Ho Chi Minh City University of Technology
Faculty of Computer Science and Engineeing

COMPUTER NETWORK

ASSIGNMENT 1

Mentor: Nguyễn Hồng Nam

Group: L09_ST3
Member:

1/ Phạm Nguyễn Anh Tài	1813897
2/ Phan Đình Sự	1813876
3/ Nguyễn Bá Tiến	1810578
4/ Châu Thanh Tân	1810501

HO CHI MINH CITY, November 2020

Contents

1. Requirement analysis:

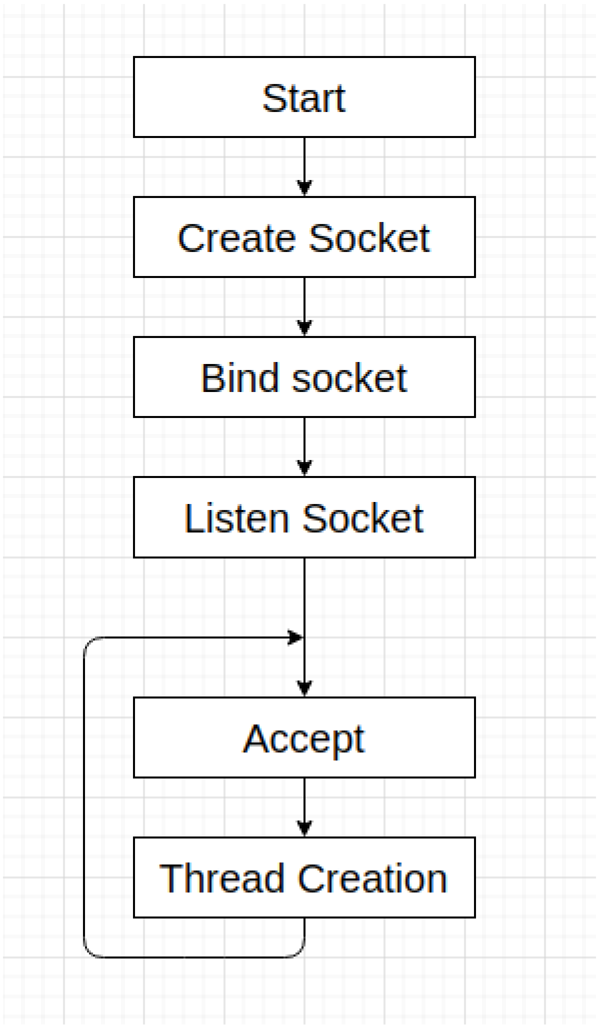
From the code that implements the RTSP protocol in the server, the RTP de-packetization in the client, and takes care of displaying the transmitted video.

Our tasks are to implement:

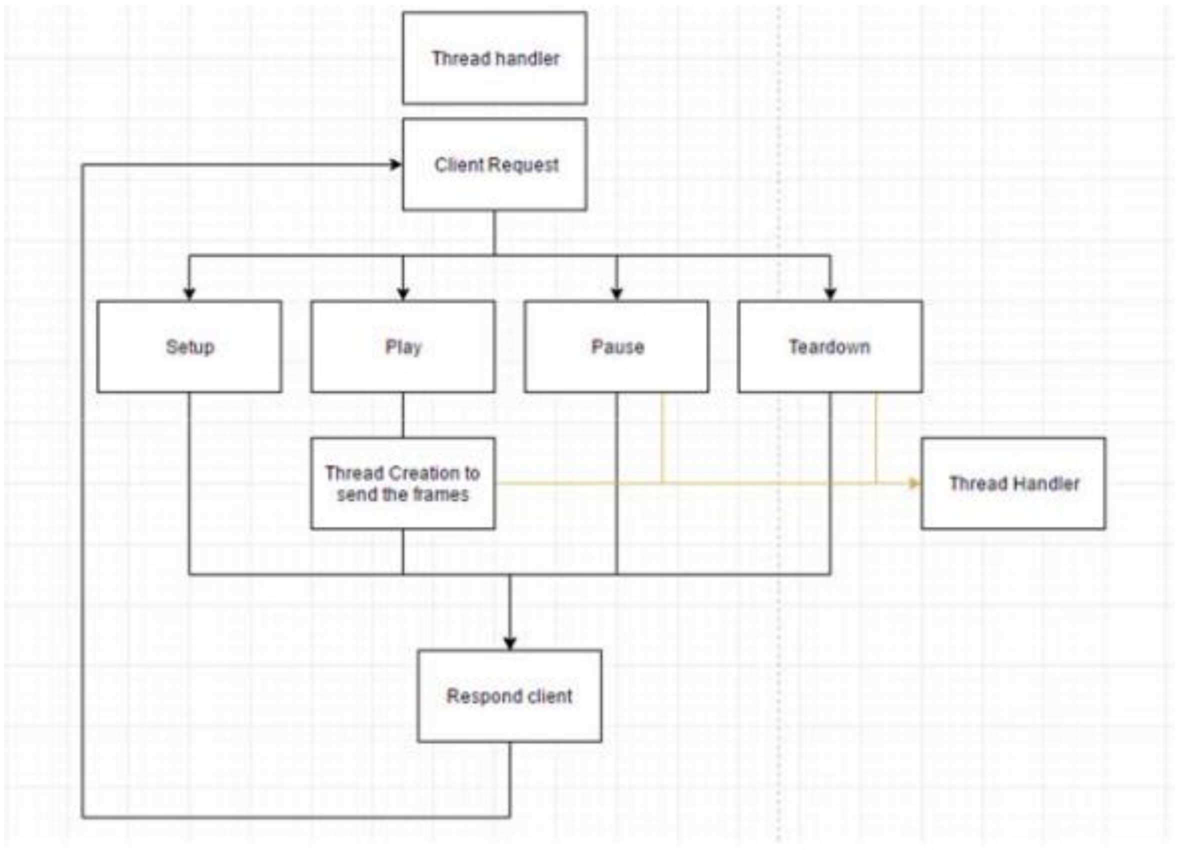
- The RTSP protocol in the client: we need to complete function sendRtspRequest , parseRtspReply , openRtpPort .
- The RTP packetization in the server: we need to complete function encode

*** FLOW CHART:**

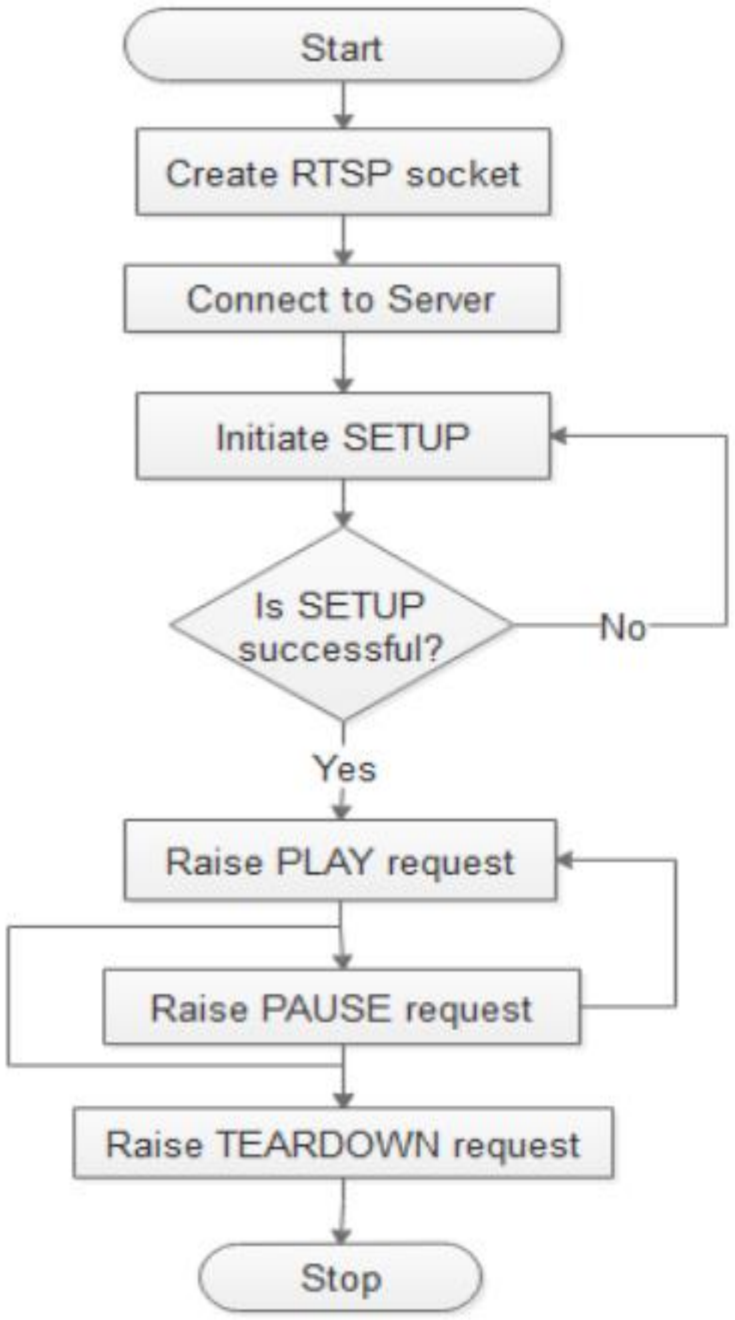
- Server:



- Client Thread Handler:



- Client:


Start
Create RTSP socket
Connect to Server
Initiate SETUP
Is SETUP successful?
No
Yes
Raise PLAY request
Raise PAUSE request
Raise TEARDOWN request
Stop

2. Function description:

- Function sendRtspRequest:

```
def sendRtspRequest(self, requestCode):
      """Send RTSP request to the server."""
      #-------------
      # TO COMPLETE
      #-------------

      # Setup request
      if requestCode == self.SETUP and self.state == self.INIT:
            threading.Thread(target=self.recvRtspReply).start()
            # Update RTSP sequence number.
            # ...
            self.rtspSeq += 1
            # Write the RTSP request to be sent.
            # request = ...
            request = "%s %s %s" % ("SETUP",self.fileName,self.RTSP_VER)
            request += "\nCSeq: %d" % self.rtspSeq
            request += "\nTransport: %s; client_port= %d" %
(self.TRANSPORT,self.rtpPort)
            # Keep track of the sent request.Bùi Phan Minh Anh
            # self.requestSent = ...
            self.requestSent = self.SETUP
      # Play request
      elif requestCode == self.PLAY and self.state == self.READY:
            # Update RTSP sequence number.
            # ...
            self.rtspSeq += 1
            # Write the RTSP request to be sent.
            # request = ...
            request = "%s %s %s" % ("PLAY",self.fileName,self.RTSP_VER)
            request+="\nCSeq: %d" % self.rtspSeq
            request+="\nSession: %d"%self.sessionId
            # Keep track of the sent request.
            # self.requestSent = ...
            self.requestSent = self.PLAY
      # Pause request
      elif requestCode == self.PAUSE and self.state == self.PLAYING:
            # Update RTSP sequence number.
            # ...
            self.rtspSeq += 1
            # Write the RTSP request to be sent.
            # request = ...
            request = "%s %s %s" % ("PAUSE",self.fileName,self.RTSP_VER)
            request+="\nCSeq: %d" % self.rtspSeq
            request+="\nSession: %d"%self.sessionId
            # Keep track of the sent request.
            # self.requestSent = ...
            self.requestSent = self.PAUSE
      # Teardown request
      elif requestCode == self.TEARDOWN and not self.state == self.INIT:
            # Update RTSP sequence number.
            # ...
            self.rtspSeq += 1
            # Write the RTSP request to be sent.
            # request = ...
            request = "%s %s %s" %('TEARDOWN',self.fileName,self.RTSP_VER)
            request+="\nCSeq: %d" % self.rtspSeq
            request+="\nSession: %d"%self.sessionId
            # Keep track of the sent request.
            # self.requestSent = ...
            self.requestSent = self.TEARDOWN
        # Describe request
        #-------------------------------
        #--------CODE-OF-MINE-----------
        #-------------------------------
        elif requestCode == self.DESCRIBE and self.state == self.READY:
            self.rtspSeq += 1
            request = "%s %s %s" %('DESCRIBE',self.fileName,self.RTSP_VER)
            request+="\nCSeq: %d" % self.rtspSeq
            request+="\nSession: %d"%self.sessionId
            self.requestSent = self.DESCRIBE
        #-----------END----------------
        else:
            return

        # Send the RTSP request using rtspSocket.
        # ...
        self.rtspSocket.send(request.encode())

        print('\nData sent:\n' + request)
```

Description: we modified it so that it does its proper job of sending the RTSP request with the correct content to server, we add request DESCRIBE to complete extend question 3, and update the value of rtspSeq and requestSent.

- Function parseRtspReply:

```
    def parseRtspReply(self, data):
        """Parse the RTSP reply from the server."""
        if self.requestSent == self.DESCRIBE:
            self.label2 = Label( text = data)
            self.label2.grid(row = 3, column=0, sticky=W)

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

```
                                          # The play thread exits. A new thread
is created on resume.
                                          self.playEvent.set()
                                    elif self.requestSent == self.TEARDOWN:
                                          # self.state = ...
                                          self.state = self.INIT
                                          # Flag the teardownAcked to close the
socket.
                                          self.teardownAcked = 1
```

Description: we modified it so that it does its proper job of updating RTSP state for each reply received from server. And also update the information for request DESCRIBE (represent by label2).

- Function openRtpPort:

```
      def openRtpPort(self):
            """Open RTP socket binded to a specified port."""
            #-------------
            # TO COMPLETE
            #-------------
            # Create a new datagram socket to receive RTP packets from the
server
            # self.rtpSocket = ...
            self.rtpSocket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
            # Set the timeout value of the socket to 0.5sec
            # ...
            self.rtpSocket.settimeout(0.5)
            try:
                  # Bind the socket to the address using the RTP port given by
the client user
                  # ...
                  self.state=self.READY
                  self.rtpSocket.bind(('',self.rtpPort))
            except:
                  tkMessageBox.showwarning('Unable to Bind', 'Unable to bind
PORT=%d' %self.rtpPort)
```

Description: we modified it so that it does its proper job of creating a new datagram socket to receive RTP packets from the server and set the timeout value of the socket to 0.5sec.

- Function encode:

```
      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 # 2 bits
            header[0] = (header[0] | padding << 5) # 1 bit
            header[0] = (header[0] | extension << 4) # 1 bit
            header[0] = (header[0] | (cc & 0x0F)) # 4 bits
            header[1] = (header[1] | marker << 7) # 1 bit
            header[1] = (header[1] | (pt & 0x7f)) # 7 bits
            header[2] = (seqnum & 0xFF00) >> 8 # 16 bits total, this is first 8
            header[3] = (seqnum & 0xFF) # second 8
            header[4] = (timestamp >> 24) # 32 bit timestamp
            header[5] = (timestamp >> 16) & 0xFF
            header[6] = (timestamp >> 8) & 0xFF
            header[7] = (timestamp & 0xFF)
            header[8] = (ssrc >> 24) # 32 bit ssrc
            header[9] = (ssrc >> 16) & 0xFF
            header[10] = (ssrc >> 8) & 0xFF
            header[11] = ssrc & 0xFF
            # header[0] = ...
            # ...

            # Get the payload from the argument
            # self.payload = ...
            self.header = header
            self.payload = payload
```

Description: we modified it so that it does its proper job of encoding the RTP packet with header fields and payload.

*** We completed two extended questions: 1 and 3. The following functions are those that we have modified to complete the question (we only added more code lines), and some of the above (as specified in the description of each function).**

*** Code added in file Client.py:**

- In function __init__:

```
 #statistics
            self.dataRate = 0 #Rate of video data received in bytes/s
            self.totalBytes = 0 #Total number of bytes received in a session
            self.startTime = 0 #Time in milliseconds when start is pressed
            self.toltalPlayTime = 0 #Time in milliseconds of video playing since
beginning
            self.fractionLost = 0 #Fraction of RTP data packets from sender lost
since the prev packet was sent
            self.sumLost = 0 #Number of packets lost
            self.expRtpNum = 0 #Expected Sequence number of RTP messages within
the session
            self.highSeqNum = 0 #Highest sequence number received in session
```

Description: initilize variables for calculating statistics.

- Code added in function createWidgets:

```
# Create Describe button
            self.describe = Button(self.master, width=20, padx=3, pady=3)
            self.describe["text"] = "Describe"
            self.describe["command"] =  self.popupInfo
            self.describe.grid(row=1, column=4, padx=2, pady=2)

# Create labels to display statistics
            self.label1 = Label(  height=4, text = "Total Bytes Received: 0" +
"\nPacket Lost Rate: 0" + "\nData Rate: 0 bytes/s")
            self.label1.grid(row = 2, column = 1)

# Create label to display Describe info
            self.label2 = Label( text = "")
            self.label2.grid(row = 3, column=0, sticky=W)
```

Description: create labels to display statistics, DESCRIBE button, DESCRIBE info.

- Code added in function playMovie:

```
# Take the time start to receive packet
                  self.startTime = time.time()
```

- Created function popupInfo:

```
def popupInfo(self):
            """Describe button handler"""
            if self.state == self.READY:
                  self.sendRtspRequest(self.DESCRIBE)
```

Description: DESCRIBE button handler.

- Code added in function listenRTP:

```
try:
                        data = self.rtpSocket.recv(20480)

                        #-------------------------------
                        #--------CODE-OF-MINE-----------
                        #-------------------------------
                        curtime = time.time()
                        self.toltalPlayTime += curtime - self.startTime
                        self.startTime = curtime
                        #-----------END----------------

                        if data:
                              rtpPacket = RtpPacket()
                              rtpPacket.decode(data)

                              currFrameNbr = rtpPacket.seqNum()
                              print("Current Seq Num: " + str(currFrameNbr))

                              if currFrameNbr > self.frameNbr: # Discard the
late packet
                                    self.frameNbr = currFrameNbr

      self.updateMovie(self.writeFrame(rtpPacket.getPayload()))

                              #-------------------------------
                              #--------CODE-OF-MINE-----------
                              #-------------------------------
                              # calculate statistics
                              if currFrameNbr > self.highSeqNum:
                                    self.highSeqNum = currFrameNbr
                              self.expRtpNum += 1
                              if self.expRtpNum < currFrameNbr:
                                    self.sumLost += 1
                                    #self.expRtpNum += 1
                              self.dataRate = 0 if (self.toltalPlayTime == 0)
else self.totalBytes / self.toltalPlayTime
                              self.fractionLost = float(self.sumLost /
self.highSeqNum)
                              self.totalBytes += len(data) - 12
                              update = "Total Bytes Received: "+
str(self.totalBytes) + "\nPacket Lost Rate: " + str(self.fractionLost) + "\nData
Rate: " + str(self.dataRate) + " bytes/s"
                              self.label1 = Label(text = update)
                              self.label1.grid(row = 2, column = 1)
                              #--------------END--------------
```

Description: calculating statistics and updating to label1.

*** Code added in file ServerWorker.py:**

- Code added in function processRtspRequest:

```
# Process DESCRIBE request
            elif requestType == self.DESCRIBE:
                  if self.state == self.READY:
                        print("processing DESCRIBE\n")

                        self.clientInfo['isDescribe'] = True
                        self.replyRtsp(self.OK_200, seq[1])
```

Description: processing DESCRIBE request.

- Code added in function replyRtsp:

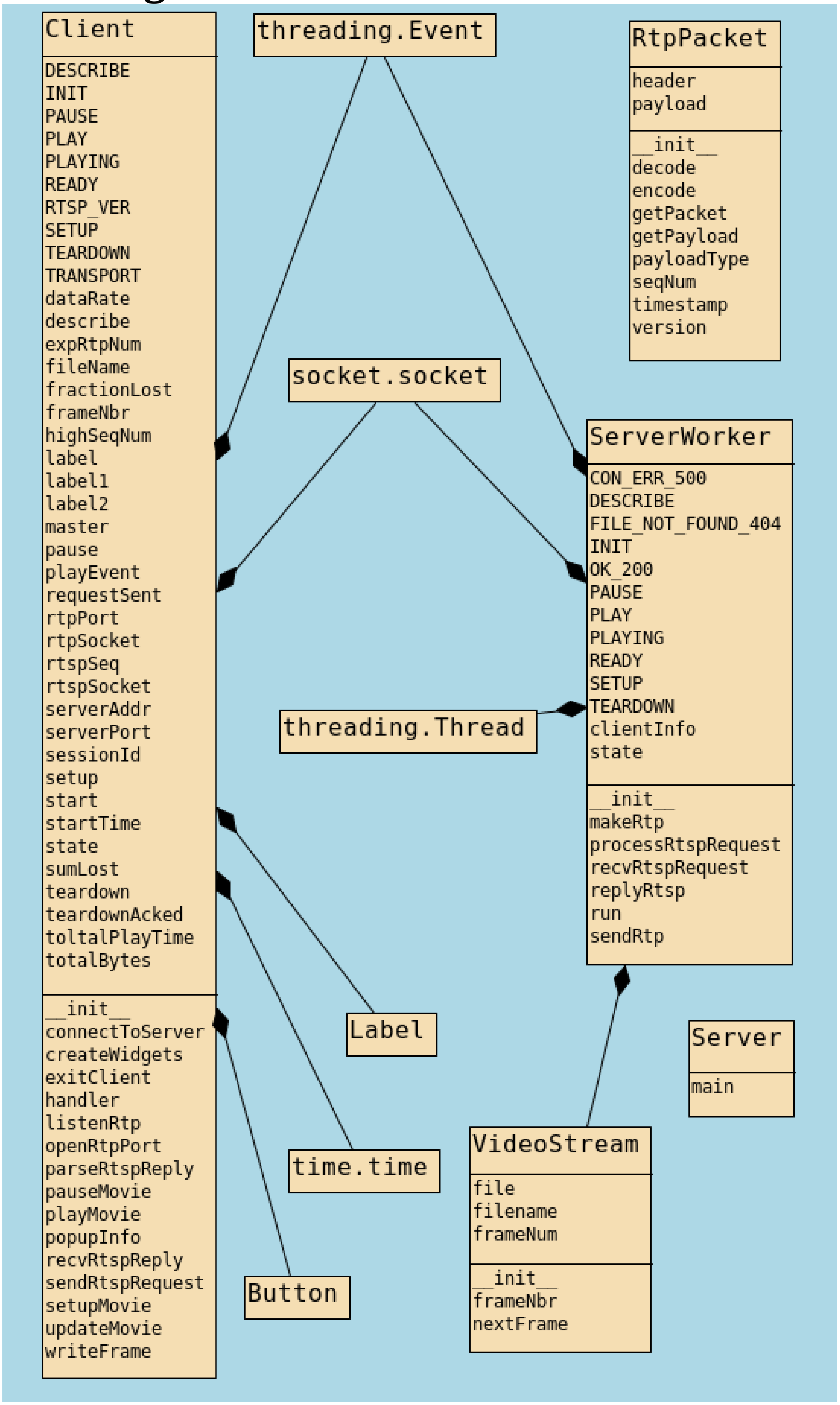
```
if self.clientInfo['isDescribe'] == True :
                  if code == self.OK_200:
                        #print("200 OK")
                        reply = 'RTSP/1.0 200 OK\nCSeq: ' + str(seq)
                        reply += '\nv=0'
                        reply += '\nm=video RPT/AVP 26'
                        reply += '\na=control:streamid=' +
str(self.clientInfo['session'])
                        reply += '\na=mimetype:string;\"video/MJPEG\"'
                        body_length = len(reply)
                        reply += '\nContent-Base: ' +
self.clientInfo['fileName']
                        reply += '\nContent-Type: application/sdp'
                        reply += '\nContent-Length: ' + str(body_length)

                        connSocket = self.clientInfo['rtspSocket'][0]
                        connSocket.send(reply.encode())

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

Description: send RTSP reply to the client, different with DESCRIBE request.

3. Class diagram:

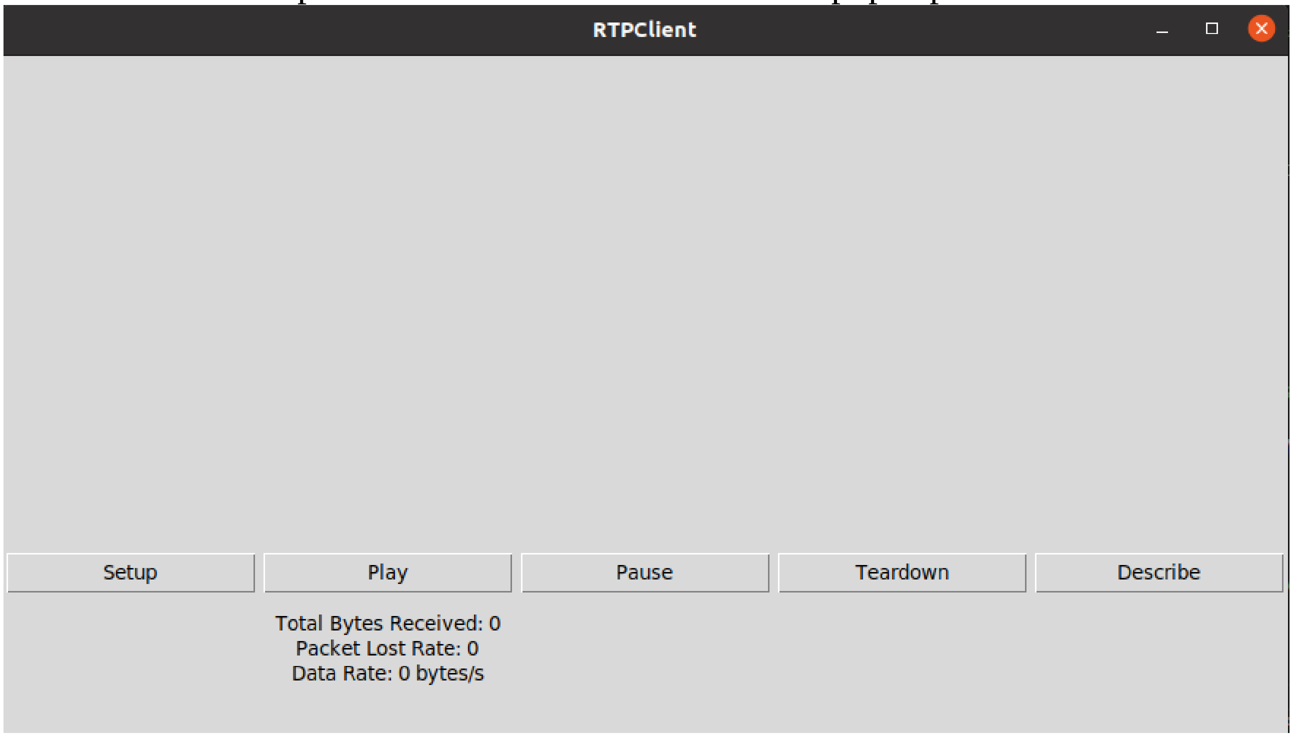


4. A Summative Evaluation of Results Achieved:

- We understand the RTP and RTSP protocol functioning in providing video streaming service to the end user.
- We have implemented a streaming video server and client that communicate using the Real-Time Streaming Protocol (RTSP) and send data using the Real-time Transfer Protocol (RTP).
- We have done two of three extended requirement: 1 and 3.
- We have not done the second extended requirement.

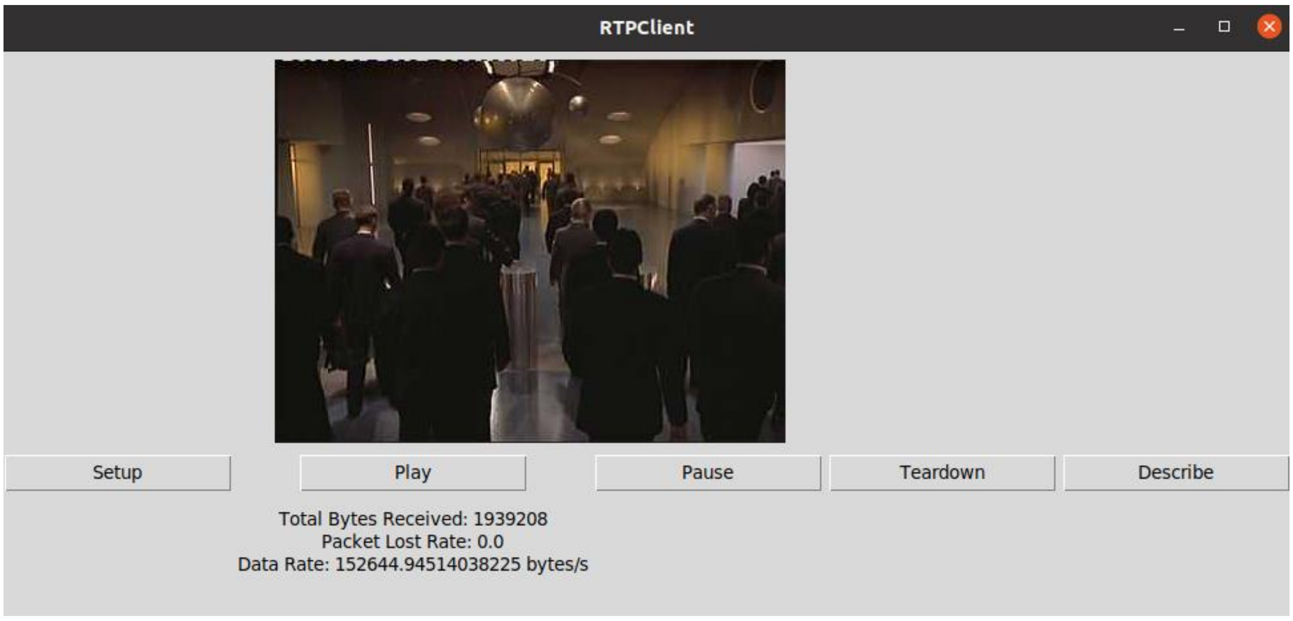
5. User manual:

- First, start the server with the command:#python Server.py server_port where server_port is the port your server listens to for incoming RTSP connections.
- Then, start the client with the command python ClientLauncher.py server_host server_port RTP_port video_file where server_host is the name of the machine where the server is running, server_port is the port where the server is listening on, RTP_port is the port where the RTP packets are received, and video_file is the name of the video file you want to request (we have provided one example file movie.Mjpeg). The file format is described in Appendix section.
- The client opens a connection to the server and pops up a window like this:



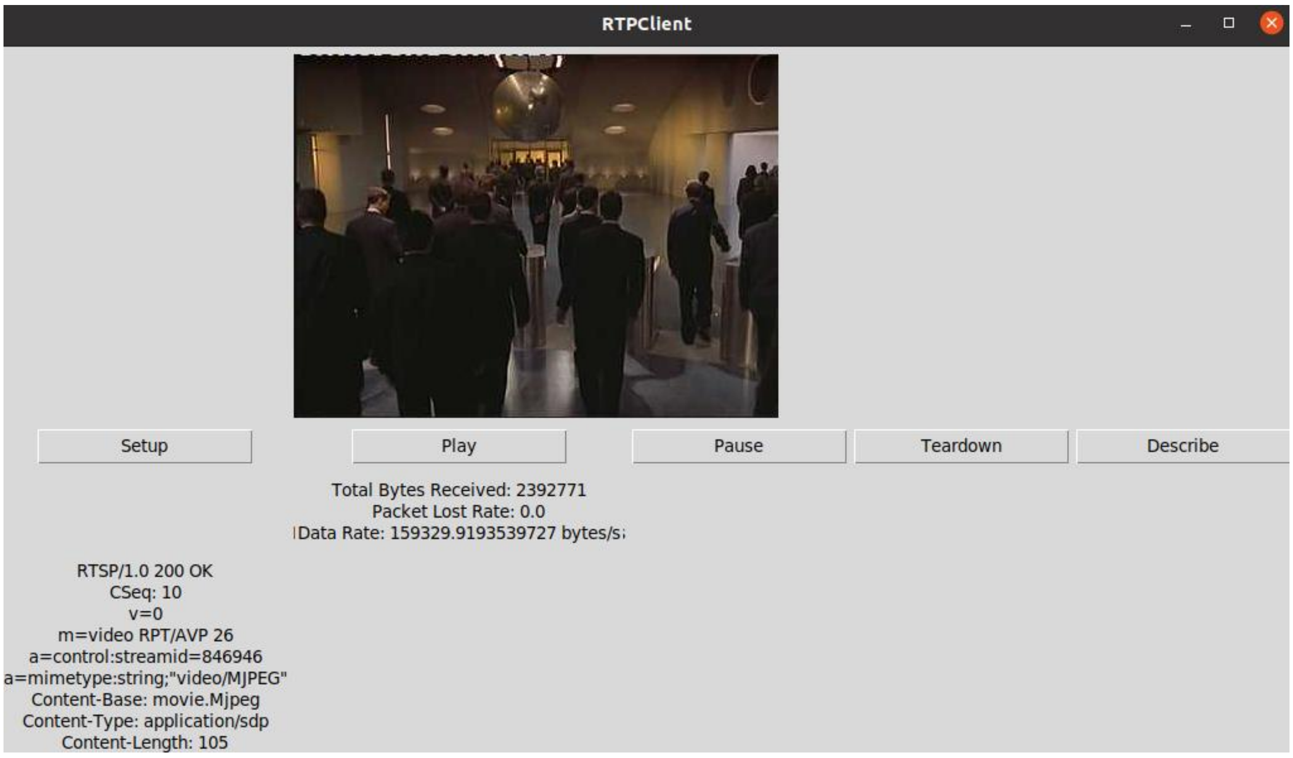
Total Bytes Received: 0
Packet Lost Rate: 0
Data Rate: 0 bytes/s

- You have to click button Setup to set the environment for streaming video.
- After that, you click button Play to run the video, you will see the statistics in the lower left of the window changes at the same time with the video playing, it shows you Total Byte Received, Packet Lost Rate and Data Rate like this:



```
Total Bytes Received: 1939208
Packet Lost Rate: 0.0
Data Rate: 152644.94514038225 bytes/s
```

- You can click the button Pause to stop the video at any time. After that, you can click button Play to continue the video, or click button Describe to see the what kinds of streams are in the session and what encodings are used in the low of window like this:



```
Total Bytes Received: 2392771
Packet Lost Rate: 0.0
Data Rate: 159329.9193539727 bytes/s

RTSP/1.0 200 OK
CSeq: 10
v=0
m=video RPT/AVP 26
a=control:streamid=846946
a=mimetype:string;"video/MJPEG"
Content-Base: movie.Mjpeg
Content-Type: application/sdp
Content-Length: 105
```

- At any time you can click button Teardown to quit the program.

---END---