

HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY
KHOA KHOA HỌC - KỸ THUẬT MÁY TÍNH



PROJECT REPORT

MẠNG MÁY TÍNH

Lecturer: Bùi Xuân Giang

Phan Thanh Trường	1814578
Nguyễn Văn Tiến	1814320
Lê Trung Sơn	1810482
Phạm Quốc Trung	1814522

Table of contents

Requirement analysis	3
Introduce the RTSP and RTP:	3
RTSP (Real Time Streaming Protocol):	3
RTP (Real-time Transport Protocol):	4
Function Description	6
Send RTSP request	6
Parse RTSP reply	8
Open RTP port	9
RTP packet	9
Extend	11
Class Diagram	23
Evaluation	25
User manual	25
References	29

I. Requirement analysis

Trong xã hội hiện đại thì việc streaming video, phát các video trực tiếp, live stream đã dần trở nên phổ biến và cấp thiết trong xã hội. Với sự phát triển xã hội thì việc đòi hỏi streaming video nhanh, mượt, tối ưu ngày càng đầy mạnh.

Kỹ thuật streaming video được thực hiện qua các bước:

- Một phần mềm (process) ở máy khách (client) cần kết nối và muốn xem một video có tồn tại ở một server.
- Yêu cầu streaming video đó được client gửi đến server.
- Process nhận nhiệm vụ streaming video ở trên máy server có nhiệm vụ chia sẻ video cho client bằng cách tách nhỏ thành các frame và gửi cho client theo các giao thức ràng buộc về thời gian (RTP, RTSP, có thể có RCTP)
- Các frame được gửi đến client dưới các vùng nhớ đệm (buffer) và nội dung các frame được giải mã (decode), sau đó hiển thị qua chương trình video ở máy client.

Yêu cầu của assignment này được đặt ra là:

- Mục tiêu cần hiện thực là phần Client và RtpPacket.

Trong phần Client (hay được hiểu rằng người dùng gửi yêu cầu đi tới server), cần hiện thực các hàm gửi yêu cầu như setup, play, pause, teardown để nhận lại data từ server.

Tương tự thế trong RtpPacket, class này dùng để xử lý RTP packet và chúng ta cần phải hiện thực phần constructor thứ nhất là RTP-Packetization của video data (constructor de-Packetize đã được hiện thực).

Phần extend:

Mục tiêu:

- 1) Sử dụng giao thức RTP để tính toán xem tỉ lệ loss rate, miss rate của các RTP packet (nguyên nhân vì RTP được sử dụng trên UDP nên chúng xuất hiện tình trạng bị miss các data frame, hoặc các frame bị hư hỏng khi gửi; đồng thời RTP cũng được dùng để đo tốc độ truyền gửi dữ liệu)
- 2) Yêu cầu hiện thực bắt buộc phải sử dụng yêu cầu SETUP trong RTSP, đồng thời đưa ra giải pháp xử lý cho việc người dùng không được cấp quyền gửi yêu cầu SETUP mà phải thực hiện tự động.
- 3) Yêu cầu hiện thực yêu cầu DESCRIBE.

II. Introduce the RTSP and RTP:

1. RTSP (Real Time Streaming Protocol):

a) Giới thiệu về RTSP:

RTSP là một giao thức truyền tin ở thời gian thực, được sử dụng ở tầng ứng dụng (Application-layer trong mô hình TCP/IP), được sử dụng chủ yếu để thiết kế các hệ thống truyền thông và giải trí, điều khiển các máy chủ chứa dữ liệu truyền tin đa phương tiện (streaming video, multimedia).

RTSP về hình thức khá giống HTTP, đều sử dụng TCP là giao thức hỗ trợ để duy trì một kết nối đầu cuối (client - server), tuy nhiên HTTP là giao thức không có trạng thái thì RTSP ngược lại, nó là một giao thức có trạng thái. Các phiên giao dịch giữa client-server đều có một định danh riêng (gọi là session).

RTSP cũng được cung cấp một port mặc định là 554.

b) Kỹ thuật sử dụng RTSP trong streaming video:

Để thực hiện kỹ thuật streaming video dùng RTSP, nhất thiết client phải gửi cái yêu cầu (request) lên server (streaming server) các request sau và theo một trình tự bắt buộc.

Đầu tiên client gửi lên server một yêu cầu bất kì, kèm theo là link đến video cần xem. Sau đó server sẽ xử lý và gửi trả về mã chấp nhận (200 - OK) hoặc từ chối (404, 403,...) cho client. Giả sử ở đây chúng ta xét đến trường hợp server chấp nhận link mà client gửi đến (gửi trả tín hiệu 200-OK).

Sau đó, client tiếp tục gửi đến server yêu cầu DESCRIBE để server phân tích đường link vừa nhận được. Thông điệp được server gửi trả về client sẽ đi qua port 554 (cổng mặc định dành cho RTSP). Port 554 là cổng có thể sử dụng cho cả TCP và UDP. Thông điệp được gửi từ server sẽ bao gồm bản miêu tả chi tiết phiên giao dịch (Session Description Protocol) gồm có streamid của luồng video và streamid của luồng âm thanh (điều quan trọng nhất). Ngoài ra, thông điệp này cũng có thể có các đường dẫn khác (link) tốt hơn cho video cần streaming.

Client sau khi nhận được thông điệp từ server (thông điệp về DESCRIBE), client tiếp tục gửi yêu cầu SETUP đến server. Yêu cầu SETUP chỉ ra cách mà một luồng byte stream bắt buộc được truyền đi như thế nào. Yêu cầu SETUP bao gồm một link dẫn đến video cần streaming, và phần đặc tả cho phần transport. Thông tin đặc tả bao gồm: một cổng nhận các gói tin audio và video (dưới dạng các gói tin RTP), một cổng nhận meta information (dạng gói tin RSTP). Server sẽ nhận thông tin yêu cầu và tiến hành điền vào các phần còn thiếu của mình, cấu hình sẵn sàng các luồng byte stream.

Đặc biệt là yêu cầu SETUP phải hoàn thành trước khi yêu cầu PLAY được client gửi đến.

Yêu cầu PLAY được client gửi đến server, sau đó server tiến hành gửi các data frame theo các luồng đã được chuẩn bị sẵn, các data frame này được lưu trong bộ nhớ đệm (buffer) của máy client, sau đó được giải mã (decode) và hiển thị bởi trình ứng dụng phát video và âm thanh (ví dụ Window Media Player, VLC,...). Yêu cầu PLAY trên căn bản khá giống với các yêu cầu trước, cũng có một đường dẫn (link) đến video cần streaming.

Ngoài ra trong quá trình streaming video, người dùng còn có thể gửi đến yêu cầu PAUSE hay STOP hay TEARDOWN.

Đối với yêu cầu PAUSE, server sẽ tạm dừng một hay nhiều các frame đang được gửi đến cho client và có thể khôi phục lại nếu yêu cầu PLAY được gửi đến.

Còn đối với STOP hay TEARDOWN, chúng đều có bản chất là yêu cầu hủy phiên giao dịch hiện tại của client-server, và server sẽ ngừng hẳn việc gửi các dataframe đến cho client.

2. RTP (Real-time Transport Protocol):

a) Giới thiệu:

RTP là giao thức truyền tải thời gian thực, dùng để chuyển các tệp tin, video, âm thanh qua mạng IP. Tương tự như RTSP, RTP cũng được sử dụng ở tầng ứng dụng (Application Layer). RTP được sử dụng rộng rãi trong các hệ thống truyền thông và giải trí liên quan đến các streaming media: như gọi điện thoại, các ứng dụng dạy học trực tuyến, các dịch vụ truyền hình và các tính năng push-to-talk dựa trên nền web.

Khác với RTSP có thể sử dụng ở cả TCP và UDP, RTP chỉ sử dụng được trên giao thức UDP (User Datagram Protocol). RTP cũng được sử dụng chủ yếu để giám sát số liệu truyền tải, kiểm tra chất lượng dịch vụ và đồng bộ đa luồng.

RTP thường được sử dụng đi kèm với RTSP và chuẩn RTP được định nghĩa là một cặp hai giao thức gồm RTP đi kèm làm việc với RTCP.

b) Miêu tả RTP:

RTP packet header							
bit offset	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 8: Header của RTP Packet

Phiên (Session): Một phiên RTP được thiết lập cho mỗi luồng dữ liệu. Một phiên bao gồm một địa chỉ IP với một cặp cổng của giao thức RTP và RTCP. Ví dụ, các luồng video và audio sẽ có các phiên RTP khác nhau, bên nhận sẽ nhận một cách riêng biệt giữa dữ liệu video và audio thông qua 2 cổng khác nhau cho 2 giao thức RTP và RTCP. Thường thì số hiệu cổng của RTP là một số chẵn trong khoảng 1024 tới 65535 và cổng của RTCP là một số lẻ kế tiếp.

Kích thước nhỏ nhất của một header của gói tin RTP là 12 bytes. Sau phần header chính, là phần header mở rộng và không cần thiết phải có phần header này. Chi tiết các trường trong một header như sau:

- Version (2 bits): Cho biết phiên bản của giao thức này. Phiên bản hiện tại là phiên bản 2.
- P (Padding) (1 bit): Cho biết số các byte mở rộng cần thêm vào cuối của gói tin RTP. Ví dụ trong trường hợp ta muốn sử dụng các thuật toán mã hóa, ta có thể thêm vào một số byte vào phần kết thúc của gói tin để tiến hành mã hóa frame trên đường truyền.
- X (Extension) (1bit): Cho biết có thêm phần header mở rộng vào sau phần header chính hay không.
- CC (CSRC Count) (4 bit): Chứa con số định danh CSRC cho biết kích thước cố định của header.
- M (Marker) (1 bit): Cho biết mức của ứng dụng và được định nghĩa bởi một profile. Nếu được thiết lập, có nghĩa là dữ liệu hiện tại đã được tính toán chi phí một cách thích hợp
- PT (Payload Type) (7 bit): Cho biết định dạng của file video. Đây là một đặc tả được định nghĩa bởi một profile RTP.
- Sequence Number (16 bits): số hiệu của frame. Và sẽ được tăng lên 1 đơn vị cho mỗi gói tin RTP trước khi gửi và được sử dụng bởi bên nhận để dò ra các gói bị lạc và có thể phục hồi lại gói có số thứ tự đó.
- Timestamp (32 bits): Được sử dụng thông báo cho bên nhận biết để phát lại frame này trong khoảng thời gian thích hợp.

III. Function Description

1. Send RTSP request

a) Mô tả

Dùng để hiển thị cho bên Server những thông tin, yêu cầu của bên Client như SETUP, PLAY, PAUSE và TEARDOWN. Bên cạnh đó còn hiển thông tin video đang phát và đếm số lần gửi yêu cầu của Client tới Server.

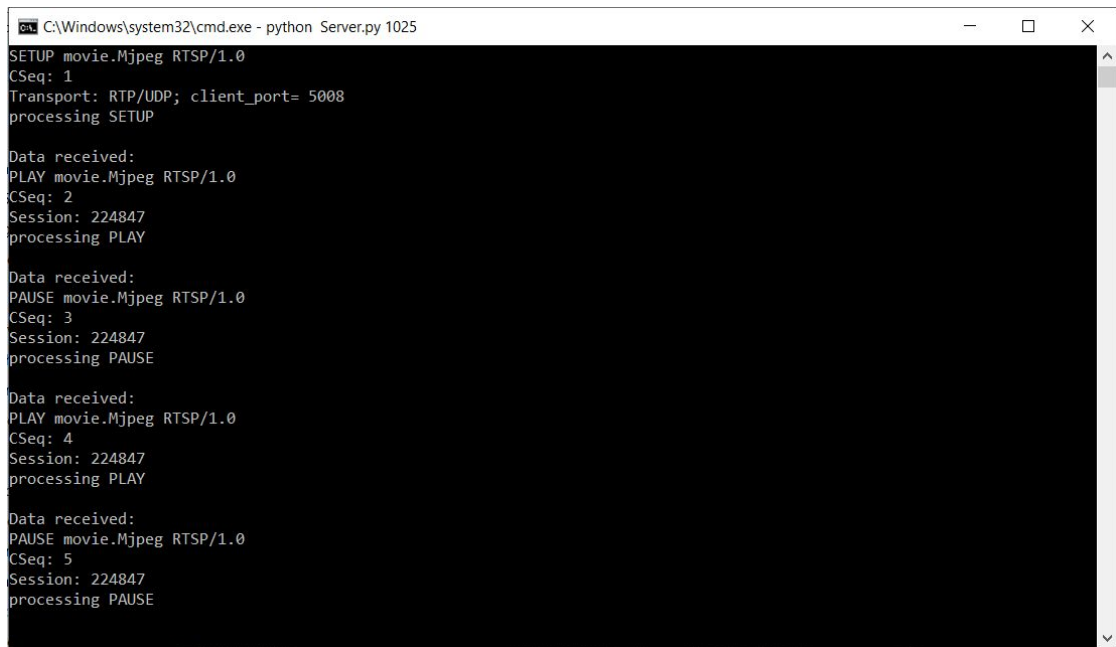
b) Hiện thực

```
def sendRtspRequest(self, requestCode):
    """Send RTSP request to the server."""
    if requestCode == self.SETUP and self.state == self.INIT:
        threading.Thread(target=self.recvRtspReply).start()
        self.rtpSeq = 1
        request = "%s %s %s" % (self.SETUP_STR, self.fileName, self.RTSP_VER)
        request += "\nCSeq: %d" % self.rtpSeq
        request += "\nTransport: %s; client_port= %d" % (self.TRANSPORT, self.rtpPort)
        self.requestSent = self.SETUP
    elif requestCode == self.PLAY and self.state == self.READY:
        self.rtpSeq += 1
        request = "%s %s %s" % (self.PLAY_STR, self.fileName, self.RTSP_VER)
        request += "\nCSeq: %d" % self.rtpSeq
        request += "\nSession: %d" % self.sessionId
        self.requestSent = self.PLAY
    elif requestCode == self.PAUSE and self.state == self.PLAYING:
        self.rtpSeq += 1
        request = "%s %s %s" % (self.PAUSE_STR, self.fileName, self.RTSP_VER)
        request += "\nCSeq: %d" % self.rtpSeq
        request += "\nSession: %d" % self.sessionId
        self.requestSent = self.PAUSE
    elif requestCode == self.TEARDOWN and not self.state == self.INIT:
        self.rtpSeq += 1
        request = "%s %s %s" % (self.TEARDOWN_STR, self.fileName, self.RTSP_VER)
        request += "\nCSeq: %d" % self.rtpSeq
        request += "\nSession: %d" % self.sessionId
        self.requestSent = self.TEARDOWN
    else:
        return
    self.rtpSocket.send(request.encode("utf-8"))
    print('\nData sent:\n' + request)
```

Hình 1.1: Hiện thực code hàm sendRtspRequest

Sau mỗi lần **request** được gửi từ client thì *self.requestCode* sẽ tăng thêm một đơn vị, Bên cạnh đó ta sẽ tạo form của *request* để xuất ra màn hình trạng thái khi Server được yêu cầu.

c) Kết quả



```
C:\Windows\system32\cmd.exe - python Server.py 1025
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: 224847
processing PLAY

Data received:
PAUSE movie.Mjpeg RTSP/1.0
CSeq: 3
Session: 224847
processing PAUSE

Data received:
PLAY movie.Mjpeg RTSP/1.0
CSeq: 4
Session: 224847
processing PLAY

Data received:
PAUSE movie.Mjpeg RTSP/1.0
CSeq: 5
Session: 224847
processing PAUSE
```

Hình 1.2: Kết quả hiển thị trạng thái của Server

2. Parse RTSP reply

a) Mô tả

Dùng để cập nhật trạng thái của Client sau mỗi lần request của Client.

b) Hiện thực

Ta sẽ cập nhật trạng thái *self.state* theo từng trường hợp cụ thể

```
if self.sessionId == session:
    if int(lines[0].split(' ')[1]) == 200:
        if self.requestSent == self.SETUP:
            self.state = self.READY
            self.openRtpPort()
        elif self.requestSent == self.PLAY:
            self.state = self.PLAYING
        elif self.requestSent == self.PAUSE:
            self.state = self.READ
            self.playEvent.set()
        elif self.requestSent == self.TEARDOWN:
            self.state = self.INIT
            self.teardownAcked = 1
```

Hình 2.1: Phần hiện thực *def parseRtspReply()* của class Client

3. Open RTP port

a) Mô tả

Dùng để mở RTP Socket đã được liên kết với một cổng port ở trước đó. Tạo một datagram socket mới để nhận RTP packet từ server, sau đó truyền thời gian chờ cho nó là 0,5s. Sau đó liên kết tới địa chỉ bằng RTP port được client user cung cấp trước đó.

b) Hiện thực

```
def openRtpPort(self):
    self.rtpSocket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    self.rtpSocket.settimeout(0.5)
    try:
        self.rtpSocket.bind((self.serverAddr, self.rtpPort))
    except:
        tkinter.messagebox.showwarning('Unable to Bind', 'Unable to bind PORT=%d' % self.rtpPort)
```

Hình 3.1: Phần hiện thực `def openRtpPort()` của class Client

4. RTP packet

a) Mô tả

Ở phần này ta sẽ set phần RTP packet cho phù hợp với từng thành phần.

Theo quan sát thì RTP packet bao gồm 3 tầng, mỗi tầng cần 32 bit để lưu

=> Total = 32*3 = 96 bit để lưu trữ => cần tổng cộng 12 byte (96bit / 8) để lưu trữ.

Chính vì thế HEADER_SIZE = 12

RTP packet header																																
Octet	0							1							2							3										
Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	Version=2		P	X	CC			M	PT							Sequence number																
32	Timestamp																															
64	synchronization source (SSRC) identifier																															
96	contributing source (CSRC) identifiers																															
	...																															

Hình 4.1: RTP packet header

(RTP Giao thức truyền tải cho các ứng dụng thời gian thực, RFC 1889).

- + RTP-version = 2 => cần 2 bit để lưu nó.

Vì mỗi khung sẽ gồm 1 byte (8 bit) mà RTP-version chỉ cần 2 bits nên ta sẽ dịch bit về đầu => **header[0] = (header[0] | version << 6) & 0xC0** // 2 bit

- + Padding (P), Extension(X), number of contributing sources (CC), và marker (M) fields đều set là 0 trong lab nên chúng ta chỉ cấp cho chúng 1 bit.

P và E header sẽ lần lượt được dời bit như sau:

header[0] = (header[0] | padding << 5); // 1 bit

header[0] = (header[0] | extension << 4); // 1 bit

CC header sẽ được dời:

header[0] = (header[0] | (cc & 0x0F)); // 4 bit

M header sẽ dời vào bit đầu của header mới:

header[1] = (header[1] | marker << 7); // 1 bit

+ **Payload type (PT)** sẽ là phần còn lại trong header 1:

header[1] = (header[1] | (pt & 0x7f)); // 7 bit

+ **Sequence number** chiếm tổng cộng 16 bits:

header[2] = (seqnum & 0xFF00) >> 8; // 8 bit đầu trong header 2

header[3] = (seqnum & 0xFF); // 8 bit sau trong header 3

+ **Timestamp** chiếm tổng cộng 32 bits:

header[4] = (timestamp >> 24); // 8 bit đầu trong header 4

header[5] = (timestamp >> 16) & 0xFF; // 8 bit sau trong header 5

header[6] = (timestamp >> 8) & 0xFF; // 8 bit sau trong header 6

header[7] = (timestamp & 0xFF); // 8 bit sau trong header 7

+ **Set the source identifier (SSRC)** chiếm tổng cộng 32 bits:

header[8] = (ssrc >> 24); // 8 bit đầu trong header 8

header[9] = (ssrc >> 16) & 0xFF; // 8 bit sau trong header 9

header[10] = (ssrc >> 8) & 0xFF; // 8 bit sau trong header 10

header[11] = ssrc & 0xFF; // 8 bit sau trong header 11

b) Hiện thực

```

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] = version + padding + extension + cc + seqnum + marker + pt + ssrc
    header[0] = (header[0] | version << 6) & 0xC0; # 0011000000 # 2 bits
    header[0] = (header[0] | padding << 5); # 1 bit
    header[0] = (header[0] | extension << 4); # 1 bit
    header[0] = (header[0] | (cc & 0x0F)); # 0000001111 # 4 bits

    header[1] = (header[1] | marker << 7); # 1 bit
    header[1] = (header[1] | (pt & 0x7F)); # 127 # 7 bits

    header[2] = (seqnum & 0xFF00) >> 8; # 16 bits total, this is first 8
    header[3] = (seqnum & 0xFF); # 11111111 # 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

```

Hình 4.2: Hiện thực code class RptPacket

c) Kết quả:

Từ đó ta có thể get được các giá trị như **decode**, **version**, **seqNum**, **timestamp**, **payloadType**, **getPayload**, **getPacket**.

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

def version(self):
    """Return RTP version."""
    return int(self.header[0] >> 6)

def seqNum(self):
    """Return sequence (frame) number."""
    seqNum = self.header[2] << 8 | self.header[3]
    return int(seqNum)

def timestamp(self):
    """Return timestamp."""
    timestamp = self.header[4] << 24 | self.header[5] << 16 | self.header[6] << 8 | self.header[7]
    return int(timestamp)

def payloadType(self):
    """Return payload type."""
    pt = self.header[1] & 127
    return int(pt)

def getPayload(self):
    """Return payload."""
    return self.payload

def getPacket(self):
    """Return RTP packet."""
    return self.header + self.payload
```

Hình 4.3: Các function của class RtpPacket

5. Extend

a) RTP packet loss rate and video data rate

- Mô tả:

- + *RTP packet loss rate*: Vì *Sequence number* sẽ được tăng lên cho mỗi gói dữ liệu RTP được gửi đi và thường sẽ được dùng để người nhận gói tin phát hiện tình trạng mất gói. Nên ta sẽ dựa vào nó để tính tỉ lệ mất gói khi truyền đi.
- + *Video data rate*: Ta sẽ dựa vào tổng size gói được truyền đi, sau đó sẽ tính khoảng thời gian tương ứng với số gói đã truyền đi để được tỉ lệ truyền data của video.

- Hiện thực:

Khởi tạo những giá trị ban đầu:

```
class Client:
    INIT = 0
    READY = 1
    PLAYING = 2
    state = INIT

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

    SETUP_STR = 'SETUP'
    PLAY_STR = 'PLAY'
    PAUSE_STR = 'PAUSE'
    TEARDOWN_STR = 'TEARDOWN'
    RTSP_VER = "RTSP/1.0"
    TRANSPORT = "RTP/UDP"

    COUNT = 0
    TOTALLOSSPACKET = 0
    TOTALSIZE = 0
    PASTSTOP = 0
    TIMEEND = 0
    TIMESTART = 0
```

Hình 5.1.1: Khởi tạo các giá trị cần thiết

Với:

COUNT = 0 để kiểm tra xem các gói có truyền liên tục hay không

TOTALLOSSPACKET = 0 tính tổng các gói mất

TOTALSIZE = 0 tính tổng size gói đã truyền đi

PASTSTOP = 0 gói cuối cùng truyền

TIMEEND = 0 time kết thúc truyền gói cuối cùng

TIMESTART = 0 time bắt đầu truyền gói

+ RTP packet loss rate:

Sau khi khởi tạo, ta sẽ tính số lượng gói mất khi truyền đi:

```

107     def listenRtp(self):
108         """Listen for RTP packets."""
109         self.TIMESTART = time.time()
110         while True:
111             try:
112                 data = self.rtpSocket.recv(20480)
113                 if data:
114                     rtpPacket = RtpPacket()
115                     rtpPacket.decode(data)
116                     self.TOTALSIZE += len(data)
117                     currFrameNbr = rtpPacket.seqNum()
118                     self.COUNT += 1
119                     if self.COUNT != currFrameNbr:
120                         self.TOTALLOSSPACKET += 1
121                         self.COUNT = currFrameNbr
122                     print("Current Seq Num: " + str(currFrameNbr))
123                     if currFrameNbr > self.frameNbr: # Discard the late packet
124                         self.frameNbr = currFrameNbr
125                     self.updateMovie(self.writeFrame(rtpPacket.getPayload()))
126                     if self.COUNT == 500:
127                         self.TIMEEND = time.time()

```

Hình 5.1.2: Thực thi phần đếm số lượng packet mất

Ta sẽ dùng biến *COUNT* để kiểm tra sự liên tục của *currFrameNbr* nếu mà hai biến này khác nhau (line 119) thì *TOTALLOSSPACKET* sẽ được tăng lên một giá trị (line 120) và biến *COUNT* sẽ được set về đúng số thứ tự gói hiện tại (line 121).

```

128     except:
129         # Stop listening upon requesting PAUSE or TEARDOWN
130         if self.playEvent.isSet():
131             if self.TIMEEND == 0:
132                 self.TIMEEND = time.time()
133                 print("Rate loss packet: "
134                       + str(round((self.TOTALLOSSPACKET/(self.COUNT - self.PASTSTOP))*100, 2)) + "%")
135                 print("Video data rate: "
136                       + str(round((self.TOTALSIZE/(self.TIMEEND - self.TIMESTART)), 2)) + str("byte/s"))
137
138                 self.PASTSTOP = self.COUNT
139                 self.TOTALLOSSPACKET = 0
140                 self.TIMEEND = 0
141                 self.TOTALSIZE = 0
142                 break
143             if self.teardownAcked == 1:
144                 self.rtpSocket.shutdown(socket.SHUT_RDWR)
145                 self.rtpSocket.close()
146                 break

```

Hình 5.1.3: Xuất ra màn hình tỉ lệ mất gói

Tính và in ra màn hình tỉ lệ mất gói RTP (line 133, 134).

+ Video data rate:

Theo *Hình 5.1.2* ở trên thì ta sẽ set *TIMESTART* từ lúc ta gọi hàm *listenRtp* (line 109), khi data được truyền ta sẽ tính size gói data đó ra và cộng dồn vào *TOTALSIZE* (line 116). Sau khi biến *COUNT* bằng 500 (line 126) tức đã truyền hết các gói, ta sẽ set *TIMEEND* (line 127)

Theo *Hình 5.1.3* ở trên ta sẽ tính được tốc độ truyền gói theo công thức ở line 135, 136.

- **Kết quả:**

Trường hợp chạy hết và ấn nút *PAUSE*

```
C:\Windows\system32\cmd.exe - python ClientLaunch
Current Seq Num: 479
Current Seq Num: 480
Current Seq Num: 481
Current Seq Num: 482
Current Seq Num: 483
Current Seq Num: 484
Current Seq Num: 485
Current Seq Num: 486
Current Seq Num: 487
Current Seq Num: 488
Current Seq Num: 489
Current Seq Num: 490
Current Seq Num: 491
Current Seq Num: 492
Current Seq Num: 493
Current Seq Num: 494
Current Seq Num: 495
Current Seq Num: 496
Current Seq Num: 497
Current Seq Num: 498
Current Seq Num: 499
Current Seq Num: 500

Data sent:
PAUSE movie.Mjpeg RTSP/1.0
CSeq: 3
Session: 370682
Rate loss packet: 0.0%
Video data rate: 136742.93byte/s
```

Hình 5.1.4: Kết quả 1

```
C:\Windows\System32\cmd.exe - python C
Current Seq Num: 479
Current Seq Num: 480
Current Seq Num: 481
Current Seq Num: 482
Current Seq Num: 483
Current Seq Num: 484
Current Seq Num: 485
Current Seq Num: 486
Current Seq Num: 487
Current Seq Num: 488
Current Seq Num: 489
Current Seq Num: 490
Current Seq Num: 491
Current Seq Num: 492
Current Seq Num: 493
Current Seq Num: 494
Current Seq Num: 495
Current Seq Num: 496
Current Seq Num: 497
Current Seq Num: 498
Current Seq Num: 499
Current Seq Num: 500

Data sent:
PAUSE movie.mjpeg RTSP/1.0
CSeq: 3
Session: 310467
Rate loss packet: 0.2%
Video data rate: 135620.69byte/s
```

Hình 5.1.5: Kết quả 2

Với kết quả của *Hình 5.1.4* ta không có sự thiếu hụt về vấn đề truyền data còn *Hình 5.1.5* thì ngược lại, từ đó ta sẽ có tỉ lệ mất gói của toàn bộ quá trình truyền data.

Trường hợp PAUSE từng đoạn nhỏ

```
Current Seq Num: 26
Current Seq Num: 27
Current Seq Num: 28
Current Seq Num: 29
Current Seq Num: 30

Data sent:
PAUSE movie.mjpeg RTSP/1.0
CSeq: 3
Session: 964054
Rate loss packet: 0.0%
Video data rate: 79228.88byte/s

Data sent:
PLAY movie.mjpeg RTSP/1.0
CSeq: 4
Session: 964054
Current Seq Num: 31
Current Seq Num: 32
Current Seq Num: 33
Current Seq Num: 34
Current Seq Num: 35
Current Seq Num: 36
Current Seq Num: 37
Current Seq Num: 38
Current Seq Num: 39
Current Seq Num: 40
Current Seq Num: 41
Current Seq Num: 42
Current Seq Num: 43

Data sent:
PAUSE movie.mjpeg RTSP/1.0
CSeq: 5
Session: 964054
Rate loss packet: 0.0%
Video data rate: 70009.62byte/s
```

Hình 5.1.6: Kết quả 3

Ta sẽ nhận được kết quả cho từng đợt truyền gói cụ thể. (Như hình 5.1.6 ở trên)

b) User interface update

- **Mô tả:** Để thu gọn xuống còn 3 buttons (PLAY, PAUSE, STOP) thì chúng ta cần khởi tạo SETUP (SETUP là bắt buộc trong RTSP) và giá trị của các biến cần thiết trong quá trình khởi tạo giao diện, tức là ta sẽ setup sau khi connect tới server thành công.

- **Hiện thực:**

```
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

    """Setup after making connection to server."""
    if self.state == self.INIT:
        self.sendRtspRequest(self.SETUP)
```

Hình 5.2.1: Cập nhật trạng thái khi connect tới server thành công

Song với việc cập nhật trạng thái, ta sẽ chỉnh lại các buttons từ 4 xuống còn 3:

```
def createWidgets(self):
    """Build GUI."""
    # Create Play button
    self.start = Button(self.master, width=20, padx=3, pady=3)
    self.start["text"] = "Play"
    self.start["command"] = self.playMovie
    self.start.grid(row=1, column=0, padx=2, pady=2)

    # Create Pause button
    self.pause = Button(self.master, width=20, padx=3, pady=3)
    self.pause["text"] = "Pause"
    self.pause["command"] = self.pauseMovie
    self.pause.grid(row=1, column=1, padx=2, pady=2)

    # Create Stop button
    self.stop = Button(self.master, width=20, padx=3, pady=3)
    self.stop["text"] = "Stop"
    self.stop["command"] = self.stopMovie
    self.stop.grid(row=1, column=2, padx=2, pady=2)

    # Create a label to display the movie
    self.label = Label(self.master, height=19)
    self.label.grid(row=0, column=0, columnspan=4, sticky=W+E+N+S, padx=5, pady=5)
```

Hình 5.2.2: Cập nhật lại các buttons

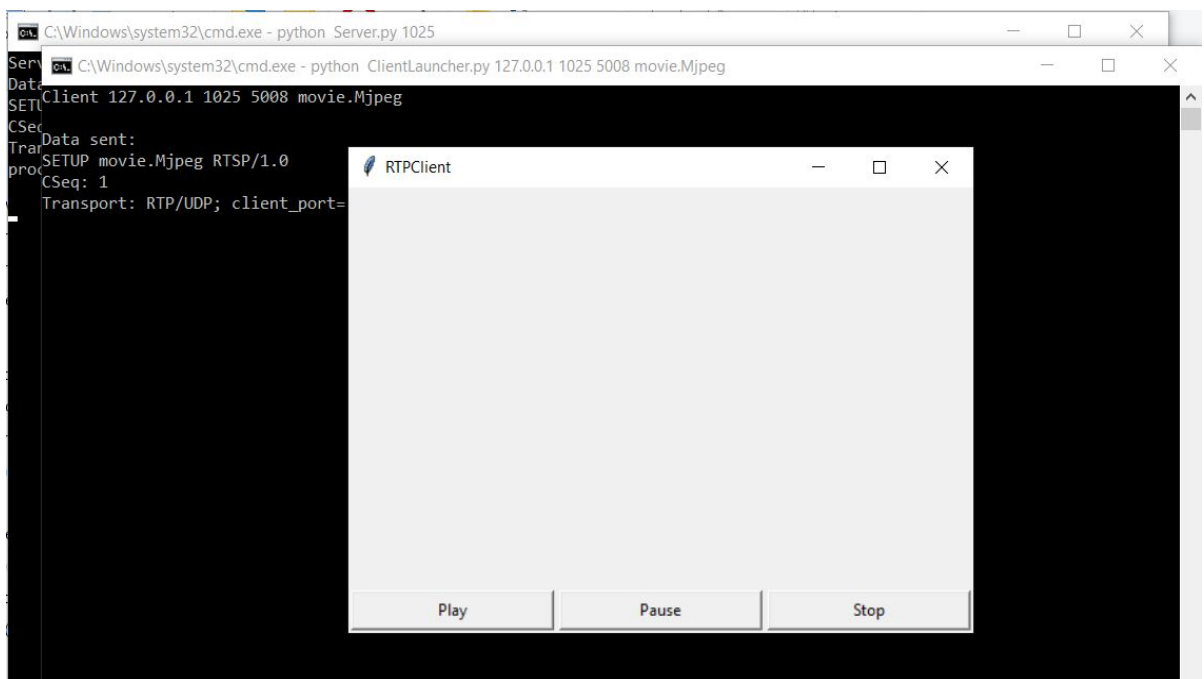
Chúng ta sẽ tạo thêm 1 hàm stopMovie() được gán cho nút Stop với chức năng hiện bảng xác nhận sau khi nhấn stop. Nếu nhấn thoát, TEARDOWN sẽ được gửi đi và trình phát sẽ tự động tắt sau đó, ngược lại, trình phát sẽ dừng phát mà không tắt giao diện.

```
def stopMovie(self):
    self.pauseMovie()
    if tkinter.messagebox.askokcancel("Quit?", "Are you sure you want to quit?"):
        self.exitClient()
    else:
        self.sendRtspRequest(self.TEARDOWN)

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

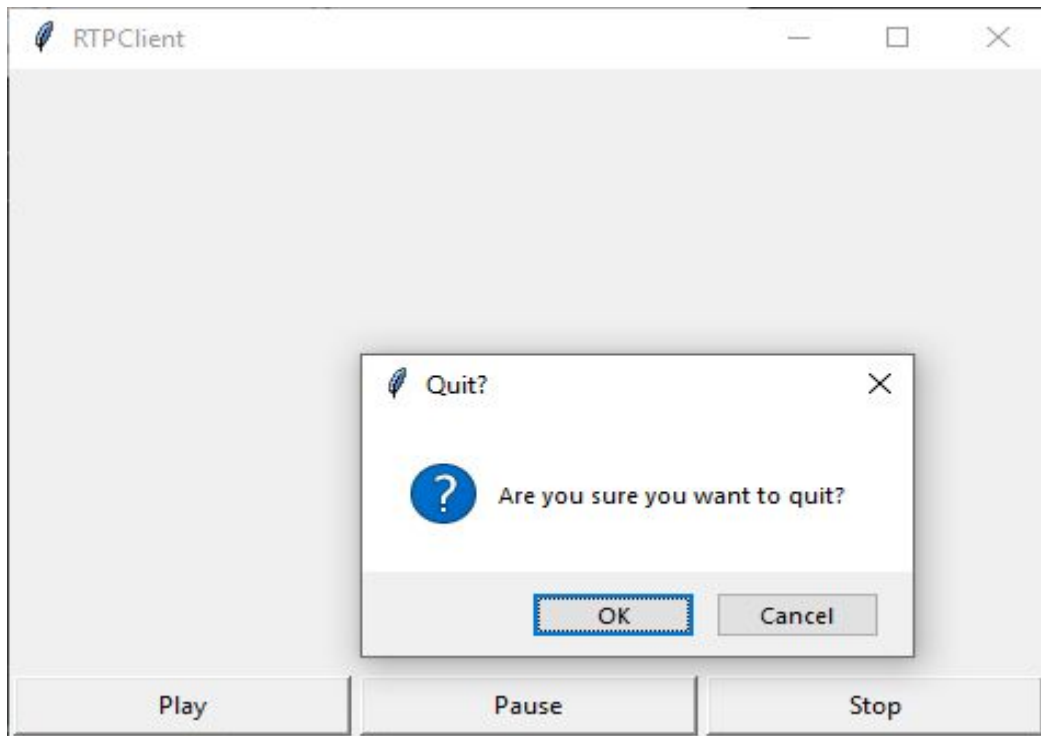
Hình 5.2.3: Nội dung hàm stopMovie

- Kết quả hiện thực:



Hình 5.2.4: UI mới sau khi sửa

- **Kết quả khi nhấn stop:**



Hình 5.2.5: Kết quả khi nhấn nút Stop

c) **Describe-request**

- **Mô tả:** Thực hiện chức năng DESCRIBE nhằm gửi yêu cầu tới server về luồng media và server sẽ trả về thông tin của phiên hiện tại bao gồm loại luồng, loại mã hóa và các thông tin khác
- **Hiện thực:**
 - + Trong giao diện trình phát, thêm nút Describe

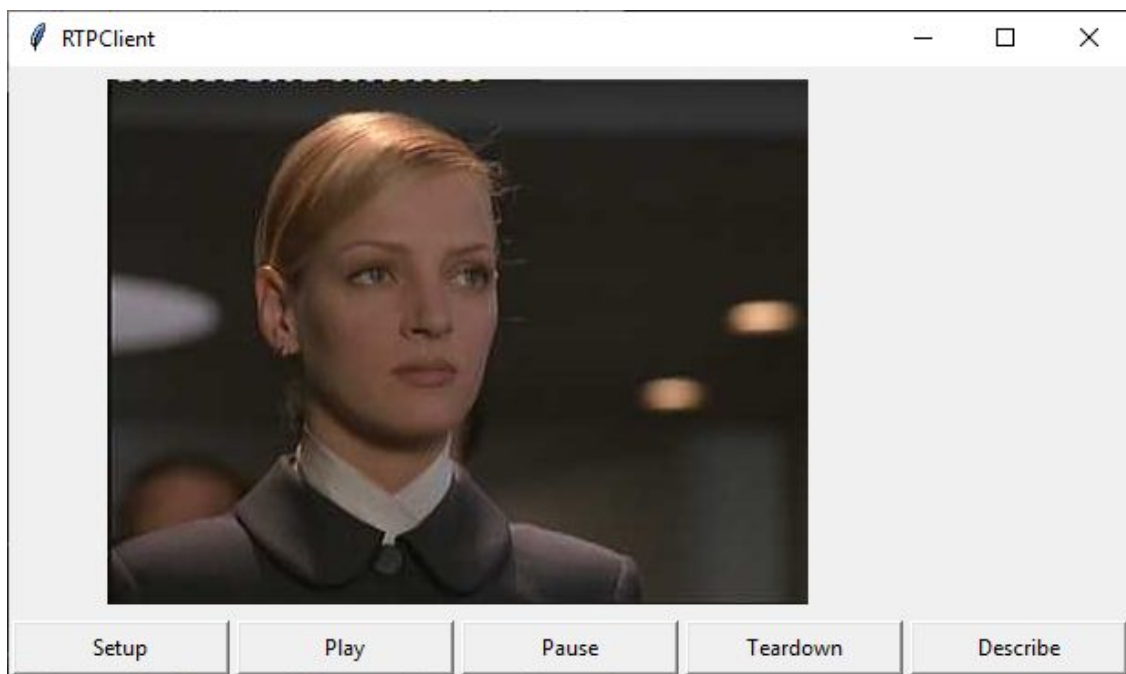
```
# Create Teardown button
self.teardown = Button(self.master, width=15, padx=3, pady=3)
self.teardown["text"] = "Teardown"
self.teardown["command"] = self.exitClient
self.teardown.grid(row=1, column=3, padx=2, pady=2)

# Create Describe button
self.describe = Button(self.master, width=15, padx=3, pady=3)
self.describe["text"] = "Describe"
self.describe["command"] = self.describeSession
self.describe.grid(row=1, column=4, padx=2, pady=2)

# Create a label to display the movie
self.label = Label(self.master, height=19)
self.label.grid(row=0, column=0, columnspan=4, sticky=W+E+N+S, padx=5)
```

Hình 5.3.1: Hiện thực nút Describe

- + Ta sẽ được giao diện chương trình như sau



Hình 5.3.2: Giao diện sau khi thêm

- + Hiện thực hàm describeSesion() cho nút Describe để gửi yêu cầu

```
def playMovie(self):  
    """Play button handler."""  
    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)  
  
    def describeSesion(self):  
        """Pause button handler."""  
        self.sendRtspRequest(self.DESCRIBE)  
  
    def listenRtp(self):  
        """Listen for RTP packets."""  
        while True:  
            try:
```

Hình 5.3.3: Nội dung hàm describeSesion

- + Trong hàm sendRtspRequest(), tạo 1 request cho chức năng description

```
# Describe request
elif requestCode == self.DESCRIBE:
    # Update RTSP sequence number.
    # ...
    self.rtspSeq+=1

    # Write the RTSP request to be sent.
    # request = ...
    request = "%s %s %s" % (self.DESCRIBE_STR, self.fileName, self.RTSP_VER)
    request+="\nCSeq: %d" % self.rtspSeq

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

else:
    return

# Send the RTSP request using rtspSocket.
# ...
self.rtspSocket.send(request.encode("utf-8"))
```

Hình 5.3.4: Nội dung request Describe

- + Trong file ServerWorker.py, ta tạo thêm 1 chức năng để nghe yêu cầu describe từ client trong hàm processRtspRequest() như sau

```
# 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()

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

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

def sendRtp(self):
```

Hình 5.3.5: Nội dung phần nghe request describe của server

- + Sau đó, thực hiện phản trả về thông tin phiên của server sẽ gửi về cho client trong hàm replyRtsp()

```
def replyRtsp(self, code, seq):
    """Send RTSP reply to the client."""
    if code == self.OK_200:
        #print("200 OK")
        reply = 'RTSP/1.0 200 OK\r\nCSeq: ' + seq + '\r\nSession: ' + str(self.clientInfo['session'])
        connSocket = self.clientInfo['rtspSocket'][0]
        connSocket.send(reply.encode())

    elif code == self.DESCRPTION:
        diff = datetime.utcnow() - datetime(1900, 1, 1, 0, 0, 0)
        timestamp = diff.days*24*60*60+diff.seconds

        reply = 'RTSP/1.0 200 OK\r\nCSeq: ' + seq + '\r\nSession: ' + str(self.clientInfo['session']) +
        '\r\nFile name: %s' %self.clientInfo['videoStream'].filename + '\r\nEncoding: UTF-8' + '\r\nCurrent
        frame: %d' %self.clientInfo['videoStream'].frameNbr()

        sdp = '\nv=0\r\no=' + socket.gethostname() + ' %d ' %timestamp + '1 IN IP4 ' + socket.gethostbyname
        (socket.gethostname()) + '\ns= SDP\r\nm=video ' + self.clientInfo['rtpPort'] + ' RTP/UDP 1.0'

        reply += '\nContent-Type: application/sdp\r\nContent-Length: ' + str(len(sdp)) + '\nDate: ' +
        datetime.now().strftime('%d %b %Y %H:%M:%S GMT') + sdp

        connSocket = self.clientInfo['rtspSocket'][0]
        connSocket.send(reply.encode())
```

Hình 5.3.6: Nội dung mà server sẽ trả về cho client

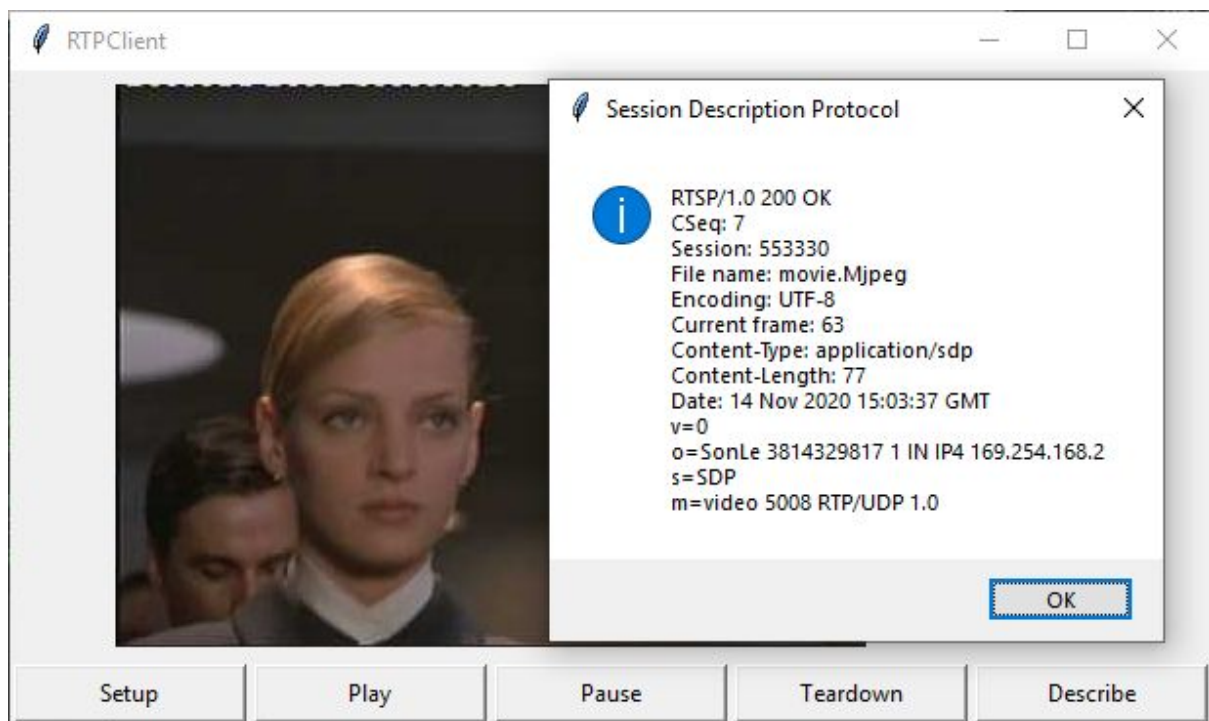
- + Cuối cùng là phần xử lý thông tin của client sau khi nhận phản hồi từ server. Khi người dùng nhấn vào nút Describe, sẽ hiện ra 1 bảng chứa thông tin mà server phản hồi

```
self.playEvent.set()
elif self.requestSent == self.TEARDOWN:
    # self.state = ...
    # no TEARDOWN state
    self.state = self.INIT

    # Flag the teardownAcked to close the socket.
    self.teardownAcked = 1
elif self.requestSent == self.DESCRIBE:
    # self.state = ...
    sdp = "\n".join(lines)
    contentLength = len(data)
    tkinter.messagebox.showinfo('Session Description Protocol', sdp)
```

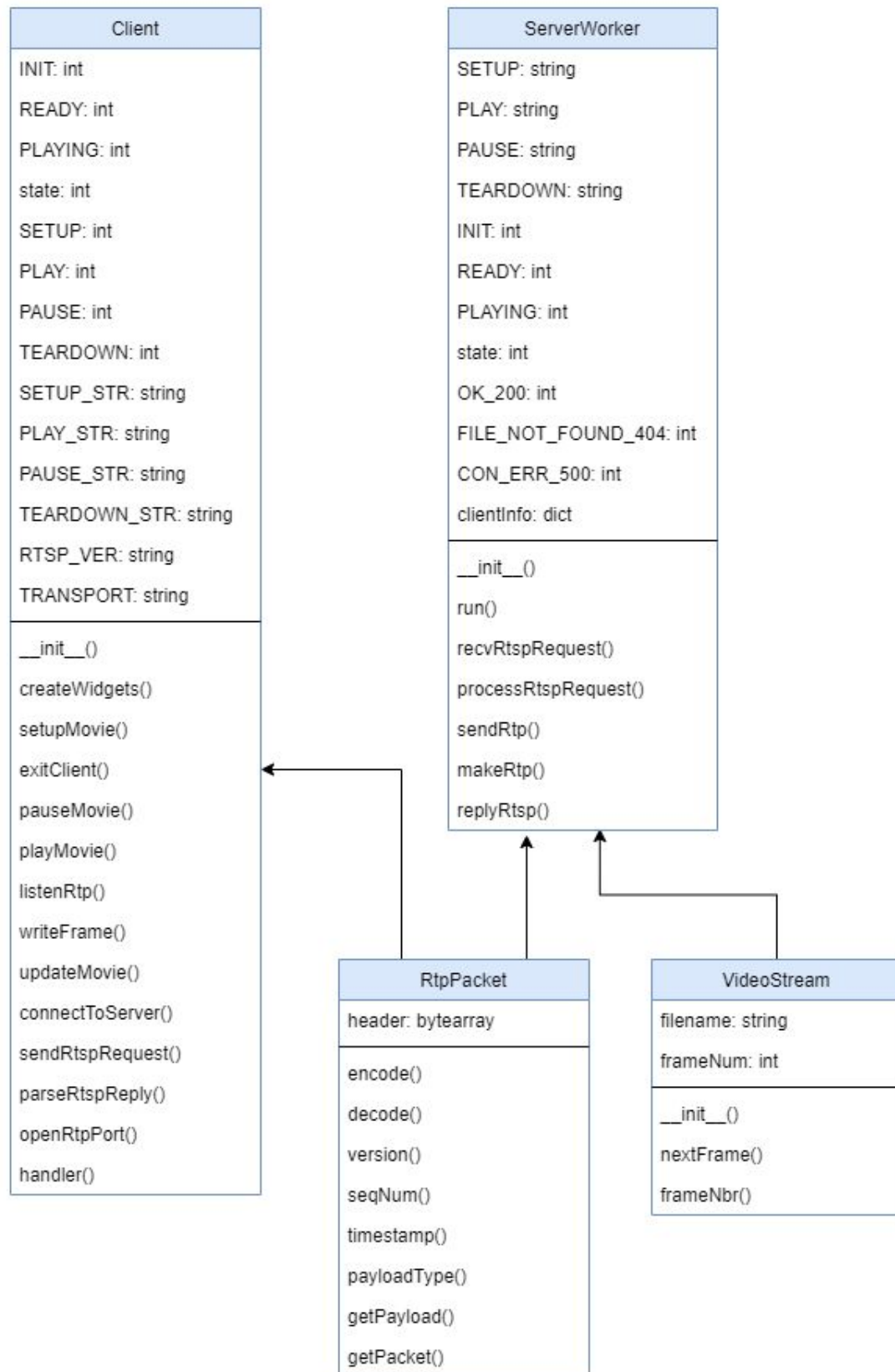
Hình 5.3.7: Hiện thực bảng hiện nội dung mà server trả về

- **Kết quả**



Hình 5.3.8: Giao diện sau khi hoàn thành

IV. Class Diagram



Hình 6.1: Class Diagram

V. Evaluation

Thành viên	Task	Evaluation
Phan Thanh Trường	<ul style="list-style-type: none"> - Client: sendRtspRequest - RtpPacket: Làm chung - Extend: Phần 1 - Report: Làm chung 	100%
Nguyễn Văn Tiến	<ul style="list-style-type: none"> - Client: parseRtspReply - RtpPacket: Làm chung - Extend: Phần 3 - Report: Làm chung 	100%
Lê Trung Sơn	<ul style="list-style-type: none"> - Fix lỗi - RtpPacket: Làm chung - Extend: Phần 2, 3 - Report: Làm chung 	100%
Phạm Quốc Trung	<ul style="list-style-type: none"> - Client: openRtpPort - RtpPacket: Làm chung - Extend: Phần 2 - Report: Làm chung 	100%

VI. User manual

Chương trình gồm 5 file:

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

Bước 1: Khởi động chương trình

Ta có thể khởi động chương trình bằng 2 cách sau

- Cách 1 (hình 7.1) ta sẽ run file *script.bat* để mở giao diện người dùng lên (chỉ sử dụng cho hệ điều hành windows).
- Cách 2 (hình 7.2) ta sẽ mở 2 **Command Prompt (terminal)** lên và chạy lần lượt 2 dòng lệnh sau:

`python Server.py 1025`

=> để khởi tạo môi trường Server với 1025 là server port (nên lớn hơn 1024)

`python ClientLauncher.py 127.0.0.1 1025 5008 movie.mjpeg`

=> để khởi tạo trình phát cho Client. Trong đó 127.0.0.1 là địa chỉ của server, 1025 là port của server, 5008 là port của các RTP packet, movie.mjpeg là tên file video cần phát.

	HK1_2021_Assignment 1	11/13/2020 10:32 PM	Microsoft Edge PD...	264 KB
	movie.Mjpeg	11/13/2020 10:32 PM	MJPEG File	4,170 KB
	RtpPacket	11/13/2020 10:32 PM	JetBrains PyCharm	3 KB
	script	11/13/2020 10:32 PM	Windows Batch File	1 KB
	Server	11/13/2020 10:32 PM	JetBrains PyCharm	1 KB

Hình 7.1: Mở bằng script

```

C:\Windows\System32\cmd.exe - python Server.py 1025
Microsoft Windows [Version 10.0.18363.1198]
(c) 2019 Microsoft Corporation. All rights reserved.

D:\TLHT\HK201\MMT\assingment1\MMT-ass1-master>python Server.py 1025

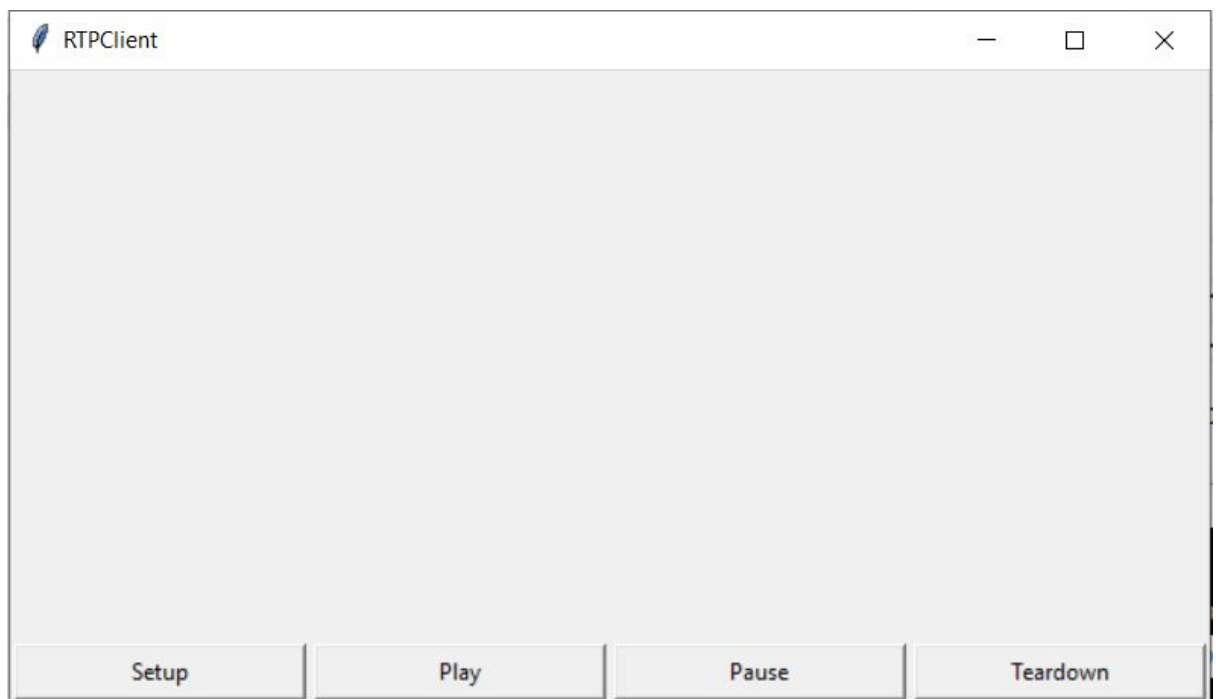
C:\Windows\System32\cmd.exe - python ClientLauncher.py 127.0.0.1 1025 5008 movie.mjpeg
Session: 310467

D:\TLHT\HK201\MMT\assingment1\MMT-ass1-master>python ClientLauncher.py 127.0.0.1 1025 5008 movie.mjpeg

```

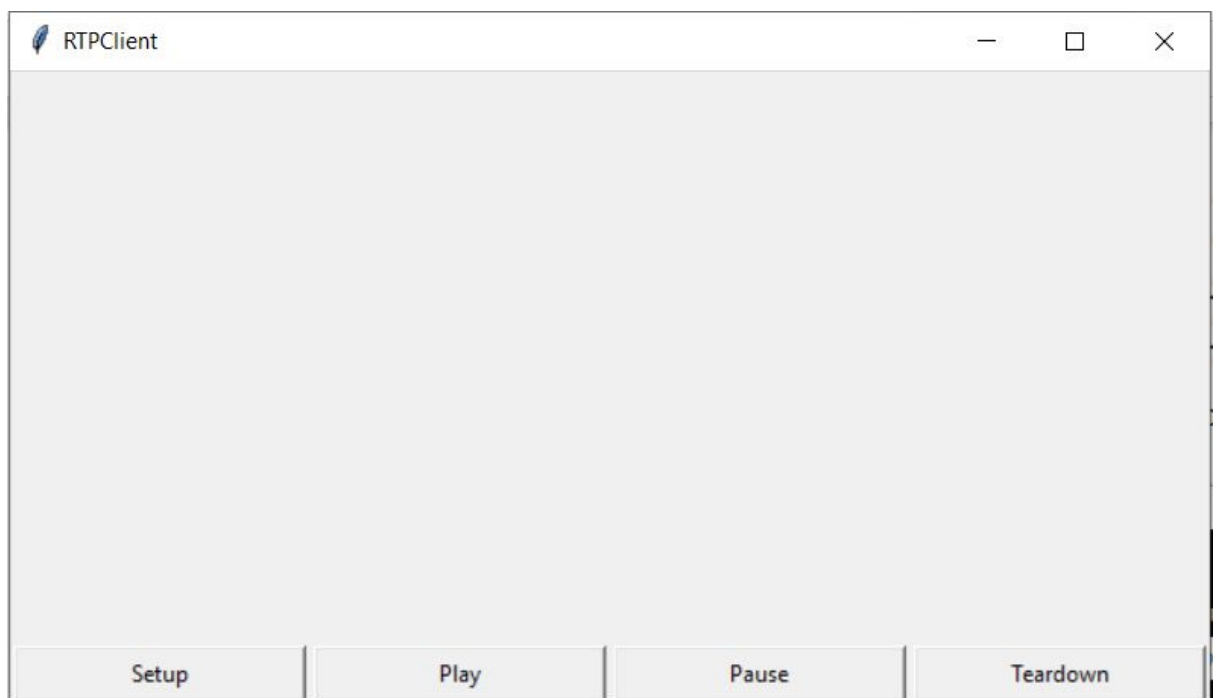
Hình 7.2: Mở bằng dòng lệnh

Sau khi thực hiện 1 trong 2 cách trên thì ta có UI như sau (hình 7.3).



Hình 7.3: Giao diện RTPClient

Bước 2: Cách dùng các buttons



Hình 7.4: Giao diện RTPClient

Chức năng của các button:

- *SETUP*: Để thông báo cho Server khởi tạo phiên.
- *PLAY*: Phát video
- *PAUSE*: Tạm dừng video
- *TEARDOWN*: Đóng ứng dụng



Hình 7.5: Giao diện RTPClient khi phát video

VII. References

1. <https://tools.ietf.org/html/rfc1889>
2. https://en.wikipedia.org/wiki/Real-time_Transport_Protocol
3. <https://sites.google.com/site/embedded247/npcourse/tim-hieu-ky-thuat-video-streaming>
4. https://vi.wikipedia.org/wiki/Real_Time_Streaming_Protocol
5. <https://www.codeproject.com/Articles/507218/Managed-Media-Aggregation-using-Rtsp-and-Rtp>
6. https://vi.wikipedia.org/wiki/Real_Time_Streaming_Protocol
7. https://vi.wikipedia.org/wiki/Real-time_Transport_Protocol