ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KHOA HỌC - KỸ THUẬT MÁY TÍNH

MẠNG MÁY TÍNH

Bài tập lớn 1

VIDEO STREAMING APPLICATION

GV ra đề và hướng dẫn: Trần Huy

Lớp L04: Nhóm 01
SV thực hiện: Nguyễn Quốc Hùng - 1913610 **(Nhóm trưởng)**
Nguyễn Anh Văn - 1915886
Nông Trọng Thuyên - 1915390
Nguyễn Ngọc Nguyên - 1914384

Tp. Hồ Chí Minh, Tháng 11/2021

Mục lục

Danh sách các thành viên & phân công, đánh giá công việc **2**

1 Phân tích yêu cầu **3**
1.1 Functional . . . 3
1.2 Non-Functional . . . 3

2 Mô tả chức năng **4**
2.1 Set up . . . 5
2.2 Play . . . 6
2.3 Pause . . . 9
2.4 TearDown . . . 10
2.5 Describe . . . 11
2.6 Backward . . . 12
2.7 FastForWard . . . 13
2.8 Packet Loss Rate và Data Rate . . . 14
2.9 Sử dụng ứng dụng không cần nhấn SETUP . . . 14

3 Danh sách các component **15**
3.1 Client . . . 15
3.2 RtpPacket . . . 16
3.3 Server . . . 16
3.4 ServerWorker . . . 16
3.5 VideoStream . . . 17

4 Mô hình dòng dữ liệu **17**

5 Class diagram **18**

6 Đánh giá kết quả **18**

7 Cách chạy ứng dụng **19**

8 Source Code **20**

Danh sách thành viên & nhiệm vụ, đánh giá

STT	Họ và tên	MSSV	Phân công công việc	Đánh giá
1	Nguyễn Quốc Hùng	1913610	Tổng hợp báo cáo thực hiện chức năng Set up, Play, Pause	100%
2	Nguyễn Anh Văn	1915886	Thực hiện chức năng TearDown và Describe	100%
3	Nông Trọng Thuyên	1915390	Thực hiện chức năng Backward và FastForWard	100%
4	Nguyễn Ngọc Nguyên	1914384	Thực hiện tính toán Packet Loss Rate và Data Rate, sử dụng ứng dụng không cần nhấn Setup	100%

Bảng 1: *Bảng phân công công việc*

1 Phân tích yêu cầu

1.1 Functional

Chương trình gồm có 5 file:

- Client.py
- ClientLauncher.py
- RtpPacket.py
- Server.py
- ServerWorker.py

Hiện thực một ứng dụng truyền video trực tuyến giữa máy chủ và khách hàng sử dụng giao thức RTSP trong máy chủ và khử nhịp độ RTP trong máy khách hàng và quan tâm đến việc hiển thị video đã truyền.

- RTSP(Real Time Streaming Protocol): Thường được sử dụng trong các hệ thống giải trí và truyền thông để điều khiển các máy chủ đa phương tiện truyền trực tiếp và kiểm soát các phiên truyền giữa các điểm cuối. RTSP chạy trên giao thức TCP.
- RTP(Real-time Transport Protocol) là giao thức dùng để chuyển tập tin video, âm thanh qua mạng IP, thường được sử dụng nhiều trong các hệ thống Streaming media. RTP chạy trên giao thức UDP.

1.2 Non-Functional

Một số yêu cầu phi chức năng như:

- Socket datagram để nhận dữ liệu RTP và thời gian chờ trên Socket tối đa là 0.5 giây.
- Sử dụng port cho server trên 1024.
- Chỉ thực hiện 3 mã trả về 200 là đã thành công trong khi đó 404 và 500 đại diện cho file not found và lỗi kết nối.
- Khi gửi khung video cho máy khách thì một khung được gửi sau tối đa 50 mili giây.
- Video được phát thì được định dạng với dạng tệp MJPEG.
- **Yêu cầu về hiệu suất**: Tỉ lệ mất gói tin (Packet loss rate) phải ít hơn 5% và dữ liệu tốc độ video (video data rate) phải lớn hơn 70kb byte/s
- **Yêu cầu về khả năng sử dụng**: RTSP cung cấp vận chuyển dữ liệu với đặc điểm là thời gian thực, chẳng hạn như âm thanh và các video tương tác. Thêm vào đó, RTP hỗ trợ vận chuyển các gói tin tới điểm đến sử dụng phân phối đa hướng nếu được cung cấp bởi các mạng cơ bản.
- **Yêu cầu về độ tin cậy**: Tỉ lệ mất gói tin (Packet loss rate) được giảm tới mức tối thiểu nhất

2 Mô tả chức năng

Một trong các bước triển khai ban đầu, máy chủ sẽ tạo TCP socket và bắt đầu lắng nghe đến một Port nhất định. Đoạn code dưới đây cho đấy cách RTSP socket và lắng nghe máy khách thông qua Port nhất định.

```
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()
```

Ở đoạn code trên cho thấy rằng thread với tên ClientInfo được tạo để giải quyết các yêu cầu trong thread mới. ServerWorker là một class được thực hiện nhằm giải quyết tất cả các yêu cầu RTSP từ máy khách.

Khi máy khách khởi động thì đồng thời mở RTSP socket đến máy chủ. Khi người dùng nhấn các nút trên giao diện thì các chức năng sẽ được socket gửi máy chủ. Những chức năng chính gồm:

- SETUP
- PLAY
- PAUSE
- TEARDOWN

Bên cạnh đó còn gồm các Extend được thêm vào gồm:

- DESCRIBE
- BACKWARD
- FASTFORWARD
- Sử dụng PLAY không cần SETUP
- Tính toán Packet Loss Rate và Data Rate
- Hiển thị Slider cho video đồng thời cập nhật liên tục

Hơn nữa còn bao gồm các Lable biểu diện thông tin của tỉ lệ mất gói tin (Packet loss rate) và dữ liệu tốc độ video (Video data rate) đồng thời là một Lable để hiện thông tin Describe của một video (DESCRIBE response).

Các lệnh này sẽ cho máy chủ biết hành động tiếp theo mà nó sẽ hoàn thành. Những gì mà máy chủ gửi đến máy khách thông qua giao thức RTSP là các tham số để khác hàng biết phản hồi của máy chủ khi nhận các lệnh một cách chính xác.

- OK_200
- FILE_NOT_FOUND_404
- CON_ERR_500

2.1 Set up

Khi máy khách gửi yêu cầu SETUP đến máy chủ, chèn thông tin truyền tải mà mình chỉ định cổng cho Socket datagram RTP mà mình vừa tạo. Máy khách sẽ nhận được phản hồi từ máy chủ và nhận được ID của phiên RTSP.

```
# Process SETUP request
if requestType == self.SETUP:
    self.clientInfo['isDescribe-request']=False
    if self.state == self.INIT:
        # Update state
        print ("\nPROCESSING SETUP\n")

        try:
            self.clientInfo['videoStream'] = VideoStream(filename)
            self.totalFrame = self.clientInfo['videoStream'].totalFrame()
            self.clientInfo['videoStream'] = VideoStream(filename)
            self.state = self.READY
            self.request = 'SETUP'

        except IOError:
            self.replyRtsp(self.FILE_NOT_FOUND_404, seq[1])

        # Generate a randomized RTSP session ID
        self.clientInfo['session'] = randint(100000, 999999)
        self.clientInfo['filename']=filename
        self.clientInfo['length'] = len(data)
        # 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]
```

Gói RTSP SETUP sẽ bao gồm:

- Lệnh SETUP
- Tên của file video phát
- Số thứ tự gói RTSP bắt đầu từ 1
- Loại giao thức: RTSP/1.0 RTP
- Giao thức truyền: UDP
- RTP port để truyền video

Khi phía máy chủ nhận được lệnh "SETUP" thì sẽ thực hiện các bước sau:

- Gán cho máy khách một Session ngẫu nhiên.
- Nếu có vấn đề xảy ra với lệnh hoặc có lỗi trong máy chủ thì sẽ trả error về phía khách.
- Nếu lệnh xử lí chính xác không lỗi thì nó sẽ trả lại OK-200 cho khách và đặt trạng thái READY. Máy chủ sẽ mở tệp video được chỉ định trong SETUP Packet và khởi tạo số frame thành 0

Nếu gói RTSP Packet được phản hồi cho lệnh SETUP thì máy khách sẽ đặt STATE của nó là READY
Sau đó mở một RTP Port để nhận luồng video.

2.2 Play

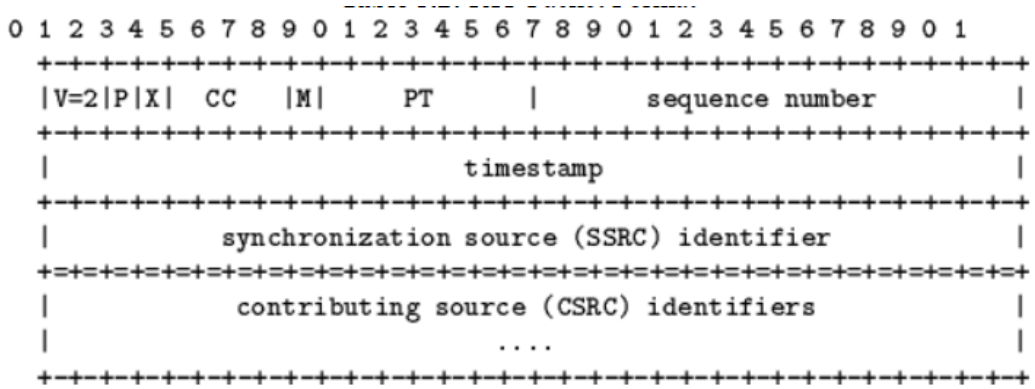
Khi yêu cầu PLAY được gửi tới máy chủ. Máy chủ sẽ tạo ra một Socket RTP/UDP và bắt đầu gửi video:
Sau đó file Videostream.py sẽ được dùng để cắt video thành các frame để gửi tới cho máy khách

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

```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()
```

Mỗi packet data sẽ được mã hóa với một header bao gồm:

- RTP-version (V=2)
- Padding (P)
- Extension (X)
- CSRC Count (CC)
- Marker (M)
- Payload Type (PT)
- Sequence Number
- Timestamp
- SSRC



Hình 1: *Packet Format*

Chúng sẽ được chèn vào RTP packet thông qua thao tác bitwise với 1 index của header sẽ chứa 8 bit của RTP packet:

```
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
```

Sau đó RTP packet sẽ được định dạng thông qua một frame và header để gửi đến cổng RTP ở phía khách:

```
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
        if self.request == self.NEXT:
            data = self.clientInfo['videoStream'].skipFrame(self.totalFrame, self.skipNo*75)
            self.skipNo = 0
            self.request = ''
        elif self.request == self.BACK:
            data = self.clientInfo['videoStream'].backFrame(self.skipNo*75)
            self.skipNo = 0
            self.request = ''
        else:
            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")
```

Máy khách lúc này sẽ lắng nghe RTP Packet đồng thời sau đó sẽ giải mã RTP Packet để lấy header và frame đồng thời tổ chức lại khung và hiện trên giao diện người dùng.

```
def listenRtp(self):
    """Listen for RTP packets."""
    start = datetime.now()
    end = datetime.now()
    total_bytes = 0
    while True:
        try:
            print("LISTENING...")
            data = self.rtpSocket.recv(20480)
            if data:
                rtpPacket = RtpPacket()
                rtpPacket.decode(data)
                # rtpPacket.
                currFrameNbr = rtpPacket.seqNum()
                print ("CURRENT SEQUENCE NUM: " + str(currFrameNbr))
                if (currFrameNbr-self.frameNbr)!=1:
                    self.numLatePacket+=1
                if currFrameNbr > self.frameNbr: # Discard the late packet
                    self.frameNbr = currFrameNbr
                    self.updateMovie(self.writeFrame(rtpPacket.getPayload()))
                    self.updateSlider(int(self.frameNbr/15))
                end = datetime.now()
                total_time = self.time_different(end,start)
                total_bytes += rtpPacket.getPayload_length()

                Packet_loss = self.computePercentLoss(self.frameNbr+self.numLatePacket,self.numLatePacket)
                video_data_rate = self.computeRateKBs(total_bytes,total_time)
                self.label1 = Label(text=str(Packet_loss),width=20,relief=tkinter.RIDGE,background="yellow"
                self.label2 = Label(text=str(video_data_rate),width=20,relief=tkinter.RIDGE,background="yel
```

2.3 Pause

Khi yêu cầu PAUSE được gửi tới máy chủ, lúc này máy khách sẽ thoát khỏi thread đang tồn tại để ngừng nhận Packet từ máy chủ gửi tới đồng thời tạo ra một thread mới để tiếp tục cho lần phát video sau.

```
        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.NEXT:
            pass
```

Đồng thời lúc này máy chủ sẽ chuyển trạng thái của ClientInfo thành READY đồng thời đóng thread lại.

```
    # Process PAUSE request
    elif requestType == self.PAUSE:
        if self.state == self.PLAYING:
            print ("\nPROCESSING P A U S E\n")
            self.state = self.READY

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

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

2.4 TearDown

Khi yêu cầu TEARDOWN được gửi tới máy chủ, thì máy khách sẽ ngăn cản máy chủ gửi các khung hình video đến máy khách và đóng thiết bị đầu cuối của máy khách và lúc đó STATE của máy khách được chuyển về trạng thái INIT:

```
    # 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 = "%s %s %s" % (self.TEARDOWN_STR, self.fileName, self.RTSP_VER)
    request+="\nCSeq: %d" % self.rtspSeq
    request+="\nSession: %d" % self.sessionId

    self.requestSent = self.TEARDOWN
```

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

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

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

        # Close the RTP socket
        self.clientInfo['rtpSocket'].close()
```

2.5 Describe

Khi yêu cầu TEARDOWN được gửi tới máy chủ, lúc này máy chủ sẽ định dạng thông tin dữ liệu rồi gửi về cho máy khách.

```
        # Close the RTP socket
        self.clientInfo['rtpSocket'].close()
    # Process DESCRIBE request
    elif requestType == self.DESCRIBE:
        self.clientInfo['isDescribe-request']=True
        self.replyRtsp(self.OK_200,seq[1])
```

```
if self.clientInfo['isDescribe-request']==True:
    #first
    segment1 =  'RTSP/1.0 200 OK\nCSeq: ' + seq
    segment2 = '\nv=1.0'
    segment3 = '\nm=video '+ str(self.clientInfo['rtpPort']) + ' RTP/AVP ' + '26'
    segment4 = '\nSession ID =' + str(self.clientInfo['session'])
    segment5 = '\na=mimetype:string;\"video/MJPEG\"'
    first_body = segment1+segment2+segment3+segment4+segment5

    #second
    segment6 = '\nContent-Base: '+self.clientInfo['filename']
    segment7 = '\nContent-Type: ' + 'application/sdp'
    segment8 = '\nContent-Length: ' + str(len(first_body))
    second_body = segment6+segment7+segment8
    reply = first_body+second_body
```

Đồng thời lúc này máy khách sẽ nhận dữ liệu giải mã dữ liệu rồi hiển thị lên màn hình của mình.

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

    #DESCRIBE REQUEST
    if self.requestSent==self.DESCRIBE:
        self.display_description(data)
        self.state=self.READY
```

2.6 Backward

Khi yêu cầu BACKWARD được gửi tới máy chủ, máy chủ sẽ xác định frame thông qua hàm BackFrame trong file VideoStreaming.py khi nhấn nút BACKWARD đồng thời gửi lên cho Client.

```
def skipFrame(self, limit, skipNo):
    """Get next frame."""
    data = self.file.read(0)  # Get the framelength from the first 5 bits
    if skipNo + self.frameNum > limit:
        skipNo = limit - self.frameNum
    for x in range(skipNo):
        # Get the framelength from the first 5 bits
        data = self.file.read(5)
        if data:
            framelength = int(data)
            # Read the last 30 frame from current
            data = self.file.read(framelength)
            self.frameNum += 1
        if self.frameNum + 1 == limit:
            break
    return data
```

```
        if self.clientInfo['event'].isSet():
            break
        if self.request == self.NEXT:
            data = self.clientInfo['videoStream'].skipFrame(self.totalFrame, self.skipNo*75)
            self.skipNo = 0
            self.request = ''
        elif self.request == self.BACK:
            data = self.clientInfo['videoStream'].backFrame(self.skipNo*75)
            self.skipNo = 0
            self.request = ''
        else:
```

2.7 FastForWard

Khi yêu cầu FASTFORWARD được gửi tới máy chủ, máy chủ sẽ xác định frame thông qua hàm SkipFrame trong file VideoStreaming.py khi nhấn nút FASTFORWARD đồng thời gửi lên cho Client.

```
    def backFrame(self, skipNo):
        """Get back frame."""
        self.file.seek(0)
        temp = self.frameNum
        self.frameNum = 0
        data = self.file.read(0)
        for x in range(temp - skipNo):
            # Get the framelength from the first 5 bits
            data = self.file.read(5)
            if data:
                framelength = int(data)
                # Read the current frame
                data = self.file.read(framelength)
                self.frameNum += 1

        return data
```

```
        if self.clientInfo['event'].isSet():
            break
        if self.request == self.NEXT:
            data = self.clientInfo['videoStream'].skipFrame(self.totalFrame, self.skipNo*75)
            self.skipNo = 0
            self.request = ''
        elif self.request == self.BACK:
            data = self.clientInfo['videoStream'].backFrame(self.skipNo*75)
            self.skipNo = 0
            self.request = ''
        else:
```

2.8 Packet Loss Rate và Data Rate

Packet Loss Rate được tính toán dựa trên số Packet đến trễ trên tổng số các Packet đã tới và Data Rate được tính dựa trên tổng số byte được gửi đến trên một đơn vị thời gian.

```
    def computePercentLoss(self,nbLostAndRecv,nbLost):
        if nbLostAndRecv ==0:
            return 0
        else:
            return (100*nbLost)/nbLostAndRecv

    def computeRateKBs(self,nBytes,intervalMs):
        if intervalMs==0:
            return 0
        else:
            return nBytes/intervalMs

    def time_different(self,dt2, dt1):
        timedelta = dt2 - dt1
        return timedelta.days * 24 * 3600 + timedelta.seconds
```

2.9 Sử dụng ứng dụng không cần nhấn SETUP

Trong các ứng dụng videostreaming thực thế như RealPlayer hoặc Windows Media Player, chúng ta chỉ thấy 3 nút nhấn cho các hoạt động là: PLAY, PAUSE, STOP (tương tự với nút nhấn TEARDOWN). Vì thế để cải thiện điều ấy cho ứng dụng nút SETUP có thể được bỏ đi và thay vào đó được gọi trực tiếp trong lúc nhấn nút PLAY để thực hiện 2 tác vụ cùng một lúc. Chúng ta có thể gọi SETUP trong lúc tạo GUI cho ứng dụng để khi PLAY thì có thể chạy được video.

```
# 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
    self.numLatePacket=0
    print("Initial the movie.")
    self.setupMovie()
    self.intervalTime = 0 # ms
    self.statTotal_bytes = 0
```

3 Danh sách các component

3.1 Client

- CreateWidgets: Tạo các button, thiết lập lệnh khi nhấn vào button, tạo không gian trình chiếu video.
- UpdateSlider: Cập nhật ví trí của thanh Slider trên ứng dụng theo thời gian video.
- NextMovie: Kiểm tra trạng thái và gửi request NEXT
- BackMovie: Kiểm tra trạng thái và gửi request BACK
- SetupMovie: Kiểm tra trạng thái và gửi request SETUP lên server.
- ExitClient: Gửi request TEARDOWN, đóng cửa sổ trình chiếu và xóa dữ liệu cache.
- PauseMovie: Kiểm tra trạng thái và gửi request PAUSE.
- PlayMovie: Kiểm tra trạng thái, chờ gói tin RTP, tạo Event object và đồng bộ, gửi request PLAY.
- DescribeMovie: Gửi request DESCRIBE
- ListenRtp: Lắng nghe gói tin từ server, nhận và tiền hành giải mã (decode) dữ liệu xuất ra từ server, frame number và kiểm tra với frame number hiện hành để update frame, đợi PAUSE hoặc TEARDOWN.
- WriteFrame: Tiến hành ghi khung hình nhận được dưới dạng tệp ảnh.

- UpdateMovie: Load hình ành thành khung hình trong video, dùng khi cần đồng bộ hóa khung hình.
- ConnectToServer: Kết nối và tạo mới RTSP/TCP session, hiện lỗi nếu kết nối thất bại.
- sendRtspRequest: Cập nhật sequence và gửi RTSP request.
- RecvRtspReply: Đọc gói tin nhận được từ server.
- ParseRtspReply: Xử lý gói tin nhận được, tiến hành cắt chuỗi để nhận các thông số.
- OpenRtpPort: Mở socket với RTP port tương ứng, cài đặt time out.
- Handler: Thiết lập việc đóng chương trình.
- ComputePercentLoss: Tính toán tỉ lệ mất gói tin.
- ComputeRateKBs: Tính toán tốc độ dữ liệu trên giây.
- Time_different: Tính toán khoảng thời gian truyền dữ liệu.
- Display_description: Hiển thị thông tin của Extend DESCRIBE.

3.2 RtpPacket

- Encode: Mã hóa gói RTP như mô tả.
- Decode: Giải mã gói RTP.
- Version: Trả về version của RTP.
- SeqNum: Trả về số thứ tự của frame.
- Timestamp: Trả về timestamp của gói RTP.
- PayloadType: Trả về loại của payload trong gói RTP.
- GetPayload: Trả về Payload của gói RTP.
- GetPacket: Trả về gói RTP đã được xử lý.
- GetPayload_length: Trả về độ dài của Payload của gói RTP.

3.3 Server

- Main: Mở cổng port server, nhận thông tin qua RTSP/TCP.

3.4 ServerWorker

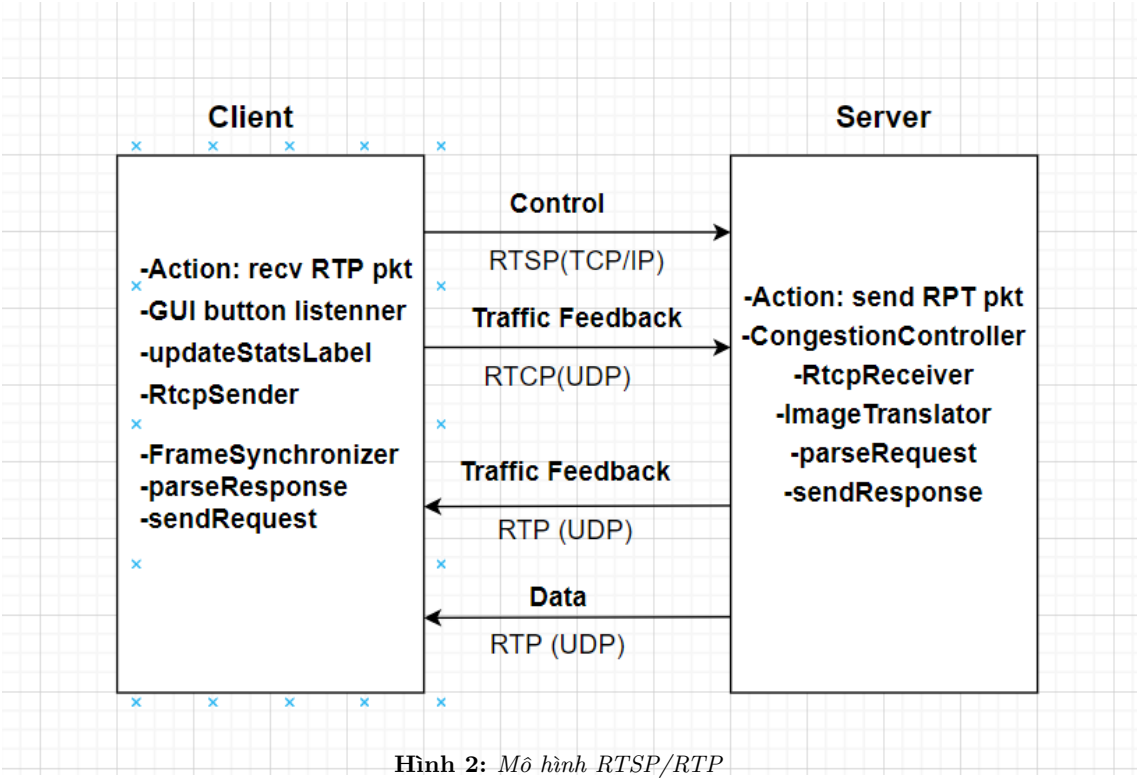
- Run: Bắt đầu nhận các yêu cầu từ Client.
- RecvRtspRequest: Nhận yêu cầu RTSP từ client.
- ProcessRtspRequest: nhận số thứ tự, lấy tên tệp phương tiện và thiết lập các giao thức.
- SendRtp: Gửi các gói tin thông qua UDP.
- MakeRtp: Tạo các gói dữ liệu của video thành các gói RTP.
- ReplyRtsp: Phản hồi những yêu cầu của client.
- ReplyRtspMsg: Gửi thông tin của gói tin cho client thoogn qua UDP.

3.5 VideoStream

- TotalFrame: Trả về số frame sẽ gửi.
- NextFrame: Lấy data của frame kế tiếp để đưa vào gói dữ liệu RTP và tăng giá trị frame number.
- SkipFrame: Lấy data của frame khi nhấn nút FASTFORWARD để đưa vào gói dữ liệu RTP và tăng giá trị frame number.
- BackFrame: Lấy data của frame khi nhấn nút BACKWARD để đưa vào gói dữ liệu RTP và tăng giá trị frame number.
- Frame nbr: Trả về số Frame.

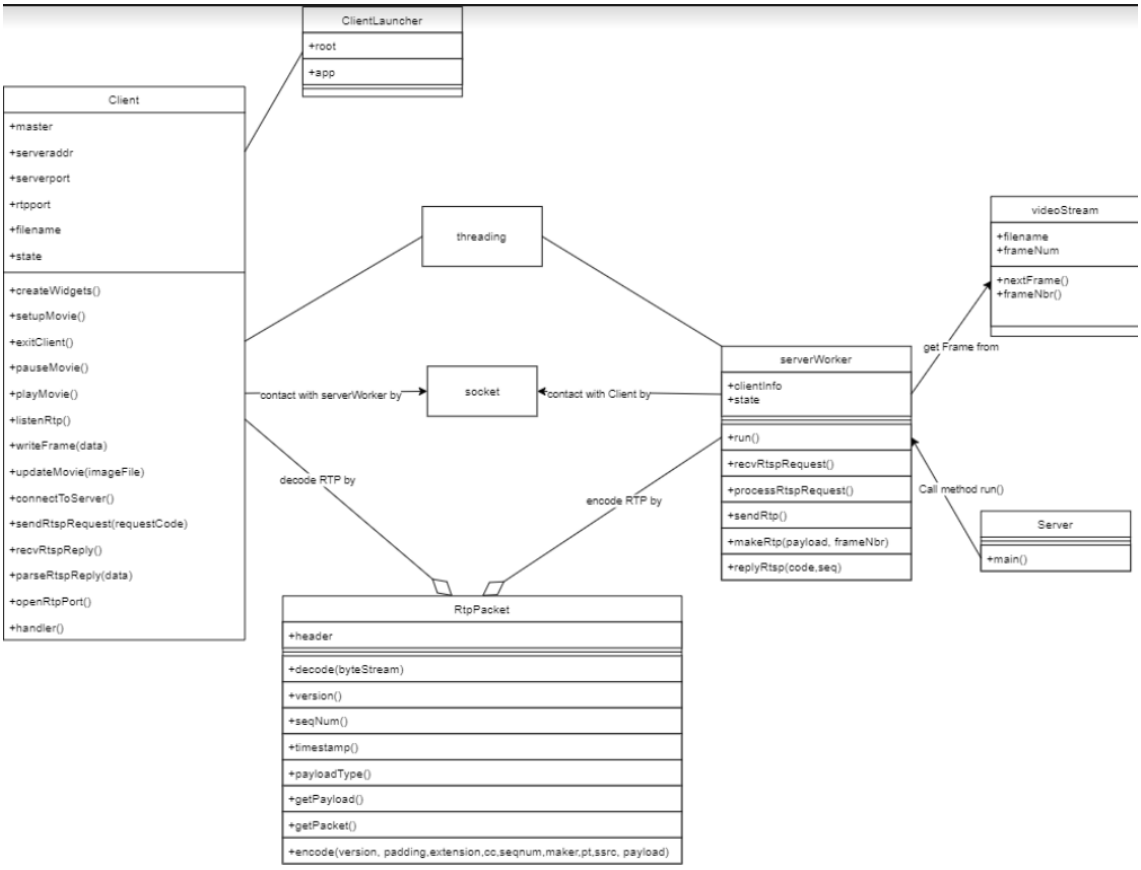
4 Mô hình dòng dữ liệu

Mô hình RTSP/RTP:



Hình 2: *Mô hình RTSP/RTP*

5 Class diagram



6 Đánh giá kết quả

Sau khi hiện thực xây dựng ứng dụng thì thu được kết quả như sau:

Giao diện người dùng gồm các nút nhấn SETUP, PLAY, PAUSE, TEARDOWN, DESCRIBE, BACKWARD, FASTFORWARD.

Khi người dùng nhấn các nút nhấn thì các message request và reply được gửi từ máy khách và máy chủ được hiển thị bên máy khách khi nhấn SETUP, PLAY, PAUSE, DESCRIBE, BACKWARD, FASTFORWARD, máy chủ sẽ trả về RTSP/1.0 200-OK nếu yêu cầu thành công. Đồng thời sẽ thực hiện những tác vụ tương ứng với các nút nhấn.

Cuối cùng khi người dùng nhấn TEARDOWN thì phiên kết thúc, tắt cửa sổ giao diện, trả message RTSP/1.0 200-OK về cho người dùng.

7 Cách chạy ứng dụng

Mở thư mục chứa Source code. Để chạy Server, mở terminal và chạy với câu lệnh

python Server.py <Server-port>

Trong đó server-port là cổng RTSP mà máy chủ nhận các kết nối RTSP đến. Cổng RTSP tiêu chuẩn là 554 nhưng cần phải chọn số cổng lớn hơn 1024.

Ví dụ:**python Server.py 1025**

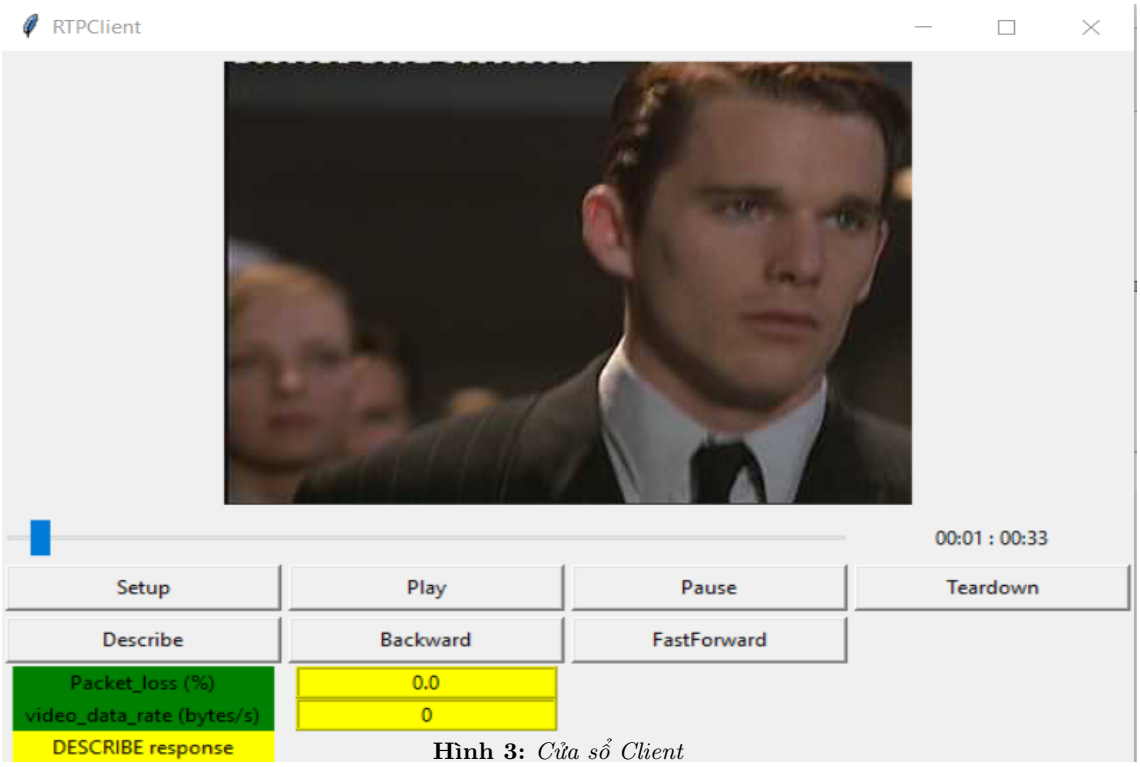
Sau đó, khởi động máy khách bằng lệnh:

python ClientLauncher.py <Server-host><Server-port><RTP-port><Video-file>

Trong đó:

- Server-host là tên của máy chủ đang chạy hoặc là ip của máy chủ
- Server-port là cổng mà nơi máy chủ đang nhận các kết nối RTSP.
- RTP-port là cổng nơi nhận các RTP packet.
- Video-file là tên của tệp video bạn muốn yêu cầu phát.

Ví dụ: **python ClientLauncher.py 127.0.0.1 1025 5008 video.Mjpeg**
Máy chủ mở một kết nối với máy khách và bật lên một cửa sổ như sau:



Hình 3: *Cửa sổ Client*

Bạn có thể gửi lệnh RTSP đến máy chủ bằng cách nhấn các nút trên giao diện. Một tương tác RTSP bình thường được diễn ra như sau:

- Step 1: Máy khách gửi SETUP. Lệnh này được sử dụng để phát lập phiên và các tham số truyền tải.
- Step 2: Máy khách gửi PLAY. Lệnh này dùng để bắt đầu phát video.
- Step 3: Máy khách có thể gửi BACKWARD hoặc FASTWARD để tua video lên xuống.
- Step 4: Máy khách gửi PAUSE nếu muốn tạm dừng video.
- Step 5: Máy khách gửi yêu cầu DESCRIBE nếu muốn xem thông tin video
- Step 6: Máy khách gửi TEARDOWN. Lệnh này có tác dụng kết thúc phiên và đóng kết nối với máy chủ, đồng thời cửa sổ giao diện cũng đóng.

8 Source Code

https://github.com/hungnguyenquocc/ComputerNetworking