ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KHOA HỌC  KỸ THUẬT MÁY TÍNH


# MẠNG MÁY TÍNH

## Assignment 1

# VIDEO STREAMING APPLICATION

GVHD: Bùi Xuân Giang
SV thực hiện : Lê Khánh Toàn - 1915541
Phạm Nhật Minh - 1910346
Hồ Minh Trí - 1915650
Đỗ Thiện Hoàng - 1913418
Hoàng Nhật Linh Kiều - 2120034

Tp. Hồ Chí Minh, Tháng 11/2021

# Mục lục

# 1 Requirement analysis:

## 1.1 Client

Hiện thực RTSP protocol. Chúng ta cần hoàn thành các function mà được gọi khi người dùng click vào các button trên giao diện người dùng. Khi client bắt đầu, RTSP socket sẽ được mở đến server và sử dụng socket này cho việc gửi các yêu cầu RTSP. Các button:

- SETUP: Lệnh này sẽ thiết lập các session và tham số truyền tải.
- PLAY: Lệnh này sẽ phát video cho client.
- PAUSE: Lệnh này sẽ tạm dừng việc phát video.
- TEARDOWN: Lệnh này sẽ kết thúc session và đóng kết nối với server.

## 1.2 Server

Chúng ta tạo ra packet, set các fields trong header của packet và copy payload vào trong packet. Khi server nhận được yêu cầu PLAY từ client, server sẽ đọc một video frame từ file và tạo ra một đối tượng RtpPacket là RTP-encapsulation của video frame và gửi frame này đến client qua UDP sau mỗi 50ms. Để đóng gói, chúng ta sẽ gọi hàm encode của class RtpPacket. Nhiệm vụ của chúng ta là hiện thực cái hàm này theo những bước sau:

- Set RTP-version (V) = 2.
- Set padding (P), extension (X), số contributing sources (CC), maker (M), tất cả đều bằng 0.
- Set payload type (PT) = 26.
- Set sequence number.
- Set timestamp.
- Set sources identifier.

# 2 Function Description

| Class | Functions | Description |
|---|---|---|
| Server worker | init | tạo constructor |
| | run | bắt đầu server |
| | processRtspRequest | xử lý dữ liệu request |
| | sendRtp | gửi các gói RTP đến máy khách |
| | makeRtp | dùng encode để đóng gói header và payload, gửi đến máy khách. Trả về gói RTP vừa được tạo |
| | replyRtsp | phản hồi thông tin về máy khách thông qua terminal (“200 OK” nếu thành công, “404 NOT FOUND” khi không tìm thấy file, “500 CONNECTION EROR” nếu bị lỗi kết nối) |
| Client | init | tạo constructor |
| | createWidgets | tạo giao diện chương trình, gồm các nút bấm và nơi hiển thị video |
| | setupMovie | gửi lệnh RTSP SETUP tới máy chủ |
| | exitClient | gửi lệnh RTSP SETUP tới máy chủ,và thoát chương trình |
| | pauseMovie | gửi lệnh RTSP PAUSE tới máy chủ để dừng video |
| | listenRTP | nhận và decode các gói RTP, cập nhật lại frame number hiện tại |
| | writeFrame | viết dataframe của video vào file ảnh |
| | updateMovie | hiển thị file ảnh lên giao diện người dùng |
| | connectToServer | kết nối với máy chủ |

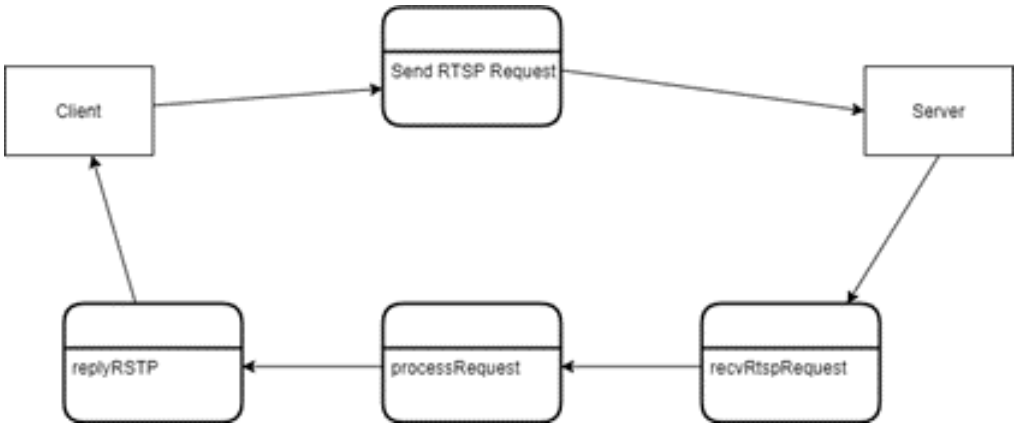
| | | |
|---|---|---|
| | sendRtspRequest | gửi lệnh RTSP đến máy chủ |
| | recvRtspRequest | nhận phản hồi RTSP từ máy chủ, decode dữ liệu và gửi data đến hàm đọc, đồng thời tắt chương trình nếu lệnh là TEARDOWN |
| | parseRtspReplay | xử lý dữ liệu nhận được từ dữ liệu đã được decode |
| | openRtpReport | tạo cổng kết nối giữa máy khách và máy chủ |
| | handler | xử lý khi người dùng bấm vào nút [X] ở góc trên |
| Video Stream | init | tạo constructor |
| | nextFrame | trả về frame tiếp theo của Video |
| | frameNbr | trả về frame number hiện tại |
| RTP Packet | init | tạo constructor |
| | encode | encode các thông số vào header theo đúng định dạng, update lại header và payload |
| | decode | tách gói RTP thành 2 phần header và payload |
| | version | trả về phiên bản của gói RTP |
| | seqNum | trả về sequence number của gói |
| | timestamp | trả về timestamp |
| | payloadType | trả về kiểu của payload |
| | getPayload | trả về payload |
| | getPacket | trả về packet (gồm header và payload) |

# 3 List of Component

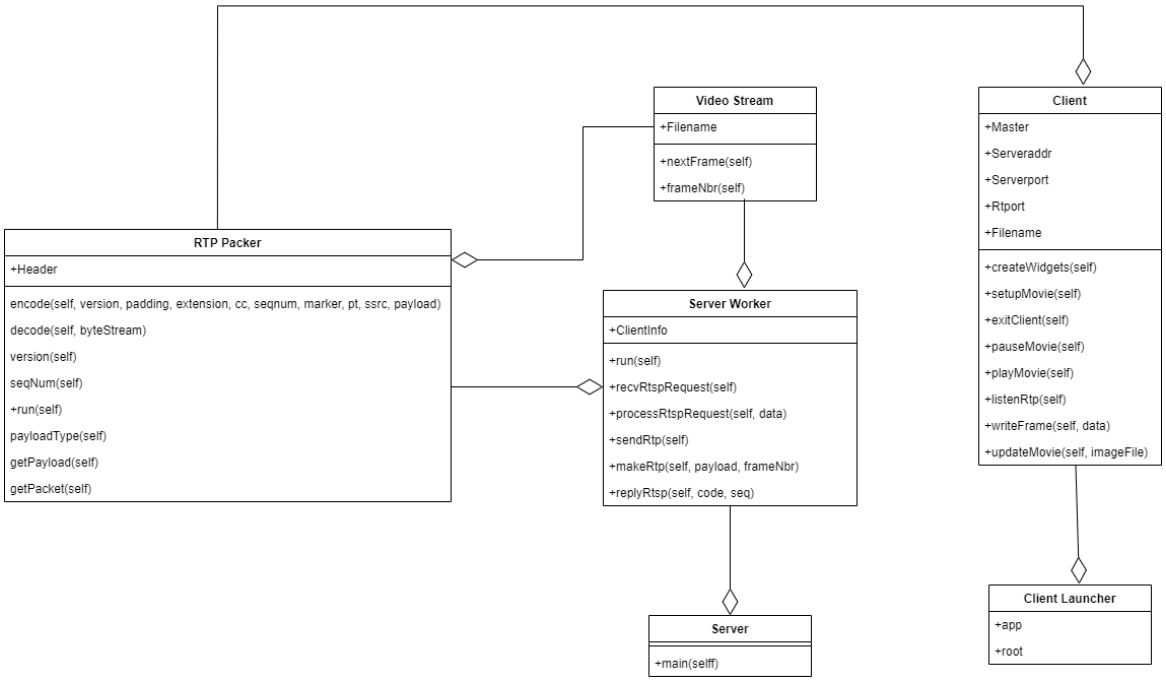
Chúng ta có 6 class:

- Client: Gửi các lệnh RTSP về phía Server
- ClientLauncher: Có chức năng khởi động máy khách
- Server: Có chức năng khởi động server
- ServerWorker: Có chức năng nhận các lệnh RTSP từ máy khách và xử lí những yêu cầu đó
- RTPPacket: Xử lí các gói RTP
- VideoStream: Đọc dữ liệu video từ ổ đĩa

# 4 Data flow diagram:



# 5 Class Diagram:




# 6 Implementation:

Client.py:

- Tạo socket theo phương thức UDP:

```
self.rtpSocket = socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
```

- Hàm setupMovie:

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

Nếu biến trạng thái state bằng INIT(0), gửi Rts packet với yêu cầu SETUP cho server bằng hàm sendRtspRequest.

- Hàm exitClient:

```
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)
```

Gửi Rts packet với yêu cầu TEARDOWN cho server bằng hàm sendRtspRequest. Đóng cửa sổ GUI bằng phương thức destroy() sau đó xóa dữ liệu hình ảnh của video trong bộ nhớ cache.

- Hàm pauseMovie:

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

Nếu trạng thái đang là PLAYING - video đang chiếu, gửi Rts packet với yêu cầu PAUSE cho server bằng hàm sendRtspRequest.

- Hàm playMovie:

```
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)
```

Nếu trạng thái đang là READY – đã SETUP video, tạo luồng phát video chạy hàm listenRtp, gửi Rts packet với yêu cầu PLAY cho server bằng hàm sendRtspRequest.

- Hàm listenRtp:

```
def listenRtp(self):
    """Listen for RTP packets."""
    while True:
        try:
            data = self.rtpSocket.recv(20480)
            if data:
                rtpPacket = RtpPacket()
                rtpPacket.decode(data)

                currFrameNbr = rtpPacket.seqNum()
                print("Current Seq Num: " + str(currFrameNbr))

                if currFrameNbr > self.frameNbr: # Discard the late packet
                    self.frameNbr = currFrameNbr
                    self.updateMovie(self.writeFrame(rtpPacket.getPayload()))
        except:
            # Stop listening upon requesting PAUSE or TEARDOWN
            if self.playEvent.isSet():
                break

            # Upon receiving ACK for TEARDOWN request,
            # close the RTP socket
            if self.teardownAcked == 1:
                self.rtpSocket.shutdown(socket.SHUT_RDWR)
                self.rtpSocket.close()
                break
```

Nhận các frame của video do server gửi từ socket UDP đã tạo với dung lượng tối đa 20480 byte mỗi lần nhận. Thay thế các frame trước bằng các frame hiện tại bằng hàm updateMovie. Dừng nhận frame (bằng phương thức isSet() để ngưng luồng phát video playEvent) nếu người dùng chọn PAUSE hoặc TEARDOWN. Nếu TEARDOWN được chọn, socket sẽ đóng lại.

- Hàm updateMovie:

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

Cập nhật biến label bằng file ảnh với tên file được trả về từ hàm writeFrame.

- Hàm writeFrame:

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

  return cachename
```

Nhận data gồm file ảnh nhận được từ hàm getPayload(). Ghi các file ảnh nhận được vào bộ nhớ cache. Trả về tên của file ảnh đã ghi.

- Hàm connectToServer:

```
def connectToServer(self):
  """Connect to the Server. Start a new RTSP/TCP session."""
  self.rtspSocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
  try:
    self.rtspSocket.connect((self.serverAddr, self.serverPort))
  except:
    tkMessageBox.showwarning('Connection Failed', 'Connection to \'%s
 \' failed.' %self.serverAddr)
```

Tạo socket gửi nhận request/reply từ server theo phương thức TCP và kết nối với server thông qua địa chỉ và số port. Xuất ra 1 thông báo khi kết nối thất bại.

- Hàm sendRtspRequest:

```
def sendRtspRequest(self, requestCode):
  """Send RTSP request to the server."""
  # 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 = ...
        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"))
        # Keep track of the sent request.
        # self.requestSent = ...
        self.requestSent = self.SETUP

    # 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 = ...
        request = "PLAY " + str(self.fileName) + " RTSP/1.0\nCSeq: " + str
    (self.rtspSeq) +"\nSession: " + str(self.sessionId)
        self.rtspSocket.send(request.encode("utf-8"))
        # Keep track of the sent request.
        # self.requestSent = ...
        self.requestSent = self.PLAY
    # Pause request
    elif requestCode == self.PAUSE and self.state == self.PLAYING:
        # Update RTSP sequence number.
        # ...
        self.rtspSeq += 1

        # Write the RTSP request to be sent.
        # request = ...
        request = "PAUSE " + str(self.fileName) + " RTSP/1.0\nCSeq: " +
    str(self.rtspSeq) + "\nSession: " + str(self.sessionId)
        self.rtspSocket.send(request.encode("utf-8"))
        # Keep track of the sent request.
        # self.requestSent = ...
        self.requestSent = self.PAUSE
    # Teardown request
    elif requestCode == self.TEARDOWN and not self.state == self.INIT:
        # Update RTSP sequence number.
        # ...
        self.rtspSeq += 1
        # Write the RTSP request to be sent.
        # request = ...
        request = "TEARDOWN " + str(self.fileName) + " RTSP/1.0\nCSeq: " +
     str(self.rtspSeq) + "\nSession: " + str(self.sessionId)
        self.rtspSocket.send(request.encode("utf-8"))
        # Keep track of the sent request.
        # self.requestSent = ...
        self.requestSent = self.TEARDOWN
    else:
        return
```

```
    # Send the RTSP request using rtspSocket.
    # ...

    print('\nData sent:\n' + request)
```

Gửi các request đến server với 4 loại request tương ứng 4 button trên GUI, request gồm các thông tin: request code, tên file, RTSP version, Cseq, ID của Session. Request sẽ được định dạng bằng phương thức encode. Tạo luồng chạy hàm recvRtspReply để nhận các reply từ server khi SETUP và trạng thái đang là INIT (trạng thái khởi đầu).

- Hàm recvRtspReply:

```
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
```

Nhận các RTS packet reply từ server với kích thước tối đa 1024 byte sau đó phân tách các thông số từ dữ liệu nhận được bằng hàm parseRtspReply. Đóng socket nếu người dùng teardown.

- Hàm parseRtspReply:

```
def parseRtspReply(self, data):
  """Parse the RTSP reply from the server."""
  print("-"*40 + "\nData received:\n" + data)
  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

    # Process only if the session ID is the same
    if self.sessionId == session:
      if int(lines[0].split(' ')[1]) == 200:
        if self.requestSent == self.SETUP:
          #-------------
          # TO COMPLETE
          #-------------
          # Update RTSP state.
          # self.state = ...
```

```
            self.state = self.READY

            # Open RTP port.
            self.openRtpPort()
          elif self.requestSent == self.PLAY:
            # self.state = ...
            self.state = self.PLAYING
          elif self.requestSent == self.PAUSE:
            # self.state = ...
            self.state = self.READY
            # The play thread exits. A new thread is created on resume.
            self.playEvent.set()
          elif self.requestSent == self.TEARDOWN:
            # self.state = ...
            self.state = self.INIT
            # Flag the teardownAcked to close the socket.
            self.teardownAcked = 1
```

Tách các thông số từ dữ liệu nhận được từ server. Đầu tiên lấy ra sequence number để kiểm tra xem có đúng với sequence number đã gửi hay không sau đó mới tiếp tục xử lý. Nếu đúng sequence number, lấy ra session ID và kiểm tra (tạo mới nếu chưa có và gán bằng session ID của server). Nếu server phản hồi code 200 OK tức là request thành công và bắt đầu thực hiện chuyển các trạng thái theo như request đã gửi. Mở cổng kết nối đến server bằng hàm openRtpPort khi SETUP.

- Hàm 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 = ...
  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.serverAddr,self.rtpPort))
  except:
    tkMessageBox.showwarning('Unable to Bind','Unable to bind PORT=%d'
  %self.rtpPort)
```

Set timeout = 0.5s cho socket. Chỉ định địa chỉ của server và port kết nối đến bằng phương thức bind. Thông báo nếu port truyền vào không thể kết nối.

- Hàm handler:

```
def handler(self):
  """Handler on explicitly closing the GUI window."""
  self.pauseMovie()
  if tkMessageBox.askokcancel("Quit?", "Are you sure you want to quit?
 "):
    self.exitClient()
  else: # When the user presses cancel, resume playing.
```

```
        self.playMovie()
```

Được gọi khi người dùng nhấn nút exit trên cửa sổ: Dừng video bằng hàm pauseMovie, hiện hộp thông báo xác nhận muốn rời khỏi. Gọi exitClient nếu người dùng xác nhận "Quit", tiếp tục phát video playMovie nếu người dùng bấm hủy (cancel).

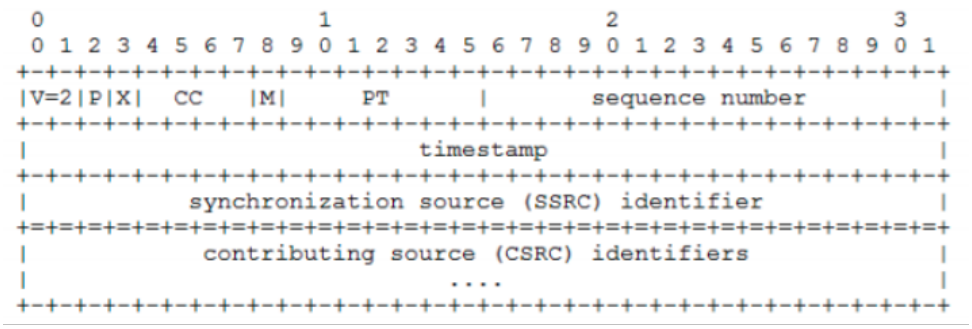
RtpPacket.py:

- Hàm encode:

```
def encode(self, version, padding, extension, cc, seqnum, marker, pt,
 ssrc, payload):
  """Encode the RTP packet with header fields and payload."""

  timestamp = int(time())
  self.header = bytearray(HEADER_SIZE)

  self.header[0] = version << 6
  self.header[0] = self.header[0] | padding << 5
  self.header[0] = self.header[0] | extension << 4
  self.header[0] = self.header[0] | cc
  self.header[1] = marker << 7
  self.header[1] = self.header[1] | pt

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

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

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


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

Mã hóa packet data với một header bao gồm các thông số:

- RTP-version: 2 bit
- Padding: 1 bit
- Extension: 1 bit
- Contributing source: 4 bit
- Marker: 1 bit
- Payload type field (pt): 7 bit
- Sequence number: 16 bit
- Timestamp: 32 bit
- SSRC: 32 bit

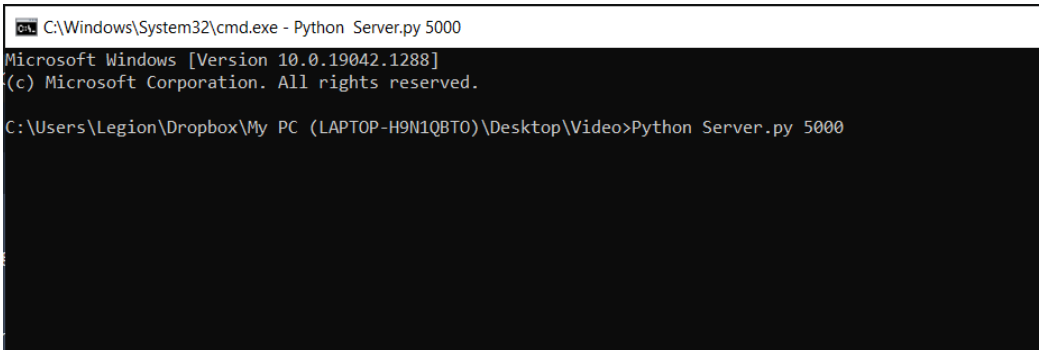



# 7 User manual:

Đầu tiên, để khởi động server, ta thực hiện lệnh sau:

**Python Server.py <server port>**

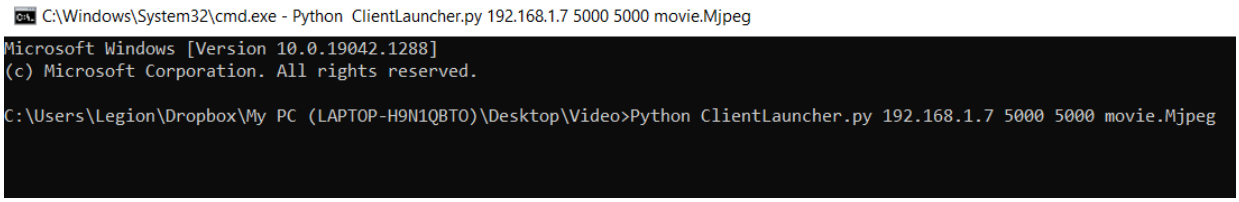
Ta chọn server port là một số bất kì > 1024, chẳng hạn: **Python Server.py 5000**



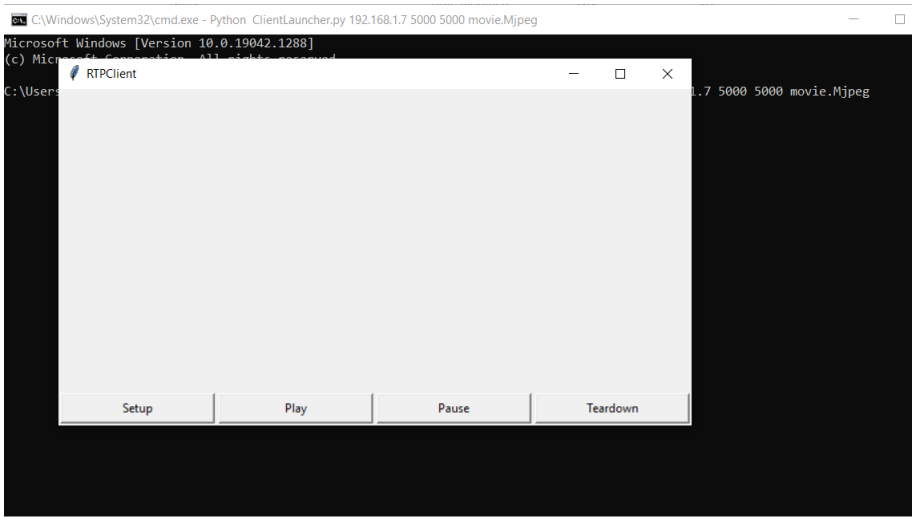

Tiếp theo ta khởi động máy khách bằng lệnh:

**Python ClientLaunchet.py <server host> <server port> <RTP port> <filename>**

Ví dụ: **Python ClientLauncher.py 192.168.1.7 5000 5000 movie.Mjpeg**

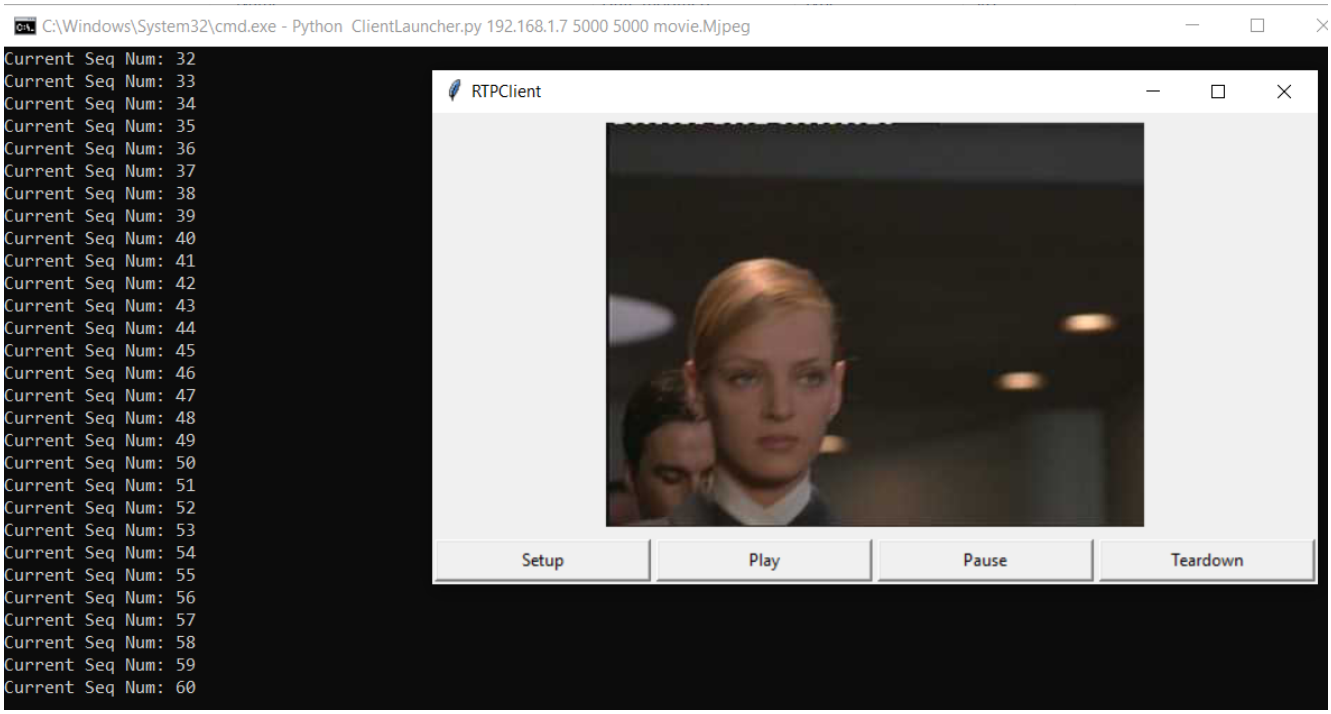



Khi này ta sẽ được một giao diện gồm 4 nút SETUP, PLAY, PAUSE và TEARDOWN.




gười dùng gửi lệnh RTSP đến máy chủ bằng cách nhấn các nút trên giao diện. Các tương tác RTSP bao gồm:

- SETUP: sử dụng để phát lập phiên và các tham số truyền tải
- PLAY: bắt đầu phát video
- PAUSE: tạm dừng video
- TEARDOWN: kết thúc phiên và đóng kết nối với máy chủ, đồng thời đóng cửa sổ GUI.



# 8 Extend

## 8.1

Để tính tốc độ truyền video và tỉ lệ RSP loss ta sẽ chạy video trong một khoảng thời gian rồi dừng, tính tổng dữ liệu truyền được trong thời gian đó rồi chia cho thời gian tính. Ta sẽ đo tổng dữ liệu truyền được và thời gian trong hàm listenRtp().

```
def listenRtp(self):
    """Listen for RTP packets."""
    self.begin = time.time()
    start=self.begin
    self.stop = False
    while True:
        try:
            data = self.rtpSocket.recv(20480)
            if data:
                rtpPacket = RtpPacket()
                rtpPacket.decode(data)
                self.sumData += len(data)
                currFrameNbr = rtpPacket.seqNum()
                print("Current Seq Num: " + str(currFrameNbr))

                try:
                    if self.frameNbr + 1 != currFrameNbr:
                        self.counter += 1
                        print('!'*60 + "\nPACKET LOSS\n" + '!'*60)
                        # currFrameNbr = rtpPacket.seqNum()
                        # version = rtpPacket.version()
                except:
                    print("seqNum() error")
                    print('-'*40)
                    traceback.print_exc(file=sys.stdout)
                    print('-'*40)
```

Khi người dùng nhấn nút Teardown thì hàm exitClient() sẽ xuất ra kết quả.

```
def exitClient(self):
    """Teardown button handler."""
    self.sendRtspRequest(self.TEARDOWN)
    self.master.destroy() # Close the gui window
    try:
        os.remove(CACHE_FILE_NAME + str(self.sessionId) + CACHE_FILE_EXT)
    except:
        "No cache file to delete."
    if not self.stop:
        self.sumOfTime += time.time() - self.begin
    if self.sumOfTime>0:
        rateData = float(int(self.sumData)/int(self.sumOfTime))
        print('-'*40 + "\nVideo Data Rate: " + str(rateData) +"\n")
        rateLoss = float(self.counter/self.frameNbr)
        print('-'*40 + "\nRTP Packet Loss Rate: " + str(rateLoss) +"\n")
```

Kết quả:

```
Data sent:
TEARDOWN movie.Mjpeg RTSP/1.0
CSeq: 4
Session: 867943

Video Data Rate: 14051.29


RTP Packet Loss Rate: 0.0
```

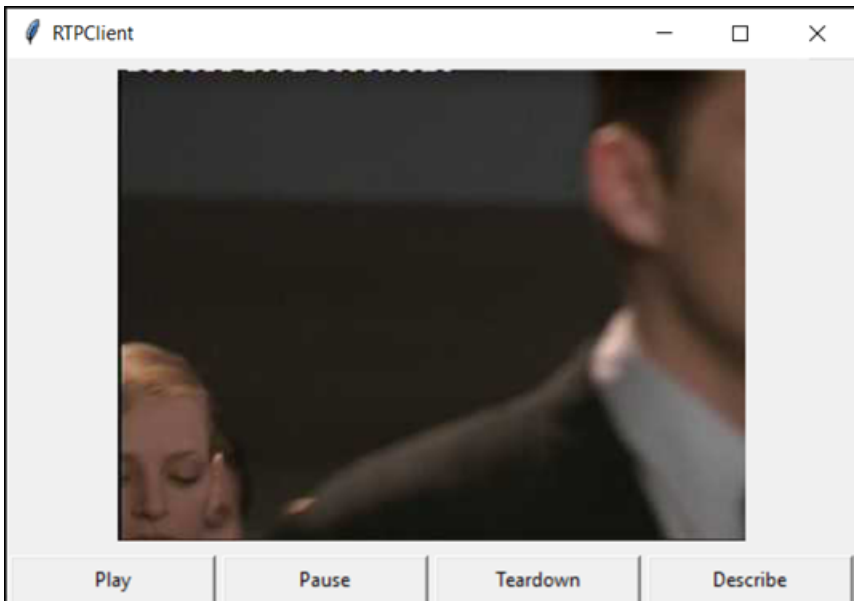
## 8.2

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

Giải pháp đưa ra là khi người dung 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.

Kết quả:




## 8.3

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

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

Để hiện thực tính năng DESCRIBE ta sẽ hiện thực tương tự như các tính năng SETUP, PLAY, PAUSE và TEARDOWN.

**Client:**

Nếu người dùng nhấn nút DESCRIBE thì hàm Describe() sẽ cho biết là khách đã chọn tính năng DESCRIBE.

```
def describe(self):
    """Describe button handler."""
    if self.trigle:
        self.sendRtspRequest(self.DESCRIBE)
```

Hàm sendRtspRequest() sẽ gửi tín hiệu DESCRIBE đến Server.

```
def sendRtspRequest(self, requestCode):
    """..."""
    elif requestCode == self.DESCRIBE:
        self.rtspSeq = self.rtspSeq + 1
        request = "DESCRIBE " + str(self.fileName) + " RTSP/1.0\nCSeq: " +
   str(self.rtspSeq) + "\nSesssion:" + str(self.sessionId)
        self.rtspSocket.send(request.encode("utf-8"))
```

**Server:**

Bên server sẽ nhận tín hiệu từ máy khách và thực thi tính năng.Hàm processRtspRequest() sẽ nhận tín hiệu và gửi về bên reply.

```
def processRtspRequest(self, data):
    """..."""
    elif requestType == self.DESCRIBE:
  self.replyDescibe(self.OK_200,seq[1])
```

Hàm replyDescribe() sẽ encode những thống kê cần phản hồi bao gồm RTSP Port, Video Encoding, RTSP ID trong trường hợp kết nối thành công. Nếu không thì sẽ báo lỗi.

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

Hàm describe () sẽ in ra cách thống kê cho máy khách.

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

```
      + str(len(seq1)) + "\n"
  return seq2 + seq1
```

Khi khách nhấn nút DESCRIBE, kết quả sẽ được in ra máy khách.

```
Data received:
RTSP/1.0 200 OK
CSeq: 4
Session: 700519
Content-Base: movie.Mjpeg
Content-Length: 86
v=0
m=video 5000 RTP/AVP 26
a=control:streamid=700519
a=mimetype:string;"video/Mjpeg"
```

## 8.4

Để hiển thị thông tin về thời gian của video ta sẽ sử dụng thư viện CV2. Chúng ta sẽ import thư viện này trong file videoStream. Chúng ta sẽ thêm 1 số câu lệnh trong 3 file videoStream, ServerWorker và Client.

**Videostream:**

Bên server sẽ nhận tín hiệu từ máy khách và thực thi tính năng.Hàm processRtspRequest() sẽ nhận tín hiệu và gửi về bên reply.

Hàm totaltime() sẽ trả về tổng thời gian của video.

```
def totaltime(self):
  video = cv2.VideoCapture(self.filename)
  self.fps = video.get(cv2.CAP_PROP_FPS)
  self.frames = self.countframe(video)
  self.duration = self.frames / self.fps
  return self.duration
```

Chúng ta sử dụng hàm countframe() để đếm số frame.

```
def countframe(self, video):
  frames = 0
  while True:
    (grabbed, frame) = video.read()

    if not grabbed:
      break
    frames += 1
  return frames
```

**ServerWorker:**

Ta sẽ thêm tổng thời gian trong biến reply ở 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\nCSeq: ' + seq + '\nSession: ' + str(self.
   clientInfo['session'])+'\nVideoTotaltime: '+str(self.clientInfo['
   videoStream'].totaltime())
```

**Client:**

Ta sẽ lấy tổng thời gian khi người dùng nhấn lệnh SETUP

```
def parseRtspReply(self, data):
    """..."""
                if self.requestSent == self.SETUP:
                # Update RTSP state.
                # self.state = ...
                self.totaltime=float(lines[3].split(' ')[1])
                self.state = self.READY
```

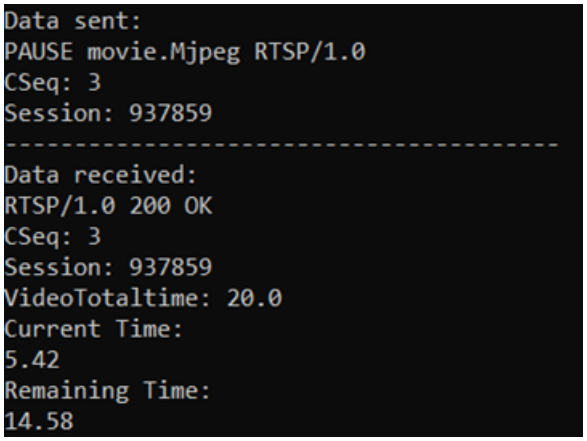
Ta sẽ đo thời gian hiện tại trong hàm listenRtp()

```
def listenRtp(self):
    """Listen for RTP packets."""
    self.begin = time.time()
    start=self.begin
```

Nếu người dùng bấm nút Pause thì hàm listenRtp() sẽ xuất ra thời gian hiện tại và thời gian còn lại của video.

```
except:
    self.sumOfTime += time.time() - self.begin
    self.stop = True
    # Stop listening upon requesting PAUSE or TEARDOWN and print time
   information
    if self.playEvent.isSet():
        print('Current Time: ')
        self.currtime=round(time.time()-start,2)
        print(str(self.currtime))
        remainingtime=round(self.totaltime-self.currtime,2)
        print('Remaining Time: ')
        print(str(remainingtime))
        break

    # Upon receiving ACK for TEARDOWN request,
    # close the RTP socket
    if self.teardownAcked == 1:
        try:
            self.rtpSocket.shutdown(socket.SHUT_RDWR)
            self.rtpSocket.close()
        finally:
            break
```

Kết quả:




# 9 A summary of achieved result

Về cơ bản, nhóm em đã hoàn thành được phần Requirement. Chương trình chạy ổn, không bị lỗi. Nhóm làm được 3 extend đầu cùng với phần hiển thị thời gian của extend 4, tuy nhiên do không đủ thời gian nên nhóm không hoàn thành được extend 5.

# 10 Bảng phân công công việc

| Thành viên | Công việc |
|---|---|
| Phạm Nhật Minh | Extend 1,2,3,4, vẽ Class Diagram |
| Lê Khánh Toàn | Functional Description, Implement |
| Hồ Minh Trí | User Manual |
| Dỗ Thiện Hoàng | Requirement Analysis |
| Hoàng Nhật Linh Kiều | Component Diagram, List of component |

# 11 Source code

https://github.com/Ez9amE0812/Video_streaming