Complete coding at RTPpacket.java file:

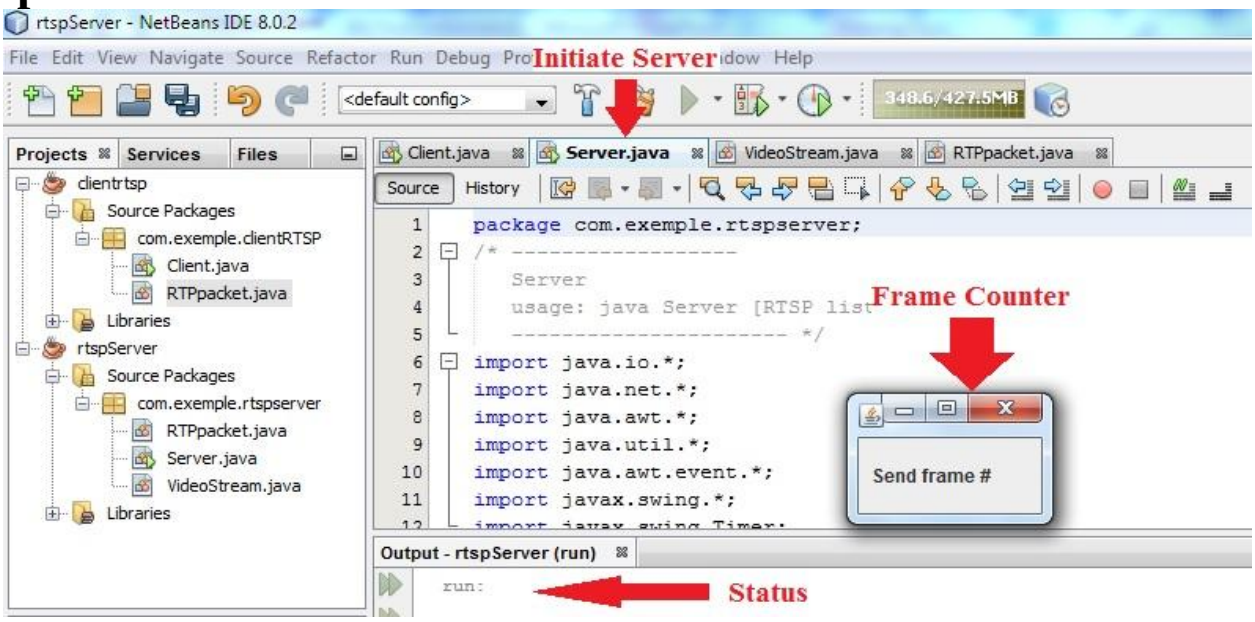
```
// build the header bistream:
            // --------------------------
            // declaring header variable
            header = new byte[HEADER_SIZE];
            // Putting marker value at header index 1
            header[1] = (byte) ((Marker << 7) | PayloadType);
            // Putting sequence number value at header index 2
            header[2] = (byte) (SequenceNumber >> 8);
            // Putting sequence number value at header index 3
            header[3] = (byte) (SequenceNumber);
            // Step 5 : server gives this number as the Time argument to the
constructor
            for (int i = 0; i < 4; i++)
                  header[7 - i] = (byte) (TimeStamp >> (8 * i));
            // Step 6 : Field identifies the server and picks up any random
value.
            for (int i = 0; i < 4; i++)
                  header[11 - i] = (byte) (Ssrc >> (8 * i));

            // Picked up header information to AARAY header and prepare
payload.
            payload_size = data_length;
            payload = new byte[data_length];
            payload = data;
```

We build the header bitstream, declaring header variable. Putting sequence number value at header index 2, Putting sequence number value at header index 3, server gives this number as the Time argument to the constructor, Field identifies the server and picks up any random value. Picked up header information to AARAY header and prepare payload.

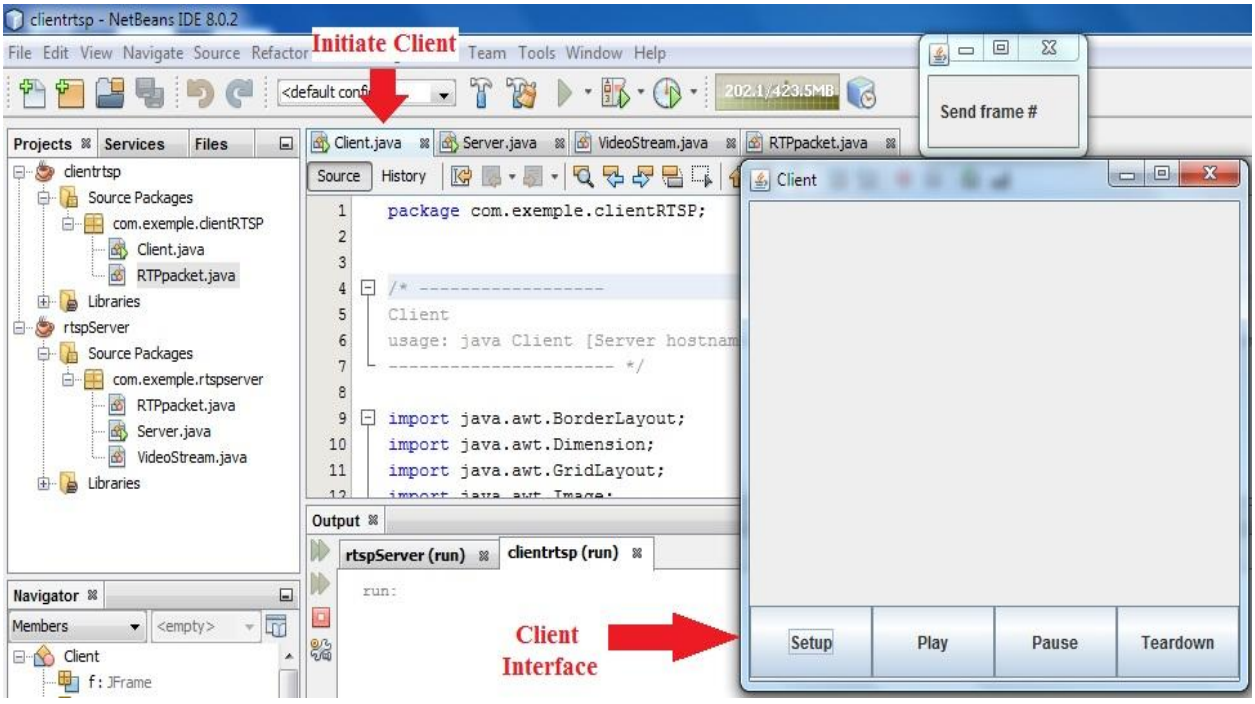
Run RTSP Server and Client:

Step#1:

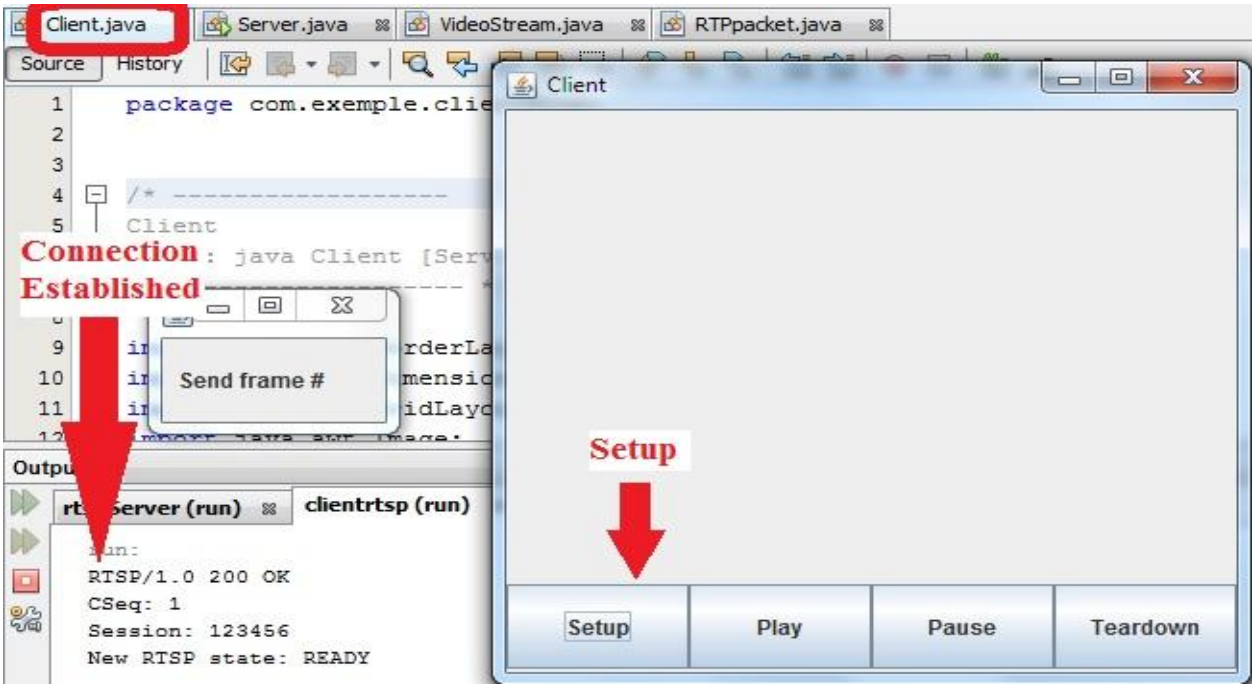


First we initialized the server. As we can see the small frame counter window pop's up. This window show us the number of packett transferred.

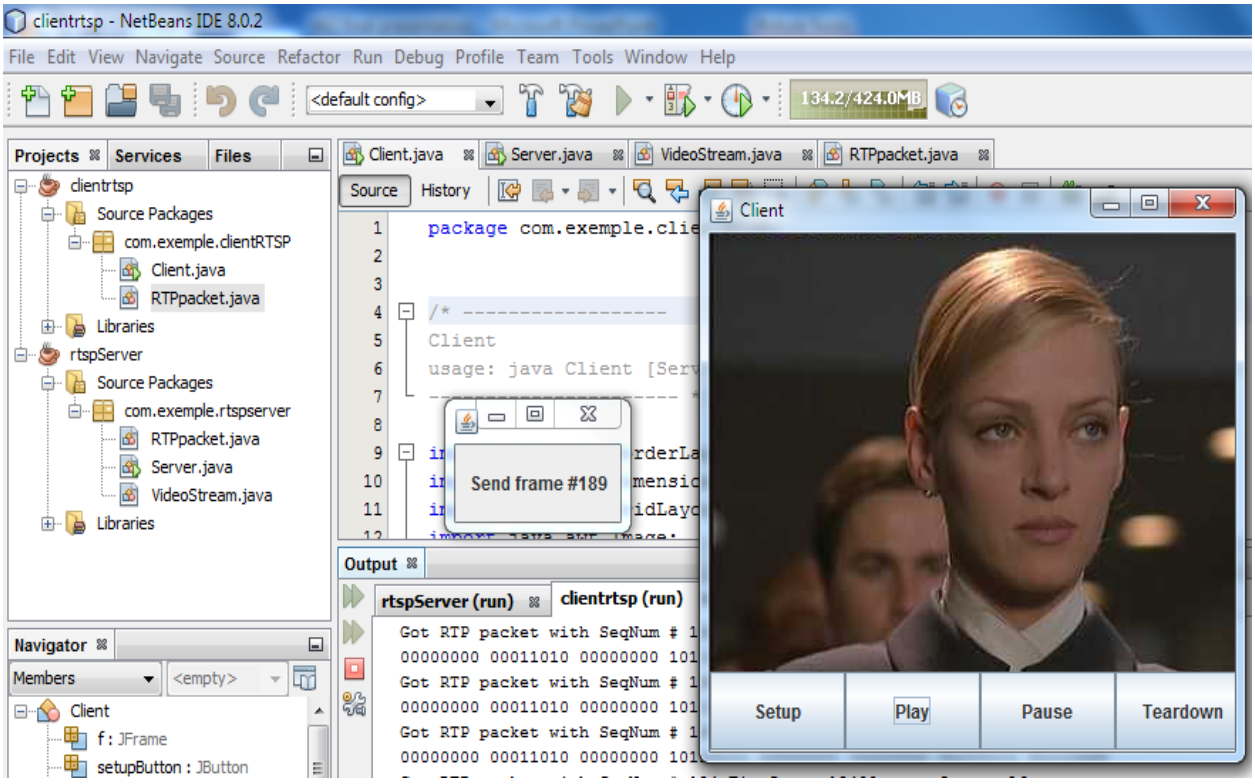
Step#2:



Step #3


Client.java
Server.java
VideoStream.java
RTPpacket.java
Source
History
Connection Established
Send frame #
Client
Setup
Setup
Play
Pause
Teardown
clientrtsp (run)
RTSP/1.0 200 OK
CSeq: 1
Session: 123456
New RTSP state: READY

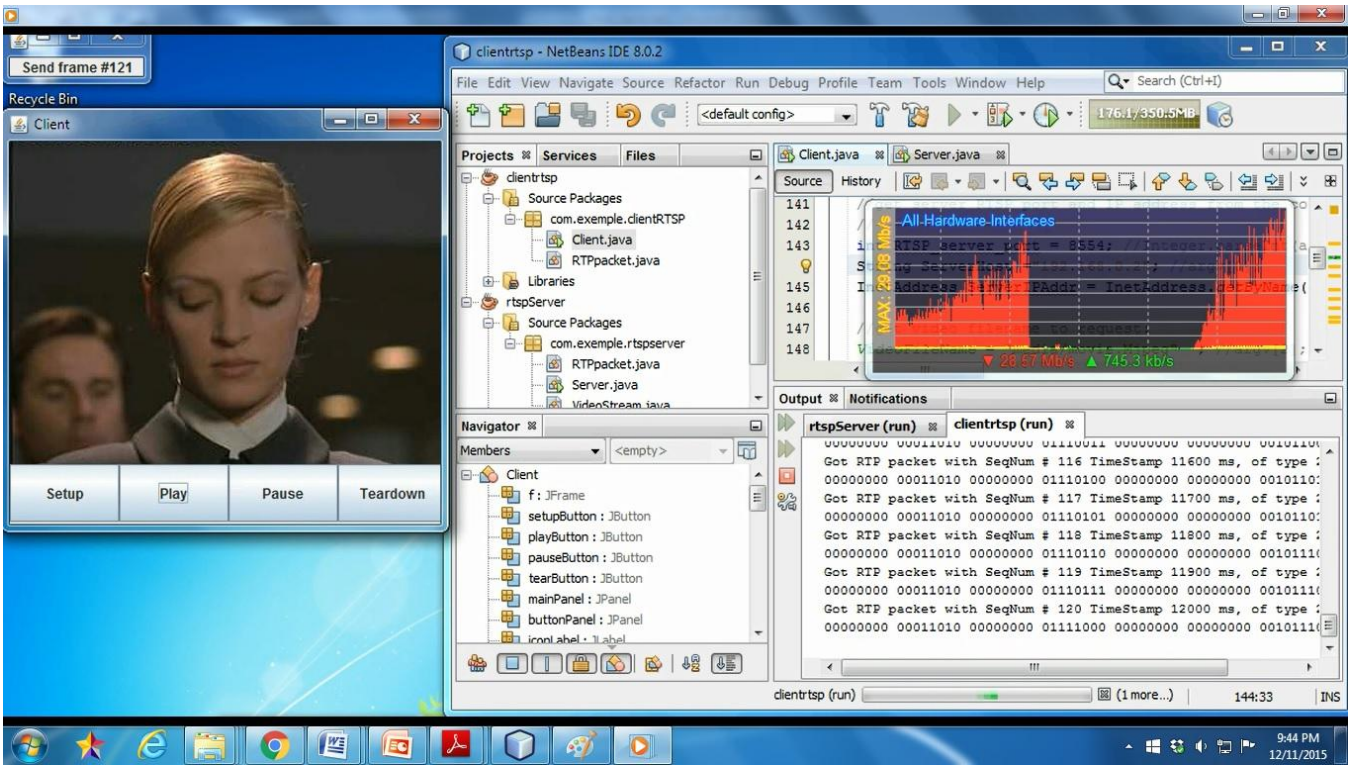
Step #4


clientrtsp - NetBeans IDE 8.0.2
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Projects
Services
Files
clientrtsp
Source Packages
com.exemple.clientRTSP
Client.java
RTPpacket.java
Libraries
rtspServer
Source Packages
com.exemple.rtspserver
RTPpacket.java
Server.java
VideoStream.java
Libraries
Navigator
Members
Client
f : JFrame
setupButton : JButton
Send frame #189
Client
Setup
Play
Pause
Teardown
rtspServer (run)
clientrtsp (run)
Got RTP packet with SeqNum # 1

Test Scenario # 1:

Test Conditions:

- Both Server & client are on the same computer using same network wired local area network.
- Additional traffic is generated at system range from 10Mb/s to 50 Mb/s to identify any packet loss.



RTSP Server and Client are running at the same computer.

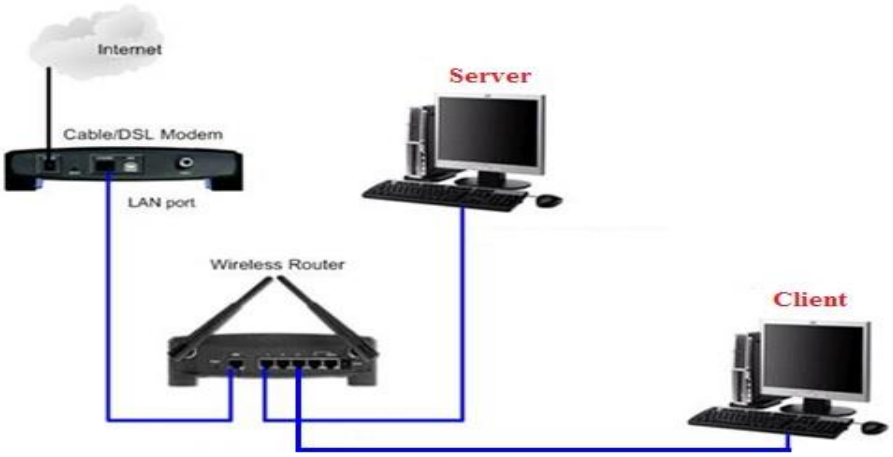
Result:

No packet loss while using a wired network under given conditions. As it was expected due to several reasons like both client and server using same computer, no transfer data from one system to another etc.

Test Scenario # 2:

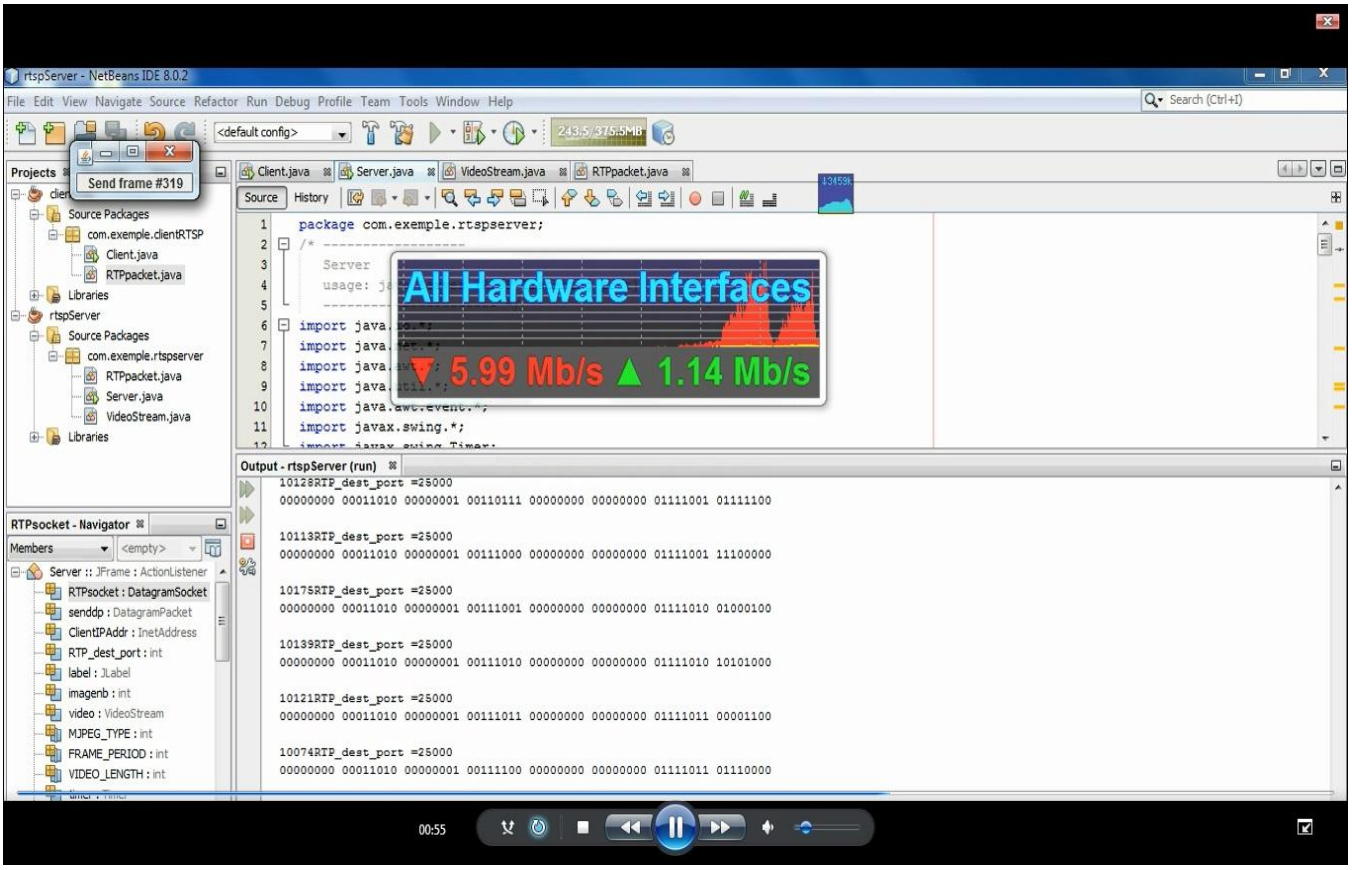
Test Conditions:

- Both Server & client are connected through wired local area network.
- Distance range between Server & Client is 6 meters.
- Additional traffic is generated at Server range from 10Mb/s to 50 Mb/s.

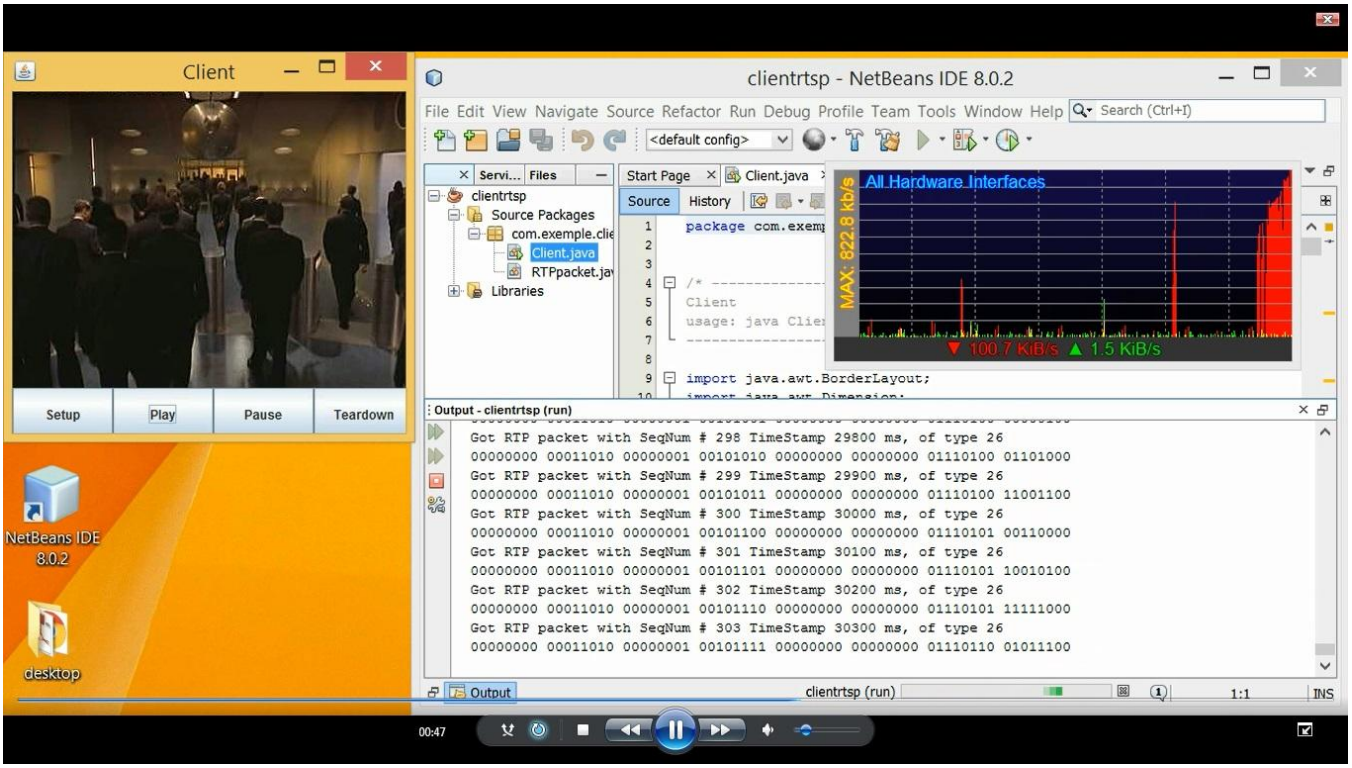


Network Topology

Additional traffic monitor



RTSP Server at Desktop (wired network)



RTSP Client at Laptop(wired network)

Result:

No packet loss while using a wired network under given conditions. As it was expected due to several reasons:

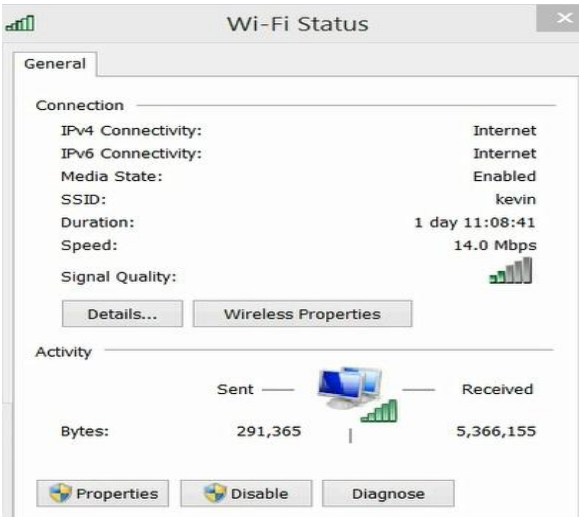
- Both client and server using same local area network and connect through wire (Ethernet cable)
- No data transfer to far destination.
- Network bandwidth was very high.

Test Scenario # 3:

Test Conditions:

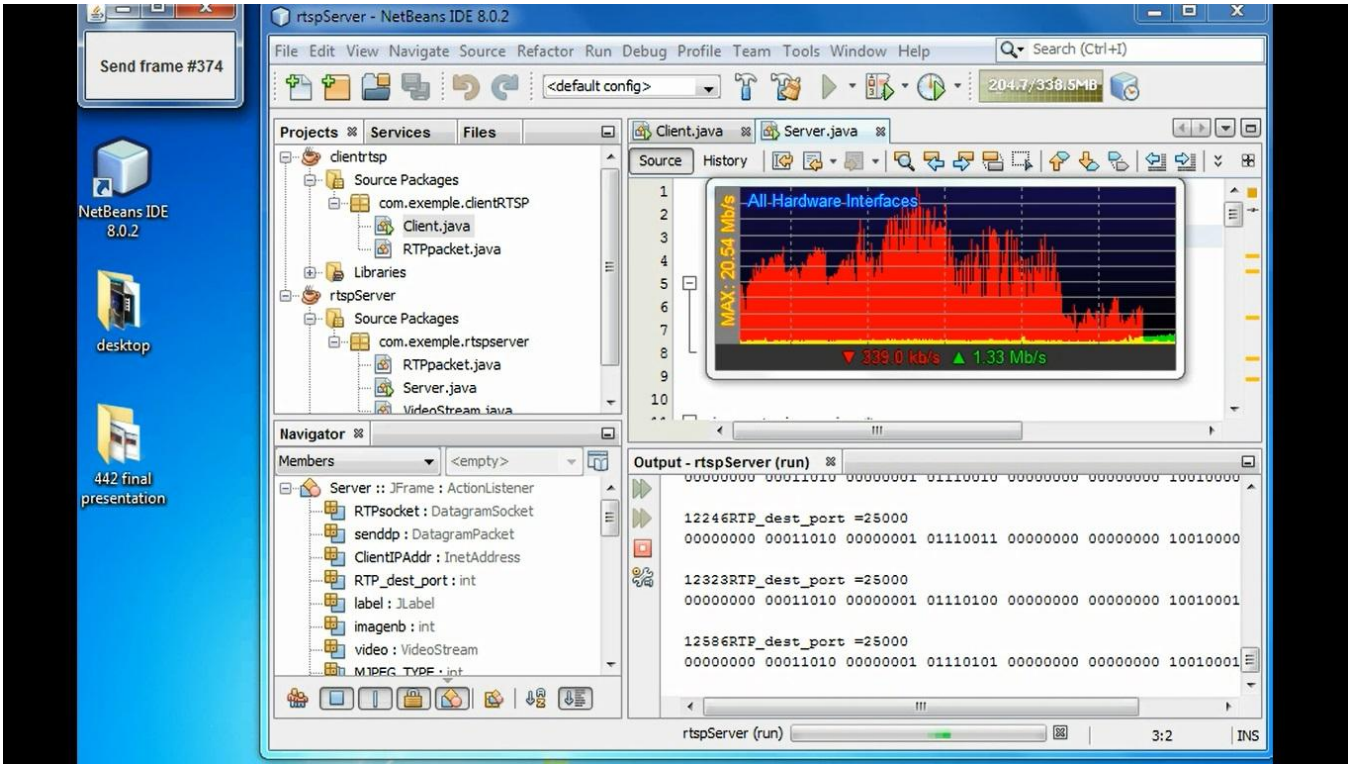
- Both Server & client are connected through Wireless network.
- Distance range between Server & Client is 20 meters.
- Used very low wireless signals to analyze packet loss.
- Additional traffic is generated at Server range from 10Mb/s to 50 Mb/s.

Network Topology(wireless)

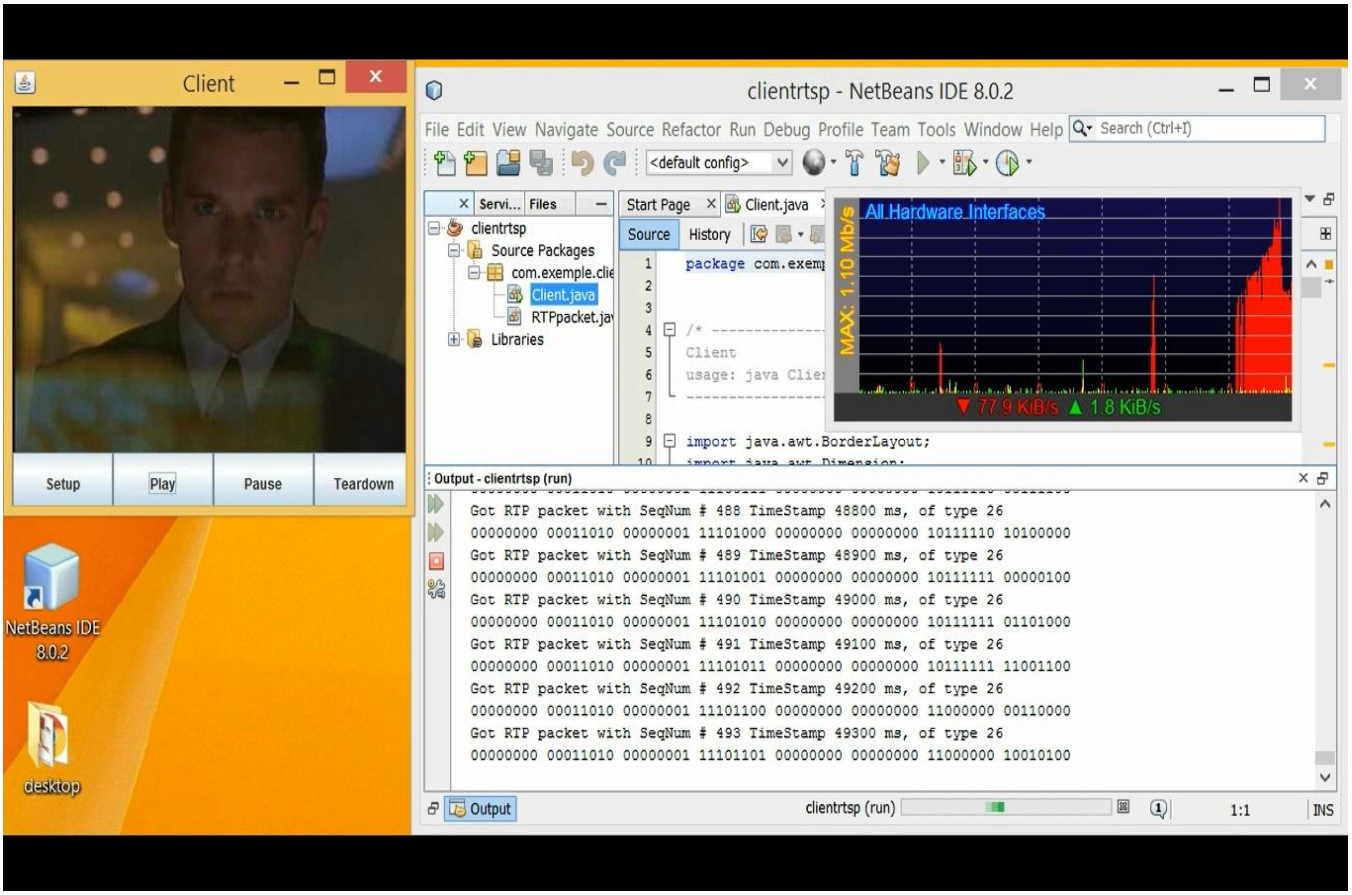


Signal strength at Client Laptop (wireless)

Additional traffic monitor



RTSP server at desktop (wireless)



RTSP Client at laptop (wireless)

Result:

After analyzing the detail results, we noticed that there was a packet loss when we used the wireless RTSP server and client.

Lost Packet #3:

```
Got RTP packet with SeqNum # 2 TimeStamp 200 ms, of type 26
00000000 00011010 00000000 00000010 00000000 00000000 00000000 11001000
Got RTP packet with SeqNum # 4 TimeStamp 400 ms, of type 26
```

Resend Packet #12 twice:

```
Got RTP packet with SeqNum # 12 TimeStamp 1200 ms, of type 26
00000000 00011010 00000000 00001100 00000000 00000000 00000100 10110000
Got RTP packet with SeqNum # 12 TimeStamp 1200 ms, of type 26
```

Lost Packet #68, #69, #70, #71:

```
Got RTP packet with SeqNum # 67 TimeStamp 6700 ms, of type 26
00000000 00011010 00000000 01000011 00000000 00000000 00011010 00101100
Got RTP packet with SeqNum # 72 TimeStamp 7200 ms, of type 26
```

Lost Packet #245, #247, #248, #250, #251, #252:

```
Got RTP packet with SeqNum # 244 TimeStamp 24400 ms, of type 26
00000000 00011010 00000000 11110100 00000000 00000000 01011111 01010000
Got RTP packet with SeqNum # 246 TimeStamp 24600 ms, of type 26
00000000 00011010 00000000 11110110 00000000 00000000 01100000 00011000
Got RTP packet with SeqNum # 249 TimeStamp 24900 ms, of type 26
00000000 00011010 00000000 11111001 00000000 00000000 01100001 01000100
Got RTP packet with SeqNum # 253 TimeStamp 25300 ms, of type 26
```

Lost Packet #295, #296:

```
Got RTP packet with SeqNum # 294 TimeStamp 29400 ms, of type 26
00000000 00011010 00000001 00100110 00000000 00000000 01110010 11011000
Got RTP packet with SeqNum # 297 TimeStamp 29700 ms, of type 26
00000000 00011010 00000001 00101001 00000000 00000000 01110100 00000100
```

Lost Packet #304:

```
Got RTP packet with SeqNum # 303 TimeStamp 30300 ms, of type 26
00000000 00011010 00000001 00101111 00000000 00000000 01110110 01011100
Got RTP packet with SeqNum # 305 TimeStamp 30500 ms, of type 26
00000000 00011010 00000001 00110001 00000000 00000000 01110111 00100100
```

Lost Packet #361:

```
Got RTP packet with SeqNum # 360 TimeStamp 36000 ms, of type 26
00000000 00011010 00000001 01101000 00000000 00000000 10001100 10100000
Got RTP packet with SeqNum # 362 TimeStamp 36200 ms, of type 26
00000000 00011010 00000001 01101010 00000000 00000000 10001101 01101000
```

Total Packet Lost = 15

Total Number Of Packets = 500

Packet loss Percentage = 3%

This result shows us clearly that data transfer over wireless network could cause the packet loss.

Conclusion:

After running the 3 test between RTSP server and client, we come to the point that in first 2 test we conducted using a wired local area network, there was not packet loss as it was pretty much expected. But after running the 3rd test in the wireless network we noticed there was a significant packet loss and the resulted video quality was poor. We calculated the packet drop ratio was 3% out of 500 total packets transferred.

There could be many reasons of this packet loss. Few of them are as follows:

- Weak signal cause a packet loss as in test 3, we had a significant low wireless signals.
- Away from the router could also cause the packet loss.
- Network conjunction as we used heavy traffic on the network.
- Low bandwidth also because we used heavy traffic.

Team Member Participation:

Design & Analysis	Babar Baig Jaymin H Patel
Coding for client and RTP packet file	Babar Baig (client file) Jaymin H Patel (RTPpacket file)
Testing	Babar Baig Jaymin H Patel
Result Analysis/ report writing	Babar Baig Jaymin H Patel

Note: Me and my group partner Jaymin Patel performed all tasks together, including power point presentation slides.

Appendix

Source code for RTPpacket.java:

```
package com.exemple.clientRTSP;

public class RTPpacket {
	// size of the RTP header:
	static int HEADER_SIZE = 12;

	// Fields that compose the RTP header
	public int Version;
	public int Padding;
	public int Extension;
	public int CC;
	public int Marker;
	public int PayloadType;
	public int SequenceNumber;
	public int TimeStamp;
	public int Ssrc;

	// Bitstream of the RTP header
	public byte[] header;

	// size of the RTP payload
	public int payload_size;
	// Bitstream of the RTP payload
	public byte[] payload;

	// --------------------------
	// Constructor of an RTPpacket object from header fields and payload
	// bitstream
	// --------------------------
	public RTPpacket(int PType, int Framenb, int Time, byte[] data,
			int data_length) {
		// fill by default header fields:
		Version = 2;
		Padding = 0;
		Extension = 0;
		CC = 0;
		Marker = 0;
		Ssrc = 0;

		// fill changing header fields:
		SequenceNumber = Framenb;
		TimeStamp = Time;
		PayloadType = PType;

		// build the header bistream:
		// --------------------------
		header = new byte[HEADER_SIZE];

		header[1] = (byte) ((Marker << 7) | PayloadType);
```

```
        header[2] = (byte) (SequenceNumber >> 8);
        header[3] = (byte) (SequenceNumber);

        for (int i = 0; i < 4; i++)
                header[7 - i] = (byte) (TimeStamp >> (8 * i));

        for (int i = 0; i < 4; i++)
                header[11 - i] = (byte) (Ssrc >> (8 * i));

        payload_size = data_length;
        payload = new byte[data_length];

        payload = data;


}

// --------------------------
// Constructor of an RTPpacket object from the packet bistream
// --------------------------
public RTPpacket(byte[] packet, int packet_size) {
        // fill default fields:
        Version = 2;
        Padding = 0;
        Extension = 0;
        CC = 0;
        Marker = 0;
        Ssrc = 0;

        // check if total packet size is lower than the header size
        if (packet_size >= HEADER_SIZE) {
                // get the header bitsream:
                header = new byte[HEADER_SIZE];
                for (int i = 0; i < HEADER_SIZE; i++)
                        header[i] = packet[i];

                // get the payload bitstream:
                payload_size = packet_size - HEADER_SIZE;
                payload = new byte[payload_size];
                for (int i = HEADER_SIZE; i < packet_size; i++)
                        payload[i - HEADER_SIZE] = packet[i];

                // interpret the changing fields of the header:
                PayloadType = header[1] & 127;
                SequenceNumber = unsigned_int(header[3]) + 256
                                * unsigned_int(header[2]);
                TimeStamp = unsigned_int(header[7]) + 256 * unsigned_int(header[6])
                                + 65536 * unsigned_int(header[5]) + 16777216
                                * unsigned_int(header[4]);
        }
}

// --------------------------
// getpayload: return the payload bistream of the RTPpacket and its size
```

```
// --------------------------
public int getpayload(byte[] data) {

        for (int i = 0; i < payload_size; i++)
                data[i] = payload[i];

        return (payload_size);
}

// --------------------------
// getpayload_length: return the length of the payload
// --------------------------
public int getpayload_length() {
        return (payload_size);
}

// --------------------------
// getlength: return the total length of the RTP packet
// --------------------------
public int getlength() {
        return (payload_size + HEADER_SIZE);
}

// --------------------------
// getpacket: returns the packet bitstream and its length
// --------------------------
public int getpacket(byte[] packet) {
        // construct the packet = header + payload
        for (int i = 0; i < HEADER_SIZE; i++)
                packet[i] = header[i];
        for (int i = 0; i < payload_size; i++)
                packet[i + HEADER_SIZE] = payload[i];

        // return total size of the packet
        return (payload_size + HEADER_SIZE);
}

// --------------------------
// gettimestamp
// --------------------------

public int gettimestamp() {
        return (TimeStamp);
}

// --------------------------
// getsequencenumber
// --------------------------
public int getsequencenumber() {
        return (SequenceNumber);
}

// --------------------------
// getpayloadtype
// --------------------------
```

```
	public int getpayloadtype() {
		return (PayloadType);
	}

	// --------------------------
	// print headers without the SSRC
	// --------------------------
	public void printheader() {

		for (int i = 0; i < (HEADER_SIZE - 4); i++) {
			for (int j = 7; j >= 0; j--)
				if (((1 << j) & header[i]) != 0)
					System.out.print("1");
				else
					System.out.print("0");
			System.out.print(" ");
		}

		System.out.println();
	}

	// return the unsigned value of 8-bit integer nb
	static int unsigned_int(int nb) {
		if (nb >= 0)
			return (nb);
		else
			return (256 + nb);
	}

}
```

Source code for Client.java:

```
package com.exemple.clientRTSP;


/* ------------------
Client
usage: java Client [Server hostname] [Server RTSP listening port] [Video file requested]
---------------------- */

import java.awt.BorderLayout;
import java.awt.Dimension;
import java.awt.GridLayout;
import java.awt.Image;
import java.awt.Toolkit;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.IOException;
```

```
import java.io.InputStreamReader;
import java.io.InterruptedIOException;
import java.io.OutputStreamWriter;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.net.Socket;
import java.net.SocketException;
import java.util.StringTokenizer;

import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.Timer;

public class Client{

//GUI
//----
JFrame f = new JFrame("Client");
JButton setupButton = new JButton("Setup");
JButton playButton = new JButton("Play");
JButton pauseButton = new JButton("Pause");
JButton tearButton = new JButton("Teardown");
JPanel mainPanel = new JPanel();
JPanel buttonPanel = new JPanel();
JLabel iconLabel = new JLabel();
ImageIcon icon;


//RTP variables:
//----------------
DatagramPacket rcvdp; //UDP packet received from the server
DatagramSocket RTPsocket; //socket to be used to send and receive UDP packets
static int RTP_RCV_PORT = 25000; //port where the client will receive the RTP packets

Timer timer; //timer used to receive data from the UDP socket
byte[] buf; //buffer used to store data received from the server

//RTSP variables
//----------------
//rtsp states
final static int INIT = 0;
final static int READY = 1;
final static int PLAYING = 2;
static int state; //RTSP state == INIT or READY or PLAYING
Socket RTSPsocket; //socket used to send/receive RTSP messages
//input and output stream filters
static BufferedReader RTSPBufferedReader;
static BufferedWriter RTSPBufferedWriter;
static String VideoFileName; //video file to request to the server
int RTSPSeqNb = 0; //Sequence number of RTSP messages within the session
int RTSPid = 0; //ID of the RTSP session (given by the RTSP Server)
```

```
final static String CRLF = "\r\n";

//Video constants:
//------------------
static int MJPEG_TYPE = 26; //RTP payload type for MJPEG video

//--------------------------
//Constructor
//--------------------------
public Client() {

 //build GUI
 //--------------------------

 //Frame
 f.addWindowListener(new WindowAdapter() {
   public void windowClosing(WindowEvent e) {
         System.exit(0);
   }
 });

 //Buttons
 buttonPanel.setLayout(new GridLayout(1,0));
 buttonPanel.add(setupButton);
 buttonPanel.add(playButton);
 buttonPanel.add(pauseButton);
 buttonPanel.add(tearButton);
 setupButton.addActionListener(new setupButtonListener());
 playButton.addActionListener(new playButtonListener());
 pauseButton.addActionListener(new pauseButtonListener());
 tearButton.addActionListener(new tearButtonListener());

 //Image display label
 iconLabel.setIcon(null);

 //frame layout
 mainPanel.setLayout(null);
 mainPanel.add(iconLabel);
 mainPanel.add(buttonPanel);
 iconLabel.setBounds(0,0,380,280);
 buttonPanel.setBounds(0,280,380,50);

 f.getContentPane().add(mainPanel, BorderLayout.CENTER);
 f.setSize(new Dimension(390,370));
 f.setVisible(true);

 //init timer
 //--------------------------
 timer = new Timer(20, new timerListener());
 timer.setInitialDelay(0);
 timer.setCoalesce(true);

 //allocate enough memory for the buffer used to receive data from the server
 buf = new byte[15000];
```

```
}

//-----------------------------------
//main
//-----------------------------------
public static void main(String argv[]) throws Exception
{
 //Create a Client object
 Client theClient = new Client();

 //get server RTSP port and IP address from the command line
 //------------------
 int RTSP_server_port = 8554; //Integer.parseInt(argv[1]);
 String ServerHost ="192.168.0.2"; //argv[0];
 InetAddress ServerIPAddr = InetAddress.getByName(ServerHost);

 //get video filename to request:
 VideoFileName = "media/movie.Mjpeg"  ; //argv[2];

 //Establish a TCP connection with the server to exchange RTSP messages
 //------------------
// theClient.RTSPsocket = new Socket(ServerIPAddr, RTSP_server_port);


 theClient.RTSPsocket = new Socket("192.168.0.2", 8554);

 //Set input and output stream filters:
 RTSPBufferedReader = new BufferedReader(new
InputStreamReader(theClient.RTSPsocket.getInputStream()) );
 RTSPBufferedWriter = new BufferedWriter(new
OutputStreamWriter(theClient.RTSPsocket.getOutputStream()) );

 //init RTSP state:
 state = INIT;
}


//-----------------------------------
//Handler for buttons
//-----------------------------------

//.............
//TO COMPLETE
//.............

//Handler for Setup button
//-----------------------
class setupButtonListener implements ActionListener{
 public void actionPerformed(ActionEvent e){

  //System.out.println("Setup Button pressed !");

  if (state == INIT)
        {
          //Init non-blocking RTPsocket that will be used to receive data
```

```
            try{
             //construct a new DatagramSocket to receive RTP packets from the server, on port
RTP_RCV_PORT
             //RTPsocket = ...
                        RTPsocket = new DatagramSocket(RTP_RCV_PORT);
                    //   set TimeOut value of the socket to 5msec.

                                RTPsocket.setSoTimeout(5);
             //set TimeOut value of the socket to 5msec.
             //....
           }
            catch (SocketException se)
             {
               System.out.println("Socket exception: "+se);
               System.exit(0);
             }

            //init RTSP sequence number
            RTSPSeqNb = 1;

            //Send SETUP message to the server
            send_RTSP_request("SETUP");

            //Wait for the response
            if (parse_server_response() != 200)
              System.out.println("Invalid Server Response");
            else
             {
               //change RTSP state and print new state
                     state = READY;
               System.out.println("New RTSP state: READY");
               //System.out.println("New RTSP state: ....");
             }
          }//else if state != INIT then do nothing
 }
}

//Handler for Play button
//-----------------------
class playButtonListener implements ActionListener {
 public void actionPerformed(ActionEvent e){

            System.out.println("Play Button pressed !");

    if (state == READY)
          {



  //increase RTSP sequence number



            RTSPSeqNb++;
```

```
//send PLAY message to the server
            send_RTSP_request("PLAY");
//   wait for the response

            if (parse_server_response() != 200)
                    System.out.println("Invalid Server Response");
            else
              {
//   change RTSP state and print out new state

                state=PLAYING;
                System.out.println("New RTSP state: PLAYING");

//   start the timer
                timer.start();
              }
           }//else if state != READY then do nothing
  }
}


//Handler for Pause button
//-----------------------
class pauseButtonListener implements ActionListener {
 public void actionPerformed(ActionEvent e){

  //System.out.println("Pause Button pressed !");

  if (state == PLAYING)
          {
            //increase RTSP sequence number
     RTSPSeqNb++;
            //........

            //Send PAUSE message to the server
            send_RTSP_request("PAUSE");

            //Wait for the response
           if (parse_server_response() != 200)
                    System.out.println("Invalid Server Response");
            else
              {
                //change RTSP state and print out new state
                //........
                    state = READY;
                //System.out.println("New RTSP state: ...");

                //stop the timer
                timer.stop();
              }
          }
  //else if state != PLAYING then do nothing
 }
}
```

```
//Handler for Teardown button
//-----------------------
class tearButtonListener implements ActionListener {
 public void actionPerformed(ActionEvent e){

  //System.out.println("Teardown Button pressed !");

  //increase RTSP sequence number
  // ..........


  //Send TEARDOWN message to the server
  send_RTSP_request("TEARDOWN");

  //Wait for the response
  if (parse_server_response() != 200)
        System.out.println("Invalid Server Response");
  else
        {
         //change RTSP state and print out new state
         //........
         //System.out.println("New RTSP state: ...");

         //stop the timer
         timer.stop();

         //exit
         System.exit(0);
        }
 }
}


//-----------------------------------
//Handler for timer
//-----------------------------------

class timerListener implements ActionListener {
 public void actionPerformed(ActionEvent e) {

  //Construct a DatagramPacket to receive data from the UDP socket
  rcvdp = new DatagramPacket(buf, buf.length);

  try{
        //receive the DP from the socket:
        RTPsocket.receive(rcvdp);

        //create an RTPpacket object from the DP
        RTPpacket rtp_packet = new RTPpacket(rcvdp.getData(), rcvdp.getLength());

        //print important header fields of the RTP packet received:
        System.out.println("Got RTP packet with SeqNum #
"+rtp_packet.getsequencenumber()+" TimeStamp "+rtp_packet.gettimestamp()+" ms, of type
"+rtp_packet.getpayloadtype());
```

```
            //print header bitstream:
            rtp_packet.printheader();

            //get the payload bitstream from the RTPpacket object
            int payload_length = rtp_packet.getpayload_length();
            byte [] payload = new byte[payload_length];
            rtp_packet.getpayload(payload);

            //get an Image object from the payload bitstream
            Toolkit toolkit = Toolkit.getDefaultToolkit();
            Image image = toolkit.createImage(payload, 0, payload_length);

            //display the image as an ImageIcon object
            icon = new ImageIcon(image);
            iconLabel.setIcon(icon);
        }
        catch (InterruptedIOException iioe){
            //System.out.println("Nothing to read");
        }
        catch (IOException ioe) {
            System.out.println("Exception caught: "+ioe);
        }
    }
}

//------------------------------------
//Parse Server Response
//------------------------------------
private int parse_server_response()
{
 int reply_code = 0;

 try{
   //parse status line and extract the reply_code:
   String StatusLine = RTSPBufferedReader.readLine();
   //System.out.println("RTSP Client - Received from Server:");
   System.out.println(StatusLine);

   StringTokenizer tokens = new StringTokenizer(StatusLine);
   tokens.nextToken(); //skip over the RTSP version
   reply_code = Integer.parseInt(tokens.nextToken());

   //if reply code is OK get and print the 2 other lines
   if (reply_code == 200)
        {
          String SeqNumLine = RTSPBufferedReader.readLine();
          System.out.println(SeqNumLine);

          String SessionLine = RTSPBufferedReader.readLine();
          System.out.println(SessionLine);

          //if state == INIT gets the Session Id from the SessionLine
          tokens = new StringTokenizer(SessionLine);
          tokens.nextToken(); //skip over the Session:
          RTSPid = Integer.parseInt(tokens.nextToken());
```

```
        }
 }
 catch(Exception ex)
  {
        System.out.println("Exception caught: "+ex);
        System.exit(0);
  }

 return(reply_code);
}

//---------------------------------
//Send RTSP Request
//---------------------------------

//.............
//TO COMPLETE
//.............

private void send_RTSP_request(String request_type)
{
        try{

                RTSPBufferedWriter.write(request_type+" "+VideoFileName+"
"+"RTSP/1.0"+CRLF);

                RTSPBufferedWriter.write("CSeq: "+RTSPSeqNb+CRLF);

                if(request_type.equals("SETUP")) {
                   RTSPBufferedWriter.write("Transport: RTP/UDP; client_port=
"+RTP_RCV_PORT+CRLF);
                }

                else {
                   RTSPBufferedWriter.write("Session: "+RTSPid+CRLF);
                }

                RTSPBufferedWriter.flush();
              }
              catch(Exception ex)
                {
                     System.out.println("Exception caught: "+ex);
                }
            }

          }//end of Class Client
```

References:

[1] http://searchnetworking.techtarget.com/definition/Real-Time-Transport-Protocol

[2] https://ackspace.nl/wiki/RTP_packet_streaming

[3] http://www.siptutorial.net/RTP/header.html

[4]http://www.slideshare.net/Ronny72/rtprtcp-ppt