TRƯỜNG ĐẠI HỌC BÁCH KHOA TP.HCM
MẠNG MÁY TÍNH

BÀI TẬP LỚN 1

REAL TIME STREAMING PROTOCOL

GVHD: BÙI XUÂN GIANG

Thành viên:

1. **Nguyễn Văn Hoàn 1711376**
2. **Ngô Đức Kiên 1812703**
3. **Trần Thanh Tuấn 2020114**

Contents

I.

1. Phân tích yêu cầu:

- Hiện thức giao thức RTSP trong client
  RTSP là một giao thức cung cấp khung để truyền dữ liệu phương tiện theo thời gian thực ở cấp ứng dụng. Nó truyền dữ liệu thời gian thực từ đa phương tiện sang thiết bị đầu cuối bằng cách giao tiếp trực tiếp với máy chủ truyền dữ liệu.
  Giao thức tập trung vào việc kết nối và kiểm soát các phiên phân phối dữ liệu trên các dòng đồng bộ hóa thời gian cho phương tiện liên tục như video và âm thanh. Tóm lại, giao thức truyền phát thời gian thực hoạt động như một điều khiển từ xa mạng cho các tệp phương tiện thời gian thực và máy chủ đa phương tiện.

- Tạo nhịp độ RTP trong máy chủ
  RTP (Giao thức Vận chuyển Thời gian Thực) đặc tả một tiêu chuẩn định dạng gói tin dùng để truyền âm thanh và hình ảnh qua internet

2. Chi tiết các chức năng của ứng dụng

1. Khởi tạo Streaming Server cho phép Client kết nối đến và giao tiếp thông qua giao thức RTSP/TCP

- Khởi tạo Server và socket RTSP/TCP, lắng nghe và cho phép tạo kết nối với nhiều máy khách kết nối đến cùng lúc.

2. Khởi tạo Client và giao diện người dùng để gửi các yêu cầu RTSP.

- Khởi tạo Client, socket để tạo kết nối với Server và khởi tạo giao diện. Sau khi kết nối thành công, người dùng thao tác gửi các yêu cầu RTSP đến máy chủ khi thao tác nhấn các nút nhấn trên giao diện.

3. Sử dụng giao thức RTP để xử lý dữ liệu video và phản hồi cho Client.

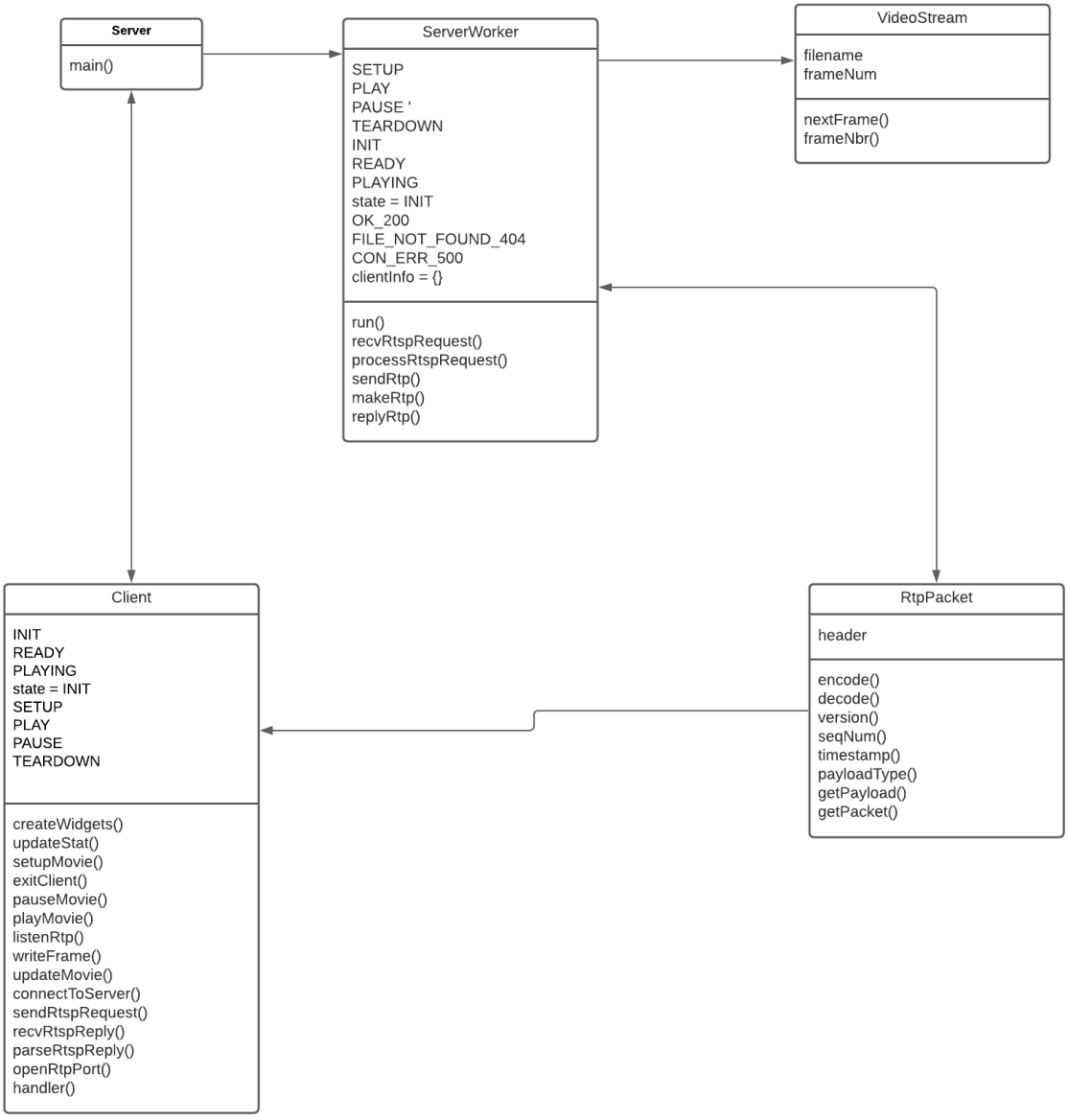
- Server nhận các yêu cầu RTSP của Client, xử lý dữ liệu video và đóng gói dưới dạng gói tin RTP, sau đó phản hồi cho Client.

4. Client nhận các gói tin RTP, xử lý và hiển thị video trên giao diện

- Client lắng nghe và nhận các gói tin RTP phản hồi từ Server, phân giải các gói tin và hiển thị video trên giao diện người dùng.

3. Thiết kế chi tiết ứng dụng

a. Class Diagram



b. Phân tích luồng của ứng dụng

- Chạy Server.py trên Command Promt tại thư mục chứa ứng dụng để khởi động server. Câu lệnh: python Server.py server_port. Server sẽ tiến hành khởi tạo socket

RTSP/TCP, gán socket với địa chỉ IP và port mà chúng ta cung cấp, sau đó bắt đầu lắng nghe kết nối từ 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()
```

- Chạy ClientLaucher.py trên Command Promt tại thư mục chứa ứng dụng để khởi động Client. Câu lệnh: python ClientLauncher.py server_host server_port PRT_port video_file. ClientLaucher.py sẽ nhận vào các thông số người dùng cung cấp, gọi đến Client. Tại Client, tiến hành khởi tạo các trạng thái, thông số, tạo socket kết nối tới Server thông qua địa chỉ và port được cung cấp.

```
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()
```

```python
class Client:
    INIT = 0
    READY = 1
    PLAYING = 2
    state = INIT

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

    # 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)
        self.rtpPort = int(rtpport)
        self.fileName = filename
        self.rtspSeq = 0
        self.sessionId = 0
        self.requestSent = -1
        self.teardownAcked = 0
        self.connectToServer()
        self.frameNbr = 0
```

```python
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:
        tkinter.messagebox.showwarning('Connection Failed', 'Connection to \'%s\' failed.' %self.serverAddr)
```

Server lắng nghe và chấp nhận kết nối khi nhận được yêu cầu kết nối từ Client. Tại ServerWorker, khởi tạo các trạng thái, thông số Server và sẵn sàng nhận các yêu cầu RTSP từ Client.

```
# Receive client info (address,port) through RTSP/TCP session
while True:
    clientInfo = {}
    clientInfo['rtspSocket'] = rtspSocket.accept()        unknown,
    ServerWorker(clientInfo).run()
```

```
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        unknown, 3 days ago

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

- Đồng thời, Client tiến hành khởi tạo giao diện người dùng.

```
def createWidgets(self):
    """Build GUI."""
    # Create Setup button
    self.setup = Button(self.master, width=20, padx=3, pady=3)
    self.setup["text"] = "Setup"
    self.setup["command"] = self.setupMovie
    self.setup.grid(row=1, column=0, padx=2, pady=2)

    # Create Play button
    self.start = Button(self.master, width=20, padx=3, pady=3)
    self.start["text"] = "Play"
    self.start["command"] = self.playMovie
    self.start.grid(row=1, column=1, padx=2, pady=2)
```

- Sau khi đã khởi tạo Server và Client, kết nối với nhau qua giao thức RTSP/TCP. Client sẽ gửi đến Server các lệnh như “SETUP”, “PLAY”, “PAUSE”, “TEARDOWN” thông qua giao thức RTSP, các lệnh này sẽ cho phía Server biết hành động tiếp theo mà nó cần hoàn thành.
- Những gì sẽ được phản hồi từ Server đến Client thông qua Giao thức RTSP là các tham số như: “OK_200”, “FILE_NOT_FOUND_404”, “CON_ERR_500” để xác nhận với Client nếu Server đã nhận được yêu cầu của nó một cách chính xác.
- Sau khi Client nhận được câu trả lời của Server, nó sẽ thay đổi trạng thái tương ứng thành: “READY”, “PLAYING”.

```
# Setup request
if requestCode == self.SETUP and self.state == self.INIT:
    threading.Thread(target=self.recvRtspReply).start()
    # Update RTSP sequence number.
    self.rtspSeq = 1      unknown, 21 hours ago • ComputerNetworkAssignm

    # Write the RTSP request to be sent.
    request = "SETUP " + str(self.fileName) + " RTSP/1.0\n"
    request += "CSeq: " + str(self.rtspSeq) + "\n"
    request += "Transport: RTP/UDP; client_port= " + str(self.rtpPort)

    # Keep track of the sent request.
    self.requestSent = self.SETUP
```

- Nếu yêu cầu "SETUP" được gửi đến Server, gói tin RTSP "SETUP" sẽ bao gồm:

  + Yêu cầu "SETUP"

  + Tên Video muốn phát

  + Protocol type: RTSP/1.0

  + Số RTSP Packet Sequence bắt đầu từ 1

  + Transport Protocol: RTP/UDP

  + Cổng RTP để truyền dòng video

  Thông tin về requestSent cũng được ghi lại.

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

        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'] = randint(100000, 999999)

        # 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]
```

- Khi Server nhận được yêu cầu "SETUP", nó sẽ gán cho Client ngẫu nhiên một Specific Session Number. Nếu có sai sót với yêu cầu hoặc trạng thái Server, nó sẽ phản hồi gói tin ERROR về cho Client. Nếu yêu cầu là chính xác, Server sẽ mở tệp video được chỉ định trong yêu cầu "SETUP" và khởi tạo số khung hình video của nó thành 0.
- Nếu yêu cầu được xử lý chính xác, Server sẽ trả lời lại OK_200 cho Client và đặt STATE của nó thành "READY".

```
class VideoStream:        unknown, a day ago • (
    def __init__(self, filename):
        self.filename = filename
        try:
            self.file = open(filename, 'rb')
        except:
            raise IOError
        self.frameNum = 0
```

- Phía Client sẽ thực hiện vòng lặp để nhận RTSP Reply của Server:

```
def recvRtspReply(self):
    """Receive RTSP reply from the server."""
    while True:
        reply = self.rtspSocket.recv(1024).decode("utf-8")

        if reply:
            print('\n[*]Received reply:\n' + reply + '\n')
            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
```

- Sau đó tiến hành phân tích cú pháp RTSP phản hồi từ máy chủ, lấy được Session Number do Server khởi tạo trước đó cho Client.

```
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 nu
    if seqNum == self.rtspSeq:
        session = int(lines[2].split(' ')[1])
        # New RTSP session ID
        if self.sessionId == 0:
            self.sessionId = session
```

- Và nếu gói tin phản hồi cho lệnh "SETUP", Client sẽ đặt trạng thái (STATE) là "READY". Sau đó, mở một RtpPort để nhận luồng video đến.

```
if int(lines[0].split(' ')[1]) == 200:
    if self.requestSent == self.SETUP:
        # Update RTSP state.
        self.state = self.READY

        # Open RTP port.
        self.openRtpPort()
```

- Khởi tạo RTP socket để nhận gói tin RTP từ server, thiết lập timeout cho socket này là 0.5 giây. Ràng buộc cho socket địa chỉ sử dụng RTP port được cung cấp bởi người dùng, nếu không thành công sẽ có cửa sổ thông báo hiện lên.

```
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.settimeout(0.5)

    try:
        # Bind the socket to the address using the RTP port given by the client user
        self.rtpSocket.bind(('',self.rtpPort))
    except:
        tkinter.messagebox.showwarning('Unable to Bind', 'Unable to bind PORT=%d' %self.rtpPort)
```

- Sau đó, nếu yêu cầu "PLAY" được gửi đến Server, Số RTSP Packet Sequence được tăng lên 1 và gói tin RTSP "PLAY" sẽ bao gồm:

  + Yêu cầu "SETUP"

  + Tên Video muốn phát

  + Protocol type: RTSP/1.0

  + Số RTSP Packet Sequence

  + SessionId (được khởi tạo ngẫu nhiên từ ServerWorker và trả về từ phản hồi cho yêu cầu "SETUP")

  Thông tin về requestSent cũng được ghi lại.

```
# 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 " + str(self.fileName) + " RTSP/1.0\n"
    request += "CSeq: " + str(self.rtspSeq) + "\n"
    request += "Session: " + str(self.sessionId)

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

- Khi nhận được yêu cầu, Server sẽ tạo một Socket để truyền RTP qua UDP và khởi tạo một luồng (threading) để gửi gói tin RTP.

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

        # Create a new socket for RTP/UDP
        self.clientInfo["rtpSocket"] = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

        self.replyRtsp(self.OK_200, seq[1])

        # Create a new thread and start sending RTP packets
        self.clientInfo['event'] = threading.Event()
        self.clientInfo['worker']= threading.Thread(target=self.sendRtp)
        self.clientInfo['worker'].start()
```

- VideoStream.py sẽ giúp cắt tệp video thành từng frame riêng biệt và đưa từng frame vào gói dữ liệu RTP.

```
def sendRtp(self):
    """Send RTP packets over UDP."""
    while True:
        self.clientInfo['event'].wait(0.05)

        # Stop sending if request is PAUSE or TEARDOWN
        if self.clientInfo['event'].isSet():
            break

        data = self.clientInfo['videoStream'].nextFrame()
        if data:
            frameNumber = self.clientInfo['videoStream'].frameNbr()
            try:
                address = self.clientInfo['rtspSocket'][1][0]
                port = int(self.clientInfo['rtpPort'])
                self.clientInfo['rtpSocket'].sendto(self.makeRtp(data, frameNumber),(address,port))
            except:
                print("Connection Error")
                #print('-'*60)
                #traceback.print_exc(file=sys.stdout)
                #print('-'*60)
```

```python
def nextFrame(self):
    """Get next frame."""
    data = self.file.read(5) # Get the framelen
    if data:
        framelength = int(data)

        # Read the current frame
        data = self.file.read(framelength)
        self.frameNum += 1
    return data
```

```python
def makeRtp(self, payload, frameNbr):
    """RTP-packetize the video data."""
    version = 2
    padding = 0
    extension = 0
    cc = 0
    marker = 0
    pt = 26 # MJPEG type
    seqnum = frameNbr
    ssrc = 0

    rtpPacket = RtpPacket()

    rtpPacket.encode(version, padding, extension, cc, seqnum, marker, pt, ssrc, payload)

    return rtpPacket.getPacket()
```

- Việc thực hiện mã hóa gói tin RTP gồm các trường của header và payload sẽ được thực hiện tại RtpPacket.py, chúng được chèn vào gói tin RTP thông qua các thao tác bitwise:

```python
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

    self.header[0] = version << 6      unknown, a day ago • ComputerNetworkAssignment1
    self.header[0] = self.header[0] | padding << 5
    self.header[0] = self.header[0] | extension << 4
    self.header[0] = self.header[0] | cc
    self.header[1] = marker << 7
    self.header[1] = self.header[1] | pt

    # 2 bytes of sequence number
    self.header[2] = (seqnum >> 8) & 0xFF
    self.header[3] = seqnum & 0xFF
```

```python
    # 4 bytes of timestamp
    self.header[4] = (timestamp >> 24) & 0xFF
    self.header[5] = (timestamp >> 16) & 0xFF
    self.header[6] = (timestamp >> 8) & 0xFF
    self.header[7] = timestamp & 0xFF

    # 4 byte of SSRC
    self.header[8] = ssrc >> 24
    self.header[9] = ssrc >> 16
    self.header[10] = ssrc >> 8
    self.header[11] = ssrc

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

- Kích thước nhỏ nhất của một header của gói tin RTP là 12 bytes. Sau phần header chính, là phần header mở rộng và không cần thiết phải có phần header này. Chi tiết các trường trong một header như sau:

RTP packet header

bit offset	0-1	2	3	4-7	8	9-15	16-31
0	Version	P	X	CC	M	PT	Sequence Number
32	Timestamp						
64	SSRC identifier						
96	CSRC identifiers ...						
96+32×CC	Profile-specific extension header ID						Extension header length
128+32×CC	Extension header ...						

+ Version (2 bits): Cho biết phiên bản của giao thức này.

+ P (Padding) (1 bit) : Cho biết số các byte mở rộng cần thêm vào cuối của gói tin RTP. Ví dụ trong trường hợp ta muốn sử dụng các thuật toán mã hóa, ta có thể thêm vào một số byte vào phần kết thúc của gói tin để tiến hành mã hóa frame trên đường truyền.

+ X (Extension) (1bit): Cho biết có thêm phần header mở rộng vào sau phần header chính hay không.

+ CC (CSRC Count) (4 bit) : Chứa con số định danh CSRC cho biết kích thước cố định của header.

+ M (Marker) (1 bit) : Cho biết mức của ứng dụng và được định nghĩa bởi một profile. Nếu được thiết lập, có nghĩa là dữ liệu hiện tại đã được tính toán chi phí một cách thích hợp

+ PT (Payload Type) (7 bit) : Cho biết định dạng của file video. Đây là một đặc tả được định nghĩa bởi một profile RTP.

+ Sequence Number (16 bits) : số hiệu của frame. Và sẽ được tăng lên 1 đơn vị cho mỗi gói tin RTP trước khi gửi và được sử dụng bởi bên nhận để dò ra các gói bị lạc và có thể phục hồi lại gói có số thứ tự đó.

+ Timestamp (32 bits): Được sử dụng thông báo cho bên nhận biết để phát lại frame này trong khoảng thời gian thích hợp.

+ SSRC (32 bits): Định danh cho nguồn streaming. Mỗi nguồn cho phép streaming video sẽ định danh bởi một phiên RTP duy nhất.

```
self.clientInfo['rtpSocket'].sendto(self.makeRtp(data, frameNumber),(address,port))
```

- Cuối cùng, gói tin RTP sẽ bao gồm một header và một frame của video được gửi đến RTP port ở phía Client.
- Phía Client sẽ lắng nghe để nhận các gói tin RTP Server gửi đến, sau đó tiến hành giải mã gói để lấy header và frame,

```
def listenRtp(self):
    """Listen for RTP packets."""
    while True:
        try:
            data = self.rtpSocket.recv(20480)
            print("[*]Received RTP packet")

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

```
def decode(self, byteStream):
    """Decode the RTP packet."""
    self.header = bytearray(byteStream[:HEADER_SIZE])
    self.payload = byteStream[HEADER_SIZE:]
```

- Các frame sau đó sẽ được hiển thị trên giao diện người dùng:

```
if currFrameNbr > self.frameNbr: # Discard the late packet
    self.frameNbr = currFrameNbr
    self.updateMovie(self.writeFrame(rtpPacket.getPayload()))
```

```python
def writeFrame(self, data):
    """Write the received frame to a temp image file. Return the image fi
    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
```

- Nếu lệnh “PAUSE” được gửi từ Client đến Server, nó sẽ ngăn Server gửi các frame đến Client.

```python
# Pause request
elif requestCode == self.PAUSE and self.state == self.PLAYING:
    # Update RTSP sequence number.
    self.rtspSeq += 1


    request = "PAUSE " + str(self.fileName) + " RTSP/1.0\n"
    request += "CSeq: " + str(self.rtspSeq) + "\n"
    request += "Session: " + str(self.sessionId)

    # Keep track of the sent request.
    self.requestSent = self.PAUSE      unknown, 21 hours ago •
```

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

        self.clientInfo['event'].set()

        self.replyRtsp(self.OK_200, seq[1])
```

```
def sendRtp(self):
    """Send RTP packets over UDP."""
    while True:
        self.clientInfo['event'].wait(0.05)

        # Stop sending if request is PAUSE or TEARDOWN
        if self.clientInfo['event'].isSet():
            break
```

- Nếu lệnh “TEARDOWN” được gửi từ Client đến Server, nó cũng sẽ ngăn Server gửi các frame video đến Client và đóng cả kết nối của Client.

```
# Process TEARDOWN request
elif requestType == self.TEARDOWN:
    print("processing TEARDOWN\n")

    self.clientInfo['event'].set()

    self.replyRtsp(self.OK_200, seq[1])
```

```
def sendRtp(self):
    """Send RTP packets over UDP."""
    while True:
        self.clientInfo['event'].wait(0.05)

        # Stop sending if request is PAUSE or TEARDOWN
        if self.clientInfo['event'].isSet():
            break
```

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

4. Hướng dẫn sử dụng

Chương trình bao gồm:

Client.py

ClientLauncher.py

RtpPacket.py

Server.py

ServerWorker.py

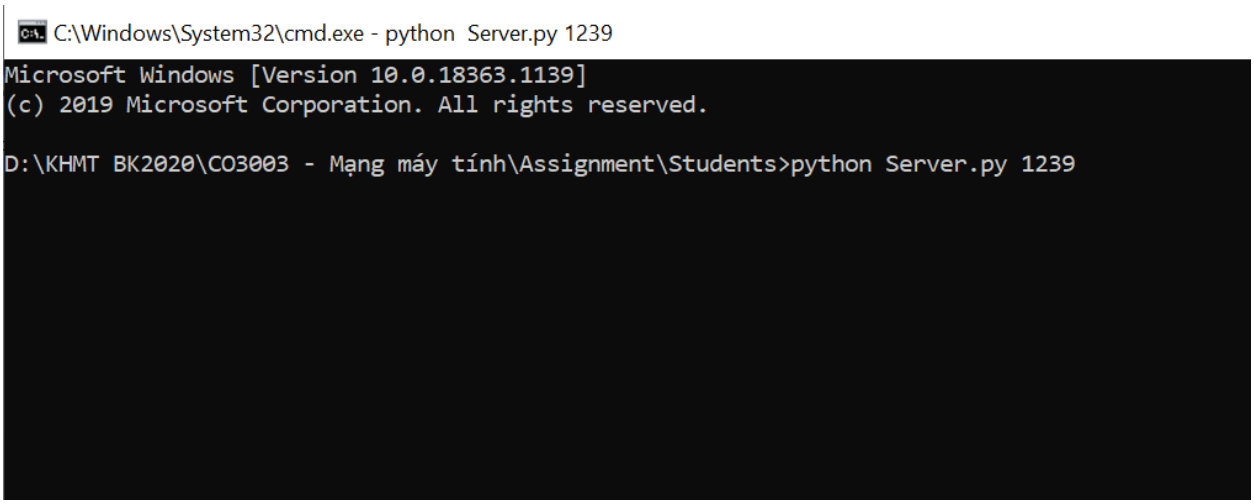
VideoStream.py

Thực hiện chương trình như sau:

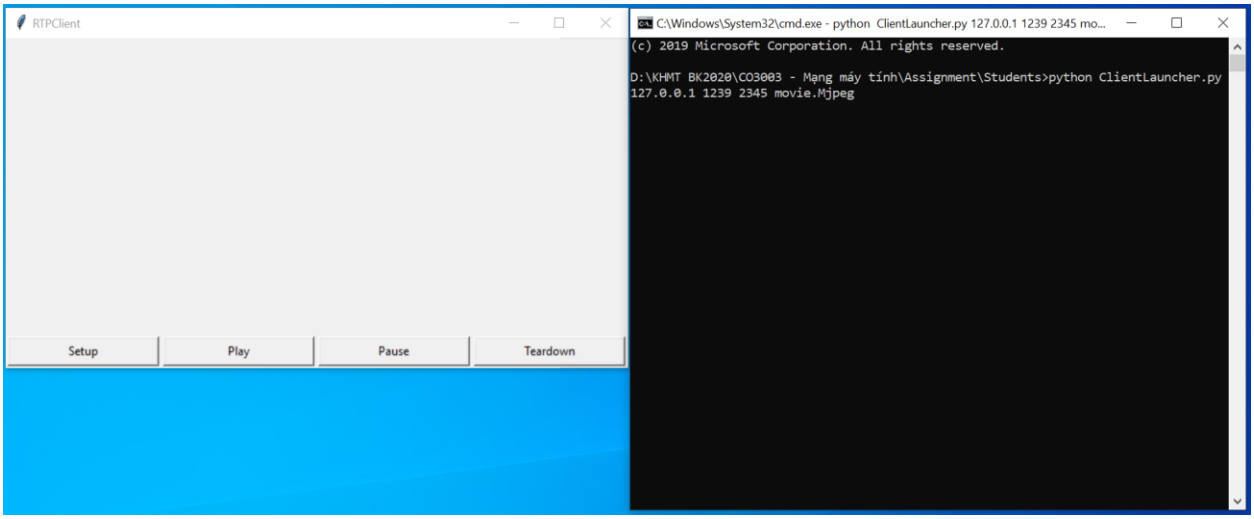
1. Chạy Server.py trên Command Promt tại thư mục chứa ứng dụng để khởi động server:

- Câu lệnh: python Server.py server_port

- server_port là cổng mà máy chủ lắng nghe các kết nối RTSP đến, trong Bài tập lớn này, chúng ta sử dụng Port có giá trị lớn hơn 1024, chúng ta cho server_port giá trị là 1239.

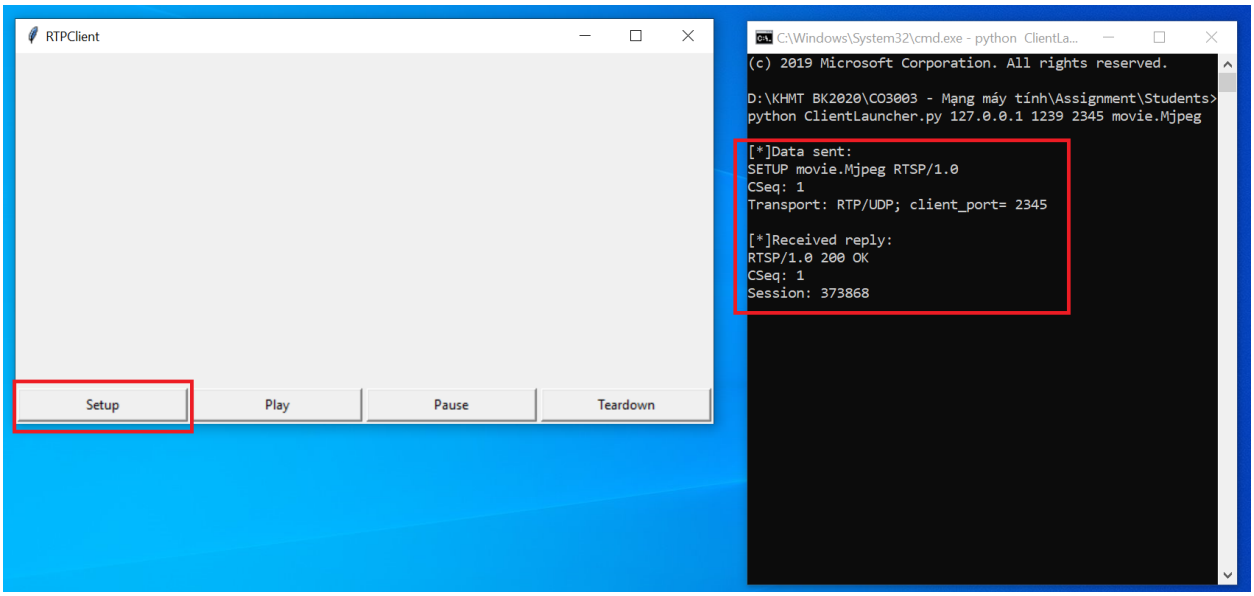


2. Chạy ClientLaucher.py trên Command Promt tại thư mục chứa ứng dụng để khởi động Client và giao diện người dùng:
   - Câu lệnh: python ClientLauncher.py server_host server_port PRT_port video_file
   - server_host là địa chỉ IP của thiết bị mà server đang chạy (chúng ta có thể sử dụng "127.0.0.1").
   - server_port là cổng mà máy chủ đang lắng nghe, ở đây là 1239.
   - RTP_port là cổng nơi các gói RTP được nhận , chúng ta cho RTP_port có giá trị là 2345.
   - video_file là tên của tệp video mà chúng ta muốn phát (ở đây là "video.mjpeg")

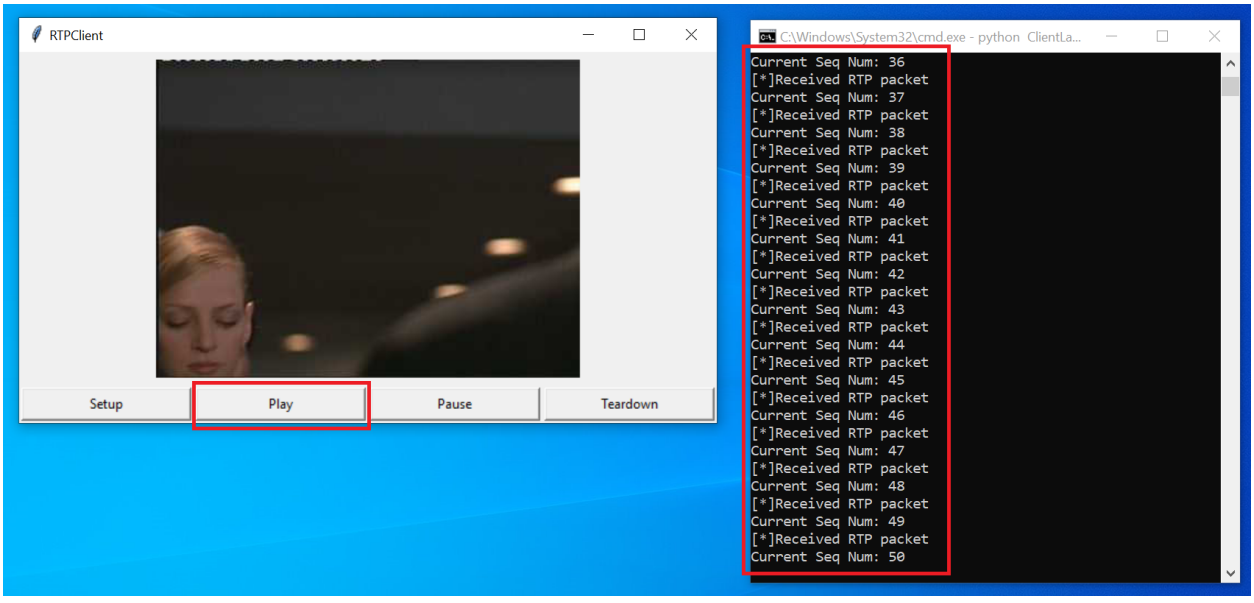


   - Nếu kết nối không thành công sẽ có thông báo hiện lên. Nếu không có thông báo hiện lên nghĩa là đã kết nối thành công.

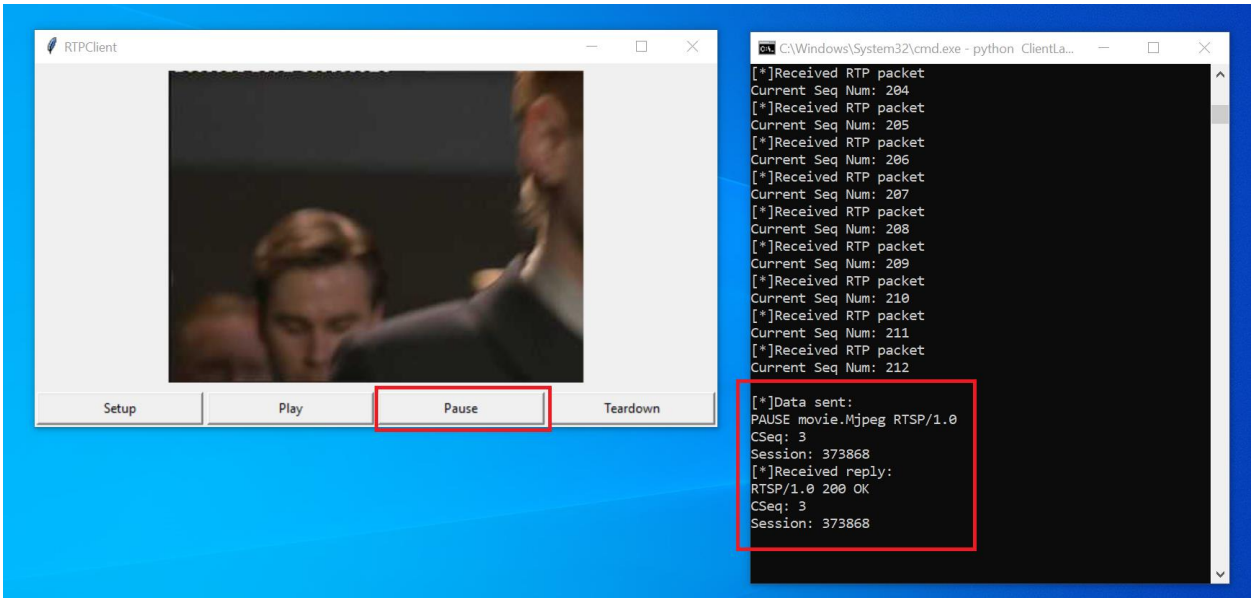
3. Thao tác trên giao diện:
   - Gửi yêu cầu “SETUP” từ Client đến Server bằng cách nhất vào nút “SETUP” trên giao diện.

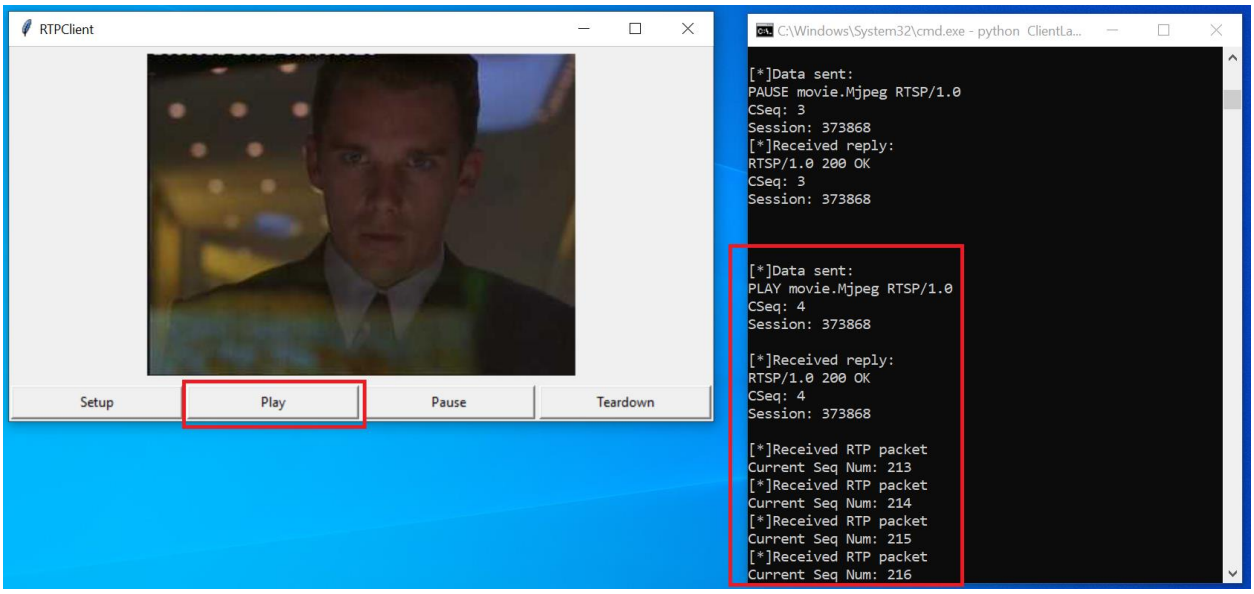


   - Sau khi gửi yêu cầu “SETUP”, để phát video, nhấn vào nút “PLAY” để gửi yêu cầu đến Server.

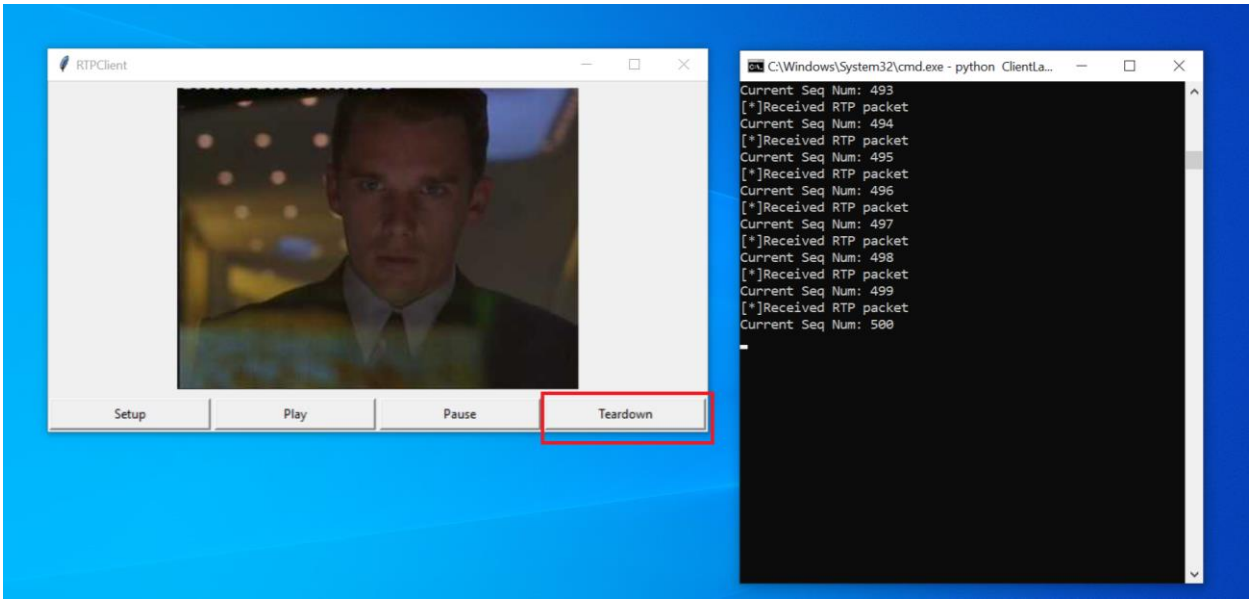


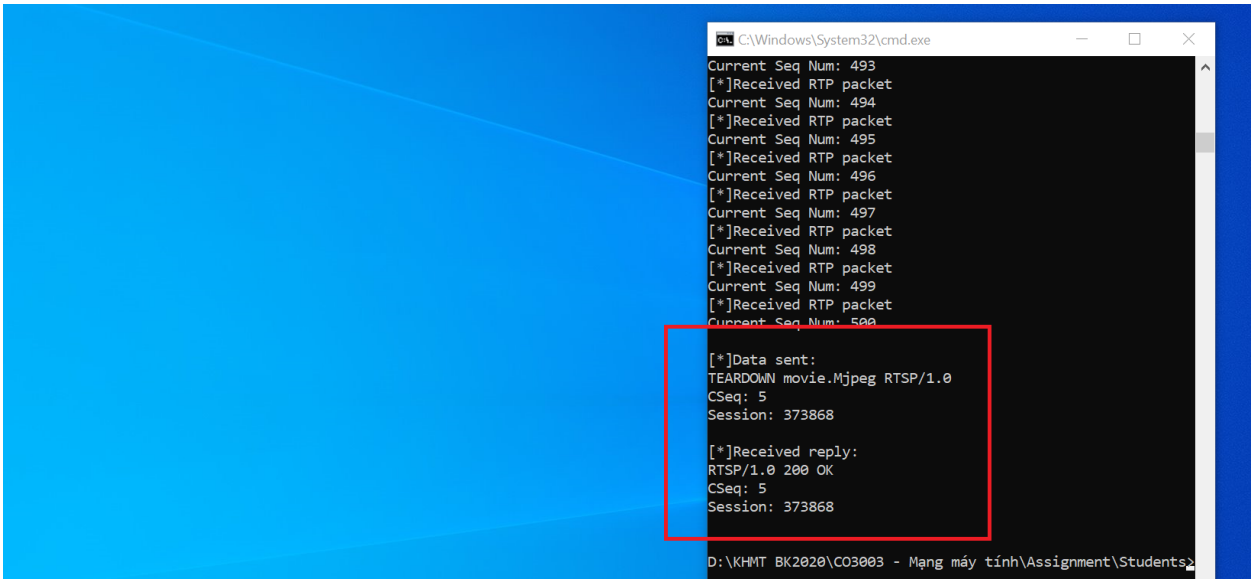
   - Để tạm dừng khi video đang phát, nhấn vào nút “PAUSE” trên giao diện, nếu muốn tiếp tục phát video, nhấn lại vào nút “PLAY”.


RTPClient
Setup
Play
Pause
Teardown
C:\Windows\System32\cmd.exe - python ClientLa...
[*]Received RTP packet
Current Seq Num: 204
[*]Received RTP packet
Current Seq Num: 205
[*]Received RTP packet
Current Seq Num: 206
[*]Received RTP packet
Current Seq Num: 207
[*]Received RTP packet
Current Seq Num: 208
[*]Received RTP packet
Current Seq Num: 209
[*]Received RTP packet
Current Seq Num: 210
[*]Received RTP packet
Current Seq Num: 211
[*]Received RTP packet
Current Seq Num: 212
[*]Data sent:
PAUSE movie.Mjpeg RTSP/1.0
CSeq: 3
Session: 373868
[*]Received reply:
RTSP/1.0 200 OK
CSeq: 3
Session: 373868


RTPClient
Setup
Play
Pause
Teardown
C:\Windows\System32\cmd.exe - python ClientLa...
[*]Data sent:
PAUSE movie.Mjpeg RTSP/1.0
CSeq: 3
Session: 373868
[*]Received reply:
RTSP/1.0 200 OK
CSeq: 3
Session: 373868
[*]Data sent:
PLAY movie.Mjpeg RTSP/1.0
CSeq: 4
Session: 373868
[*]Received reply:
RTSP/1.0 200 OK
CSeq: 4
Session: 373868
[*]Received RTP packet
Current Seq Num: 213
[*]Received RTP packet
Current Seq Num: 214
[*]Received RTP packet
Current Seq Num: 215
[*]Received RTP packet
Current Seq Num: 216

- Để dừng phát video, ngắt kết nối với Server và tắt giao diện người dùng, nhấn vào nút "TEARDOWN" trên giao diện.





5. Đánh giá kết quả đạt được

- Có được kiến thức tổng quát về hai giao thức RTSP (Real Time Streaming Protocol) và RTP (Real-time Transport Protocol):

  + RTSP: sử dụng cho hệ thống giải trí và truyền thông để kiểm soát các streaming media servers; thiết lập và kiểm soát các phiên truyền giữa các điểm cuối; sử dụng TCP.

  + RTP: giao thức để vận chuyển dữ liệu video và audio qua mạng IP; sử dụng UDP.
- Hiểu được cách thức tạo Socket để kết nối giữa Server và Client.
- Hiện thực được giao thức RTSP tại Client (Client.py) để thực hiện gửi các yêu cầu "SETUP", "PLAY", "PAUSE", "TEARDOWN" đến Server thông qua các thao tác nhấn các nút tương ứng trên giao diện của người dùng. Các thông tin gửi đi theo đúng yêu cầu đề bài, và phản hồi từ Server cũng được đọc.
- Hiện thực được việc đóng gói gói tin RTP từ các frame dữ liệu của video (RtpPacket.py, phương thức encode()). Hiểu về các trường trong một header của một gói tin RTP.

II. Phần mở rộng (Extend)

1. Tính RTP packet loss rate và Video data rate (bytes per second):

Phân tích:

- RTP packet loss rate: được định nghĩa là tỷ lệ của tổng số gói được truyền nhưng không đến máy nhận. Để tính tỷ lệ này ta sẽ đếm số lượng gói bị mất đi (Client không nhận được), sau đó chia cho số gói đã được gửi đi đến hiện tại.
- Video data rate (bytes per second): để tính thông số này, ta sẽ tính tổng số lượng bytes của các frame đã nhận được và thời gian từ lúc gửi yêu cầu "PLAY" đến khi nhận được gói tin RTP mới nhất.

Thực hiện:

- Thêm các thuộc tính sau tại Client để tính toán các thông số trên. Thêm các Label vào giao diện để hiện các thông số Total Bytes, Packet loss rate, Data Rate.

```
        self.totalBytes = 0
        self.lostNum = 0
        self.statLost = 0
        self.startTime = 0
        self.totalPlayTime = 0
        self.dataRate = 0
```

```
    # Create a label to display the stats
    self.label2 = Label(self.master)
    self.label2["text"] = "Total Bytes Received: 0"
    self.label2.grid(row=2, column=0, columnspan=4, padx=5, pady=5)
    self.label3 = Label(self.master)
    self.label3["text"] = "Packets Lost Rate: 0"
    self.label3.grid(row=3, column=0, columnspan=4, padx=5, pady=5)
    self.label4 = Label(self.master)
    self.label4["text"] = "Data Rate (bytes/sec): 0"
    self.label4.grid(row=4, column=0, columnspan=4, padx=5, pady=5)
```

- Đầu tiên, khi thực hiện gửi yêu cầu "PLAY", ta sẽ tiến hành lưu lại thời điểm bắt đầu:

```
# Play request
elif requestCode == self.PLAY and self.state == self.READY:

    # save the time start
    self.startTime = time.time()
```

- Khi nhận được gói tin RTP từ Server, ta sẽ thực hiện ghi lại thời điểm nhận được gói tin, từ đó tính được tổng thời gian từ lúc yêu cầu "PLAY" đến thời điểm nhận được gói tin RTP theo cú pháp sau:

```
try:
    data = self.rtpSocket.recv(20480)
    print("[*]Received RTP packet")

    curTime = time.time()
    self.totalPlayTime += curTime - self.startTime
    self.startTime = curTime
```

- Đồng thời, ta tính được tổng số byte nhận được bằng cách cộng dồn size của payload của mỗi gói tin RTP nhận được. Từ đó tính được thông số Data Rate bằng cách lấy tổng byte nhận được chia cho tổng thời gian từ lúc yêu cầu "PLAY":

```
# Tính tổng số byte nhận được bằng cách cộng dồn size của payload
self.totalBytes += len(rtpPacket.getPayload())

# Tính data rate
self.dataRate = round(self.totalBytes/self.totalPlayTime, 2)
```

- Về Packet Loss, ta có thể xác định bằng cách so sánh số frame mà Client đang lưu với số frame của gói tin mà Server trả về, nếu số frame mà Client đang lưu cộng thêm 1 mà không bằng với số frame của gói tin mà Server trả về, có nghĩa đã có gói tin mà Client không nhận được. Từ đó ta tính được số gói tin Client không nhận được, đồng thời số gói tin RTP mà Server đã gửi đến thời điểm nhận gói tin mới nhất cũng chính là số frame của gói tin mà Server trả về. Do đó mà ta tính được thông số Packet Loss Rate:

```
currFrameNbr = rtpPacket.seqNum()        unknown, 3 days ago • ComputerN
print("Current Seq Num: " + str(currFrameNbr))

# xác định mất gói và số gói bị mất  (Packet loss rate is defined as t
arrive at the receiver.)
if self.frameNbr + 1 != currFrameNbr:
    self.lostNum += currFrameNbr - self.frameNbr - 1
    print("[*]Packet loss!")

 # tính packet loss rate
 self.statLost = 0 if self.lostNum == 0 else self.lostNum/currFrameNbr
```

- Sau đó ta thực hiện cập nhật lại các thông số trên lên giao diện:

```
 # Update lại các thông số
 self.updateStat()        You, 7 minutes ago • Uncommitted change


def updateStat(self):
    self.label2["text"] = "Total Bytes Received: " + str(self.totalBytes)
    self.label3["text"] = "Packets Lost Rate: " + str(self.statLost)
    self.label4["text"] = "Data Rate (bytes/sec): " + str(self.dataRate)
```