

ĐẠI HỌC QUỐC GIA VIỆT NAM, THÀNH PHỐ HỒ CHÍ MINH
ĐẠI HỌC BÁCH KHOA
KHOA KHOA HỌC VÀ KỸ THUẬT MÁY TÍNH



MẠNG MÁY TÍNH (CO3093)

Bài tập lớn

Video Streaming

GVHD: Nguyễn Quang Sang
Nhóm SV thực hiện: Nguyễn Khoa Gia Cát - 1912749
Trần Nguyễn Hữu Thọ - 1915347

TP. HỒ CHÍ MINH, 11/2021



Contents

1	Thành viên và nhiệm vụ	2
2	Kiến thức chuẩn bị	3
2.1	Giao thức RTSP	3
2.2	Giao thức RTP	3
3	Phân tích yêu cầu	3
3.1	Yêu cầu chức năng	3
3.2	Yêu cầu phi chức năng	3
4	Mô tả các chức năng của ứng dụng	4
4.1	Setup	4
4.2	Play	5
4.3	Pause	5
4.4	Teardown	5
5	Danh sách các thành phần	6
5.1	Client	6
5.2	RtpPacket	6
5.3	Server	6
5.4	ServerWorker	7
5.5	VideoStream	7
6	Model và data flow	8
6.1	Model	8
6.2	Data flow	8
7	Class diagram	9
8	Hiện thực ứng dụng	10
8.1	Hiện thực giao thức RTSP phía Client	10
8.1.1	SETUP	11
8.1.2	PLAY	12
8.1.3	PAUSE	13
8.1.4	TEARDOWN	14
8.1.5	Nhận gói tin từ server và phân tích	15
8.1.6	Nhận RTP packet từ server và phát video	17
8.2	Hiện thực giao thức RTP ở Client	19
9	Kết quả thu được	20
10	Hướng dẫn sử dụng	23
11	Mở rộng	25
11.1	Thống kê về phiên	25
11.2	Lược bỏ SETUP	27
11.3	Thêm chức năng DESCRIBE	29
12	Source code	32



1 Thành viên và nhiệm vụ

STT	MSSV	Thành viên	Lớp	Nhiệm vụ	Đóng góp
1	1912749	Nguyễn Khoa Gia Cát			
2	1915347	Trần Nguyễn Hữu Thọ			

2 Kiến thức chuẩn bị

2.1 Giao thức RTSP

- RTSP là một giao thức truyền tin thời gian thực (viết tắt từ tiếng Anh - Real Time Streaming Protocol) được sử dụng để kiểm soát các máy chủ phát trực tiếp sử dụng trong các hệ thống giải trí và truyền thông. Máy chủ RTSP nằm giữa luồng trực tiếp và người xem, đưa ra các lệnh “phát”, “tạm dừng” và “tua lại”. Khi RTSP kiểm soát kết nối máy chủ đến máy khách, các luồng video sẽ được truyền theo yêu cầu sử dụng.

2.2 Giao thức RTP

- 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.

3 Phân tích yêu cầu

3.1 Yêu cầu chức năng

- Hiện thực giao thức RTSP (Real Time Streaming Protocol) (File Client.py)
- Hiện thực giao thức RTP (Real-time Transport Protocol) (File ClientLauncher.py)

3.2 Yêu cầu phi chức năng

- Máy chủ luôn trả lời tất cả các tin nhắn mà khách hàng gửi. Mã 200 có nghĩa là yêu cầu đã thành công trong khi mã 404 và 500 đại diện cho lỗi FILE-NOT-FOUND và lỗi kết nối tương ứng.
- Socket datagram để nhận dữ liệu RTP và thời gian chờ trên Socket tối đa là 0.5 giây.
- Khi gửi khung video cho máy khách thì một khung được gửi sau 50 mili giây.
- Các video được định dạng là MJPEG (Motion JPEG)
- Server tiếp nhận thông tin client thông qua RTSP/TCP session

4 Mô tả các chức năng của ứng dụng

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:

- Setup
- Play
- Pause
- Teardown

các câu lệnh sẽ gửi đến máy chủ để hoàn thành nhiệm vụ và thông qua giao thức RTSP gửi đến người dùng các số để người dùng biết phản hồi của máy chủ:

- 200 - kết nối thành công
- 404 - FILE-NOT-FOUND
- 500 - báo lỗi ERROR

4.1 Setup

khi gửi yêu cầu SETUP, chèn tiêu đề truyền tải cổng cho Socket datagram RTP mà mình vừa tạo, người dùng sẽ nhận được phản hồi và ID của phiên RTSP. RTSP SETUP bao gồm:

- Lệnh SETUP
- tên video
- số thứ tự gói RPST
- loại giao thức RTSP/1.0 RTP
- Giao thức UDP
- RTP port để truyền video

khi nhận được lệnh SETUP thì máy chủ sẽ thực hiện các bước:

- Gán cho người dùng một số phiên ngẫu nhiên
- nếu xử lý lỗi trả về ERROR
- nếu xử lý không lỗi sẽ trả về OK, đặt trạng thái READY. Máy chủ sẽ mở tệp video được chỉ định trong SETUP Packet và khởi tạo số frame = 0.

Phía máy người dùng sẽ lặp lại để nhận RTSP Reply từ máy chủ. Nếu gói RTSP Packet được phản hồi cho lệnh SETUP thì máy người dùng sẽ đặt STATE của nó là READY. Sau đó mở một RTP Port để nhận luồng video.

4.2 Play

Khi lệnh PLAY được gọi từ người dùng 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. Mỗi packet data sẽ được mã hóa với một header bao gồm:

- RTP-version
- Padding
- Extension
- Contributing source
- Marker
- Type
- Sequence number
- Timestamp
- SSRC

Chúng sẽ được chèn vào RTP packet thông qua thao tác bitwise, RTP packet sẽ gồm một khung video và một header được gửi đến cổng RTP của người dùng.

4.3 Pause

nếu gửi lệnh Pause thì máy chủ sẽ không gửi các khung video đến người dùng. máy người dùng sẽ đặt lại STATE là READ và đợi lệnh tiếp theo.

4.4 Teardown

lệnh Teardown sẽ ngăn máy chủ gửi các khung hình video đến người dùng và đóng thiết bị đầu cuối của người dùng. STATE được chuyển về trạng thái INIT.

5 Danh sách các thành phần

5.1 Client

- Create widgets: tạo các button, thiết lập lệnh khi nhấn vào button, tạo không gian trình chiếu video
- Setup movie: kiểm tra trạng thái và gửi request SETUP lên server
- Exit client: gửi request TEARDOWN, đóng cửa sổ trình chiếu và xóa dữ liệu cache
- Pause movie: kiểm tra trạng thái và gửi request PAUSE
- Play movie: kiểm tra trạng thái, chờ các gói tin RTP, tạo Event object và đồng bộ, gửi request PLAY
- Listen Rtp: chờ gói tin từ server, nhận và tiến hành decode dữ liệu, xuất ra từ server, frame number và kiểm tra với frame number hiện hành để update, đợi PAUSE hay TEARDOWN
- Write frame: tiến hành ghi khung hình nhận được dưới dạng tệp ảnh
- Update movie: Load hình ảnh thành khung hình trong video dùng khi cần đồng bộ hóa khung hình
- Connect to server: kết nối và tạo mới RTSP/TCP session, hiện cảnh báo nếu kết nối thất bại
- Send rtsp request: cập nhật sequence và gửi RTSP request
- Recv rtsp reply: đọc gói tin nhận được từ server
- Parse rtsp reply: xử lý gói tin nhận được, tiến hành cắt chuỗi để nhận các thông số
- Open rtp port: mở socket với RTP port tương ứng, cài đặt time out
- Handler: thiết lập việc đóng chương trình

5.2 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
- Payload type: trả về loại của payload trong gói RTP
- Get payload: Trả về gói RTP đã được xử lý

5.3 Server

- Main: kết nối với server, nhận thông tin qua RTSP/TCP



5.4 ServerWorker

- Run: Bắt đầu nhận các yêu cầu từ client.
- Recv rtsp request: Nhận yêu cầu RISP từ client.
- Process rtsp request: nhận số thứ tự, lấy tên tệp phương tiện và thiết lập các giao thức
- Send rtp: Gửi các gói tin thông qua UDP.
- Make rtp: Tạo các gói dữ liệu của video thành các gói RTP
- Reply rtsp: Phản hồi những yêu cầu của client.
- Reply rtsp msg: Gửi thông tin của gói tin cho client thông qua UDP.
- Set speed: Thiết lập tốc độ truyền gói tin.

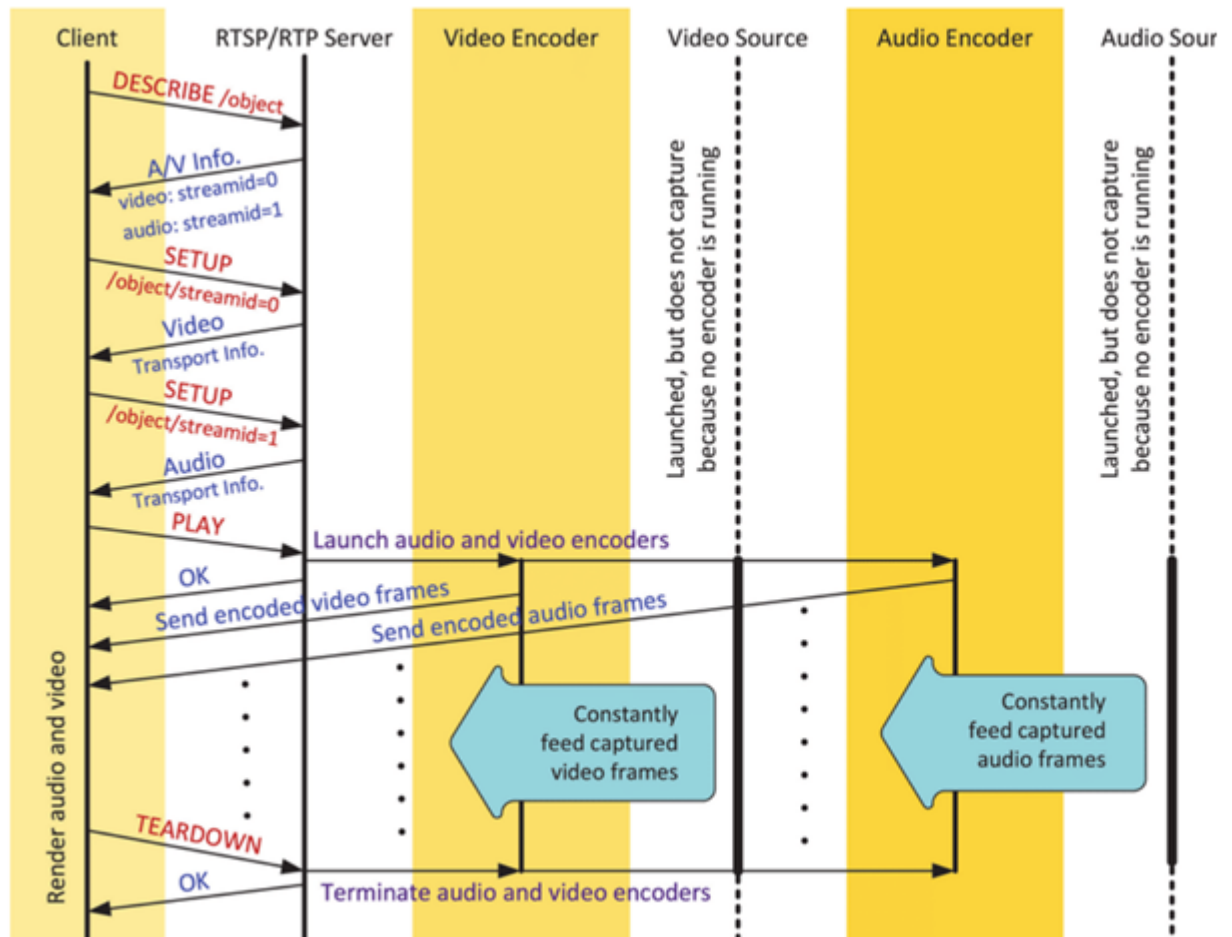
5.5 VideoStream

- Next frame: Lấy data của frame kế tiếp để đưa vào gói dữ liệu Rtp và tăng giá trị frameNbr
- Frame nbr: Trả về số frame

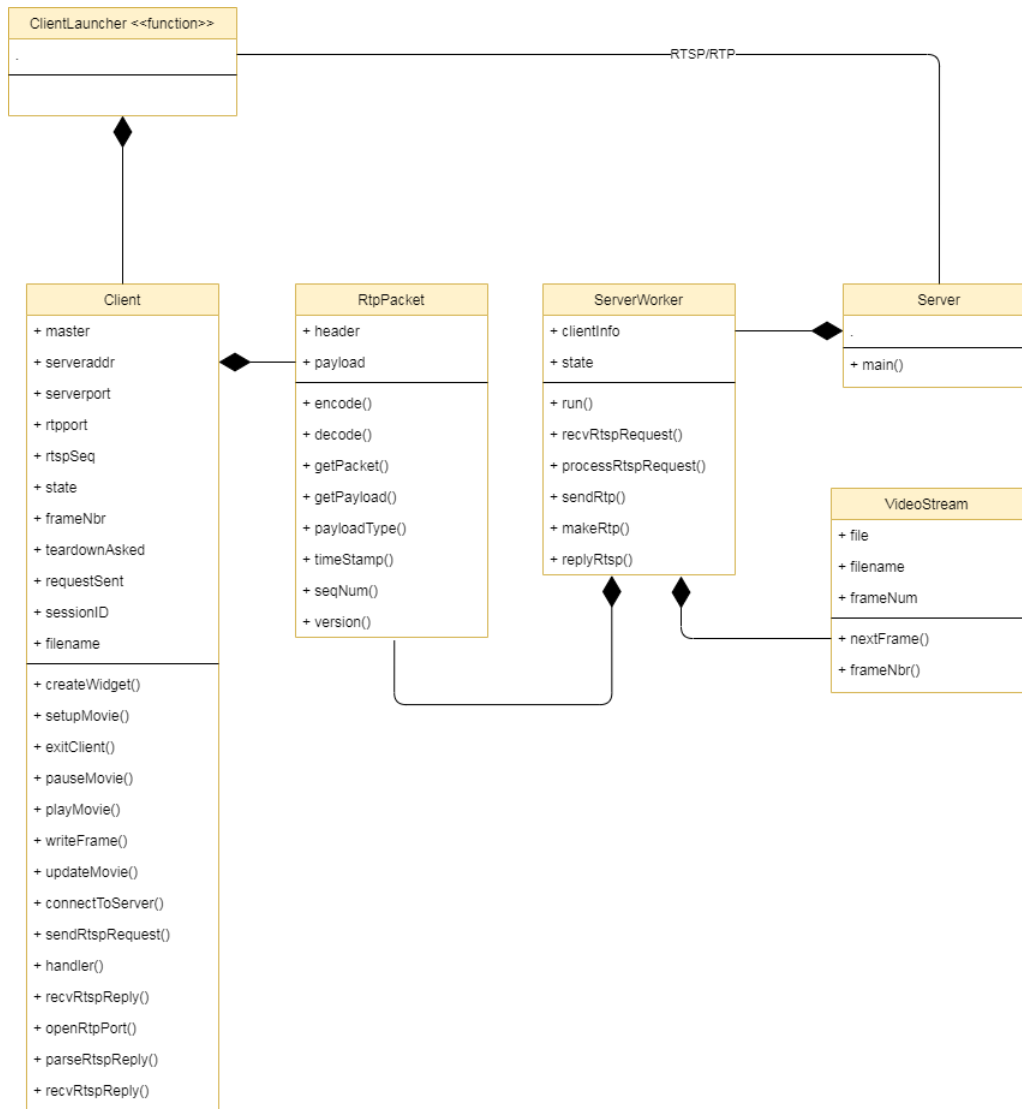
6 Model và data flow

6.1 Model

6.2 Data flow



7 Class diagram



8 Hiện thực ứng dụng

8.1 Hiện thực giao thức RTSP phía Client

- Hiện thực bốn chức năng được gọi khi người dùng nhấn vào 4 nút: "SETUP", "PLAY", "PAUSE", "TEARDOWN"
- Khi Client được khởi tạo thì nó sẽ kết nối đến RTSP port của server

```
def __init__(self, master, serveraddr, serverport, rtpport, filename):  
    ...  
    self.connectToServer()  
    ...  
    self.rtpSocket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)  
    ...  
  
def connectToServer(self):  
    """Connect to the Server. Start a new RTSP/TCP session."""  
    #TODO  
    self.rtspSocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)  
    try:  
        server_address = (self.serverAddr, self.serverPort)  
        self.rtspSocket.connect(server_address)  
    except:  
        tkinter.messagebox.showwarning('Connection Failed', 'Connection to \'%s\'  
            failed' %self.serverAddr)
```

8.1.1 SETUP

- Khi người dùng nhấn vào nút SETUP, hàm setupMovie() sẽ được thực thi.

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

- Hàm này sẽ gửi gói tin RTSP thông điệp SETUP đến máy chủ

```
def sendRtspRequest(self, requestCode):  
    # SETUP  
    if requestCode == self.SETUP and self.state == self.INIT:  
        threading.Thread(target = self.recvRtspReply).start()  
  
        # Init sequence number  
        self.rtspSeq = 1  
  
        # Create and send request  
        request = "SETUP " + str(self.fileName) + " RTSP/1.0\nCSeq: " +  
            str(self.rtspSeq) + "\nTransport: RTP/UDP; client_port=" +  
            str(self.rtpPort)  
        self.rtspSocket.send(request.encode("utf-8"))  
  
        self.requestSent = self.SETUP  
  
        print("\nRequest: " + request + "\n")  
    ...  
https://www.overleaf.com/project/617785278ccf7d51224d2bbf
```

- Bên cạnh đó, tạo ra một luồng song song thực thi hàm recvRtspReply() (Xem mục 8.1.5) để nhận gói tin RTSP được trả về từ server

8.1.2 PLAY

- Khi người dùng nhấn vào nút PLAY, hàm playMovie() sẽ được thực thi

```
def playMovie(self):
    """Play button handler."""
    #TODO
    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)
```

- Hàm này sẽ khởi tạo cơ chế đồng bộ giữa các thread và gửi gói tin RTSP đến máy chủ

```
def sendRtspRequest(self, requestCode):
    ...
    # PLAY
    elif requestCode == self.PLAY and self.state == self.READY:
        # Update sequence number
        self.rtspSeq += 1

        # Create and send request
        request = "PLAY " + str(self.fileName) + " RTSP/1.0\nCSeq: " +
            str(self.rtspSeq) + "\nSession: " + str(self.sessionId)
        self.rtspSocket.send(request.encode("utf-8"))

        self.requestSent = self.PLAY

        print("\nRequest: " + request + "\n")
    ...
```

8.1.3 PAUSE

- Khi người dùng nhấn vào nút PAUSE, hàm pauseMovie() sẽ được thực thi

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

- Hàm này sẽ gửi gói tin RTSP đến máy chủ

```
def sendRtspRequest(self, requestCode):  
    ...  
    # PAUSE  
    elif requestCode == self.PAUSE and self.state == self.PLAYING:  
        # Update sequence number  
        self.rtpSeq += 1  
  
        # Create and send request  
        request = "PAUSE " + str(self.fileName) + " RTSP/1.0\nCSeq: " +  
            str(self.rtpSeq) + "\nSession: " + str(self.sessionId)  
        self.rtpSocket.send(request.encode("utf-8"))  
  
        self.requestSent = self.PAUSE  
  
        print("\nRequest: " + request + "\n")  
    ...
```

8.1.4 TEARDOWN

- Khi người dùng nhấn vào nút TEARDOWN, hàm exitClient() sẽ được thực thi

```
def exitClient(self):  
    """Teardown button handler."""  
    #TODO  
    #if self.state != self.INIT:  
    self.sendRtspRequest(self.TEARDOWN)  
  
    # Close window  
    self.master.destroy()  
  
    # Delete cache  
    os.remove(CACHE_FILE_NAME + str(self.sessionId) + CACHE_FILE_EXT) # Delete  
    the cache image from video
```

- Hàm này sẽ gửi gói tin RTSP đến máy chủ, sau đó đóng cửa sổ và xóa file CACHE

```
def sendRtspRequest(self, requestCode):  
    ...  
    # TEARDOWN  
    elif requestCode == self.TEARDOWN and not self.state == self.INIT:  
        # Update sequence number  
        self.rtspSeq += 1  
  
        # Create and send request  
        request = "TEARDOWN " + str(self.fileName) + " RTSP/1.0\nCSeq: " +  
            str(self.rtspSeq) + "\nSession: " + str(self.sessionId)  
        self.rtspSocket.send(request.encode("utf-8"))  
  
        self.requestSent = self.TEARDOWN  
  
        print("\nRequest: " + request + "\n")  
    ...
```

8.1.5 Nhận gói tin từ server và phân tích

- Khi người dùng ấn vào SETUP thì một luồng song song được tạo ra thực thi hàm `recvRtspReply()` để nhận gói tin từ server

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

        if rtsp_reply:
            self.parseRtspReply(rtsp_reply.decode("utf-8"))

        if self.requestSent == self.TEARDOWN:
            self.rtspSocket.shutdown(socket.SHUT_RDWR)
            self.rtspSocket.close()
            break
```

- Hàm này sẽ chạy vòng lặp để luôn trong trạng thái nhận. Mỗi khi nhận được gói tin, hàm sẽ tiến hành thực thi `parseRtspReply()` để phân tích gói tin.
- Ngoài ra, nếu khi nhận được gói tin mà biến `requestSent` là `TEARDOWN` thì tiến hành đóng socket và thoát khỏi vòng lặp

```
def parseRtspReply(self, data):
    """Parse the RTSP reply from the server."""
    #TODO
    print("-----" + 'Data received-----\n' + data)
    mess = data.split('\n')
    seq = int(mess[1].split(' ')[1])

    # Check whether the sequence number equal to RTSP request
    if seq == self.rtspSeq:
        session = int(mess[2].split(' ')[1])

        # New session
        if self.sessionId == 0:
            self.sessionId = session

        # Check whether session receive is the same as RTSP sessionId
        if self.sessionId == session:
            if (int(mess[0].split(' ')[1])) == 200:
                if self.requestSent == self.SETUP:

                    # Update RTSP state
                    self.state = self.READY

                    # Open port
                    self.openRtpPort()

    ...
```

- Trong hàm phân tích gói tin `parseRtspReply()`, nếu gói tin nhận được là 200 (OK) và tín hiệu gửi là SETUP thì tiến hành mở RTP port bằng hàm `openRtpPort()` để nhận được dữ liệu khi tiến hành PLAY

```
def openRtpPort(self):
    self.rtpSocket.settimeout(0.5)

    try:
        server_address = (self.serverAddr, self.rtpPort)
        self.rtpSocket.bind(server_address)
    except:
        tkinter.messagebox.showwarning("Can't bind", "Can't bind to PORT %d"
                                       %self.rtpPort)
```

- Bên cạnh đó, nếu gói tin nhận được là 200 (OK) và tín hiệu gửi là PLAY, PAUSE hoặc TEARDOWN thì tiến hành đặt trạng thái của client lần lượt là PLAYING, READY và INIT. Ngoài ra, nếu là TEARDOWN thì ta sẽ đặt biến `teardownAked` về 1

```
def parseRtspReply(self, data):
    ...
        elif self.requestSent == self.PLAY:
            # Update RTSP state
            self.state = self.PLAYING

    elif self.requestSent == self.PAUSE:
        # Update RTSP state
        self.state = self.READY

        # The play thread exits. A new thread is created on resume.
        self.playEvent.set()

    elif self.requestSent == self.TEARDOWN:
        # Update RTSP state
        self.state = self.INIT

        # Turn on the teardown flag
        self.teardownAked = 1
```

8.1.6 Nhận RTP packet từ server và phát video

- Khi người dùng bấm Play, một luồng song song được tạo ra thực thi hàm listenRtp() để nhận gói tin RTP từ server và phát video

```
def listenRtp(self):
    #TODO
    while True:
        try:
            data = self.rtpSocket.recv(20480)

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

                current_Frame_Number = rtpPacket.seqNum()
                print("Current Frame " + str(current_Frame_Number))

                if current_Frame_Number > self.frameNbr:
                    self.frameNbr = current_Frame_Number
                    self.updateMovie(self.writeFrame(rtpPacket.getPayload()))
        except:
            if self.playEvent.isSet():
                break

            # If already sent teardown request -> close the rtpsocket
            if self.teardownAcked == 1:
                self.rtpSocket.shutdown(socket.SHUT_RDWR)
                self.rtpSocket.close()
                break
```

- Hàm này sẽ tiến hành nhận gói tin RTP và phân tích nó

- Qua quá trình phân tích, sẽ lấy được chỉ mục của bức ảnh và thực thi hàm updateMovie() để cập nhật khung ảnh

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

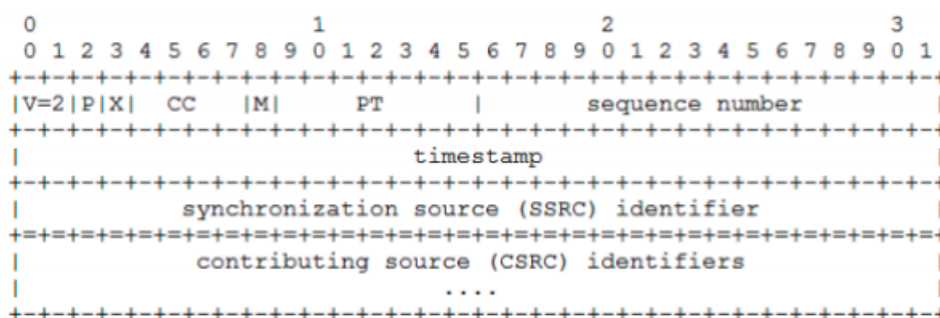
    return cache

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

- Nếu người dùng bấm PAUSE thì flag của event là true, khi đó sẽ ngưng nhận gói tin RTP
- Ngoài ra, nếu người dùng bấm TEARDOWN thì đóng RTP Socket và ngưng nhận gói tin RTP

8.2 Hiện thực giao thức RTP ở Client

- Ta sẽ hiện thực hàm `encode()` ở file `RtpPacket.py` để tạo ra header cho gói tin RTP
- Cấu trúc header của gói tin RTP như sau:



- Ta sẽ áp dụng các toán thao tác bit như dịch trái (`<<`), dịch phải (`>>`), hoặc (`|`), và (`&`)

```
def encode(self, version, padding, extension, cc, seqnum, marker, pt, ssrc,
    payload):
    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 & 0xFF
    self.header[3] = seqnum & 0xFF

    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

    self.payload = payload
```

9 Kết quả thu được

- Khi khởi chạy xong, ta có giao diện



- Khi bấm SETUP, kết quả từ Client và Server lần lượt là

```
Request: SETUP movie.Mjpeg RTSP/1.0
CSeq: 1
Transport: RTP/UDP; client_port= 8000

-----Data received-----
RTSP/1.0 200 OK
CSeq: 1
Session: 465641
```

```
Data received:
SETUP movie.Mjpeg RTSP/1.0
CSeq: 1
Transport: RTP/UDP; client_port= 8000
processing SETUP
```

- Khi bấm PLAY, kết quả từ giao diện, Client và Server lần lượt là



```
-----Data received-----  
RTSP/1.0 200 OK  
CSeq: 2  
Session: 465641  
Current Frame 1  
Current Frame 2  
Current Frame 3  
Current Frame 4  
Current Frame 5  
Current Frame 6  
Current Frame 7  
Current Frame 8  
Current Frame 9  
Current Frame 10  
Current Frame 11  
Current Frame 12
```

```
Data received:  
PLAY movie.Mjpeg RTSP/1.0  
CSeq: 2  
Session: 465641  
processing PLAY
```

- Khi bấm PAUSE, kết quả từ Client và Server lần lượt là

```
Request: PAUSE movie.Mjpeg RTSP/1.0
CSeq: 3
Session: 465641
```

```
Data received:
PAUSE movie.Mjpeg RTSP/1.0
CSeq: 3
Session: 465641
processing PAUSE
```

- Khi bấm TEARDOWN, giao diện đóng, kết quả từ Client và Server lần lượt là

```
Request: TEARDOWN movie.Mjpeg RTSP/1.0
CSeq: 6
Session: 465641

-----Data received-----
RTSP/1.0 200 OK
CSeq: 6
Session: 465641
```

```
Data received:
TEARDOWN movie.Mjpeg RTSP/1.0
CSeq: 6
Session: 465641
processing TEARDOWN
```

Kết quả thu được giống với mong đợi

10 Hướng dẫn sử dụng

- Đầu tiên, khởi động server bằng lệnh (Port Server phải lớn hơn 1024)

```
python Server.py server_port
```

Ví dụ:

```
python Server.py 2000
```

- Sau đó, khởi động Client bằng lệnh

```
python ClientLauncher.py server_host server_port RTP_port  
video_file
```

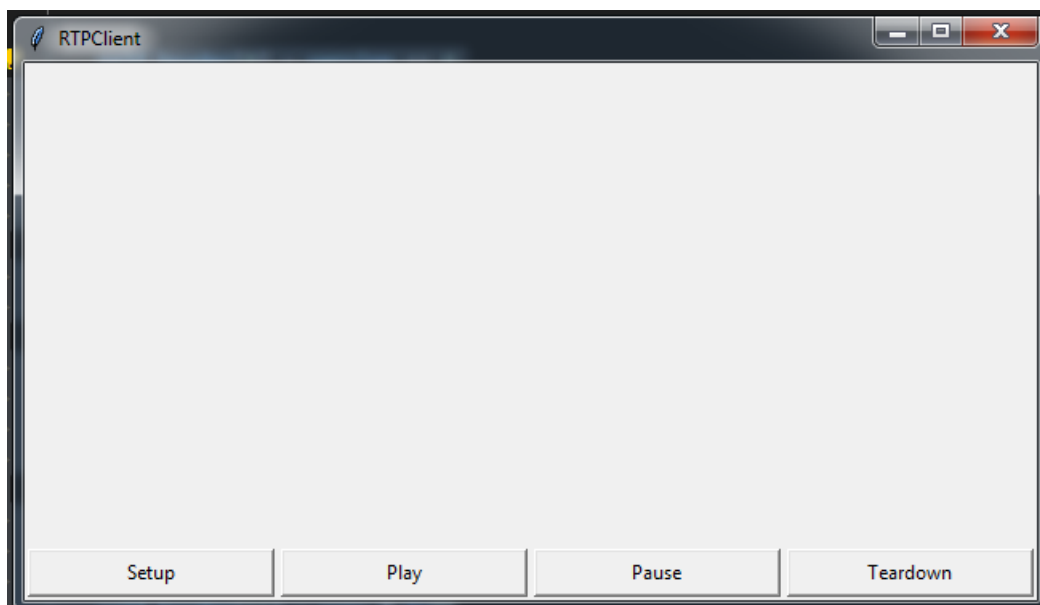
Trong đó:

- server_host là tên hoặc địa chỉ của server
- server_port là port mà server mở RTSP
- RTP_port là port để nhận gói tin RTP
- video_file là tên của video yêu cầu

Ví dụ

```
python ClientLauncher.py 127.0.0.1 2000 3000 movie.Mjpeg
```

- Client sẽ mở một kết nối đến server và hiện một pop-up





- Bạn có thể gửi lệnh RTSP đến server bằng cách nhấn các nút trên giao diện.
 - SETUP: Lệnh này được sử dụng để thiết lập phiên và các tham số truyền tải
 - PLAY: Lệnh này dùng để bắt đầu phát video
 - PAUSE: Lệnh này dùng để tạm dừng video
 - TEARDOWN: Lệnh này có tác dụng kết thúc phiên và đóng kết nối với server
- Server sẽ luôn phản hồi tất cả các thông điệp mà client gửi:
 - Phản hồi mã 200: Yêu cầu thành công
 - Phản hồi mã 404: FILE_NOT_FOUND
 - Phản hồi mã 500: Kết nối bị lỗi

11 Mở rộng

11.1 Thống kê về phiên

- Để tính toán về tỉ lệ mất gói tin RTP và tốc độ truyền tải dữ liệu (Bit/Bytes Per Second), ta thêm vào các số liệu sau

```
class Client:
    ...
    start_time = 0
    total_time = 0
    total_data = 0
    num_frame_lost = 0
    stop = False
    ...
```

- Trong đó:
 - start_time: Thời gian bắt đầu phát video
 - total_time: Tổng thời gian phát video
 - total_data: Tổng số byte nhận được
 - num_frame_lost: Tổng số khung hình bị mất
 - exit: Kiểm tra xem người dùng đã thoát hay chưa
- Khi người dùng nhấn PLAY, thiết lập thời gian trong hàm listenRtp() và đến khi người dùng nhấn PAUSE hoặc TEARDOWN, tính tổng cộng thời gian. Trong khi thực thi hàm, mỗi lần nhận được gói tin, đếm số khung hình bị mất

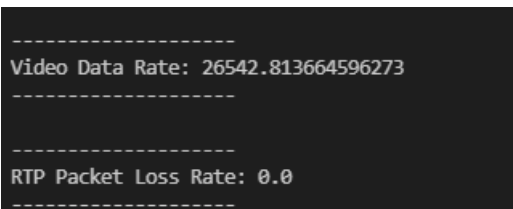
```
def listenRtp(self):
    self.start_time = time.time()
    self.stop = False
    while True:
        try:
            data = self.rtpSocket.recv(20480)

            if data:
                ...
                self.total_data += len(data)
                if self.frameNbr + 1 != current_Frame_Number:
                    self.num_frame_lost += 1
                    print('-----' + "\nLOSS 1 PACKET\n" +
                        '-----')
                ...
            except:
                self.total_time += time.time() - self.start_time
                self.stop = True
                if self.playEvent.isSet():
                    break
                ...
            self.total_time += time.time() - self.start_time
            self.stop = True
```

- Khi người dùng nhấn TEARDOWN, tiến hành tính các thông số cần thiết

```
def exitClient(self):  
    ...  
    if not self.stop:  
        self.total_time += time.time() - self.start_time  
  
    if self.total_time > 0:  
        dataRate = float(int(self.total_data)/int(self.total_time))  
        print("\n-----\nVideo Data Rate: " + str(dataRate)  
              + "\n-----")  
        lossRate = float(self.num_frame_lost/self.frameNbr)  
        print("\n-----\nRTP Packet Loss Rate: " + str(lossRate)  
              + "\n-----")
```

- Kết quả:



```
-----  
Video Data Rate: 26542.813664596273  
-----  
  
-----  
RTP Packet Loss Rate: 0.0  
-----
```

11.2 Lược bỏ SETUP

- Giao diện người dùng của Rtp Client có 4 button cho 4 chức năng
- Nếu so sánh với các media player chuẩn, có thể thấy chúng chỉ có 3 nút với cùng hành động: PLAY, PAUSE và STOP. Không có nút SETUP khả dụng cho người dùng
- **Câu hỏi** Cho biết SETUP là bắt buộc đối với giao tiếp RTSP. Hiện thực RtpClient giống như các media player ? Lúc nào thì Client gửi SETUP ?

- **Trả lời**

- Ta sẽ hiện thực điều này bằng cách, lúc người dùng nhấn vào PLAY. Client sẽ gửi thông điệp SETUP đến Server trước, sau đó sẽ gửi tiếp thông điệp PLAY
- Lúc đó, state của Client sẽ là : INIT -> READY -> PLAYING
- Đầu tiên, loại bỏ button SETUP và thêm vào biến boolean finish_setup để xem xét quá trình SETUP khi nhấn vào PLAY

```
class Client:
...
    finish_setup = True
...
```

- Khi người dùng nhấn vào PLAY, một thông điệp SETUP sẽ được gửi đến server và thiết lập một vòng lặp cho đến khi nhận được gói tin phản hồi

```
def playMovie(self):
    if self.state == self.INIT:
        self.finish_setup = False
        self.sendRtspRequest(self.SETUP)

    while not self.finish_setup:
        pass
    ...
```

- Ở hàm parseRtspReply(), khi nhận được phản hồi về SETUP thành công, sẽ đặt biến finish_setup về True để tiếp tục gửi thông điệp PLAY đến Server

```
def parseRtspReply(self, data):
    ...
    # Check whether session receive is the same as RTSP sessionId
    if self.sessionId == session:
        if (int(mess[0].split(' ')[1])) == 200:
            if self.requestSent == self.SETUP:

                # Update RTSP state
                self.state = self.READY

                # Open port
                self.openRtpPort()
                self.finish_setup = True
    ...
```

- Kết quả: Khi nhấn vào nút PLAY, video sẽ được phát ngay sau đó



11.3 Thêm chức năng DESCRIBE

- Hiện tại, Client và Server chỉ có hiện thực các tương tác RTSP cơ bản và PAUSE.
- **Yêu cầu:** Hiện thực chức năng DESCRIBE để truyền thông tin về media stream. Khi Server nhận được yêu cầu DESCRIBE, nó sẽ trả về mô tả của phiên stream
- **Trả lời**

- Đầu tiên, ở phía client, thiết lập thêm button Describe

```
def createWidgets(self):  
    """Build GUI."""  
    ...  
  
    # Create Describe button  
    self.setup = Button(self.master, width=16, padx=3, pady=3)  
    self.setup["text"] = "Describe"  
    self.setup["command"] = self.describe  
    self.setup.grid(row=1, column=4, padx=2, pady=2)  
  
    ...
```

- Thêm hàm describe() thực thi khi người dùng nhấn nút Describe và chỉnh sửa hàm sendRtspRequest() để gửi thông điệp Describe đến Server

```
def describe(self):  
    if (self.play_yet) and (self.state == self.READY):  
        self.sendRtspRequest(self.DESCRIBE)
```

```
def sendRtspRequest(self, requestCode):  
    ...  
    # DESCRIBE  
    elif requestCode == self.DESCRIBE and self.state == self.READY:  
        # Update sequence number  
        self.rtspSeq += 1  
  
        # Create and send request  
        request = "DESCRIBE " + str(self.fileName) + " RTSP/1.0\nCSeq: " +  
            str(self.rtspSeq) + "\nSession: " + str(self.sessionId)  
        self.rtspSocket.send(request.encode("utf-8"))  
  
        self.requestSent = self.DESCRIBE  
        print("\nRequest: " + request + "\n")  
    ...
```

- Ta chỉ cho phép thực hiện chức năng Describe khi đã từng thực hiện chức năng Play trước đó. Và trong khi Play, không cho phép Describe bằng cách thiết lập thêm biến play_yet

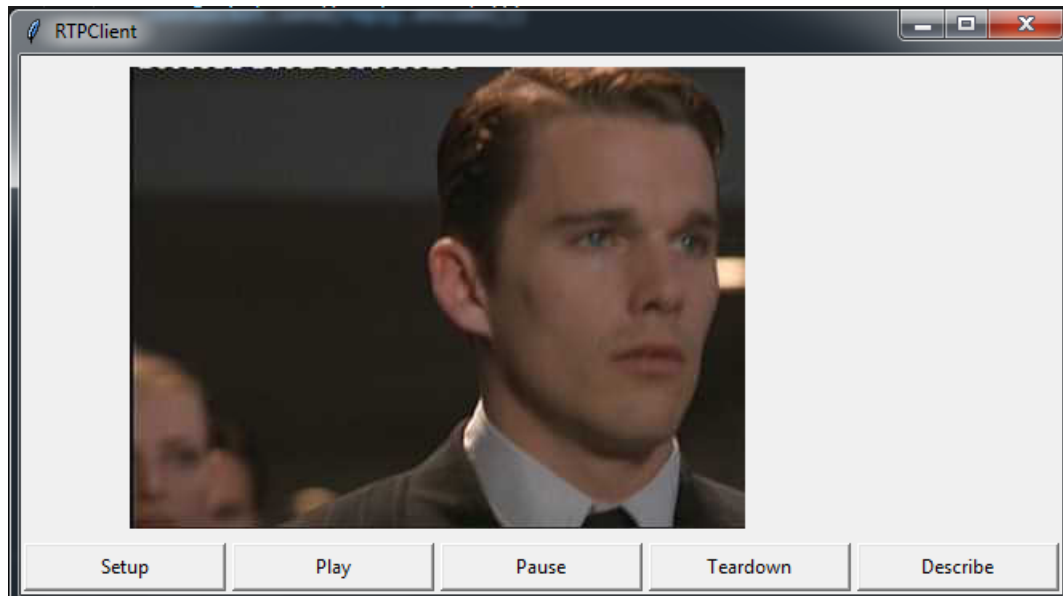
- Tiếp theo, ở phía Client, ta chỉnh sửa hàm processRtspRequest() để nhận yêu cầu Describe

```
def processRtspRequest(self, data):  
    ...  
    # Process DESCRIBE request  
    elif requestType == self.DESCRIBE:  
        print("processing DESCRIBE\n")  
  
    self.replyRtspDescribe(self.OK_200, seq[1])
```

- Cuối cùng, thêm vào hàm replyRtspDescribe() để trả về thông tin về phiên stream

```
def replyRtspDescribe(self, code, seq):  
    """Send RTSP reply to the client."""  
    if code == self.OK_200:  
        rtpPort = "\nRTP Port: " + str(self.clientInfo['rtpPort'])  
        streamid = "\nStream ID: " + str(self.clientInfo['session'])  
        videotype = "\nVideo Type: " +  
            str(self.clientInfo['videoStream'].filename) + "\n"  
  
        reply = 'RTSP/1.0 200 OK\nCSeq: ' + seq + '\nSession:1 ' +  
            str(self.clientInfo['session']) + '\n' + rtpPort + streamid +  
            videotype  
        connSocket = self.clientInfo['rtspSocket'][0]  
        connSocket.send(reply.encode())  
  
    # Error messages  
    elif code == self.FILE_NOT_FOUND_404:  
        print("404 NOT FOUND")  
    elif code == self.CON_ERR_500:  
        print("500 CONNECTION ERROR")
```

- Kết quả: Giao diện và phản hồi từ Server là:



```
-----Data received-----  
RTSP/1.0 200 OK  
CSeq: 6  
Session:1 723126  
  
RTP Port: 3000  
Stream ID: 723126  
Video Type: movie.Mjpeg
```




12 Source code

- Link Github: [Video Streaming](#)



References

- [1] [Giao thức RTSP](#)