NAME: RAKESH DATTA
TEAM ID: 12
Class ID: 15
SJSU ID: 010808447

1. What Extra Problem did I solve?

In our main project we have implemented concurrent RTP video streaming. Here client requests server for a static video. Server accepts the connection and always unicasts one specific video file using RTP.

Problem 1: Client can't request for different videos at different connection requests, like we do it in youtube.com.

Problem 2: Clients can't control the state (play/pause/terminate) of the incoming video stream, like we do it in youtube.com.

In my individual feature, I have implemented a layer of Signaling Protocol (RTSP) on top of the media protocol (RTP), that allows me to solve the above mentioned problems.

2. Detailed Explanation

2.1 Current Server Architecture (basic course project)

A thread is spawned on accepting each Client TCP Connect Request. To start with, the thread would close the TCP socket first and then would open up a UDP socket to push RTP video packets back to the client.

2.2 Modified Server Architecture (individual work)

The Main thread keeps listening for new connection request from Client(s). On receiving a connection request, the main thread spawns two threads for handling the client- a master (Class rtspThread) and a worker thread (Class threadOfClient).
The RTSP master thread keeps listening for RTSP messages on the TCP socket, parses them and invokes relevant message handlers.
The RTP worker thread opens up a UDP socket to push RTP video packets back to the client.

So the server maintains two sockets for each Client Connection. One TCP socket for signaling (RTSP) and one UDP socket for media stream(RTP).

2.3. What is RTSP?

Real Time Streaming Protocol (RTSP) is a Layer 7 protocol that is used to setup and control real-time media streams. It is used to establish a media session between clients and servers. Clients use

various RTSP message like SETUP, PLAY, PAUSE and TERMINATE to control the media streaming. Transmission of data packets is taken care of by the protocols such as RTP. Using RTSP client/server can manage (play, pause, record etc.) a RTP media stream.

2.4 Solution for Problem 1

The worker thread class maintains a member variable called *video_src* that is initialized with a default video source file path. On receiving a RTSP SETUP message, the master thread parses it and finds the file name mentioned in the message request line. If such a file is existing in the sever, the master thread sets *video_src* variable as the new file name. Worker thread uses this modified *video_src* variable as the source of video stream to be played back to the client. In this way, clients can request for different video files at different times, as part of RTSP SETUP messages. For example,

C->S: **SETUP** video.mjpeg RTSP/1.0
CSeq: 200
Transport: RTP/UDP;unicast;client_port=7000

S->C: RTSP/1.0 **200 OK**
CSeq: 200
SessionID: 1000

2.4 Solution for Problem 2

When the RTP worker thread is spawned, it continuously tries to send a video packet to the client. However, before sending the packet, the worker thread checks the state of a member variable called *keepStreaming,* present in the worker thread class. If *keepStreaming* is true, server sends a UDP video packet back to the client. If false, server does not send a UDP packet to the client.

PLAY: On receiving RTSP Play request, the RTSP master thread sets the *keepStreaming* variable in RTP Worker thread to TRUE. This ensures that the worker thread keeps transmitting RTP video packet back to the client, until the *keepStreaming* variable becomes FALSE.

C->S: **PLAY** RTSP/1.0
CSeq: 201
Transport: RTP/UDP;unicast;client_port=7000
SessionID: 1000

PAUSE: On receiving RTSP Pause request, the RTSP master thread sets the *keepStreaming* variable to FALSE. This ensures that the worker thread does not transmit RTP video packets back to the client, until the *keepStreaming* variable becomes TRUE.

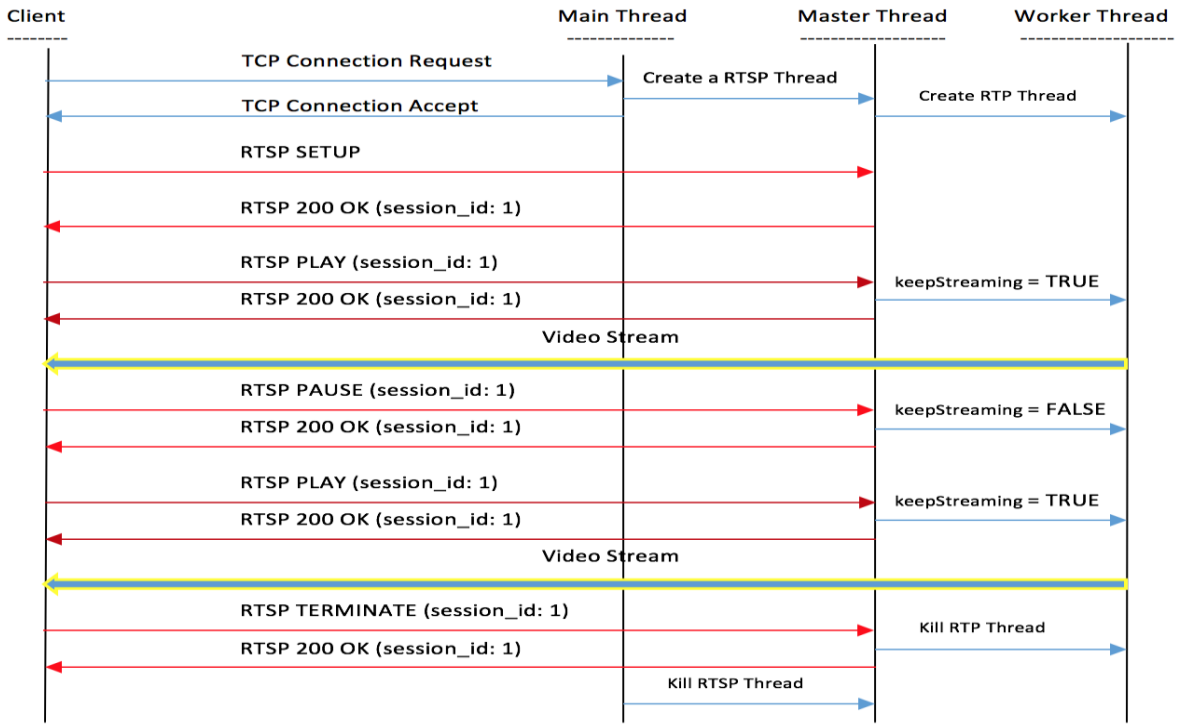
C->S: **PAUSE** RTSP/1.0
CSeq: 202
Transport: RTP/UDP;unicast;client_port=7000
SessionID: 1000

TERMINATE: On receiving RTSP Terminate request, the master thread sets the *keepStreaming*

variable to FALSE. Additionally, it closes the UDP Socket and kills the worker thread. Thereafter, the master thread kills its own TCP socket, and thus the connection between client and server is completely terminated.

C->S: **TERMINATE** RTSP/1.0
CSeq: 203
Transport: RTP/UDP;unicast;client_port=7000
SessionID: 1000

2.5 Overall Feature Ladder Diagram:



3.Conclusion

3.1 Unit Testing:

TC#	TEST CASE	EXPECTATION	RESULT
1	**Client sets up a connection with Server and sends a RTSP SETUP request to play 'video.mjpeg'**	**Server sends back RTSP 200 OK**	**Pass**
2	**Client sends RTSP PLAY on the established connection**	**Server sends back RTSP 200 OK and video is being played in the client App**	**Pass**
3	**Client sends RTSP PAUSE on the established connection**	**Server sends back RTSP 200 OK and video is being paused in the client App**	**Pass**
4	**Client sends RTSP PLAY on the established connection**	**Server sends back RTSP 200 OK and video is being re-played in the client App**	**Pass**
	Client sends RTSP TERMINATE on the established connection	**Server sends back RTSP 200 OK and subsequently the connection is terminated**	**Pass**

5	**Two clients are spawned in parallel. While one client is playing video, other client does play& pause**	**Video keeps playing in Client1. Video is paused/played in Client2, based on the action performed in Client2**	**Pass**

3.2 Code:

rtspThread: This Class is coded as part of individual Feature work and used for creating RTSP master threads. In the run method, it creates a RTP worker thread using threadOfClient Class. It maintains the TCP socketID in *socket* variable.

threadOfClient: This Class is coded as part of Main project. It is used for creating RTP Worker Threads. As part of individual feature, two new functions *setStreaming*() and *setVideoSrc*() are coded, which sets the variables *keepStreaming* and *video_src* respectively.

```
public static void main() {
    //RAKESH EXTRA CODE: START
    rtspThread rThread = new rtspThread(a, SocketOfClient[a], threads);
    rThread.start();
    System.out.println("Created RTSP Thread ID: " + rThread.getId());
    //RAKESH EXTRA CODE: END
}

// Worker/RTP Thread
class threadOfClient extends Thread {
    //RAKESH EXTRA CODE: START
    private String video_src;
    private boolean keepStreaming;
    public void setStreaming(boolean flag) {
        this.keepStreaming = flag;
    }
    public void setVideoSrc(String vid) {
         video_src = String("/Users/rdatta/CMPE207/Group12/”)
         video_src += vid;
          video = new StreamVideo(video_src);
    }

    //RAKESH EXTRA CODE: END

    public threadOfClient(Socket SocketOfClient, threadOfClient[] threads) {
        this.SocketOfClient = SocketOfClient;
        this.threads = threads;
        maxClientsCount = threads.length;
        //RAKESH EXTRA CODE: START
        video_src = "/Users/rdatta/CMPE207/Group12/video.mjpeg";
        video = new StreamVideo(video_src);;
        this.keepStreaming = false;
        //RAKESH EXTRA CODE: END
    }

}

//RAKESH EXTRA CODE: START
// Master/RTSP Thread
class rtspThread extends Thread {
    private Socket socket = null;
    private StreamVideo video;
    private final threadOfClient[] threads;
    private int sessid;
    private String video_file;

    public rtspThread(int sessID, Socket rtspSocket, threadOfClient[] threads, StreamVideo vid) {
        this.socket = rtspSocket;
        this.threads = threads;
        video = vid;
        sessid = sessID;
```

```
    }
    public void run() {
        try {
            System.out.println("RTSP Thread Created");
            System.out.println("Client IP:" + socket.getInetAddress());
            System.out.println("Client Port " + socket.getPort());
            int port = socket.getPort();
            BufferedWriter outStr = new BufferedWriter(new OutputStreamWriter(socket.getOutputStr
eam()));
            String msg = port + "n";
            outStr.write(msg);
            outStr.flush();
            BufferedReader inputStr = new BufferedReader(new InputStreamReader(socket.getInputStr
eam()));
            /* Extract the Message headers and parse*/
            System.out.println("Reached here");
            // Keep Reading RTSP Packets
            threadOfClient t = new threadOfClient(this.socket, this.threads, this.video);
            t.start();
            int thread_id = (int) t.getId();
            System.out.println("RTP Thread created; Thread ID: " + thread_id);
            while (true) {
                if (inputStr.ready()) {
                    System.out.println("Received a msg in RTSP thread");
                    String requestLine = inputStr.readLine();
                    String cseqLine = inputStr.readLine();
                    String transportLine = inputStr.readLine();
                    String[] strTokens = requestLine.split(" ");
                    String messageType = strTokens[0];
                     String video_file= strTokens[1];
                    strTokens = cseqLine.split(" ");
                    String seqNum = strTokens[1];
                    strTokens = transportLine.split(" ");
                    String[] members = strTokens[1].split(";");
                    boolean multicast = ((members[1] == "multicast") ? true : false);
                    String clientRTPPort = (members[2].split("="))[1];
                    int sID = 0;
                    if (!messageType.equals("SETUP")) {
                        String SID = inputStr.readLine();
                        strTokens = SID.split(" ");
                        sID = Integer.parseInt(strTokens[1]);
                    }
                    if (sID != this.sessid) {
                        System.out.println("ERROR: Session ID mismatch");
                    }
                    /* Print the Incoming Message*/
                    System.out.println("====================================");
                    System.out.println("MESSAGE RECIEVED: " + "n");
                    System.out.println(requestLine);
                    System.out.println(cseqLine);
                    System.out.println(transportLine);
                    /* Let RTSP Handler take care of this message and send response*/
                    System.out.println("====================================");
                    System.out.println("INFO : messageType = " + messageType);
                    System.out.println("INFO : seqNum = " + seqNum);
                    System.out.println("INFO : sessId = " + sessid);
                    System.out.println("INFO : clientRTPPort = " + clientRTPPort);
                    System.out.println("INFO : multicast = " + ((multicast) ? "true" : "false"));
                    PrintWriter resp = new PrintWriter(socket.getOutputStream(), true);
                    resp.write("RTSP/1.0 200 OK" + "rn");
                    resp.write("CSeq: " + seqNum + "rn");
                    //resp.write("Date: "+date+NEWLINE);
                    resp.write("SessionID: " + sessid + "rn");
                    resp.flush();
                    switch (messageType) {
                        case "SETUP":
                        t.setVideoSrc(video_file);
                        System.out.println("Thread " + thread_id + " Streaming is Set up");
                         System.out.println("Video File to be played: "+ video_file);
                        break;
                        case "PLAY":
```

```
                        t.setStreaming(true);
                        System.out.println("Thread " + thread_id + " Streaming Started");
                        break;
                        case "PAUSE":
                        System.out.println("Thread " + thread_id + " Streaming Paused");
                        t.setStreaming(false);
                        break;
                        case "TEARDOWN":
                        System.out.println("Thread " + thread_id + " Streaming destroyed");
                        t.setStreaming(false);
                        break;
                        default:
                        System.out.println("ERROR: Unknown Message Type");
                        break;
                    }
                }
            }
            } catch (IOException ioe) {
            System.out.println(ioe);
            } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
//RAKESH EXTRA CODE: END
```

3.3 Snapshots: