VIETNAM NATIONAL UNIVERSITY, HO CHI MINH CITY
UNIVERSITY OF TECHNOLOGY
FACULTY OF COMPUTER SCIENCE AND ENGINEERING

COMPUTER NETWORKS (CO3094)

Assignment

Real-Time Streaming Protocol (RTSP) and Real-time Transfer Protocol (RTP)

Advisor: Nguyễn Mạnh Thìn
Students: Trần Việt Hoàng - 1852145
Hoàng Nhật Quang - 1852691

HO CHI MINH CITY, NOVEMBER 2021

Contents

1 Member list & Workload

No.	Fullname	Student ID	Problems	Percentage of work
1	Hoàng Nhật Quang	1852691	2.1, 3, Code RtpPacket.py	50%
2	Trần Việt Hoàng	1852145	2.2, 4, Code Client.py	50%

2 Requirements analysis

2.1 Functional requirements

2.1.1 System-side

- The system can stream video.
- The system can communicate with user via RTSP/RTP protocol.

2.1.2 User-side

- User can connect to the server via the terminal.
- User can play video from server, pause and teardown.
- User can view the basic parameters of the video such as the time of the video.

2.2 Non-functional requirements

- The extension of the video must be .Mjpeg
- Response time from server less than or equal 0.5s

3 Description of the functions' tasks

Class Name	Function	Parameter	Description
ServerWorker	_init_(self, clientInfo)	self, clientInfo	Constructor
	run(self)	self	Begining the server
	processRtspRequest(self, data)	self, data	Process the Rtsp request
	sendRtp(self)	self	Send RTP packets over UDP
	makeRtp(self, payload, frameNbr)	self, payload, frameNbr	Make RPT for the video data
	replyRtsp(self, code, seq)	self, code, seq	Send RTSP to the Client
Sever	main(self)	self	Main function
VideoStream	_init_(self, filename)	self, filename	Constructor
	nextFrame(self)	self	Get next frame
	frameNbr(self)	self	Get frame number
Client	_init_(self, master, serveraddr, serverport, rtpport, filename)	self, master, serveraddr, serverport, rtpport, filename	Constructor
	createWidgets(self)	self	Build GUI
	setupMovie(self)	self	Setup button handler
	exitClient(self)	self	Teardown button handler
	pauseMovie(self)	self	Pause button
	playMovie(self)	self	Play button
	take_time(self, bufftime)	self, bufftime	Time format minutes : seconds
	listenRtp(self)	self	Listen for RTP packets and analysis
	writeFrame(self, data)	self, data	Write the received frame to a temp image file
	updateMovie(self, imageFile)	self, imageFile	Update the image file as video frame in the GUI
	connectToServer(self)	self	Connect to the Server. Start a new RTSP/TCP session
	sendRtspRequest(self, requestCode)	self, requestCode	Send RTSP request to the server
	recvRtspReply(self)	self	Receive RTSP reply from the server.
	parseRtspReply(self, data)	self, data	Parse the RTSP reply from the server
	openRtpPort(self)	self	Open RTP socket bined to a specified port
	handler(self)	self	Handler on explicity closing the GUI window
RtpPacket	_init_(selft)	self	constructor
	encode(self, version, padding, extension, cc, seqnum, marker, pt, ssrc, payload)	self, version, padding, extension, cc, seqnum, marker, pt, ssrc, payload	Encode the RTP packet with header fields and payload
	decode(self, byteStream)	self	Decode the RTP packet
	version(self)	self	Return RTP version
	seqNum(self)	self	Return sequence (frame) number
	timestamp(self)	self	Return timestamp
	payloadType(self)	self	Return payload type
	getPayload(self)	self	Return payload
	getPacket(self)	self	Return RTP packet

Figure: Table contain description of each functions corresponding to the classes.

4 Class diagram

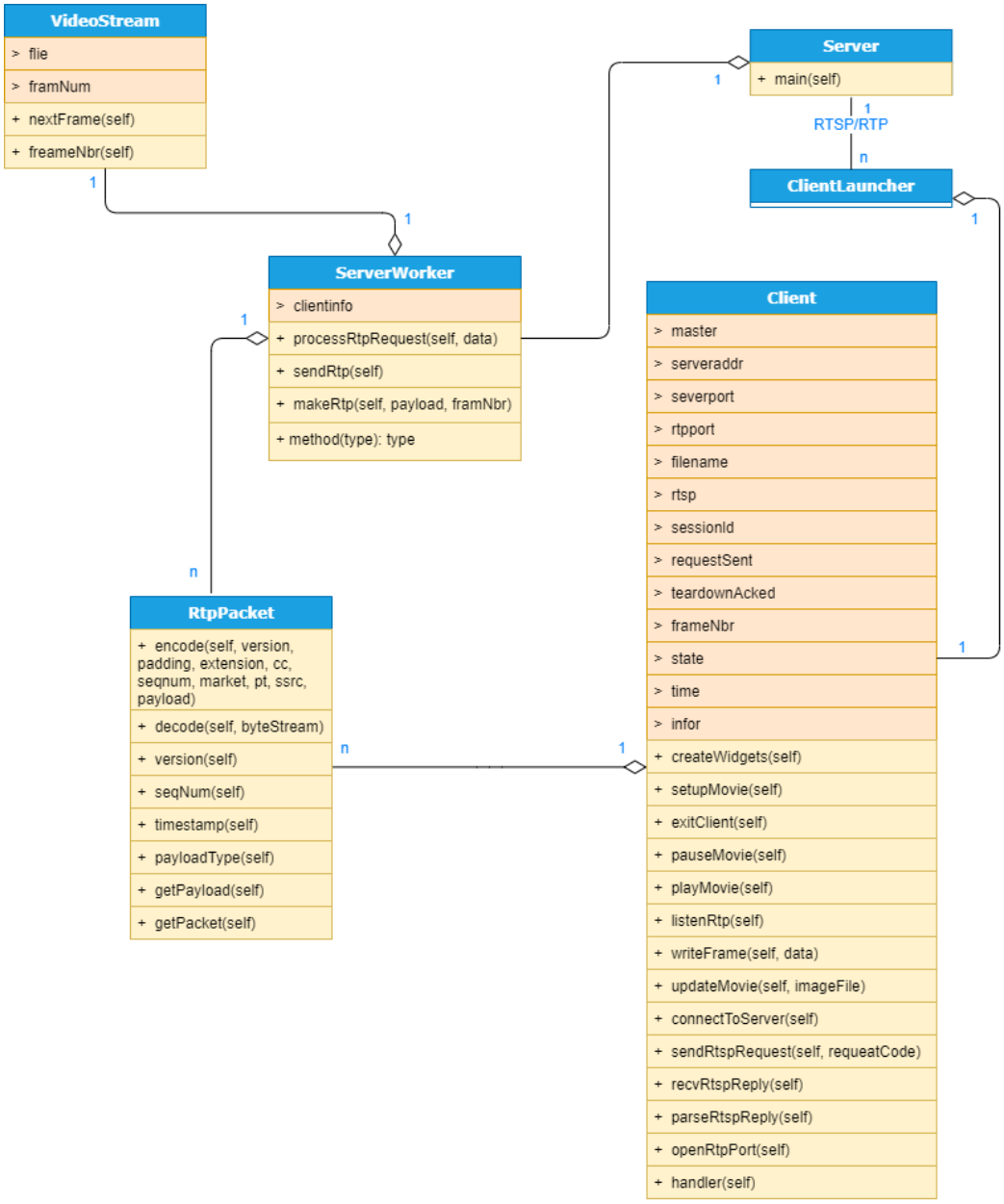


Figure 2: Class diagram

5 Implementation

Implementation of Client.py:

```
def setupMovie(self):
                """Setup button handler."""
                if self.state == self.INIT:
                        self.sendRtspRequest(self.SETUP)

def exitClient(self):
        """Teardown button handler."""
        self.sendRtspRequest(self.TEARDOWN)

        # Delete the cache image from video
        if(self.sessionId != 0):
                os.remove(CACHE_FILE_NAME + str(self.sessionId) + CACHE_FILE_EXT)

        # Stop mainloop() and quit the program
        self.master.destroy()

def pauseMovie(self):
        """Pause button handler."""
        if self.state == self.PLAYING:
                self.sendRtspRequest(self.PAUSE)

def playMovie(self):
        """Play button handler."""
        if self.state == self.READY:
                # Create a new thread to listen for RTP packets
                threading.Thread(target=self.listenRtp).start()
                self.playEvent = threading.Event()
                self.playEvent.clear()
                self.sendRtspRequest(self.PLAY)

def listenRtp(self):
        """Listen for RTP packets."""
        while True:
                try:
                        print("LISTENING...")
                        data = self.rtpSocket.recv(20480)
                        if data:
                                rtpPacket = RtpPacket()
                                rtpPacket.decode(data)

                                currFrameNbr = rtpPacket.seqNum()
                                print ("CURRENT FRAME NUM: " + str(currFrameNbr))

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

```
                                        self.updateMovie(self.writeFrame(rtpPacket.
                                            ↪ getPayload()))
                except:
                        # Stop listening upon requesting PAUSE or TEARDOWN
                        if self.playEvent.isSet():
                                break

                        # Upon receiving ACK for TEARDOWN request,
                        # close the RTP socket
                        if self.teardownAcked == 1:
                                self.rtpSocket.shutdown(socket.SHUT_RDWR)
                                self.rtpSocket.close()
                                break

def writeFrame(self, data):
        """Write the received frame to a temp image file. Return the image file
            ↪ ."""
        cachename = CACHE_FILE_NAME + str(self.sessionId) + CACHE_FILE_EXT
        file = open(cachename, "wb")
        file.write(data)
        file.close()
        return cachename

def updateMovie(self, imageFile):
        """Update the image file as video frame in the GUI."""
        photo = ImageTk.PhotoImage(Image.open(imageFile))
        self.label.configure(image = photo, height=288)
        self.label.image = photo

def connectToServer(self):
        """Connect to the Server. Start a new RTSP/TCP session."""
        self.rtspSocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        try:
                self.rtspSocket.connect((self.serverAddr, self.serverPort))
        except:
                messagebox.showwarning('Connection Failed', 'Connection to \'%s\'
                    ↪  failed.' % self.serverAddr)

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 = "SETUP %s %s\n" % (self.fileName, self.RTSP_VER)
            request += "CSeq: %d\n" % self.rtspSeq
            request += "Transport: %s; client_port= %d\n" % (self.TRANSPORT,
                ↪ self.rtpPort)

            # Keep track of the sent request.
            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 = "PLAY %s %s\n" % (self.fileName,self.RTSP_VER)
            request += "CSeq: %d\n" % self.rtspSeq
            request += "Session: %d\n" % self.sessionId


            # Keep track of the sent request.
            self.requestSent = self.PLAY


#### Pause request ####
    elif requestCode == self.PAUSE and self.state == self.PLAYING:

            # Update RTSP sequence number.
            self.rtspSeq += 1

            request = "PAUSE %s %s\n" % (self.fileName,self.RTSP_VER)
            request += "CSeq: %d\n" % self.rtspSeq
            request += "Session: %d\n" % self.sessionId

            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 = "TEARDOWN %s %s\n" % (self.fileName, self.RTSP_VER)
            request += "CSeq: %d\n" % self.rtspSeq
            request += "Session: %d\n" % self.sessionId
```

```
                self.requestSent = self.TEARDOWN

        else:
                return

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

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


def recvRtspReply(self):
        """Receive RTSP reply from the server."""
        while True:
                reply = self.rtspSocket.recv(1024)

                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.decode().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

                                        # The play thread exits. A new thread is
                                            ↪ created on resume.
                                        self.playEvent.set()
                                elif self.requestSent == self.TEARDOWN:
                                        self.state = self.INIT

                                        # Flag the teardownAcked to close the
                                            ↪ socket.
                                        self.teardownAcked = 1

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 = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

        # Set the timeout value of the socket to 0.5sec
        self.rtpSocket.Csettimeout(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:
                messagebox.showwarning('Unable to Bind', 'Unable to bind PORT = %
                    ↪ d' % self.rtpPort)


def handler(self):
        """Handler on explicitly closing the GUI window."""
        from tkinter.messagebox import WARNING
        self.pauseMovie()

        if messagebox.askokcancel(title = "Confirmation", message = "Do you want
            ↪  to quit?", icon = WARNING):
                self.exitClient()
```

Implementation of RtpPacket.py:

```
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

        self.header = header

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

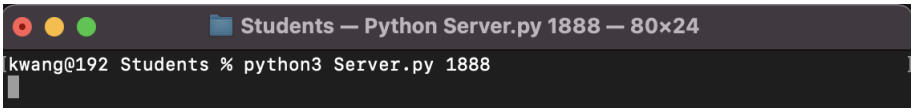
6 Result

- Completing the RTSP protocol at the client
- Complete RTP protocol at server

7 User manual

- Step 01: We must run the server first: run the terminal in the directory containing the file Server.py

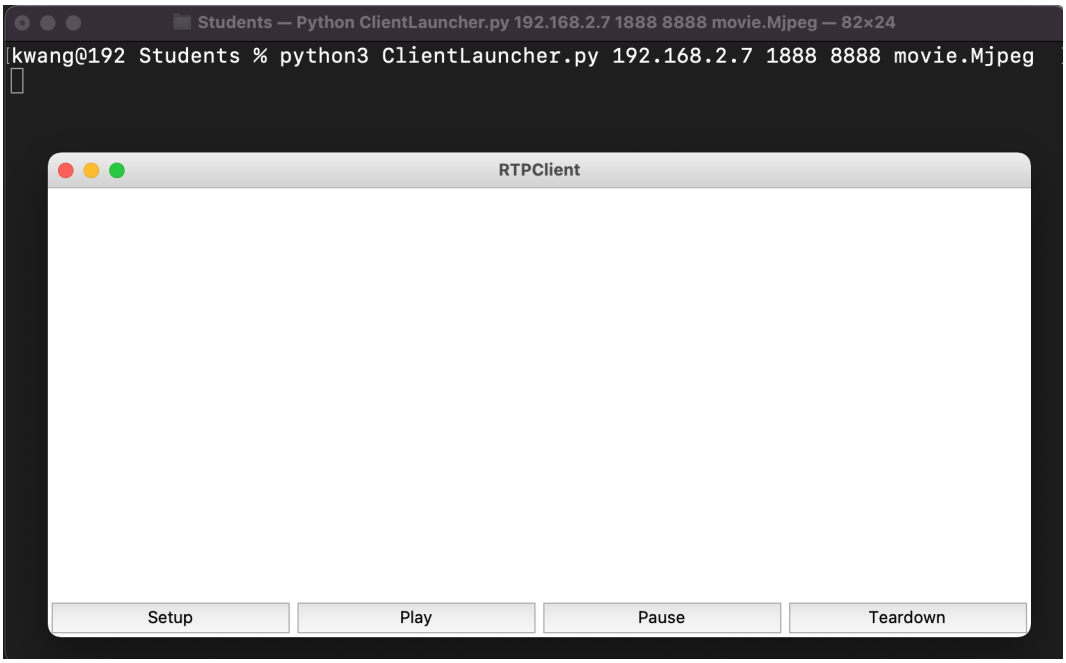
  Run command in form: python Server.py 1888 (Where port_server should greater than 1024).



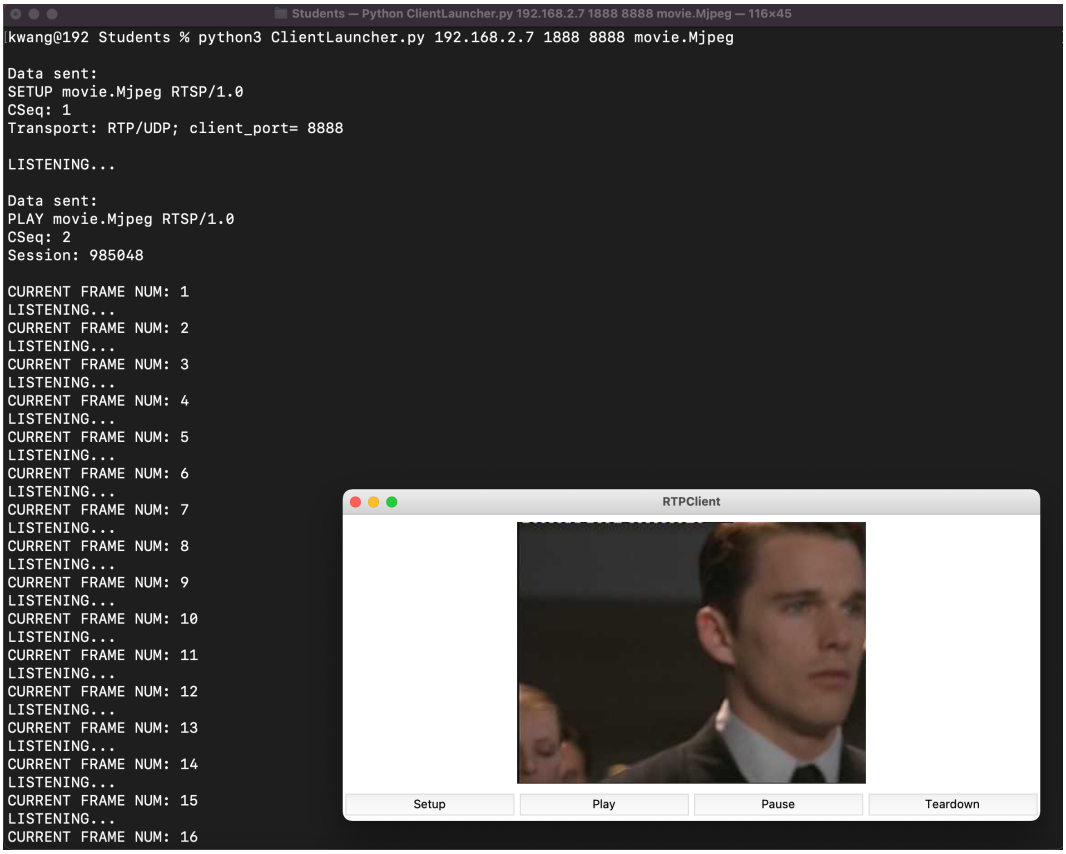
- Step 02: We open a new terminal in the folder containing the ClientLauncher.py file to connect to the Server we opened in step 01.

  Run command in form: python ClientLauncher.py 192.168.2.7 1888 8888 movie.Mjpeg

  - «host_name»: is the IP of the Server on the computer you are using, here is "192.168.2.7"
  - «port_server»: is the port initialized in step 1, here is 1200
  - «port_RTP»: for example, we choose 5008
  - «name_video»: the name of the video, here is movie.Mjpeg



- Step 03: Click **Setup** to create RTP stream and press **Play** to watch video, **Pause** to stop and **Teardown** to finish.


Students — Python ClientLauncher.py 192.168.2.7 1888 8888 movie.Mjpeg — 116×45
kwang@192 Students % python3 ClientLauncher.py 192.168.2.7 1888 8888 movie.Mjpeg
Data sent:
SETUP movie.Mjpeg RTSP/1.0
CSeq: 1
Transport: RTP/UDP; client_port= 8888
LISTENING...
Data sent:
PLAY movie.Mjpeg RTSP/1.0
CSeq: 2
Session: 985048
CURRENT FRAME NUM: 1
LISTENING...
CURRENT FRAME NUM: 2
LISTENING...
CURRENT FRAME NUM: 3
LISTENING...
CURRENT FRAME NUM: 4
LISTENING...
CURRENT FRAME NUM: 5
LISTENING...
CURRENT FRAME NUM: 6
LISTENING...
CURRENT FRAME NUM: 7
LISTENING...
CURRENT FRAME NUM: 8
LISTENING...
CURRENT FRAME NUM: 9
LISTENING...
CURRENT FRAME NUM: 10
LISTENING...
CURRENT FRAME NUM: 11
LISTENING...
CURRENT FRAME NUM: 12
LISTENING...
CURRENT FRAME NUM: 13
LISTENING...
CURRENT FRAME NUM: 14
LISTENING...
CURRENT FRAME NUM: 15
LISTENING...
CURRENT FRAME NUM: 16
RTPClient
Setup
Play
Pause
Teardown