ĐẠI HỌC QUỐC GIA TP. HCM

TRƯỜNG ĐẠI HỌC BÁCH KHOA

BÀI TẬP LỚN MÔN HỌC

MÔN MẠNG MÁY TÍNH

ĐỀ TÀI

VIDEO STREAMING

LỚP L20--- NHÓM 06 --- HK 211

NGÀY NỘP 08/11/2021

Giảng viên hướng dẫn: Nguyễn Quang Sang

Sinh viên thực hiện	Mã số sinh viên	Điểm số
Bùi Xuân Hùng	1913600	
Nguyễn Quốc Trọng	1915676	
Trương Phi Trường	1915749	
Quách Vũ Giang Nam	1914251	

Thành phố Hồ Chí Minh – 2021

Mục lục

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 cuôi. RTSP chạy trên giao thước 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 tôi đa là 0.5 giây.
- Cần cài đặt 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.

2. Mô tả chức năng

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

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 gửi yêu cầu SETUP đến máy chủ, chèn tiêu đề truyền tải mà mình chỉ định cổng cho Socket datagram RTP mà mình vừa tạo. Khách hàng sẽ nhận được phản hồi của máy chủ và nhận được ID của phiên RTSP.

```
# 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]
```

Gói RTSP SETUP sẽ bao gồm:

- Lệnh SETUP
- Tên của video sẽ phát
- Số thứ tự của 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 số phiên cụ thể một cách ngẫu nhiên.
- Nếu có vấn đề xảy ra với lệnh hoặc có lỗi trong máy chủ thì máy chủ 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.

Phía máy khách sẽ lặp lại để nhận RTSP Reply của máy chủ:

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 lệnh PLAY được gửi từ khách hàng tới máy chủ:

Máy chủ sẽ tạo ra một Socket để truyền RTP qua UDP và bắt đầu một bước gửi packet video:

Sau đó file VideoStream.py có chức năng cắt video thành từng khung riêng biệt và đưa từng khung vào packet data RTP:

```
        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)

                #Reply with OK_200
                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()
```

```
    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
- Padding
- Extension
- Contributing source
- Marker
- Type
- Sequence number
- Timestamp
- SSRC

RTP packet header

Bit offset[b]	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 ...						

Hình 1: *RTP packet header*

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] = version + padding + extension + cc + seqnum + marker + pt + ssrc
self.header[0] = version << 6
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

self.header[2] = seqnum >> 8
self.header[3] = seqnum

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

self.header[8] = (ssrc >> 24) & 0xFF
self.header[9] = (ssrc >> 16) & 0xFF
self.header[10] = (ssrc >> 8) & 0xFF
self.header[11] = ssrc & 0xFF
```

Sau đó RTP packet sẽ bao gồm một khung video và một header được gửi đến cổng RTP ở phía máy 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

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

Sau đó máy khách sẽ giải mã RTP packet để lấy header và khung video, tổ chức lại các khung và hiển thị trên giao diện người dùng:

```
def listenRtp(self):
    """Listen for RTP packets."""
    while True:
        try:
            data = self.rtpSocket.recv(20480)
            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()))
        except:
            # Stop listening upon requesting PAUSE or TEARDOWN
            if self.playEvent.isSet():
                break

            # Upon receiving ACK for TEARDOWN request
```

2.3. Pause

Nếu một lệnh PAUSE được gửi từ máy khách đến máy chủ thì nó sẽ ngăn các máy chủ không gửi khung video đến máy khách nữa:

```
# 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 = "PAUSE " + str(self.fileName) + " RTSP/1.0\nCSeq: " + str(self.rtspSeq) + "\nSession: " + str(self.sessionId)
    self.rtspSocket.send(request.encode("utf-8"))
    # Keep track of the sent request.
    # self.requestSent = ...
    self.requestSent = self.PAUSE
```

Và máy khách sẽ được đặt lại STATE của nó là READY và đợi lệnh tiếp theo từ máy khách:

```
# 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])
```

2.4. Teardown

Nếu một lệnh TEARDOWN được gửi đến từ máy khách đến máy chủ thì nó 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:

```
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 = "TEARDOWN " + str(self.fileName) + " RTSP/1.0\nCSeq: " + str(self.rtspSeq) + "\nSession: " + str(self.sessionId)
    self.rtspSocket.send(request.encode("utf-8"))
    # Keep track of the sent request.
    # self.requestSent = ...
    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()
```

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.
- 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.
- listenRtp: chờ 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.

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 gói RTP đã được xử lý.

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.
- setSpeed: thiết lập tốc độ truyền gói tin.

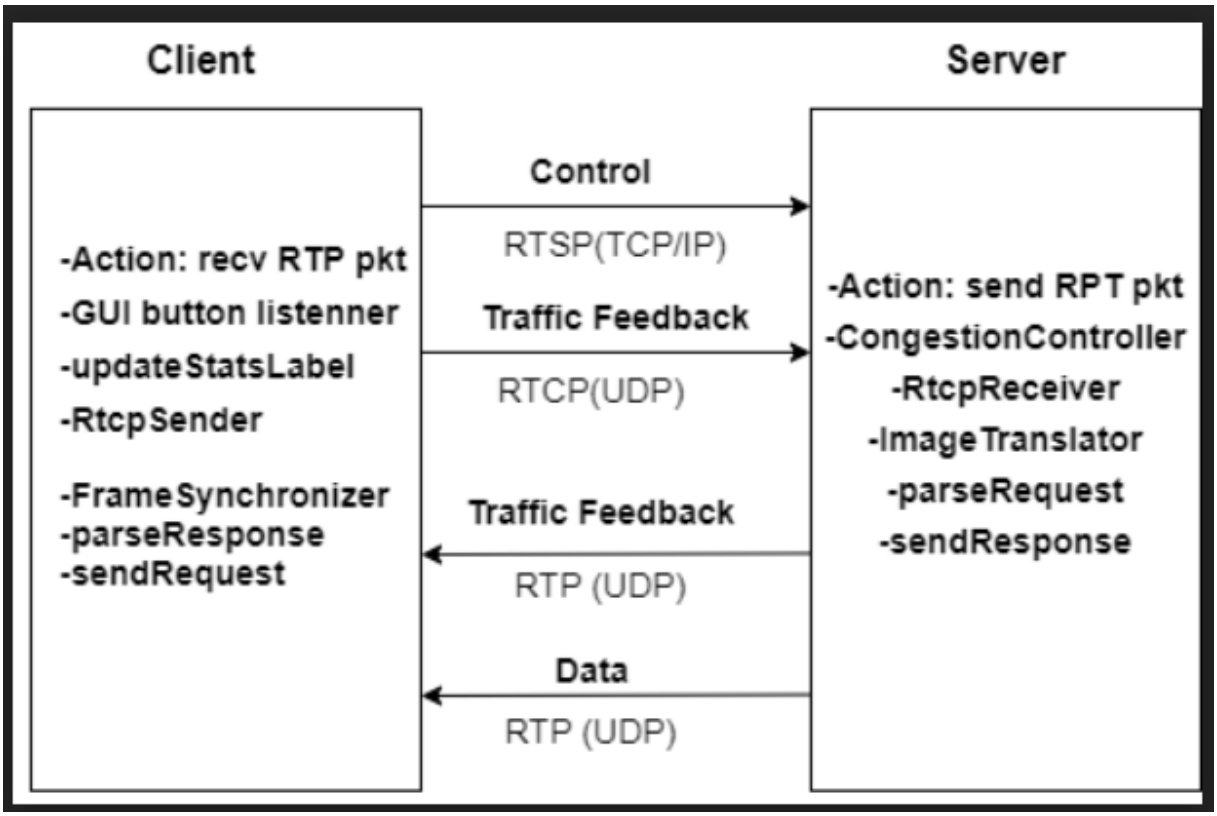
3.5. videoStream

- nextFrame: lấy dât của frame kế tiếp để đưa vào gói dữ liệu RTP và tăng giá trị frame number.
- Frame nbr: trả về số Frame.

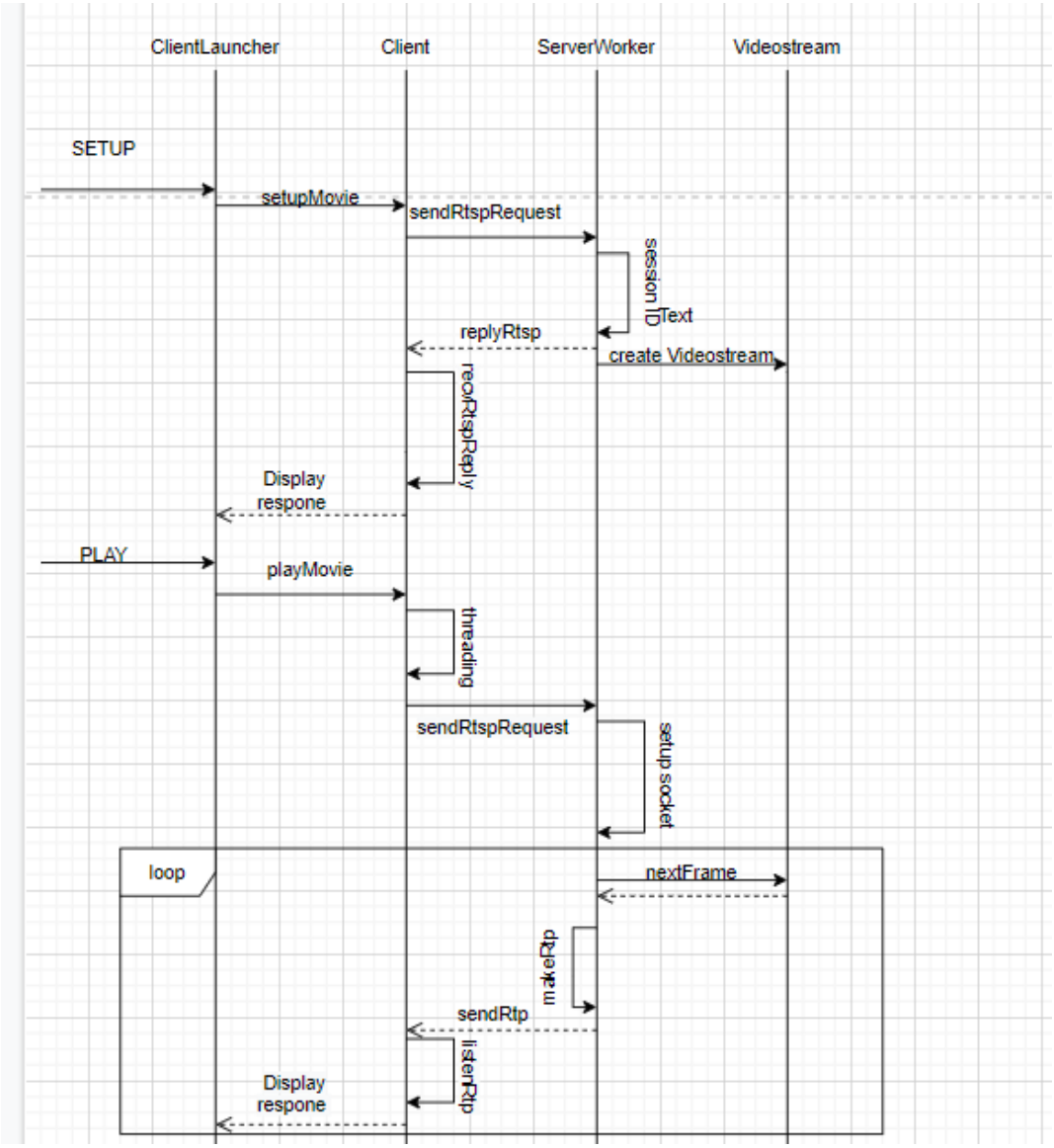
4. Mô hình và dòng dữ liệu

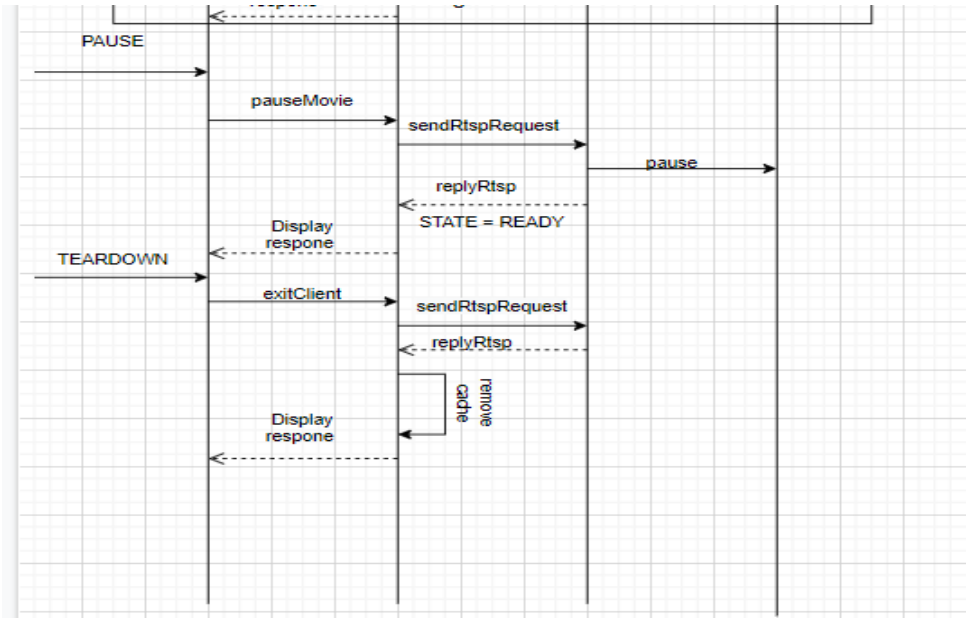
Mô hình RTSP/RTP:

- Link drive: model.drawio - Google Drive

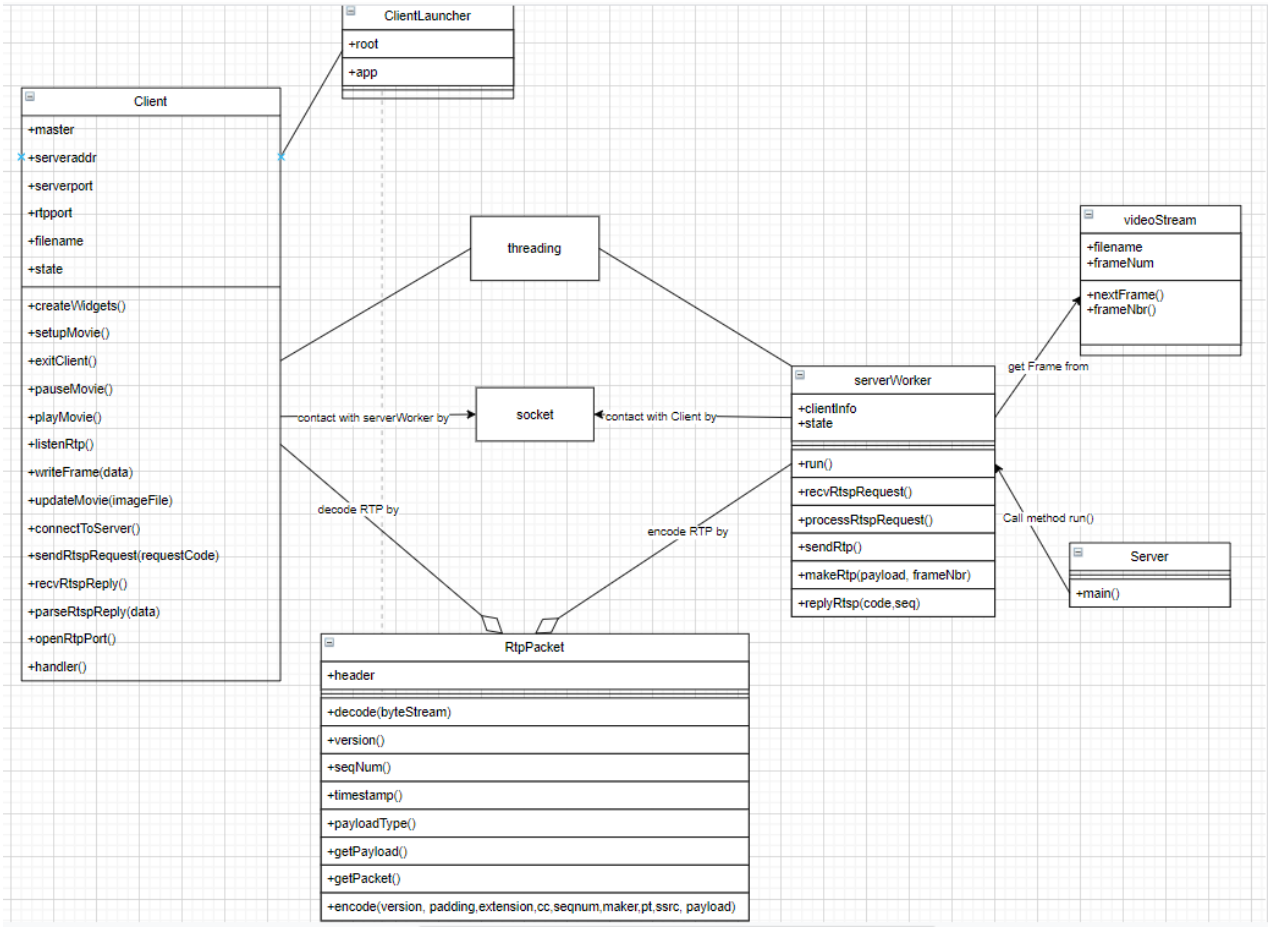


Dòng dữ liệu: - Link drive: [data_flow.drawio - Google Drive]



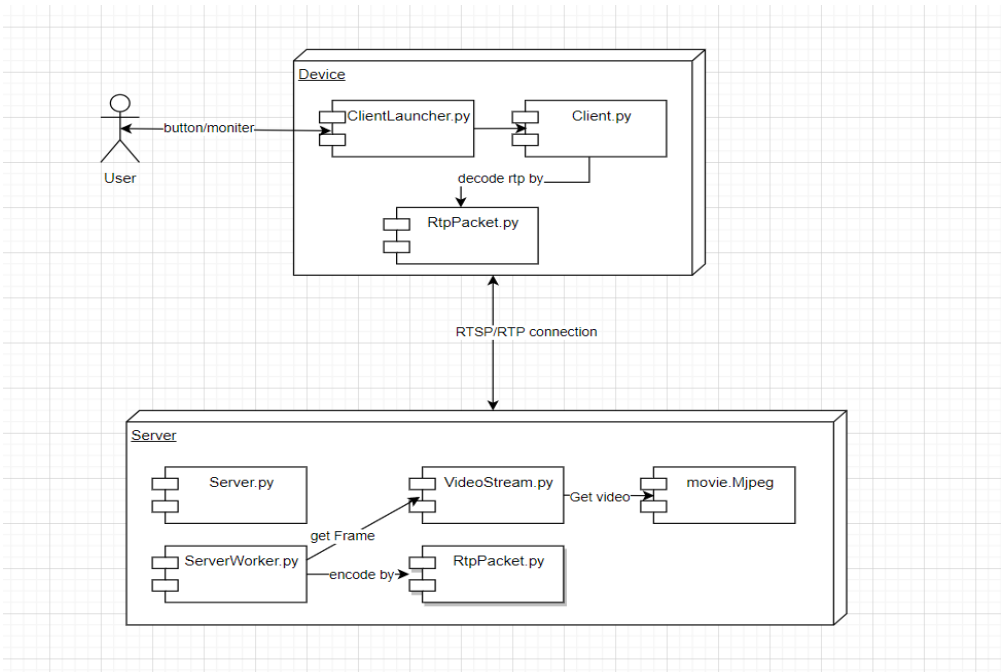


5. Class diagram



Link drive: [class_diagram.drawio - Google Drive]

6. Implementation



7. Đánh giá kết quả

Sau khi thực hiện xong các Class và chạy ứng dụng thì kết quả thu được như sau:

Giao diện người dùng RTPClient khi nhấn SETUP và PLAY.

Các message request và reply được gửi từ máy khách và máy chủ được hiển thị bên máy khách khi nhấn SETUP và PLAY, máy chủ sẽ trả về RTSP/1.0 200-OK nếu yêu cầu thành công.

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

8. Hướng dẫn sử dụng

Đầu tiên mở thư mục chứa source code. Để chạy server, mở Command line 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 8000**

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 10.0.175.20 8000 8000 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 gửi PAUSE nếu muốn tạm dừng video.
- Step 4: 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.

9. Thêm chức năng cho video streaming

9.1. Số liệu thống kê của phiên

Tính toán số liệu thống kê về phiên, thực hiện tính tỷ lệ mất RTP packet và tốc độ truyền dữ liệu của video(được tính theo byte trên giây).

```
if self.sumOfTime>0:
    rateData = float(int(self.sumData)/int(self.sumOfTime))
    print('-'*40 + "\nVideo Data Rate: " + str(rateData) +"\n" + '-'*40)
    rateLoss = float(self.counter/self.frameNbr)
    print('-'*40 + "\nRTP Packet Loss Rate: " + str(rateLoss) +"\n" + '-'*40)
```

Kết quả tính toán được hiển thị sau khi kết thúc phiên và hiển thị trên máy khách:

```
Data sent:
TEARDOWN movie.Mjpeg RTSP/1.0
CSeq: 3
Session: 509040
----------------------------------------
Data received:
RTSP/1.0 200 OK
CSeq: 3
Session: 509040
----------------------------------------
Video Data Rate: 18686.056603773584
----------------------------------------
----------------------------------------
RTP Packet Loss Rate: 0.0
----------------------------------------
PS D:\HK211\ComputerNetwork\Ass\Source_Ass1\video_stream\Extend1>
```

9.2. Loại bỏ phím SETUP

Giao diện người dùng trên RTPClient có 4 nút như đã thực hiện phía trên nhưng tiêu chuẩn của các media player chẳng hạn như RealPlayer hoặc Windows Media Player chỉ có 3 nút là: PLAY,PAUSE, và TEARDOWN, không có nút SETUP.

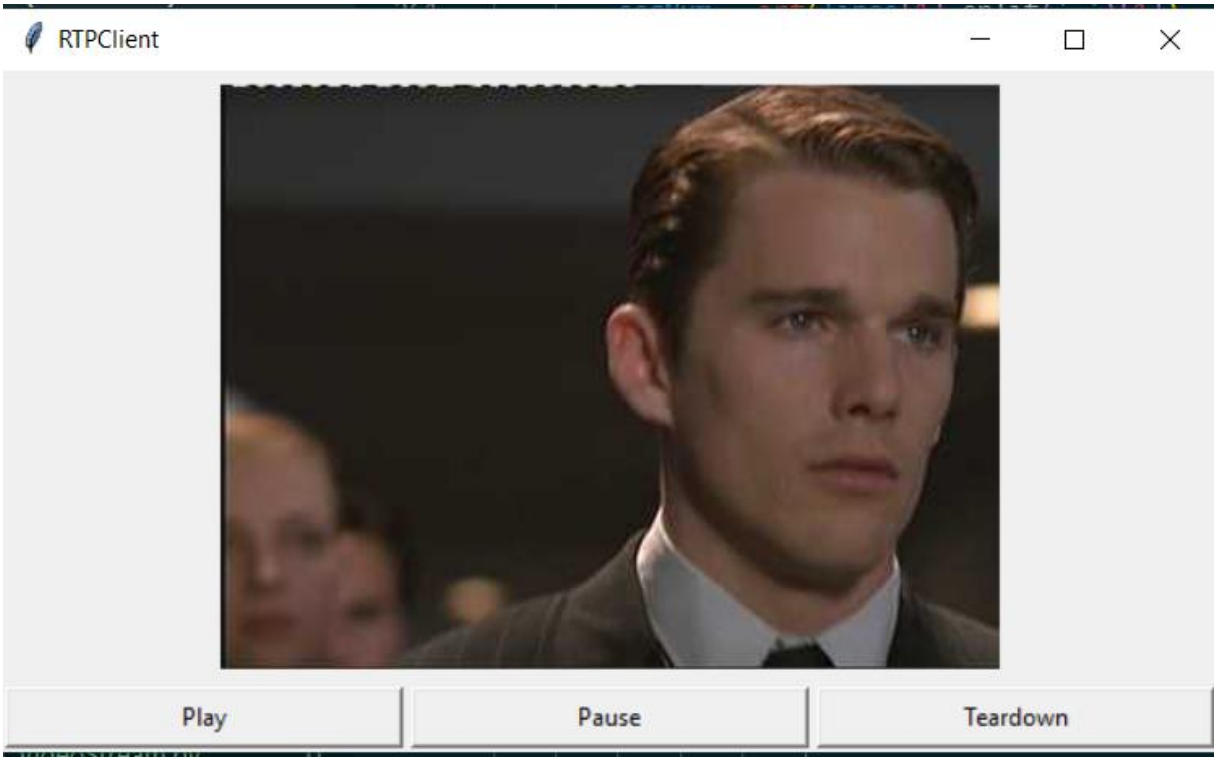
Nhiệm vụ của phần này là hiện thực giao diện RTPClient giống như media player. Giải pháp đưa ra là khi người dung nhấn PLAY thực hiện liên tiếp hai chức năng SETUP và PLAY.

```python
def playMovie(self):
    """Play button handler."""
    if self.state == self.INIT:
        self.sendRtspRequest(self.SETUP)
        return
    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.start = time.time()
        self.sendRtspRequest(self.PLAY)
```

```
            # 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()
                        self.playMovie()
```

Kết quả thu được:



9.3. Thêm chức năng DESCRIBE

Ngoài những tương tác RTSP và PAUSE, DESCRIBE được sử dụng để truyền những thông tin về video stream. Khi server nhận được DESCRIBE request, nó sẽ gửi lại một file mô tả session - nói cho client loại stream trong session và encoding được sử dụng là gì. Cụ thể, khi server gửi phản hồi cho client thì có những thống kê về:

- RTSP port của server

- Video encoding của video được truyền qua RTP

- RTSP ID của session đã được thiết lập

```
def describe(self):
    seq1 = "v=0\nm=video " + str(self.clientInfo['rtpPort']) + " RTP/AVP 26\na=control:streamid=" \
        + str(self.clientInfo['session']) +"\na=mimetype:string;\"video/Mjpeg\"\n"
    seq2 = "Content-Base: " + str(self.clientInfo['videoStream'].filename) + "\nContent-Length: " \
        + str(len(seq1)) + "\n"
    return seq2 + seq1

def replyDescribe(self,code,seq):
    des = self.describe()
    if code == self.OK_200:
        reply = "RTSP/1.0 200 OK\nCSeq: " + seq + "\nSession: " + str(self.clientInfo['session']) + "\n" + des
        connSocket = self.clientInfo['rtspSocket'][0]
        connSocket.send(reply.encode())
    elif code == self.FILE_NOT_FOUND_404:
        print("404 NOT FOUND")
    elif code == self.CON_ERR_500:
        print("500 CONNECTION ERROR")
```

Kết quả thu được:

```
Data sent:
DESCRIBE movie.Mjpeg RTSP/1.0
CSeq: 5
Sesssion: 477433
----------------------------------------
Data received:
RTSP/1.0 200 OK
CSeq: 5
Session: 477433
Content-Base: movie.Mjpeg
Content-Length: 86
v=0
m=video 8000 RTP/AVP 26
a=control:streamid=477433
a=mimetype:string;"video/Mjpeg"
```

10. Source code

[bxhung2410/computer-network-assignment (github.com)]