HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY

Faculty of Computer Science & Engineering

Computer Networks
Assignment 1

Instructor: Mr. Nguyen Manh Thin

Group: *4 - CC04*
Students: *Mai Ton Nhat Khanh - 1852038*
Vo Le Hai Dang - 1852321
Cao Hoang Kiet - 2053165
Cao Tuan Kiet - 2053166

Ho Chi Minh City, November 2021

Contents

1 Requirement Analysis

1.1 The Client

Our first task is to implement the RTSP protocol on the client side. To do this, we need to complete the functions that are called when the user clicks on the buttons on the user interface. we will need to implement the actions for the following request types. When the client starts, it also opens the RTSP socket to the server. Use this socket for sending all RTSP requests.

1.1.1 SET UP

- Send SETUP request to the server. We will need to insert the Transport header in which we specify the port for the RTP data socket we just created.
- Read the server's response and parse the Session header (from the response) to get the RTSP session ID.
- Create a datagram socket for receiving RTP data and set the timeout on the socket to 0.5 seconds.

1.1.2 PLAY

- Send PLAY request. We must insert the Session header and use the session ID returned in the SETUP response. We must not put the Transport header in this request.
- Read the server's response.

1.1.3 PAUSE

- Send PAUSE request. We must insert the Session header and use the session ID returned in the SETUP response. We must not put the Transport header in this request.
- Read the server's response.

1.1.4 TEARDOWN

- Send TEARDOWN request. We must insert the Session header and use the session ID returned in the SETUP response. We must not put the Transport header in this request.

- Read the server's response. Note: We must insert the CSeq header in every request you send. The value of the CSeq header is a number which starts at 1 and is incremented by one for each request We send.

1.2 The Server

On the server side, we will need to implement the packetization of the video data into RTP packets. We 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. When the server receives the PLAY-request from the client, the server reads one video frame from the file and creates a RtpPacket-object which is the RTP-encapsulation of the video frame. It then sends the frame to the client over UDP every 50 milliseconds. For the encapsulation, the server calls the encode function of the RtpPacket class. My task is to write this function. We will need to do the following: (the letters in parenthesis refer to the fields in the RTP packet format below).

- Set the RTP-version field (V). You must set this to 2.
- Set padding (P), extension (X), number of contributing sources (CC), and marker (M) fields. These are all set to zero in this lab.
- Set payload type field (PT). In this lab we use MJPEG and the type for that is 26.
- Set the sequence number. The server gives this the sequence number as the frameNbr argument to the encode function.
- Set the timestamp using the Python's time module.
- 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, or the first three lines from the diagram below.

1.3 Functional

- The movie can be stopped, played, paused.
- Multiple people can use the server and watch the movie at the same time.

1.4 Non-functional

- The server needs to respond within 1 second.

2 Function Description

2.1 Client.py

- __init__: Initialize the program
- createWidgets: Build the users interface to play the video
- setupMovie: Set up the movie ready to be played
- exitClient: Terminate the program
- pauseMovie: Handle the pause signal sent by the user
- playMovie: Handle the play signal sent by the user
- listenRtp: Listen for RTP packets
- writeFrame: Write the received frame to a temp image file
- updateMovie: Update the image file as video frame in the GUI
- connectToServer: Connect to the Server. Start a new RTSP/TCP session
- sendRtspRequest: Send RTSP request to the server
- recvRtspReply: Receive RTSP reply from the server
- parseRtspReply: Parse the RTSP reply from the server
- openRtpPort: Open RTP socket binded to a specified port
- handler: Handler on explicitly closing the GUI window

2.2 RtpPacket.py

- __init__: pass(didn't implement)
- encode: Encode the RTP packet with header fields and payload
- decode: Decode the RTP packet
- version: Return RTP version
- seqNum: Return sequence (frame) number
- timestamp: Return timestamp
- payloadType: Return payload type
- getPayload: Return payload
- getPacket: Return RTP packet

2.3 Server.py

- main: Initialize the server

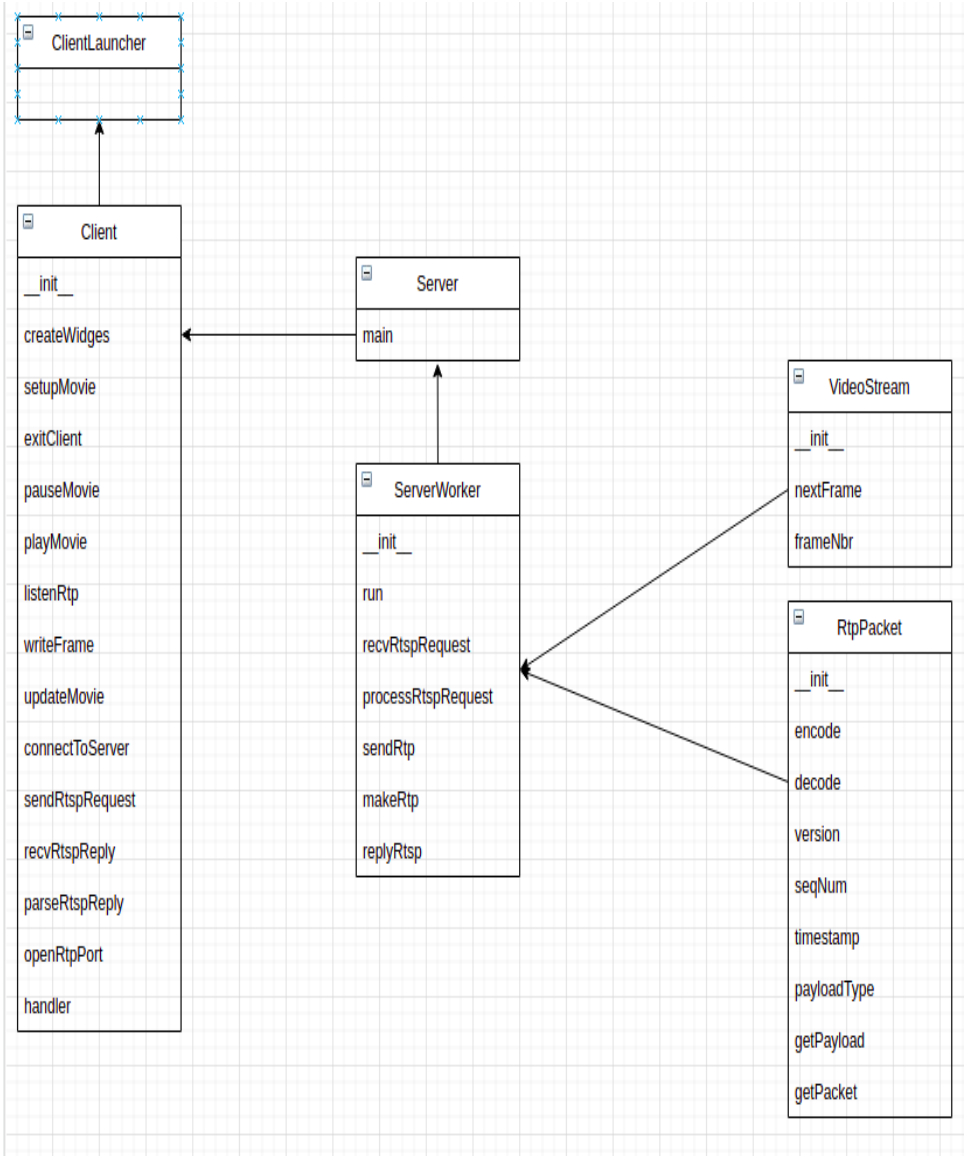
2.4 ServerWorker.py

- __init__: Initialize the client info
- run: Initialize the thread to run the program
- recvRtspRequest: Receive RTSP request from the client
- processRtspRequest: Process RTSP request sent from the client
- sendRtp: Send RTP packets over UDP
- makeRtp: RTP-packetize the video data
- replyRtsp: Send RTSP reply to the client

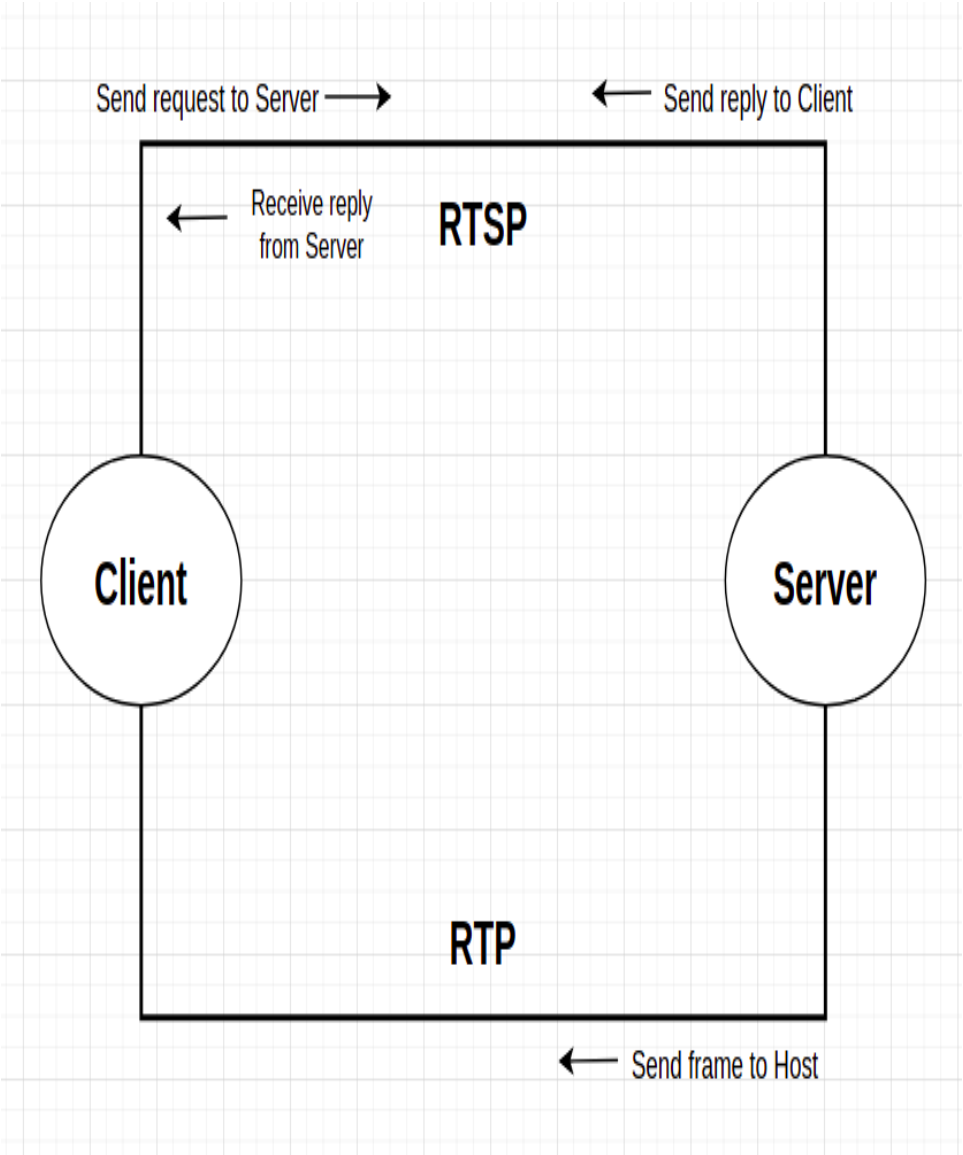
2.5 VideoStream.py

- __init__: Initialize the filename
- nextFrame: Get next frame
- frameNbr: Get frame number

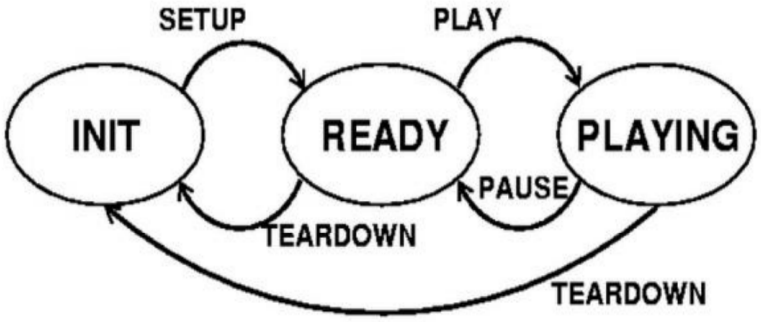
3 Class diagram



4 Model and data flow



5 Summary results



As shown in the figure above, there are 3 state of the Client launcher, INIT, READY, PLAYING. If we start the program, it will automatically set up the movie for streaming. The users can press PLAY to watch the movie, press PAUSE to temporarily pause the movie and then continue to watch it. If the users press STOP, it will stop the movie and set it back to its initial state which is the beginning.

After finishing this assignment, we've study on how a server operate and how to implement a server. Furthermore, we also learned on how to implement a small program that allows us to connect to server and stream a movie.

6 User manual

First you need to open the program folder with Visual Studio Code and go to the terminal, type in the following command:

- python2 Server.py <Port number> (Port number is any number greater than 1024)

Then start a new terminal and type in the following command:

- python2 ClientLauncher.py <Server's IP> <Server's port number> <RTP's port> <Video file>

7 Full source code

7.1 Client.py

```
from Tkinter import *
from PIL import Image, ImageTk
import socket, threading, sys, traceback, os

from RtpPacket import RtpPacket

CACHE_FILE_NAME = "cache-"
CACHE_FILE_EXT = ".jpg"

class Client:

    SETUP_STR = 'SETUP'
    PLAY_STR = 'PLAY'
    PAUSE_STR = 'PAUSE'
    TEARDOWN_STR = 'TEARDOWN'
    INIT = 0
    READY = 1
    PLAYING = 2
    state = INIT

    SETUP = 0
    PLAY = 1
    PAUSE = 2
    TEARDOWN = 3

    RTSP_VER = "RTSP/1.0"
    TRANSPORT = "RTP/UDP"



        # Initiation..
    def __init__(self, master, serveraddr, serverport, rtpport, filename):
        self.master = master
        self.master.protocol("WM_DELETE_WINDOW", self.handler)
        self.createWidgets()
        self.serverAddr = serveraddr
        self.serverPort = int(serverport)
```

```
38          self.rtpPort = int(rtpport)
39          self.fileName = filename
40          self.rtspSeq = 0
41          self.sessionId = 0
42          self.requestSent = -1
43          self.teardownAcked = 0
44          self.connectToServer()
45          self.frameNbr = 0
46
47      def createWidgets(self):
48      """Build GUI."""
49      # Create Setup button
50          self.setup = Button(self.master, width=20, padx=3,pady=3)
51          self.setup["text"] = "Setup"
52          self.setup["command"] = self.setupMovie
53          self.setup.grid(row=1, column=0, padx=2, pady=2)
54
55          # Create Play button
56          self.start = Button(self.master, width=20, padx=3, pady=3)
57          self.start["text"] = "Play"
58          self.start["command"] = self.playMovie
59          self.start.grid(row=1, column=1, padx=2, pady=2)
60
61          # Create Pause button
62          self.pause = Button(self.master, width=20, padx=3,
63                  pady=3)
64          self.pause["text"] = "Pause"
65          self.pause["command"] = self.pauseMovie
66          self.pause.grid(row=1, column=2, padx=2, pady=2)
67
68          # Create Teardown button
69          self.teardown = Button(self.master, width=20, padx=3, pady=3)
70          self.teardown["text"] = "Teardown"
71          self.teardown["command"] =  self.exitClient
72          self.teardown.grid(row=1, column=3, padx=2, pady=2)
73
74          # Create a label to display the movie
75          self.label = Label(self.master, height=19)
76          self.label.grid(row=0, column=0,
77          columnspan=4,sticky=W+E+N+S,padx=5, pady=5)
78
```

```
79      def setupMovie(self):
80          """Setup button handler."""
81          if self.state == self.INIT:
82              self.sendRtspRequest(self.SETUP)
83
84      def exitClient(self):
85          """Teardown button handler."""
86          self.sendRtspRequest(self.TEARDOWN)
87
88          # Close the gui window
89          self.master.destroy()
90
91          # Delete the cache image from video
92          os.remove(CACHE_FILE_NAME + str(self.sessionId) + CACHE_FILE_EXT)
93
94      def pauseMovie(self):
95          """Pause button handler."""
96          if self.state == self.PLAYING:
97              self.sendRtspRequest(self.PAUSE)
98
99      def playMovie(self):
100         """Play button handler."""
101         if self.state == self.READY:
102             # Create a new thread to listen for RTP packets
103             threading.Thread(target=self.listenRtp).start()
104             self.playEvent = threading.Event()
105             self.playEvent.clear()
106             self.sendRtspRequest(self.PLAY)
107
108     def listenRtp(self):
109         """Listen for RTP packets."""
110         while True:
111             try:
112                 print("LISTENING...")
113                 data = self.rtpSocket.recv(20480)
114                 if data:
115                     rtpPacket = RtpPacket()
116                     rtpPacket.decode(data)
117
118                     currFrameNbr = rtpPacket.seqNum()
119                     print ("CURRENT SEQUENCE NUM: " + str(currFrameNbr))
```

```
120
121                     if currFrameNbr > self.frameNbr:
122                         self.frameNbr = currFrameNbr
123                         self.updateMovie(self.writeFrame(
124                         rtpPacket.getPayload()))
125                 except:
126                 # Stop listening upon requesting PAUSE or TEARDOWN
127                 if self.playEvent.isSet():
128                     break
129
130                 # Upon receiving ACK for TEARDOWN request,
131                 # Close the RTP socket
132                 if self.teardownAcked == 1:
133                     self.rtpSocket.shutdown(socket.SHUT_RDWR)
134                     self.rtpSocket.close()
135                     break
136
137     def writeFrame(self, data):
138         """Write the received frame to a temp image file.
139         Return the image file."""
140         cachename = CACHE_FILE_NAME + str(self.sessionId) + CACHE_FILE_EXT
141         file = open(cachename, "wb")
142         file.write(data)
143         file.close()
144
145         return cachename
146
147     def updateMovie(self, imageFile):
148         """Update the image file as video frame in the GUI."""
149         photo = ImageTk.PhotoImage(Image.open(imageFile))
150         self.label.configure(image = photo, height=288)
151         self.label.image = photo
152
153     def connectToServer(self):
154         """Connect to the Server. Start a new RTSP/TCP session."""
155         self.rtspSocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
156         try:
157             self.rtspSocket.connect((self.serverAddr, self.serverPort))
158         except:
159             messagebox.showwarning('Connection Failed',
160             'Connection to \'%s\' failed.' %self.serverAddr)
```

```
161
162     def sendRtspRequest(self, requestCode):
163         """Send RTSP request to the server."""
164         #-------------
165         # TO COMPLETE
166         #-------------
167
168         # Setup request
169         if requestCode == self.SETUP and self.state == self.INIT:
170             threading.Thread(target=self.recvRtspReply).start()
171
172             # Update RTSP sequence number.
173             self.rtspSeq+=1
174
175             # Write the RTSP request to be sent.
176             request = "%s %s %s" %
177             (self.SETUP_STR,self.fileName,self.RTSP_VER)
178             request+="\nCSeq: %d" % self.rtspSeq
179             request+="\nTransport: %s; client_port= %d" %
180             (self.TRANSPORT,self.rtpPort)
181
182             # Keep track of the sent request.
183             self.requestSent = self.SETUP
184
185             # Play request
186         elif requestCode == self.PLAY and self.state == self.READY:
187
188             # Update RTSP sequence number.
189             self.rtspSeq+=1
190
191             # Write the RTSP request to be sent.
192             request = "%s %s %s" %
193             (self.PLAY_STR,self.fileName,self.RTSP_VER)
194             request+="\nCSeq: %d" % self.rtspSeq
195             request+="\nSession: %d"%self.sessionId
196
197
198             # Keep track of the sent request.
199             self.requestSent = self.PLAY
200
201
```

```
202             # Pause request
203         elif requestCode == self.PAUSE and self.state == self.PLAYING:
204
205             # Update RTSP sequence number.
206             self.rtspSeq+=1
207
208             request = "%s %s %s" %
209             (self.PAUSE_STR,self.fileName,self.RTSP_VER)
210             request+="\nCSeq: %d" % self.rtspSeq
211             request+="\nSession: %d"%self.sessionId
212
213             self.requestSent = self.PAUSE
214
215             # Teardown request
216         elif requestCode == self.TEARDOWN and not self.state == self.INIT:
217
218             # Update RTSP sequence number.
219             self.rtspSeq+=1
220
221             # Write the RTSP request to be sent.
222             request = "%s %s %s" % (self.TEARDOWN_STR,
223             self.fileName, self.RTSP_VER)
224             request+="\nCSeq: %d" % self.rtspSeq
225             request+="\nSession: %d" % self.sessionId
226
227             self.requestSent = self.TEARDOWN
228
229         else:
230             return
231
232         # Send the RTSP request using rtspSocket.
233         self.rtspSocket.send(request)
234
235         print ('\nData Sent:\n' + request)
236
237     def recvRtspReply(self):
238         """Receive RTSP reply from the server."""
239         while True:
240             reply = self.rtspSocket.recv(1024)
241
242             if reply:
```

```
                self.parseRtspReply(reply)

            # Close the RTSP socket upon requesting Teardown
            if self.requestSent == self.TEARDOWN:
                self.rtspSocket.shutdown(socket.SHUT_RDWR)
                self.rtspSocket.close()
                break

    def parseRtspReply(self, data):
        """Parse the RTSP reply from the server."""
        lines = data.split('\n')
        seqNum = int(lines[1].split(' ')[1])

        # Process only if the server reply's sequence
        number is the same as the request's
        if seqNum == self.rtspSeq:
            session = int(lines[2].split(' ')[1])
            # New RTSP session ID
            if self.sessionId == 0:
                self.sessionId = session

            # Process only if the session ID is the same
            if self.sessionId == session:
                if int(lines[0].split(' ')[1]) == 200:
                    if self.requestSent == self.SETUP:
                        #-------------
                        # TO COMPLETE
                        #-------------


                        # Update RTSP state.
                        self.state = self.READY

                        # Open RTP port.
                        self.openRtpPort()
                    elif self.requestSent == self.PLAY:
                        self.state = self.PLAYING
                    elif self.requestSent == self.PAUSE:
                        self.state = self.READY


```

```
284             # The play thread exits. A new thread is created on resume.
285                         self.playEvent.set()
286                     elif self.requestSent == self.TEARDOWN:
287                         self.state = self.INIT
288 
289                         # Flag the teardownAcked to close the socket.
290                         self.teardownAcked = 1
291 
292     def openRtpPort(self):
293         """Open RTP socket binded to a specified port."""
294 
295         #-------------
296         # TO COMPLETE
297         #-------------
298 
299 
300         # Create a new datagram socket to receive RTP packets from
301         # the server
302         self.rtpSocket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
303 
304 
305         # Set the timeout value of the socket to 0.5sec
306         self.rtpSocket.settimeout(0.5)
307 
308         try:
309         # Bind the socket to the address using the RTP port given by
310         # the client user.
311             self.state=self.READY
312             self.rtpSocket.bind(('',self.rtpPort))
313         except:
314             messagebox.showwarning('Unable to Bind', 'Unable to bind
315             PORT=%d' %self.rtpPort)
316 
317     def handler(self):
318         """Handler on explicitly closing the GUI window."""
319         self.pauseMovie()
320         if messagebox.askokcancel("Quit?", "Are you sure you want to quit?"):
321             self.exitClient()
322         else: # When the user presses cancel, resume playing.
323             self.playMovie()
```

7.2 ClientLauncher.py

```
import sys
from Tkinter import Tk
from Client import Client

if __name__ == "__main__":
    try:
        serverAddr = sys.argv[1]
        serverPort = sys.argv[2]
        rtpPort = sys.argv[3]
        fileName = sys.argv[4]
    except:
        print("[Usage: ClientLauncher.py Server_name Server_port
RTP_port Video_file]\n")

    root = Tk()

    # Create a new client
    app = Client(root, serverAddr, serverPort, rtpPort, fileName)
    app.master.title("RTPClient")
    root.mainloop()
```

7.3 Server.py

```
import sys, socket
from ServerWorker import ServerWorker
class Server:

    def main(self):
        try:
            SERVER_PORT = int(sys.argv[1])
        except:
            print("[Usage: Server.py Server_port]\n")
        rtspSocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        rtspSocket.bind(('', SERVER_PORT))
        rtspSocket.listen(5)

        # Receive client info (address,port) through RTSP/TCP session
        while True:
            clientInfo = {}
```

```
            clientInfo['rtspSocket'] = rtspSocket.accept()
            ServerWorker(clientInfo).run()

if __name__ == "__main__":
    (Server()).main()
```

7.4 RtpPacket.py

```
import sys
from time import time
HEADER_SIZE = 12


class RtpPacket:
    header = bytearray(HEADER_SIZE)

    def __init__(self):
        pass

    def encode(self, version, padding, extension, cc,
seqnum, marker, pt, ssrc, payload):
        """Encode the RTP packet with header fields and
payload."""

        timestamp = int(time())
        header = bytearray(HEADER_SIZE)

        # --------------
        # TO COMPLETE
        # --------------

        # Fill the header bytearray with RTP header fields
        header[0] = (header[0] | version << 6) & 0xC0
        header[0] = (header[0] | padding << 5)
        header[0] = (header[0] | extension << 4)
        header[0] = (header[0] | (cc & 0x0F))
        header[1] = (header[1] | marker << 7)
        header[1] = (header[1] | (pt & 0x7f))
        header[2] = (seqnum & 0xFF00) >> 8
        header[3] = (seqnum & 0xFF)
        for i in range(4):
```

```
34              header[7-i] = (timestamp >> 8 * i) & 0xFF
35              header[11-i] = (ssrc >> 8 * i) & 0xFF
36
37          self.header = header
38
39          # Get the payload from the argument
40          self.payload = payload
41
42      def decode(self, byteStream):
43          """Decode the RTP packet."""
44
45          self.header = bytearray(byteStream[:HEADER_SIZE])
46          self.payload = byteStream[HEADER_SIZE:]
47
48      def version(self):
49          """Return RTP version."""
50          return int(self.header[0] >> 6)
51
52      def seqNum(self):
53          """Return sequence (frame) number."""
54          seqNum = self.header[2] << 8 | self.header[3]
55          return int(seqNum)
56
57      def timestamp(self):
58          """Return timestamp."""
59          timestamp = self.header[4] << 24 | self.header[5] << 16 |
60          self.header[6] << 8 | self.header[7]
61          return int(timestamp)
62
63      def payloadType(self):
64          """Return payload type."""
65          pt = self.header[1] & 127
66          return int(pt)
67
68      def getPayload(self):
69          """Return payload."""
70          return self.payload
71
72      def getPacket(self):
73          """Return RTP packet."""
74          return self.header + self.payload
```

7.5 ServerWorker.py

```
from random import randint
import sys
import traceback
import threading
import socket

from VideoStream import VideoStream
from RtpPacket import RtpPacket


class ServerWorker:
    SETUP = 'SETUP'
    PLAY = 'PLAY'
    PAUSE = 'PAUSE'
    TEARDOWN = 'TEARDOWN'

    INIT = 0
    READY = 1
    PLAYING = 2
    state = INIT

    OK_200 = 0
    FILE_NOT_FOUND_404 = 1
    CON_ERR_500 = 2

    clientInfo = {}

    def __init__(self, clientInfo):
        self.clientInfo = clientInfo

    def run(self):
        threading.Thread(target=self.recvRtspRequest).start()

    def recvRtspRequest(self):
        """Receive RTSP request from the client."""
        connSocket = self.clientInfo['rtspSocket'][0]
        while True:
            data = connSocket.recv(256)
            if data:
```

```
                self.processRtspRequest(data)

    def processRtspRequest(self, data):
        """Process RTSP request sent from the client."""
        # Get the request type
        request = data.split('\n')
        line1 = request[0].split(' ')
        requestType = line1[0]

        # Get the media file name
        filename = line1[1]

        # Get the RTSP sequence number
        seq = request[1].split(' ')

        # Process SETUP request
        if requestType == self.SETUP:
            if self.state == self.INIT:
                # Update state
                print("\nPROCESSING SETUP")

                try:
                    self.clientInfo['videoStream'] = VideoStream(filename)
                    self.state = self.READY
                except IOError:
                    self.replyRtsp(self.FILE_NOT_FOUND_404, seq[1])

                # Generate a randomized RTSP session ID
                self.clientInfo['session'] = 123456

                # Send RTSP reply
                self.replyRtsp(self.OK_200, seq[1])

                # Get the RTP/UDP port from the last line
                self.clientInfo['rtpPort'] = request[2].split(' ')[3]

        # Process PLAY request
        elif requestType == self.PLAY:
            if self.state == self.READY:
                print("\nPROCESSING PLAY")
                self.state = self.PLAYING
```

```
81
82                  # Create a new socket for RTP/UDP
83                  self.clientInfo["rtpSocket"] = socket.socket(socket.
84                  AF_INET, socket.SOCK_DGRAM)
85
86                  self.replyRtsp(self.OK_200, seq[1])
87
88                  # Create a new thread and start sending RTP packets
89                  self.clientInfo['event'] = threading.Event()
90                  self.clientInfo['worker'] = threading.Thread(target=
91                  self.sendRtp)
92                  self.clientInfo['worker'].start()
93
94          # Process PAUSE request
95          elif requestType == self.PAUSE:
96              if self.state == self.PLAYING:
97                  print("\nPROCESSING PAUSE")
98                  self.state = self.READY
99
100                 self.clientInfo['event'].set()
101
102                 self.replyRtsp(self.OK_200, seq[1])
103
104         # Process TEARDOWN request
105         elif requestType == self.TEARDOWN:
106             print("\nPROCESSING TEARDOWN")
107
108             self.clientInfo['event'].set()
109
110             self.replyRtsp(self.OK_200, seq[1])
111
112             # Close the RTP socket
113             self.clientInfo['rtpSocket'].close()
114
115     def sendRtp(self):
116         """Send RTP packets over UDP."""
117         while True:
118             self.clientInfo['event'].wait(0.05)
119
120             # Stop sending if request is PAUSE or TEARDOWN
121             if self.clientInfo['event'].isSet():
```

```
122                 break
123
124             data = self.clientInfo['videoStream'].nextFrame()
125             if data:
126                 frameNumber = self.clientInfo['videoStream'].frameNbr()
127                 try:
128                     address = self.clientInfo['rtspSocket'][1][0]
129                     port = int(self.clientInfo['rtpPort'])
130                     self.clientInfo['rtpSocket'].sendto(self.makeRtp(data,
131 frameNumber), (address, port))
132                 except:
133                     print("Connection Error")
134
135     def makeRtp(self, payload, frameNbr):
136         """RTP-packetize the video data."""
137         version = 2
138         padding = 0
139         extension = 0
140         cc = 0
141         marker = 0
142         pt = 26  # MJPEG type
143         seqnum = frameNbr
144         ssrc = 0
145
146         rtpPacket = RtpPacket()
147
148         rtpPacket.encode(version, padding, extension, cc,seqnum, marker,
149         pt, ssrc, payload)
150
151         return rtpPacket.getPacket()
152
153     def replyRtsp(self, code, seq):
154         """Send RTSP reply to the client."""
155         if code == self.OK_200:
156             # print "200 OK"
157             reply = 'RTSP/1.0 200 OK\nCSeq: ' + seq + \ '\nSession: ' +
158 str(self.clientInfo['session'])
159             print(reply)
160             connSocket = self.clientInfo['rtspSocket'][0]
161             connSocket.send(reply)
162
```

```
163             # Error messages
164             elif code == self.FILE_NOT_FOUND_404:
165                 print("404 NOT FOUND")
166             elif code == self.CON_ERR_500:
167                 print("500 CONNECTION ERROR")
```

7.6 Result:

7.6.1 Whenever click on Setup button, we receive the following results:

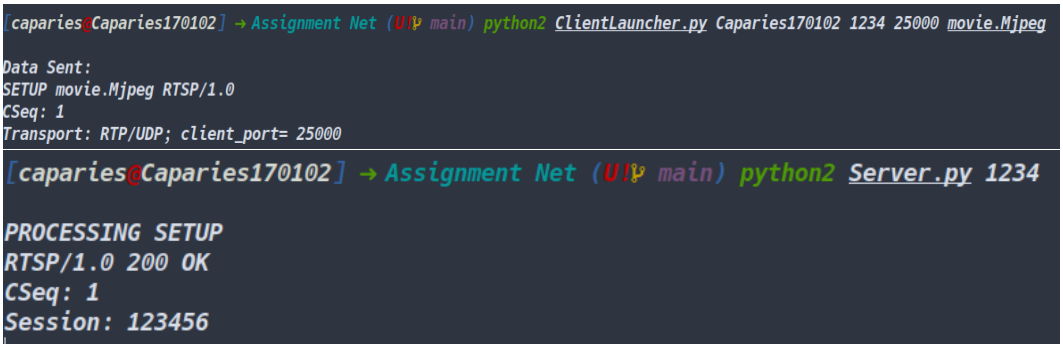


Figure 1: Click SETUP button

7.6.2 Whenever click on Play button, we receive the following results:

```
PROCESSING SETUP
RTSP/1.0 200 OK
CSeq: 1
Session: 123456

PROCESSING PLAY
RTSP/1.0 200 OK
CSeq: 2
Session: 123456
```

```
Data Sent:
SETUP movie.Mjpeg RTSP/1.0
CSeq: 1
Transport: RTP/UDP; client_port= 25000
LISTENING...

Data Sent:
PLAY movie.Mjpeg RTSP/1.0
CSeq: 2
Session: 123456
CURRENT SEQUENCE NUM: 1
LISTENING...
CURRENT SEQUENCE NUM: 2
LISTENING...
CURRENT SEQUENCE NUM: 3
LISTENING...
CURRENT SEQUENCE NUM: 4
LISTENING...
CURRENT SEQUENCE NUM: 5
LISTENING...
```

Setup | Play | Pause | Teardown

Figure 2: Click PLAY button

7.6.3 Whenever click on Pause button, we receive the following results:

Figure 3: Click PAUSE button

7.6.4 Whenever click on Teardown button, we receive the following results:

```
Data Sent:
PAUSE movie.Mjpeg RTSP/1.0
CSeq: 3
Session: 123456

Data Sent:
TEARDOWN movie.Mjpeg RTSP/1.0
CSeq: 4
Session: 123456
```

```
PROCESSING SETUP
RTSP/1.0 200 OK
CSeq: 1
Session: 123456

PROCESSING PLAY
RTSP/1.0 200 OK
CSeq: 2
Session: 123456

PROCESSING PAUSE
RTSP/1.0 200 OK
CSeq: 3
Session: 123456

PROCESSING TEARDOWN
RTSP/1.0 200 OK
CSeq: 4
Session: 123456
```

Figure 4: Click TEARDOWN button