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

COMPUTER NETWORKING - ASSIGNMENT 1

VIDEO STREAMING WITH RTSP AND RTP

GV: Nguyễn Hoài Nam
Nhóm CN: Nguyễn Tiến Phát - 1712572
Trịnh Đức Trường - 1713759

TP. HỒ CHÍ MINH, THÁNG 11/2020

Mục lục

1 Requirements Analysis

Hiện thực RTSP protocol phía server và RTP de-packetization phía client để hệ thống hiển thị được video đã truyền.

1.1 Nhấn vào button SETUP

- Hệ thống sẽ gửi yêu cầu SETUP tới server (cần chèn Transport header - chỉ rõ port cho RTP data socket)
- Hệ thống nhận phản hồi từ server và phân tích Session header (lấy RTSP session ID)
- Tạo datagram socket để nhận RTP data
- Gán timeout cho socket là 0.5s

1.2 Nhấn vào button PLAY

- Hệ thống sẽ gửi yêu cầu PLAY tới server (phải chèn Session header)
- Hệ thống nhận phản hồi từ serve

1.3 Nhấn vào button PAUSE

- Hệ thống sẽ gửi yêu cầu PAUSE tới server (phải chèn Session header)
- Hệ thống nhận phản hồi từ serve

1.4 Nhấn vào button TEARDOWN

- Hệ thống sẽ gửi yêu cầu TEARDOWN tới server (phải chèn Session header)
- Hệ thống nhận phản hồi từ server

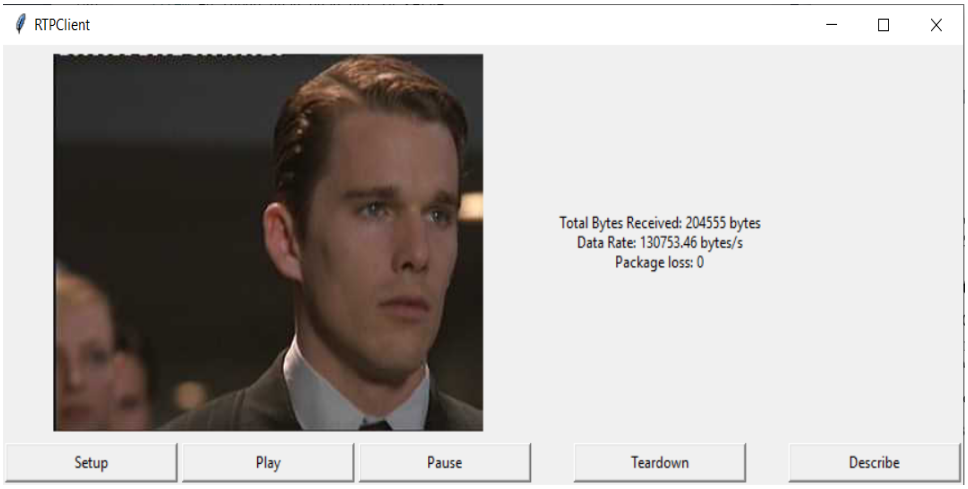
2 Function description

2.1 File Client.py

File này thực hiện các chức năng sau đây: Xây dựng GUI, xử lý sự kiện các button, kết nối với server, decode các Rtp package nhận được từ server

2.1.1 Xây dựng GUI

Hình ảnh bên dưới là GUI được xây dựng ở Client.py



GUI được code trong function

```
def createWidgets(self)
```

Ví dụ hiện thực button Setup và Label để hiển thị Image bằng Tkinter trong Python 3:

```
# Create Setup button
self.setup = Button(self.master, width=20, padx=3, pady=3)
self.setup["text"] = "Setup"
self.setup["command"] = self.setupMovie
self.setup.grid(row=1, column=0, 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=3, sticky=W+E+N+S, padx=5, pady=5)
```

Các widget khác được hiện thực tương tự. Chi tiết trong function createWidgets(self) trong source code bên dưới.

2.1.2 Xử lý sự kiện các button

Các hàm này xử lý các sự kiện tương ứng khi click vào các button trên UI. Các sự kiện hoạt động như thế nào thì đã được mô tả ở mục Requirements Analysis.

2.1.2.a SETUP

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

Khi click vào button SETUP trên GUI, hàm setupMovie(self) sẽ được gọi,và sẽ chạy hàm sendRtspRequest(self), cách thức hoạt động của hàm sendRtspRequest(self.SETUP) sẽ được mô tả bên dưới.

2.1.2.b PLAY

```
def playMovie(self):
  """Play button handler."""
  if self.state == self.READY:
     # Set time when button is pressed
     self.startTimePlay = time.time()
     # 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)
```

Khi click vào button PLAY, hàm playMovie(self) được gọi. Đầu tiên check xem self.state đã READY hay chưa? Nếu đúng thì tạo một thread để listen các package RTP được gửi từ server thông qua UDP.Sau đó gọi hàm sendRtspPackage(self.PLAY) để gửi yêu cầu cho Server.
self.startTimePlay lưu thời điểm từ lúc click butto PLAY để hổ trợ tính toán cho một số thông tin ở phần Describe.

2.1.2.c PAUSE

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

Khi click button PAUSE trên GUI, hàm pauseMovie(self) sẽ được gọi. Đầu tiên kiểm tra xem trạng thái có đang là PLAYING hay không? Nếu đúng thì gọi hàm sendRtspRequest(self.PAUSE) để gửi yêu cầu đến Server.

2.1.2.d 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)
```

Khi click button TEARDOWN trên GUI, hàm exitClient(self) sẽ được gọi. Đầu tiên, gọi hàm sendRtspRequest(self.TEARDOWN) để gửi yêu cầu đến Server thông qua RTSP. Câu lệnh self.master.destroy() dùng để đóng GUI, cụ thể hơn là dừng vòng lặp tạo GUI của Tkinter. Cuối cùng remove file *.jpg được tạo ra ở hàm writeFrame(self,data) ở trong file Client.py.

2.1.2.e DESCRIBE

```
def describeMedia(self):
   if self.state not self.INIT:
      self.sendRtspRequest(self.DESCRIBE)
```

Khi click button DESCRIBE ở trên GUI, hàm describeMedia(self) được gọi. Đầu tiên, check xem trạng thái có phải là INIT hay không? nếu không thì gọi hàm sendRtspRequest(self.DESCRIBE) để gửi yêu cầu đến Server.

Cách hoạt động của hàm sendRtspRequest(self,requestCode) trong file Client.py

```
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 = 'SETUP ' + self.fileName + ' RTSP/1.0\nCSeq ' + str(self.rtspSeq) + '\nTransport:
          RTP/UDP; client_port= ' + str(self.rtpPort)
      # Keep track of the sent request.
      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 = 'PLAY ' + self.fileName + ' RTSP/1.0\nCSeq: ' +str(self.rtspSeq) + '\nSession: '
          +str(self.sessionId)
      # Keep track of the sent request.
      self.requestSent = self.PLAY
    self.rtspSocket.send(request.encode("utf-8"))
```

Code bên trên mô tả cách xử lý sự kiện gửi request đến Server thông qua RTSP khi click button SETUP hoặc PLAY, xử lý các sự kiện khác cũng tương tự.Đầu tiên kiểm trả xem request của sự kiện nào bằng cách kiếm trả requestCode và state tương ứng. Sau đấy update RTSP sequence number lên 1. Tạo một request theo format sau:

Listing 1: Đối với SETUP

```
request = 'SETUP ' + self.fileName + ' RTSP/1.0\nCSeq ' + str(self.rtspSeq) + '\nTransport:
    RTP/UDP; client_port= ' + str(self.rtpPort)
```

Listing 2: Các sự kiện khác

```
request = '[Event] ' + self.fileName + ' RTSP/1.0\nCSeq: ' +str(self.rtspSeq) + '\nSession: '
    +str(self.sessionId)
```

Sau đó sẽ giữ lại yêu cầu sắp gửi bằng cách lưu code event vào biến self.requestSent. Cuối cùng, gửi yêu cầu thông qua RTSP từ Client đến Server bằng self.rtspSocket.send(request.encode("utf-8")

2.1.3 Kết nốt với Server

Để có thể streaming video từ Server, Client cần phải có một kết nối đến Server thông qua mạng. Để làm điều này, file Client.py đã tạo một số hàm connectToServer(self) để hiện thực chức năng này.

```
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:
      tkinter.messagebox.showwarning('Connection Failed', 'Connection to \'%s\' failed.'
          %self.serverAddr)
```

Khi run ClientLauncher.py, class Client() được tạo đồng thời hàm connectToServer(self) được gọi tại phương thức __init__(self).Đầu tiên tạo một rtspSocket bằng module socket được cung cấp trong Python3 kết nối thông qua TCP. Sau đó thử kết nốt với địa chỉ và số cổng của Server.Nếu thành công, màn hình sẽ không hiển thị, nếu thất bại, màn hình sẽ hiển thị một messagebox cảnh báo.

2.1.4 Nhận dữ liệu từ Server và xử lý

Sau khi kết nối thành công Server thông qua TCP. Client.py sẽ có một số hàm để xử lý các thông tin từ Rtp Package nhận được từ Server. Các hàm xử lý sau đây: listenRtp(self),writeFrame(self,data),updateMovie(self, imageFile)

```
def listenRtp(self):
   """Listen for RTP packets."""
   while True:
      try:
         data = self.rtpSocket.recv(20480)
         self.lengtTimeRecvPkg = time.time() - self.startTimePlay
         self.startTimePlay = time.time()
         if data:
            rtpPacket = RtpPacket()
            rtpPacket.decode(data)
            currFrameNbr = rtpPacket.seqNum()
            if currFrameNbr > self.frameNbr: # Discard the late packet
               self.frameNbr = currFrameNbr
               self.updateMovie(self.writeFrame(rtpPacket.getPayload()))
```

Hàm này xử lý việc nhận dữ liệu từ Rtp Package. Dữ liệu nhận từ Server được lưu trong biến data. rtpPackage.decode(data) sẽ giải mã dữ liệu của data. Sau đấy lấy data ảnh bằng rtpPacket.getPayload() để làm tham số đầu vào cho hàm wirteFrame()

```
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
```

Hàm này có chức năng convert dữ liệu của biến data, cụ thể payload được truyền vào thành một ảnh kiểu jpg rồi lưu vào file và trả về tên của ảnh đấy.

```
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
```

Hàm này thực hiện việc render ảnh từ file vừa được tạo trong hàm writeFrame()lên GUI.

2.2 ServerWorker.py

Server có chức năng nhận yêu cầu từ Client, xử lý và trả lời yêu cầu, xử lý dữ liệu và gửi về cho Client.

2.2.1 Nhận và xử lý yêu cầu của Client

Để thực hiện chức năng này. File ServerWorker.py có một số hàm:recvRtspRequest(self),processRtspRequest(self, data)

```
def recvRtspRequest(self):
    """Receive RTSP request from the client."""
    connSocket = self.clientInfo['rtspSocket'][0]
    while True:
        data = connSocket.recv(256)
        if data:
            print("Data received:\n" + data.decode("utf-8"))
            self.processRtspRequest(data.decode("utf-8"))
```

Hàm này nhận dữ liệu request được gửi từ Client và lưu vào data. Sau đấy gọi hàm xử lý sự kiện processRtspRequest(data.decode("utf-8"))

```
def processRtspRequest(self, data):
    """Process RTSP request sent from the client."""
    request = data.split('\n')
    line1 = request[0].split(' ')
    requestType = line1[0]
    filename = line1[1]
    seq = request[1].split(' ')
    # Process SETUP request
    if requestType == self.SETUP:
        if self.state == self.INIT:
            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]
        # print("clientInfo['rptPort']: {0}".format(self.clientInfo['rptPort']))
```

Code bên trên mô tả việc xử lý request của sự kiện SETUP. Dữ liệu được truyền vào hàm processRtspRequest(self,data) là một string. Xử lý nhiều bước ta thu được thông tin cần thiết là lưu vào các biến: requestType(để check loại request),filename(để lấy dữ liệu video với tên tương ứng),seq(lưu RTSP sequence number). Code sẽ được giải thích chi tiết ở demo.Các loại request khác xử lý tương tự, chi tiết trong source code.

2.2.2 Trả lời yêu cầu của Client

Sau khi xử lý xong hoặc không xử lý được yêu cầu của Client, Server sẽ trả lời thông tin chi tiết của yêu cầu về cho Client thông qua RTSP

```
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'])
    elif code == self.OK_200_DECR:
        reply = 'OK_200_DECS\nContent-Base: '+self.filename+''
    connSocket = self.clientInfo['rtspSocket'][0]
    connSocket.send(reply.encode("utf-8"))
    # Error messages
    if code == self.FILE_NOT_FOUND_404:
        print("404 NOT FOUND")
    elif code == self.CON_ERR_500:
        print("500 CONNECTION ERROR")
```

Nếu yêu cầu được xử lý thành công, Client sẽ nhận được trả lời từ Server với forrmat:

```
Data recieve:
RTSP/1.0 200 OK
CSeq: 1
Session: 427899
```

còn nếu không xử lý được sẽ trả về 404 NOT FOUND hoặc 500 CONNECTION ERROR

2.2.3 Gói thông tin và gửi cho Client

Để thực hiện chức năng này, ServerWorker.py có một số hàm sendRtp,makeRtp. Khi requestType là PLAY, bên hàm xử lý request tạo một Thread để thực hiện việc gửi Rtp từ Server đến Client thông qua UDP. Bằng

cách chạy qua hàm sendRtp():

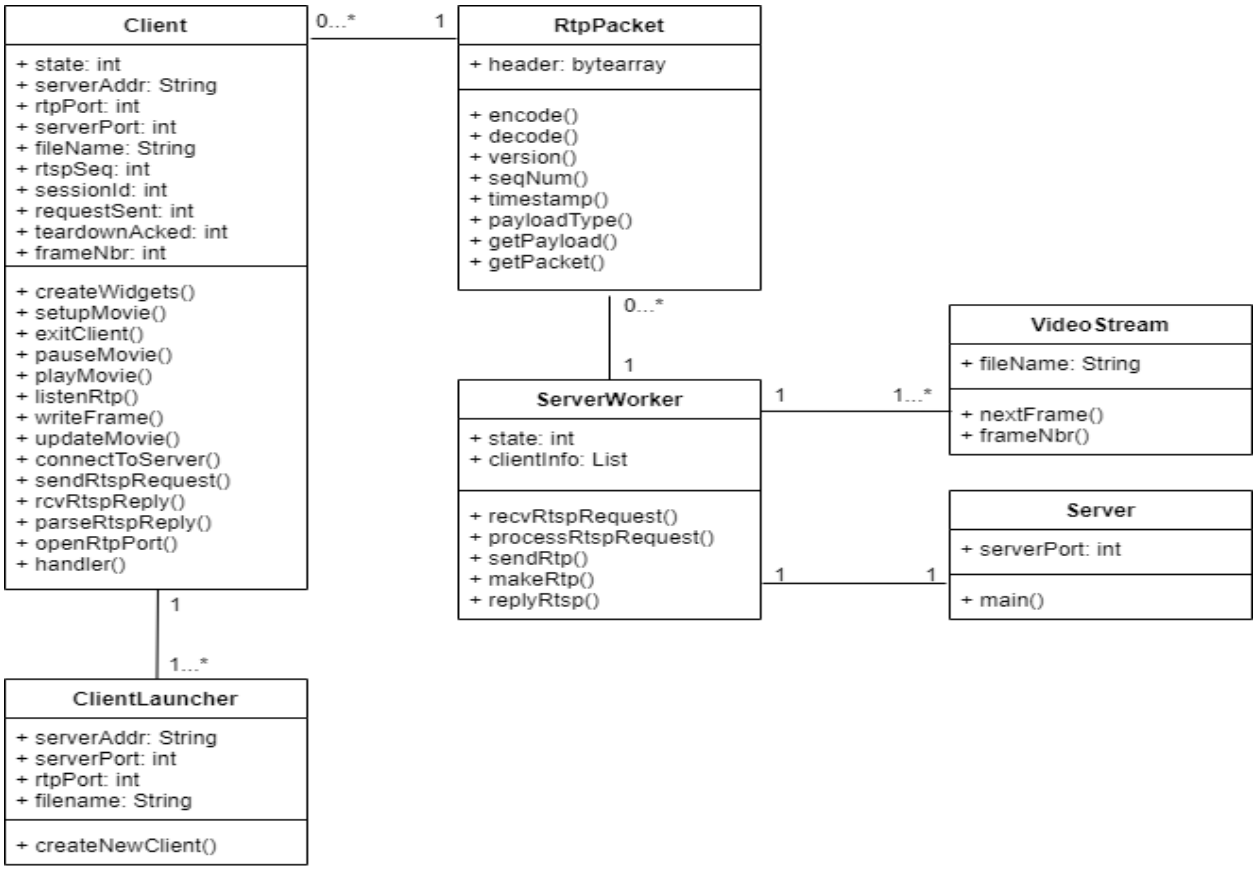
```
def sendRtp(self):
"""Send RTP packets over UDP."""
   while True:
      self.clientInfo['event'].wait(0.05)

      # Stop sending if request is PAUSE or TEARDOWN
      if self.clientInfo['event'].isSet():
         break

      data = self.clientInfo['videoStream'].nextFrame()
      if data:
         frameNumber = self.clientInfo['videoStream'].frameNbr()
         try:
            address = self.clientInfo['rtspSocket'][1][0]
            port = int(self.clientInfo['rtpPort'])
            self.clientInfo['rtpSocket'].sendto(self.makeRtp(data, frameNumber),(address,port))
         except:
            print("Connection Error")
```

Ở đây có một số câu lệnh cần lưu ý. Đầu tiên, dữ liệu của video được lấy từ class VideoStream.py thông qua phương thức frameNbr() và lưu vào biến data. Nếu có data, thì sẽ lấy tiếp số frame thông qua phương thức frameNbr(). Khi đã có data cần thiết, bắt đầu thực hiện việc đóng gói bằng hàm makeRtp(data,frameNumber)

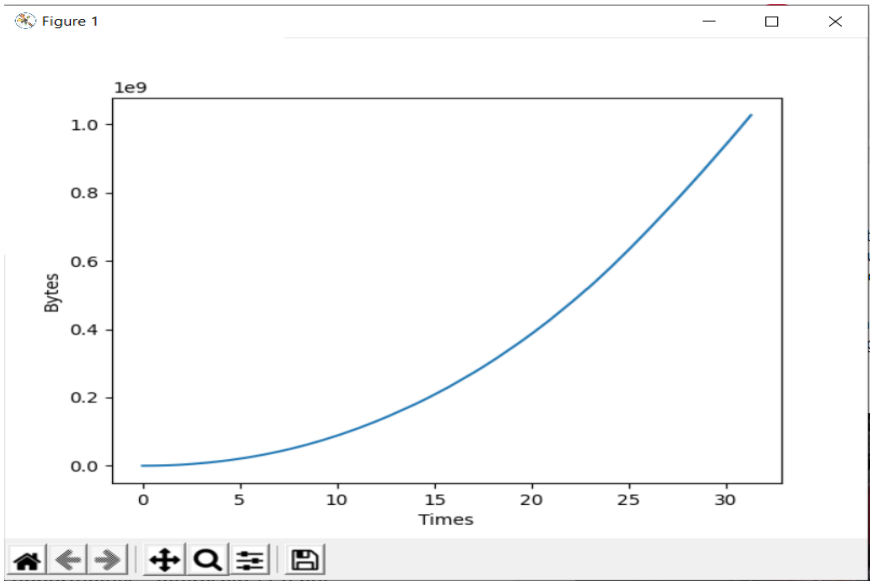
```
def makeRtp(self, payload, frameNbr):
   """RTP-packetize the video data."""
   version = 2
   padding = 0
   extension = 0
   cc = 0
   marker = 0
   pt = 26 # MJPEG type
   seqnum = frameNbr
   ssrc = 0
   rtpPacket = RtpPacket()
   rtpPacket.encode(version, padding, extension, cc, seqnum, marker, pt, ssrc, payload)
   return rtpPacket.getPacket()
```

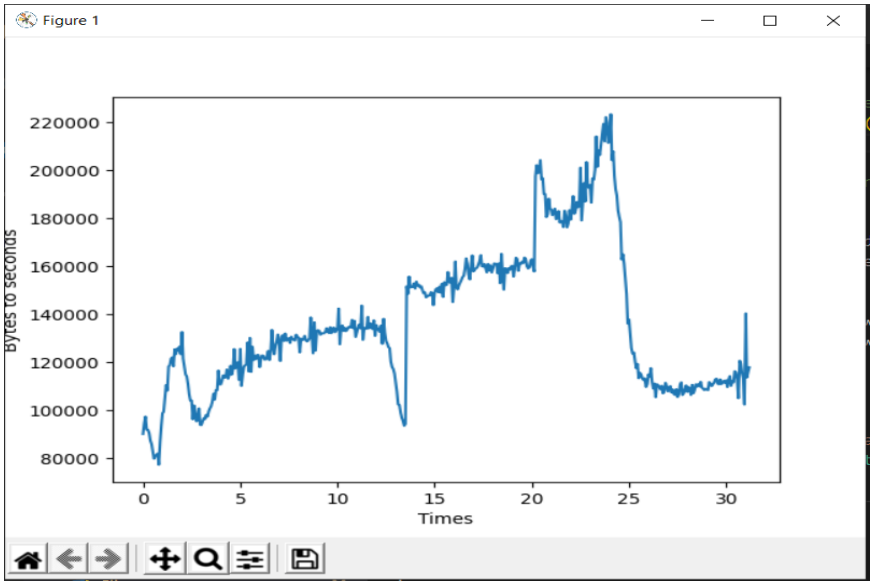
Sau khi đóng gói xong, thực hiện công việc gửi gói tin đến Client từ địa chỉ IP client và số port của Client cung cấp được lưu tương ứng vào address và port trong hàm def sendRtp(self)

3 Class diagram


Client
+ state: int
+ serverAddr: String
+ rtpPort: int
+ serverPort: int
+ fileName: String
+ rtspSeq: int
+ sessionId: int
+ requestSent: int
+ teardownAcked: int
+ frameNbr: int
+ createWidgets()
+ setupMovie()
+ exitClient()
+ pauseMovie()
+ playMovie()
+ listenRtp()
+ writeFrame()
+ updateMovie()
+ connectToServer()
+ sendRtspRequest()
+ rcvRtspReply()
+ parseRtspReply()
+ openRtpPort()
+ handler()
0...*
1
RtpPacket
+ header: bytearray
+ encode()
+ decode()
+ version()
+ seqNum()
+ timestamp()
+ payloadType()
+ getPayload()
+ getPacket()
0...*
1
ServerWorker
+ state: int
+ clientInfo: List
+ recvRtspRequest()
+ processRtspRequest()
+ sendRtp()
+ makeRtp()
+ replyRtsp()
1
1...*
VideoStream
+ fileName: String
+ nextFrame()
+ frameNbr()
1
1
Server
+ serverPort: int
+ main()
1
1...*
ClientLauncher
+ serverAddr: String
+ serverPort: int
+ rtpPort: int
+ filename: String
+ createNewClient()

4 A Summative Evaluation of Results Achieved





Đánh giá:Bên trên là biểu đồ quan hệ giữa Số lượng byte Client nhận được theo thời gian và Byte trên giây theo thời gian. Nhận thấy tốc độ truyền tải khá nhanh, mặc dù biên độ dao động lớn ở biểu đồ 2. Biểu đồ 1 cho thấy số lượng bytes nhận được theo thời gian khá ổn định, biểu đồ có độ dốc cao, cho nên dễ dàng biết được chất lượng video stream tốt, không giật lag.
Kết quả đạt được: Hệ thống chạy khá ổn định, tốc độ truyền tải tốt,báo lỗi kịp thời khi xảy ra.

5 User manual

5.1 Running server

Đầu tiên, khởi động server với dòng command sau:

```
# python Server.py server_port
```

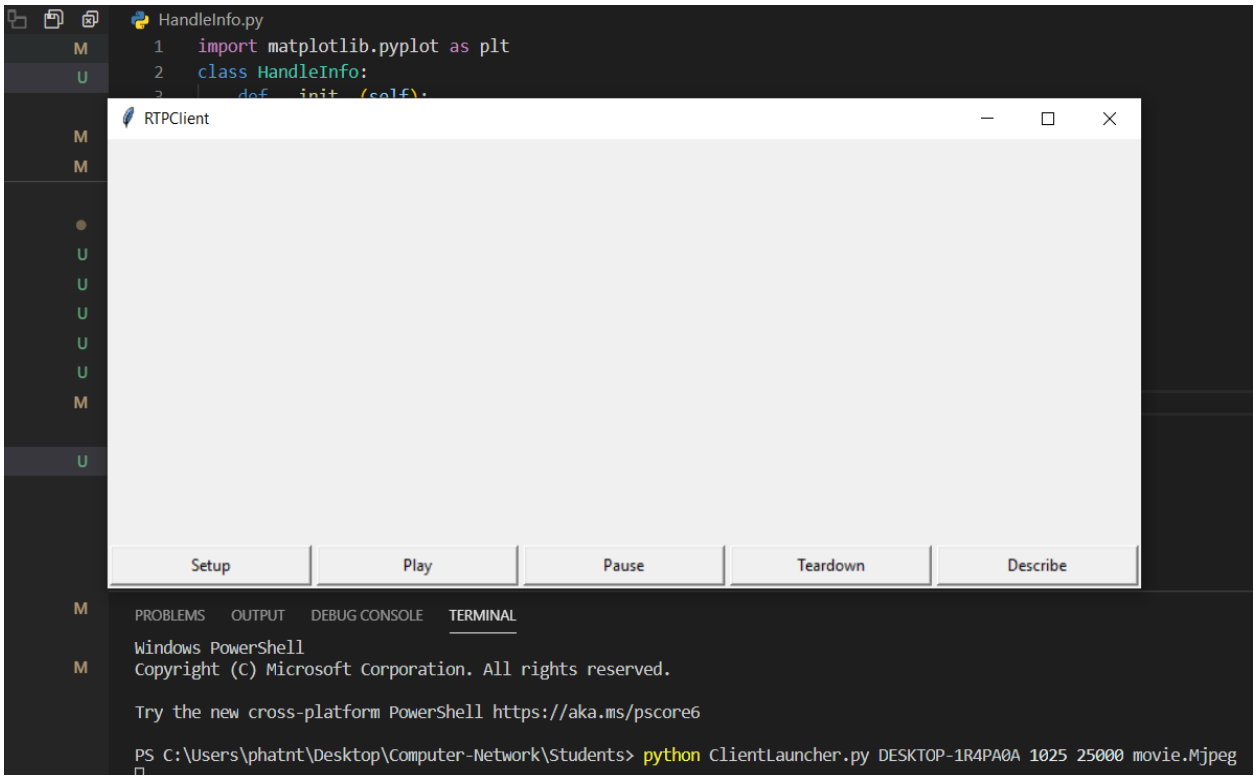
server_port là port mà server lắng nghe các kết nối RTSP đến. Port RTSP chuẩn là 554, nhưng ta cần chọn port lớn hơn 1024.

5.2 Running client

Sau đó, khởi động client với command python sau:

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

server_host là tên của máy nơi mà server đang chạy.
server_port là port nơi server lắng nghe.
RTP_port là port nơi các gói RTP được nhận.
video_file là tên của video file muốn yêu cầu (movie.Mjpeg).



Hình ảnh minh họa giao diện client sau khi running client.

Ta có thể gửi RTSP commands tới server bằng cách nhấn các button. Một tương tác RTSP bình thường diễn ra như các mục sau.

5.2.1 The clients sends SETUP

Command này dùng để thiết lập session và truyền tải tham số.

5.2.2 The client sends PLAY

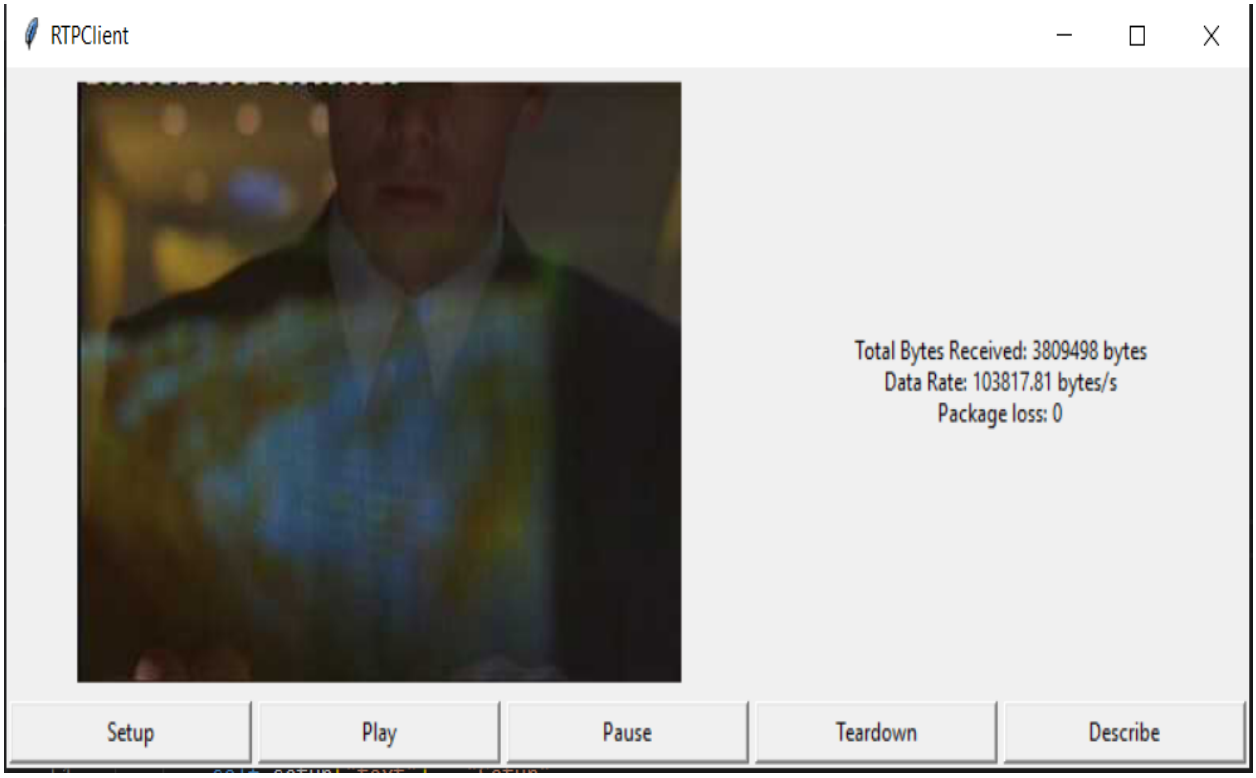
Command này bắt đầu trình phát.

5.2.3 The client may send PAUSE

Nếu muốn dừng trong quá trình phát.

5.2.4 The client sends TEARDOWN

Command này kết thúc các sessions và đóng kết nối. Server luôn phản tất cả các thông điệp mà client gửi.
Mã 200 nghĩa là yêu cầu thành công.
Mã 404 và 500 đại diện tương ứng cho lỗi FILE_NOT_FOUND và lỗi kết nối.



Hình ảnh minh họa giao diện client sau khi gửi request SETUP PLAY thành công.