TRƯỜNG ĐẠI HỌC BÁCH KHOA TP HCM

ASSIGNMENT 1

Computer Network

Nguyễn Duy Thanh - 1813967

Nguyễn Quang Tùng - 1814701

Nguyễn Ngọc Tân – 1813942

GVHD: Hoàng Nguyễn Minh Đức

Contents

1.Requirement analysis

SEVER

- Cho phép máy khách(client) kết nối, giao tiếp và truyền dữ liệu video.

CLIENT

- Kết nối, giao tiếp với máy chủ và nhận dữ liệu video với tên video đã biết trước.

def sendRtspRequest(requestCode)

If requestCode == SETUP && state == INIT

- Client thiết lập kết nối TCP với máy chủ, thường là trên cổng TCP 554 còn được biết là cổng cho RTSP.
- Máy khách sau đó sẽ bắt đầu phát hành một loạt lệnh tiêu đề RTSP có định dạng tương tự như HTTP.

> Mô tả chi tiết các yêu cầu phiên bản đối với máy chủ.

> Ví dụ: Phiên bản RTSP mà nó hỗ trợ.

> Phương tiện được sử dụng cho luồng dữ liệu.

> Mọi thông tin cổng UDP hoặc TCP được liên kết.

> Những thông tin này được chuyển bằng cách sử dụng tiêu đề DESCRIBE và SETUP và được bổ sung trên phản hồi của máy chủ với phiên bản ID mà máy khách và bất kỳ thiết bị proxy tạm thời nào có thể sử dụng để xác định luồng trong các trao đổi trong tương lai.

Sau khi các thông số thương lượng đã hoàn thành

> Máy khách đưa ra lệnh PLAY để hướng dẫn máy chủ bắt đầu phân phối luồng dữ liệu RTP.

Sau khi máy khách quyết định đóng luồn, lệnh TEARDOWN được đưa ra cùng với phiên bản ID hướng dẫn máy chủ ngừng phân phối RTP được liên kết với ID đó.

SETUP

***** Gửi yêu cầu SETUP đến máy chủ. Sinh viên cần chèn “Transport header” trong đó sinh viên chỉ định cổng ổ cắm dữ liệu RTP mà sinh viên vừa tạo.

* Đọc phản hồi của máy chủ và phân tích cú pháp tiêu đề Session (từ phản hồi) để nhận phiên bản ID.

* Tạo một ổ cắm datagram để nhận dữ liệu RTP và đặt thời gian chời trên ổ cắm là 0.5 giây.

PLAY

* Gửi yêu cầu PLAY. Sinh viên phải cần chèn “Session header” và sử dụng ID phiên bản được trả về trong phản hồi SETUP. Sinh viên không được đặt “Transport header” trong yêu cầu này

*Đọc phản hồi của máy chủ.

PAUSE

*Gửi yêu cầu PAUSE. Sinh viên phải chèn “Session header” và sử dụng ID phiên bản được trả về trong phản hồi SETUP. Sinh viên không được đặt “Transport header” trong yêu cầu này

* Đọc phản hồi của máy chủ.

TEARDOWN

*Gửi yêu cầu TEARDOWN. Sinh viên phải chèn “Session header” và sử dụng ID phiên bản được trả về trong phản hồi SETUP. Sinh viên không được đặt “Transport header” trong yêu cầu này.

* Đọc phản hồi của máy chủ. Lưu ý: Sinh viên phải chèn tiêu đề CSeq trong mọi yêu cầu sinh viên gửi. Giá trị của tiêu đề CSeq là một số bắt đầu từ 1 và được tăng lên một cho mỗi yêu cầu sinh viên gửi

2. Function Description

```
class ServerWorker:
        SETUP = 'SETUP'
        PLAY = 'PLAY'
        PAUSE = 'PAUSE'
        TEARDOWN = 'TEARDOWN'

        INIT = 0
        READY = 1
        PLAYING = 2
        state = INIT

        OK_200 = 0
        FILE_NOT_FOUND_404 = 1
        CON_ERR_500 = 2
```

```
class Client:

        SETUP_STR = 'SETUP'
        PLAY_STR = 'PLAY'
        PAUSE_STR = 'PAUSE'
        TEARDOWN_STR = 'TEARDOWN'
        INIT = 0
        READY = 1
        PLAYING = 2
        state = INIT

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

        RTSP_VER = "RTSP/1.0"
        TRANSPORT = "RTP/UDP"
```

Những gì sẽ được gửi từ máy khách đến máy chủ thông qua giao thức RTSP là các lệnh như SETUP, PLAY, PAUSE, TEARDOWN. Các lệnh này sẽ cho phía máy chủ biết hành động tiếp theo mà nó sẽ hoàn thành.

Những gì sẽ được trả lời từ máy chủ đến máy khách thông qua giao thức RTSP là các tham số như: OK_200, FILE_NOT_FOUND_404, CON_ERR để thông báo cho client nếu server nhận lệnh chính xác.

Sau khi máy khách nhận được câu trả lời của máy chủ, nó sẽ thay đổi trạng thái tương ứng thành READY, nếu lệnh SETUP được gửi từ máy khách đến máy chủ.

```
def sendRtspRequest(self, requestCode):
        """Send RTSP request to the server."""
        #-------------
        # TO COMPLETE
        #-------------

        # Setup request
        if requestCode == self.SETUP and self.state == self.INIT:
                threading.Thread(target=self.recvRtspReply).start()

                # Update RTSP sequence number.
                self.rtspSeq = 1

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

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

Gói “SETUP” RTSP sẽ bao gồm:

> Lệnh SETUP

> Tên tệp video sẽ phát

> RTSP Pakcet Sequence Number bắt đầu từ 1

> Loại giao thức: RTSP/1.0 RTP

> Giao thức truyền: UDP

> Cổng RTP để truyền dòng 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 Server nhận được lệnh "SETUP" nó sẽ

> Gán cho khách hàng một số Session cụ thể ngẫu nhiên

> Nếu có gì đó sai với lệnh này hoặc trạng thái của máy chủ, nó sẽ trả lười gói Lỗi trở lại máy khách

> Nếu lệnh xử lý chính xác, nó sẽ trả lời lại OK_200 cho máy khách và đặt STATE của nó thành READY

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

Máy chủ sẽ mở tệp video được chỉ định trong gói SETUP Packet và khởi tạo frame của video thành 0.

Phía Client sẽ lặp lại để nhận RTSP Reply của Server

```
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
```

Sau đó phân tích cú pháp gói RTSP Reply

Lấy Session Number

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

        # Process only if the server reply's sequence number is the same as the request's
        if seqNum == self.rtspSeq:
                session = int(lines[2].split(' ')[1])
                # New RTSP session ID
                if self.sessionId == 0:
                        self.sessionId = session
```

Và nếu Reply Packet phản hồi cho lệnh SETUP. Máy chủ sẽ chuyển trạng thái sang READY

```
# 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.READY

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

Sau đó openRTPPort() nhận luồng video đến

```
try:
        # Bind the socket to the address using the RTP port given by the client user.
        self.state=self.READY
        self.rtpSocket.bind(('',self.rtpPort))
```

Sau đó nếu lệnh PLAY RTSP được gửi đến server:

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

        # Update RTSP sequence number.
        self.rtspSeq += 1

        # Write the RTSP request to be sent.
        request = ("PLAY " + str(self.fileName) + " RTSP/1.0 " + "\n" +
        "CSeq: " + str(self.rtspSeq) + "\n" +
        "Session: " + str(self.sessionId))


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

Máy chủ sẽ tạo một Socket để truyền RTP qua UDP và bắt đầu một bước để gửi gói luồng video

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

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

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

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

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

```
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
```

Mỗi gói dữ liệu cũng sẽ được mã hóa với một tiêu đề, tiêu đề sẽ bao gồm

> RTP-version

> Padding extension

> Contributing source

> Marker Type Field 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 ...						

Chúng đã được chèn vào gói RTP thông qua các thao tác bitwise

```
self.header[0] = version << 6
self.header[0] = header[0] | padding << 5
self.header[0] = header[0] | extension << 4
self.header[0] = header[0] | cc
self.header[1] = marker << 7
self.header[1] = header[1] | pt
self.header[2] = (seqnum & 0xFF00) >> 8
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
self.header[9] = ssrc >> 16
self.header[10] = ssrc >> 8
self.header[11] = ssrc
```

Cuối cùng, RTP Packet sẽ bao gồm một tiêu đề và một khung video được gửi đến RTP Port ở phía client.

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

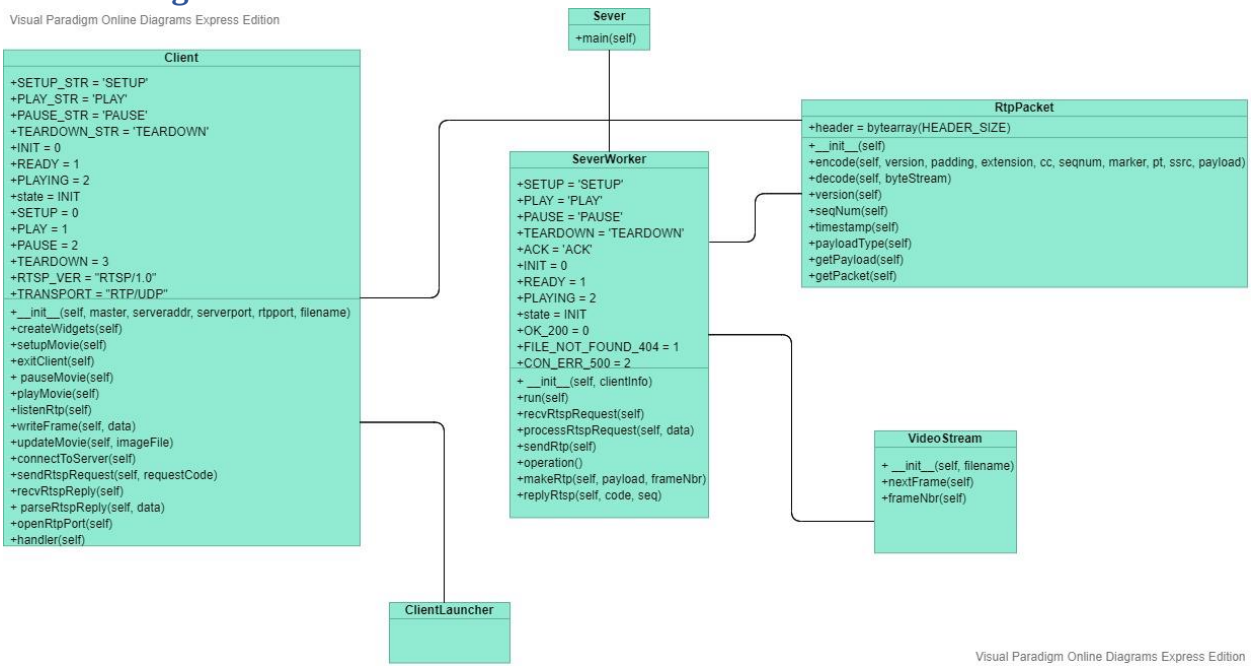
Sau đó, Client giải mã RTP Packet để lấy tiêu đề và frame, tổ chức lại cái fram và hiển thị trên giao diện người dùng

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

Nếu lệnh PAUSE được gửi từ máy khách đến máy chủ, nó sẽ ngăn máy chủ gửi các khung hình video đến máy khách

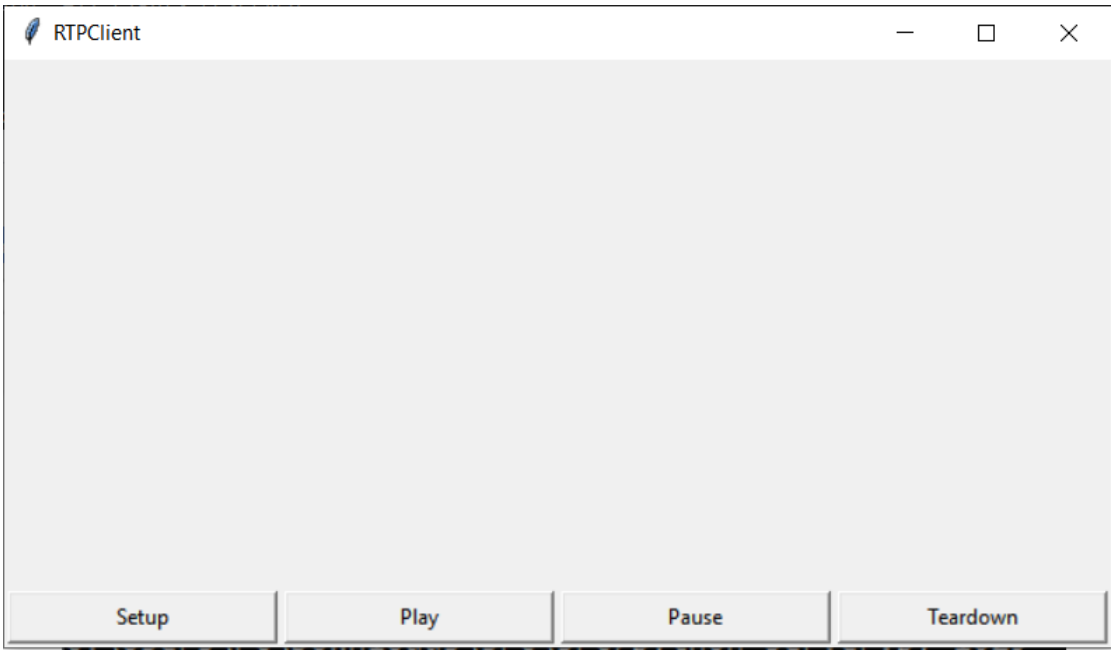
Nếu một lệnh TEARDOWN được gửi từ máy khách đến máy chủ, nó cũng sẽ ngăn máy chủ gửi các khung hình video đến máy khách và đóng cả thiết bị đầu cuối của máy khách

3. Class diagram



4. Achiedved results:

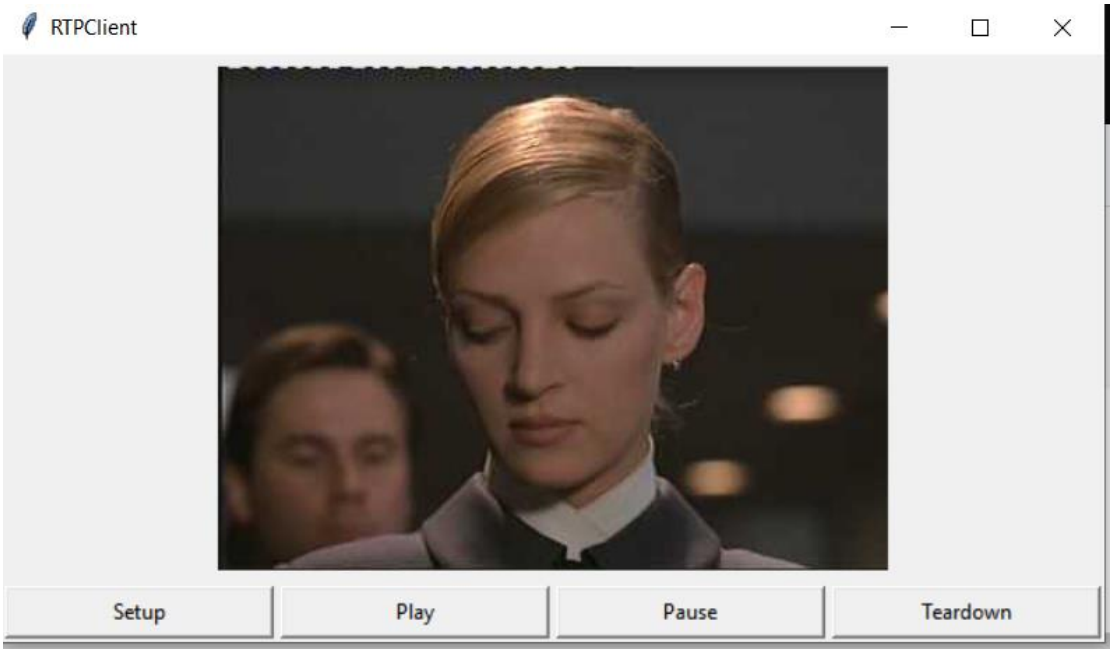
Sau khi chạy dòng lệnh “python ClientLauncher.py 127.0.0.1 1025 5008 movie.Mjpeg”, ta thu được giao diện sau:



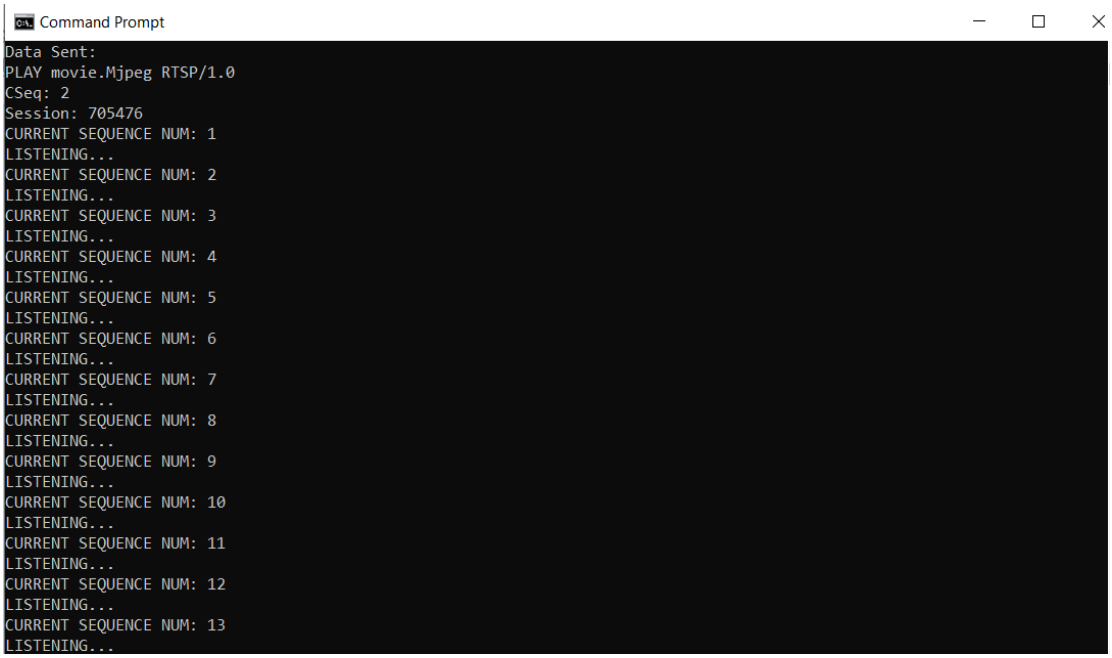
Ta kích hoạt nút Setup:

```
C:\Users\PC\Downloads\src\src>python Server.py 1025
Data received:
SETUP movie.Mjpeg RTSP/1.0
CSeq: 1
Transport: RTP/UDP; client_port= 5008
processing SETUP
```

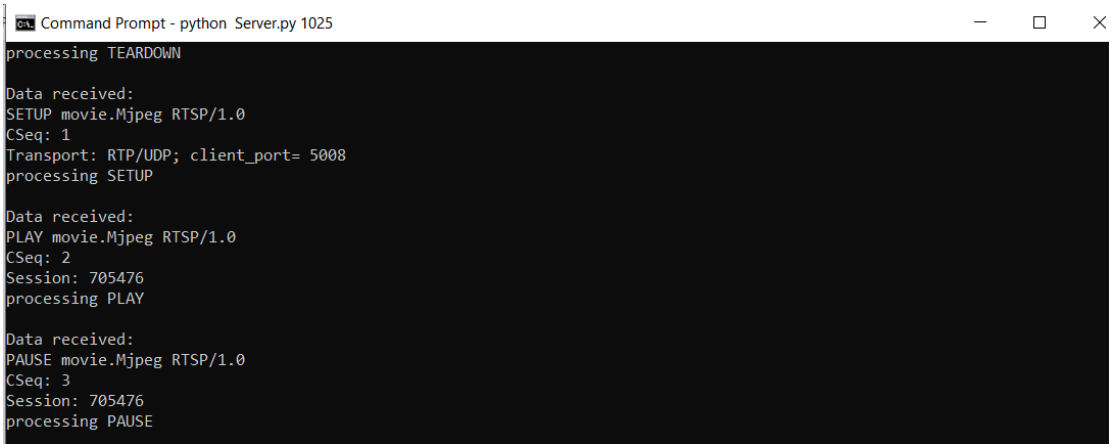
Tiếp đó ta kích nút PLAY:


RTPClient
Setup
Play
Pause
Teardown

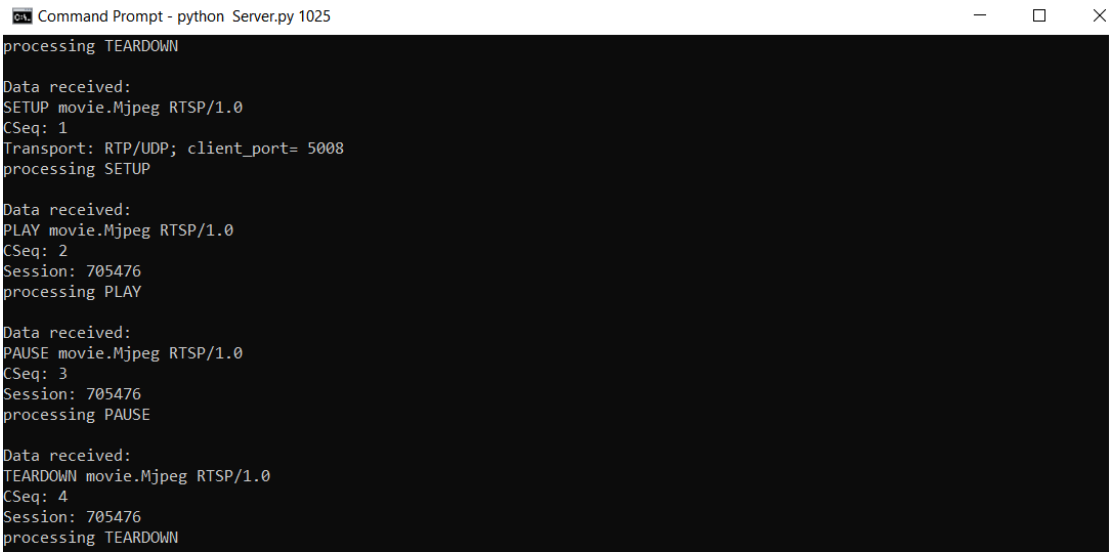

Command Prompt - python Server.py 1025
processing TEARDOWN
Data received:
SETUP movie.Mjpeg RTSP/1.0
CSeq: 1
Transport: RTP/UDP; client_port= 5008
processing SETUP
Data received:
PLAY movie.Mjpeg RTSP/1.0
CSeq: 2
Session: 705476
processing PLAY

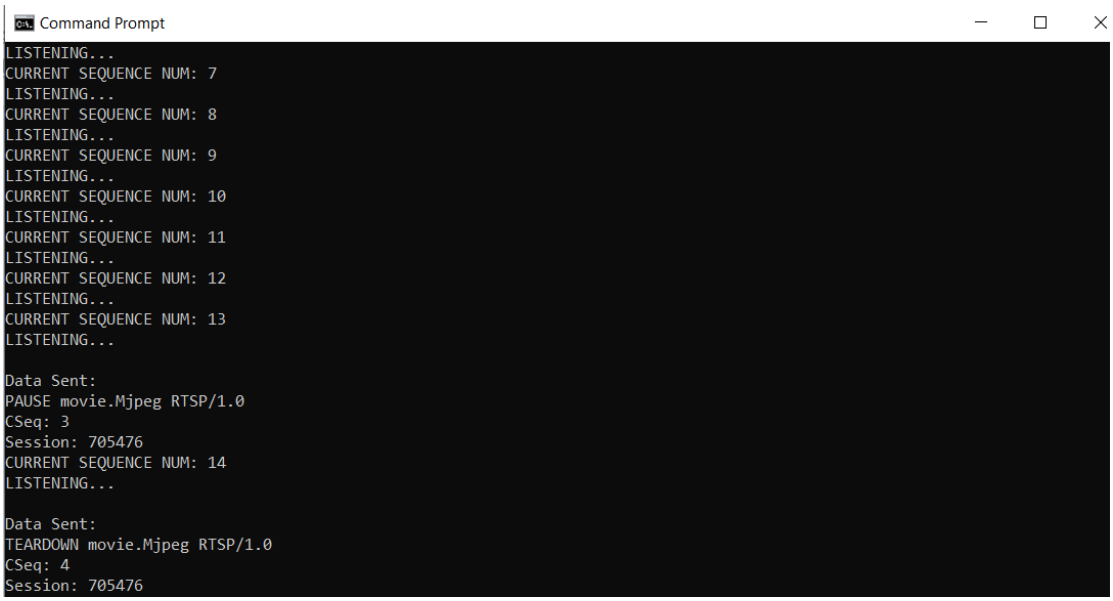


Sau đó ta bấm nút PAUSE:



Cuối cùng ta nhấn nút TEARDOWN để ngắt kết nối:



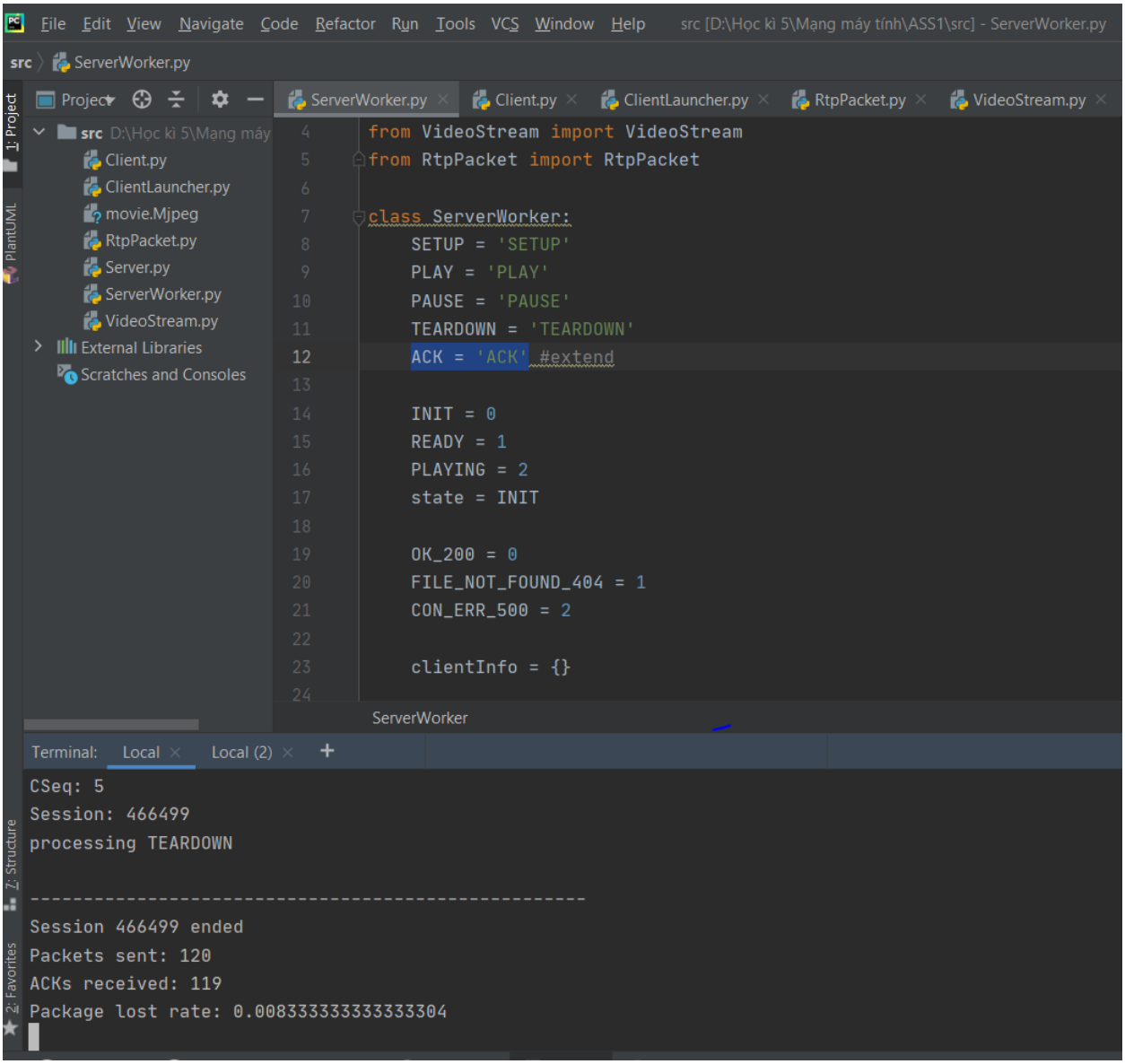


Chương trình được thiết kế để khi chưa nhấn nút SETUP hoặc PAUSE thì người sử dụng không thể PLAY video

Và chương trình sẽ báo lỗi nếu TEARDOWN chương trình khi chưa nhấn PLAY

Extend:

Nhóm đã tính được tỉ lệ mất gói (lost rate):



Khi ta làm ngắt quãng việc truyền dữ liệu có khả năng gây mất gói. Sau khi người dừng nhất TEARDOWN thì chương trình sẽ thông báo số gói gửi và nhận được cùng với tỉ lệ mất gói.

5 .User manual

Chương trình gồm 6 files:

> Client.py

> ClientLauncher.py

> Server.py

> ServerWorker.py

> RtpPacket.py

Để thực thi chương trình ta làm các bước sau

1. Chạy chương trình Server.py trên Terminal để bắt đầu server:
E.g: python Server.py server_port

Server_port là cổng mà máy chủ của bạn nghe các kết nối RTSP đến.

> Trong project này chúng ta sẽ cho giá trị > 1024

> Ở phần demo, nhóm sẽ cho port có giá trị là 1025

> Standard RTSP port là 554

```
C:\Users\PC\Downloads\src\src>python Server.py 1025
```

2. Chạy chương trình ClientLauncher.py trên Client Terminal để bắt đầu client:
E.g: python ClientLauncher.py server_host server_port PRT_port video_file

Server_host là IP của địa chỉ máy cục bộ (ta có thể sử dụng "127.0.0.1")

RTP_port là cổng mà RTP packet được nhận (ở đây là ta sử dụng cổng "5008")

Video_file là tên của video file mà ta muốn chạy (ở đây ta sử dụng "video.Mjpeg")

```
C:\Users\PC\Downloads\src\src>python ClientLauncher.py 127.0.0.1 1025 5008 movie.Mjpeg
```