

TRƯỜNG ĐẠI HỌC BÁCH KHOA TP HCM  
KHOA KHOA HỌC VÀ KỸ THUẬT MÁY TÍNH

-----



# MẠNG MÁY TÍNH

---

Báo cáo

## VIDEO STREAMING

---

Giảng viên hướng dẫn: Vũ Văn Tiến  
Sinh viên: Võ Quý Long - 1914011  
Nguyễn Xuân Vũ - 1915983  
Trần Thọ Nhân - 1910405  
Bùi Nguyễn Đức Tùng - 1910676

Tp. Hồ Chí Minh, Tháng 10/2021

# Mục lục

<b>1</b>	<b>Bảng phân công nội dung</b>	<b>2</b>
<b>2</b>	<b>Phân tích yêu cầu</b>	<b>2</b>
2.1	Functional . . . . .	2
2.2	Non-Functional . . . . .	2
<b>3</b>	<b>Mô tả chức năng</b>	<b>3</b>
3.1	Set up . . . . .	3
3.2	Play . . . . .	4
3.3	Pause . . . . .	8
3.4	Teardown . . . . .	8
<b>4</b>	<b>Class Diagram</b>	<b>9</b>
<b>5</b>	<b>Đánh giá kết quả</b>	<b>9</b>
<b>6</b>	<b>Hướng dẫn sử dụng</b>	<b>9</b>
<b>7</b>	<b>Phần mở rộng</b>	<b>11</b>
7.1	Số liệu thống kê về phiên . . . . .	11
7.2	Media player . . . . .	12
7.3	Thêm chức năng DESCRIBE . . . . .	13
<b>8</b>	<b>Source code</b>	<b>14</b>
<b>9</b>	<b>Tài liệu tham khảo</b>	<b>14</b>

# 1 Bảng phân công nội dung

Tên thành viên	Nhiệm vụ	Hoàn Thành
Võ Quý Long (Leader)	Basic + Report	100
Nguyễn Xuân Vũ	Function Description + Extend 1	100
Trần Thọ Nhân	Class Diagram + Extend 2	100
Bùi Nguyễn Đức Tùng	Requirements Analysis + Extend 3	100

## 2 Phân tích yêu cầu

### 2.1 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.

### 2.2 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.
- 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.

### 3 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 khách hàng biết phản hồi của máy chủ khi nhận các lệnh một cách chính xác:

- OK-200
- FILE-NOT-FOUND-404
- CON-ERR-500

#### 3.1 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

```
# 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 phía máy chủ nhận được lệnh "SETUP" thì sẽ thực hiện các 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 lý 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:

## 3.2 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:

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

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

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

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

    # 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:
                # Update RTSP state.
                self.state = self.READY

                # Open RTP port.
                self.openRtpPort()

def openRtpPort(self):
    """Open RTP socket binded to a specified port."""
    # 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)

# 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 ' + self.fileName + ' RTSP/1.0\nCSeq: ' + str(self.rtspSeq) + '\nSession'

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

# 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()

def nextFrame(self):
    """Get next frame."""
    data = self.file.read(5) # Get the framelength from the first 5 bits
    if data:
        framelength = int(data)

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

    return data
```

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

RTP packet header							
Bit offset <sup>[b]</sup>	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:

```
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)

    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

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

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

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

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



### 3.3 Pause

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

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

    # Write the RTSP request to be sent.
    request = 'PAUSE ' + self.fileName + ' RTSP/1.0\nCSeq: ' + str(self.rtpSeq) + '\nSession: ' +

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

Và máy khách sẽ được đặt lại STATE của nó là READ 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])
```

### 3.4 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:

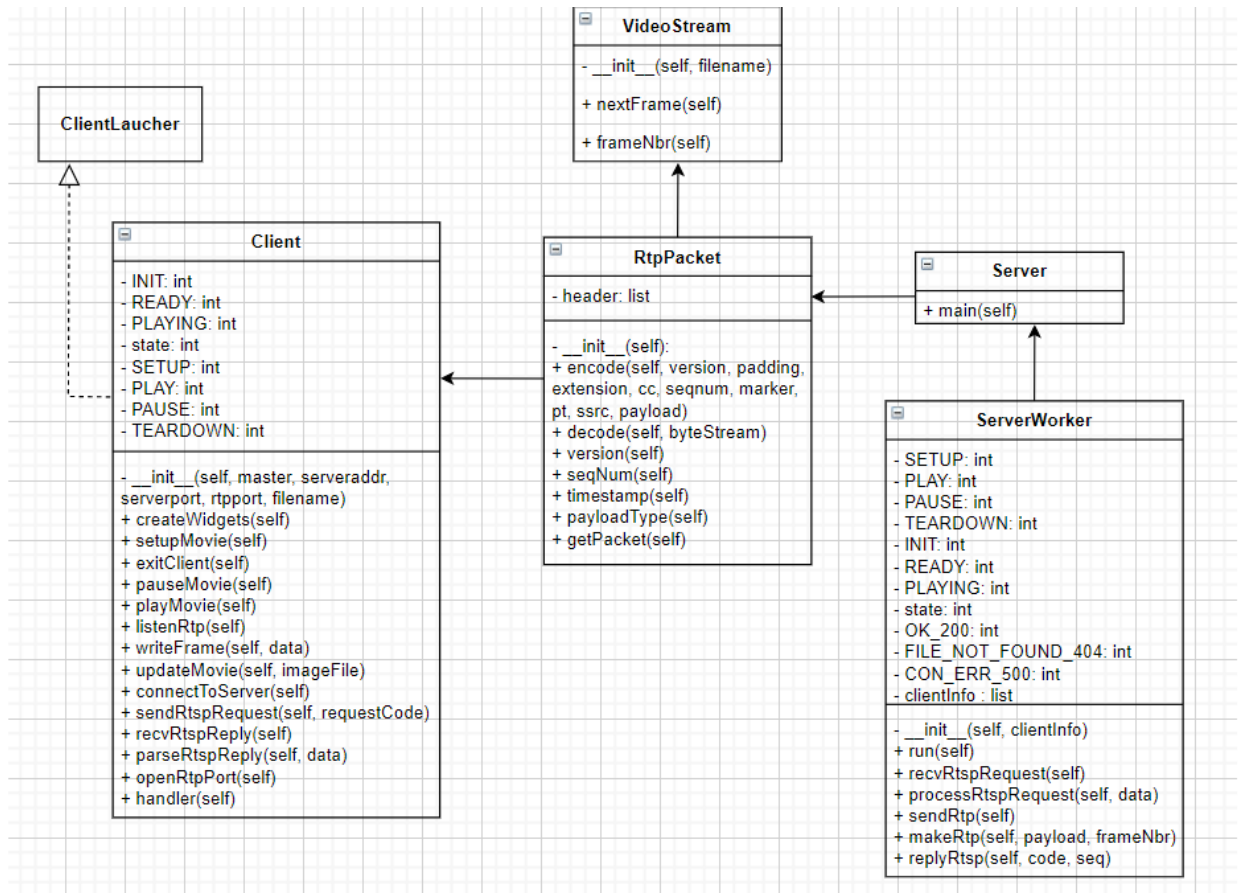
```
# Process TEARDOWN request
elif requestType == self.TEARDOWN:
    print("processing TEARDOWN\n")

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

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

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

## 4 Class Diagram



Hình 2: Class Diagram

## 5 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:

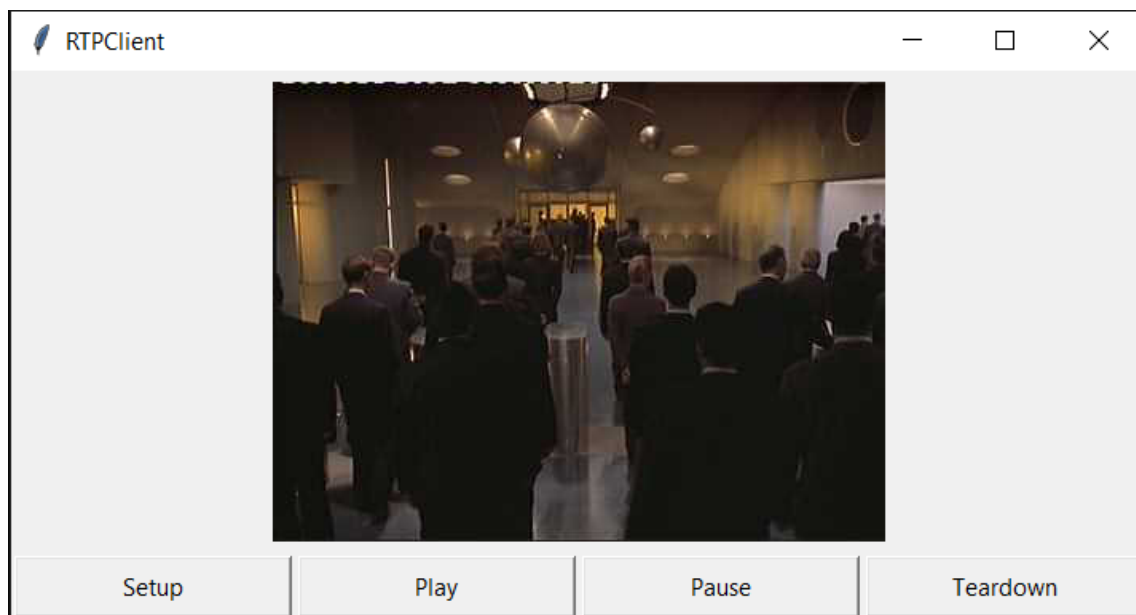
Các message request và reply được gửi từ máy khách và máy chủ được hiển thị bên máy khách khi nhấn SETUP và PLAY, máy chủ sẽ trả về RTSP/1.0 200 OK nếu yêu cầu thành công:

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

## 6 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>**



```

Microsoft Windows [Version 10.0.19042.1288]
(c) Microsoft Corporation. All rights reserved.

C:\Users\DMX DAK MIL>cd C:\Users\DMX DAK MIL\Downloads\Initcode

C:\Users\DMX DAK MIL\Downloads\Initcode>cd ./Basic

C:\Users\DMX DAK MIL\Downloads\Initcode\Basic>python ClientLauncher.py localhost 3000 80 movie.Mjpeg

Data sent:
SETUP movie.Mjpeg RTSP/1.0
CSeq: 1
Transport: RTP/UDP; client_port= 80

Data sent:
PLAY movie.Mjpeg RTSP/1.0
CSeq: 2
Session: 637720
Current Seq Num: 1
Current Seq Num: 2
Current Seq Num: 3
Current Seq Num: 4
Current Seq Num: 5
Current Seq Num: 6
    
```

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**

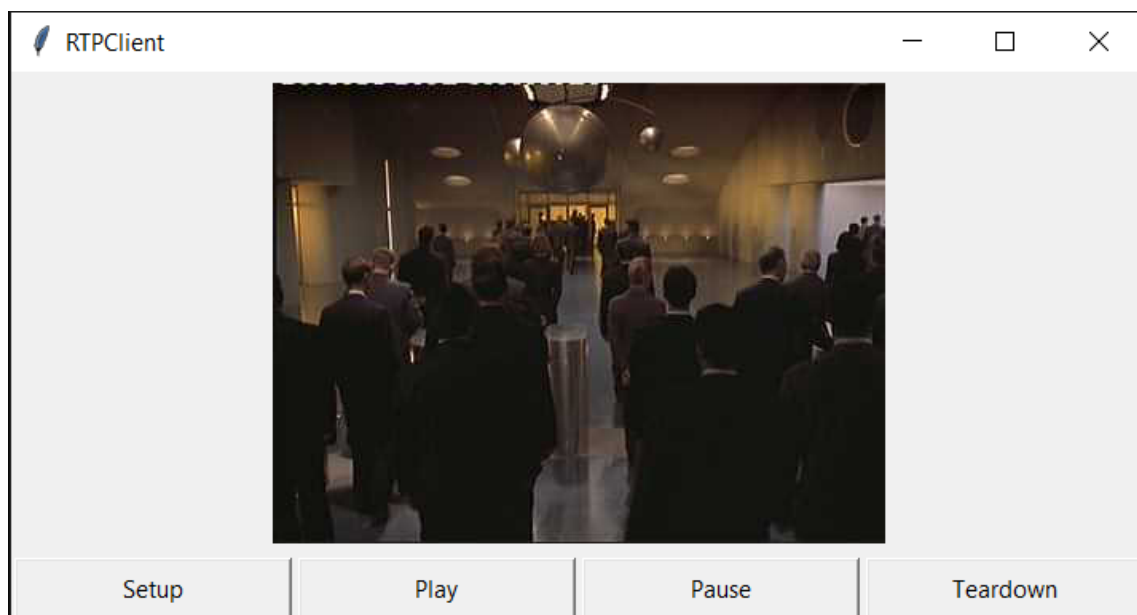
Sau đó, khởi động máy khách bằng lệnh:

**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:

1. Máy khách gửi SETUP. Lệnh này được sử dụng để phát lập phiên và các tham số truyền tải.
2. Máy khách gửi PLAY. Lệnh này dùng để bắt đầu phát video.
3. Máy khách gửi PAUSE nếu muốn tạm dừng video.
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.

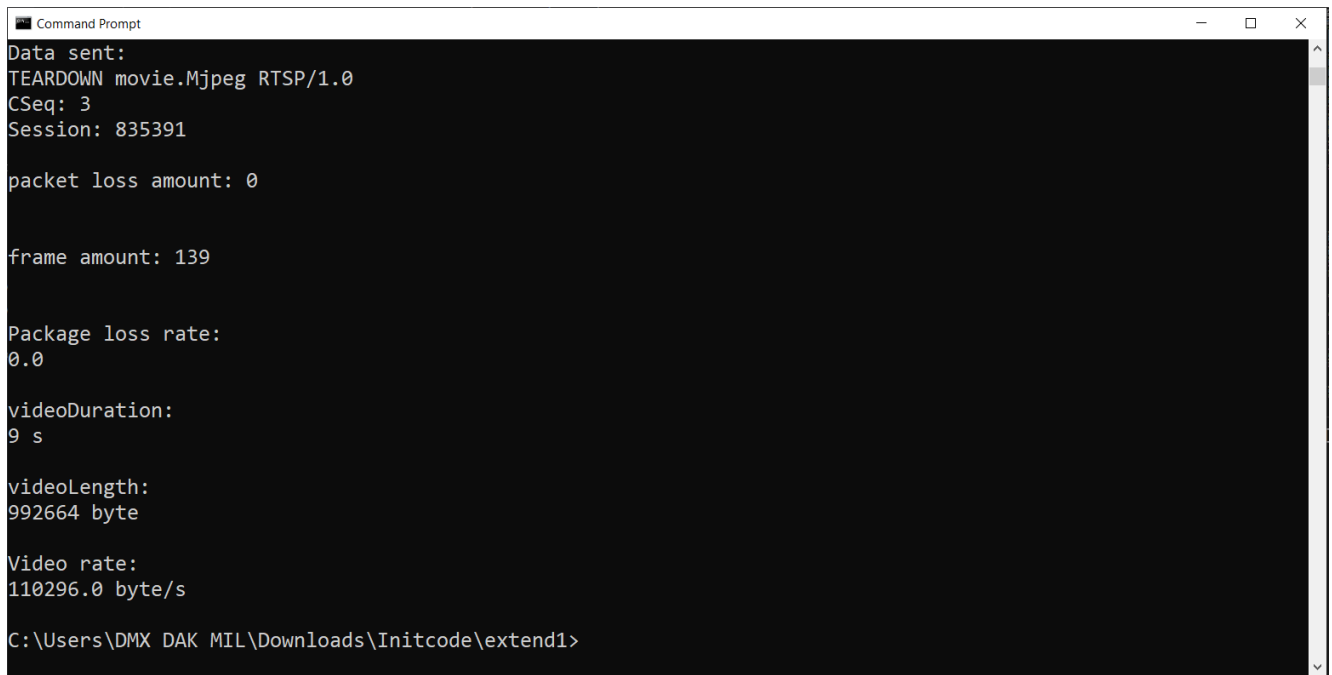
## 7 Phần mở rộng

### 7.1 Số liệu thống kê về 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).

```
rate = float(self.lostFrameCounter)/float(self.frameNbr)
print ("\npacket loss amount: " + str(self.lostFrameCounter)+"\n")
print ("\nframe amount: " + str(self.frameNbr)+"\n")
print ('\nPackage loss rate:\n' + str(rate))
```

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



```

Command Prompt
Data sent:
TEARDOWN movie.Mjpeg RTSP/1.0
CSeq: 3
Session: 835391

packet loss amount: 0

frame amount: 139

Package loss rate:
0.0

videoDuration:
9 s

videoLength:
992664 byte

Video rate:
110296.0 byte/s

C:\Users\DMX DAK MIL\Downloads\Initcode\extend1>
    
```

Hình 4: Thống kê về session

## 7.2 Media player

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.

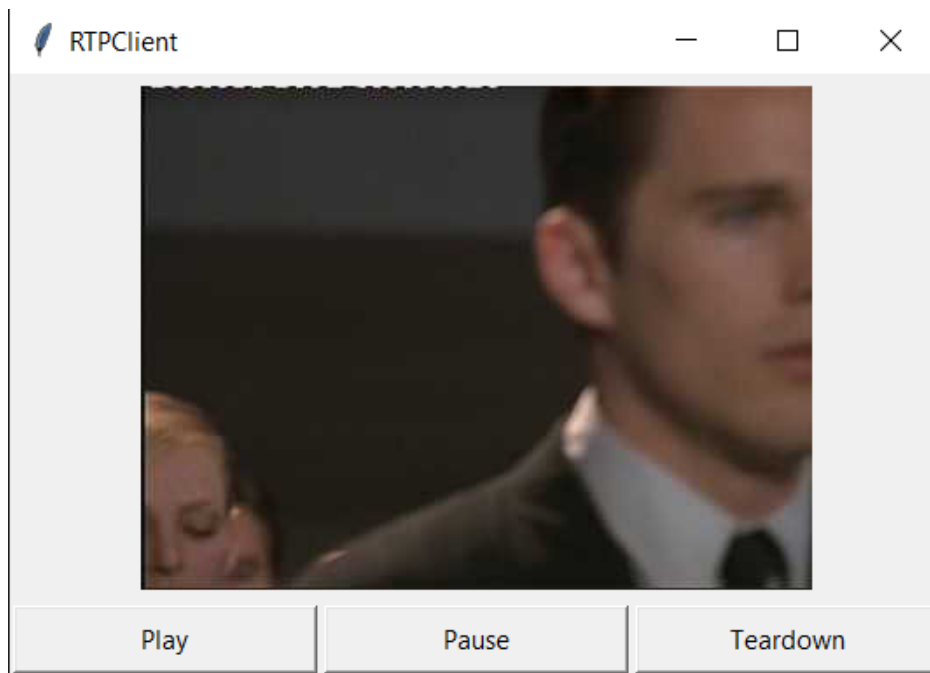
Giải pháp đưa ra là khi người dùng nhấn PLAY thực hiện liên tiếp hai chức năng SETUP và PLAY:

- + 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.
- + Máy chủ sẽ tạo ra một Socket để truyền RTP qua UDP và bắt đầu một bước gửi packet video.
- + STATE của máy khách sẽ chuyển từ INIT qua READY qua PLAYING.

```

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

Kết quả thu được:



Hình 5: Media player

### 7.3 Thêm chức năng DESCRIBE

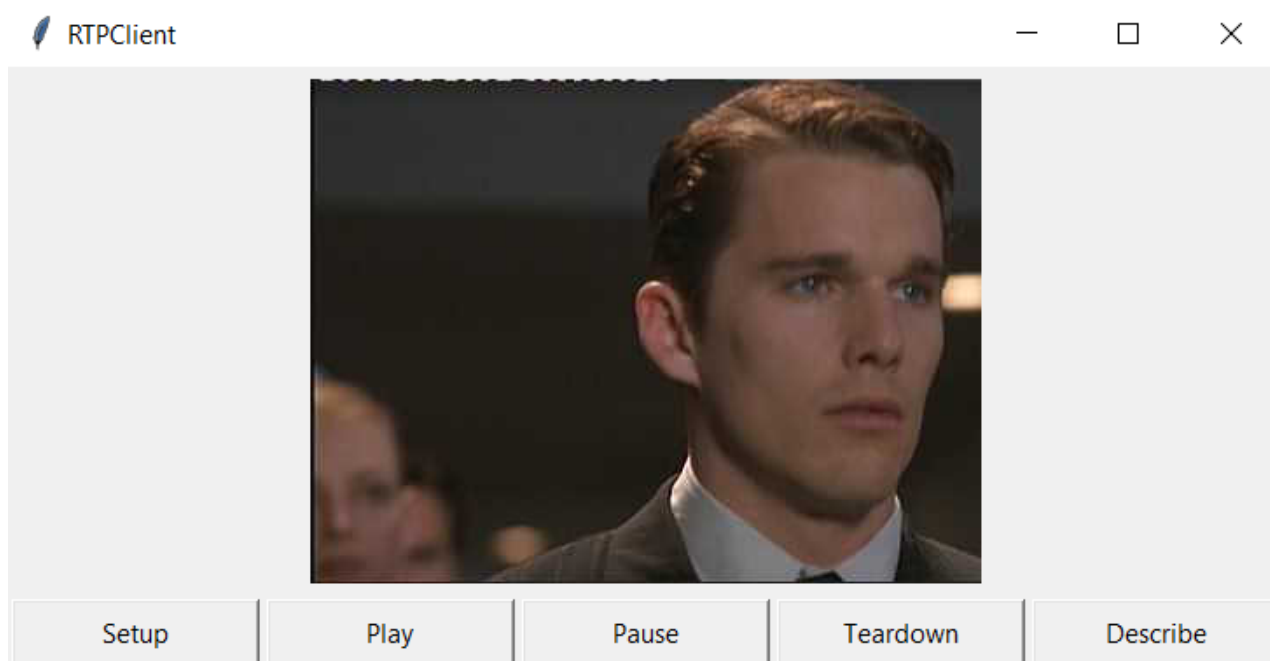
Ngoài những tương tác RTSP và PAUSE, DESCRIBE được sử dụng để truyền những thông tin về video stream. Khi server nhận được DESCRIBE request, nó sẽ gửi lại một file mô tả session - nói cho client loại stream trong session và encoding được sử dụng là gì.

Cụ thể, khi server gửi phản hồi cho client thì có những thông kê về:

- RTSP port của server
- Video encoding của video được truyền qua RTP
- RTSP ID của session đã được thiết lập

```
def describeMovie(self):
    """Describe button handler."""
    if self.state != self.INIT:
        self.info = True
        self.sendRtspRequest(self.DESCRIBE)
```

Kết quả thu được:



Hình 6: Chức năng Describe

## 8 Source code

Link source code github: <https://github.com/zogvm/Video-Stream-with-RTSP-and-RTP>

## 9 Tài liệu tham khảo

### Tài liệu

- [1] Environment for video streaming - [http://www.scielo.org.co/scielo.php?script=sci\\_arttext&pid=S0122-34612016000200005](http://www.scielo.org.co/scielo.php?script=sci_arttext&pid=S0122-34612016000200005)
- [2] RTSP Streaming with RTP/RTCP - [https://pro.sony/s3/2020/02/20093012/ SRG-360SHE\\_RTSP-Streaming-with-RTP-and-RTCP\\_V1.0.pdf](https://pro.sony/s3/2020/02/20093012/ SRG-360SHE_RTSP-Streaming-with-RTP-and-RTCP_V1.0.pdf)
- [3] RTSP/RTP streaming server - <https://www.medialan.de/usecase0001.html>
- [4] Wikipedia - Session Description Protocol - [https://en.wikipedia.org/wiki/Session\\_Description\\_Protocol](https://en.wikipedia.org/wiki/Session_Description_Protocol)
- [5] Python - Gui With Tinker - <https://realpython.com/python-gui-tkinter/>