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

Mạng Máy Tính - CO3093

VIDEO STREAMING APP

GVHD : Trần Huy

Sinh viên :
- Lê Viết Hòa - 1911186
- Võ Văn Tiến Dũng - 1910984
- Nguyễn Hoàng Minh Châu - 1910865
- Quách Minh Tuấn - 1910663

Tp.Hồ Chí Minh, 11/2021

Mục lục

1 Phân tích yêu cầu

Trong phần đầu tiên, chúng ta sẽ phân tích tổng quan các yêu cầu cần phải làm trong ứng dụng Video Streaming lần này.

1.1 Hiện thực giao thức RTSP ở phía Client

Ở phía Client, chúng ta cần hoàn thành các hàm để xử lí các hành động của người dùng theo yêu cầu như setupMovie, playMovie, pauseMovie và exitClient. Để hoàn thành các hàm kể trên ta phải đảm bảo rằng hàm sendRtspRequest phải được hiện thực sao cho các gói yêu cầu RTSP phải được gửi đúng với từng loại yêu cầu.
Chúng ta cũng sẽ hiện thực các hàm recvRtspReply, parseRtspReply để giúp cho phía Client nhận, phân tích và phản hồi với các RTSP reply từ phía Server.
Cuối cùng chúng ta sẽ mở cổng RTP thông qua hàm openRtpPort.

1.2 Đóng gói RTP ở phía Server

Ở phía Server, chúng ta sẽ thực hiện hàm encode trong RtpPacket.py. Đúng với tên gọi của hàm, nhiệm vụ của chúng ta là đóng gói các thông tin của RTP packet vào trong phần header và payload thông qua các thao tác với bit và byte để đặt chúng vào đúng vị trí như đã được cho trong diagram.

1.3 Thiết kế giao diện người dùng để chiếu video theo yêu cầu

Giao diện người dùng đã được tạo sẵn thông qua hàm createWidget trong Client.py, chúng ta chỉ cần thay đổi và hiện thực một số chức năng để hoàn thiện chúng.

2 Mô tả chức năng các hàm trong ứng dụng

Trong phần này ta sẽ chỉ trình bày về các hàm có ảnh hưởng lớn đến ứng dụng trong Client.py, ServerWorker.py và RtpPacket.py.

2.1 Client

Hàm khởi tạo init

- **master**: có chức năng như một root để lưu trữ toàn bộ giao diện người dùng

- **serverAddr**: chứa địa chỉ RTSP server.
- **serverPort**: chứa Port của RTSP server.
- **rtpPort**: chứa Port của RTP.
- **filename**: tên của video dùng để stream.
- **sesionId**: chứa sessionId của RTSP.
- **requestSent**: chứa thông tin của request được gửi đi.
- **teardownAcked**: khi nhận được TEARDOWN requst từ server sẽ được set về 1.
- **frameNbr**: cho biết frame hiện tại của video đang được stream.

connectToServer

Mở một TCP socket ở phía Client và kết nối đến TCP socket ở Server. Nếu kết nối thất bại sẽ hiện ra thông báo lỗi.

createWidget

Tạo ra 4 nút (SETUP, PLAY, PAUSE, TEARDOWN) và 1 label trong giao diện người dùng.

writeFrame và UpdateMovie

Hai hàm này sẽ lưu dữ liệu (ở đây là hình ảnh của từng frame) vào cache sau nó hiển thị ra giao diện người dùng.

setupMovie, playMovie, pauseMovie, exitClient

Đây là 4 hàm dùng để hiện thực chức năng cho 4 nút tương ứng SETUP, PLAY, PAUSE và TEARDOWN.

Ở hàm setupMovie sẽ có thêm phương thức connectToServer. Mục đích đặt hàm connectToServer ở SETUP là vì nhóm muốn sau khi TEARDOWN thì không cần mở lại ứng dụng mà vẫn có thể tiếp tục được.

Hàm playMovie chỉ được thực thi khi đang ở trạng thái READY và khi người dùng nhấn vào nút PLAY, Client sẽ thực thi hàm listenRtp trên 1 luồng riêng để nhận các gói tin RTP (trình bày rõ hơn ở bên dưới).

Hàm exitClient sẽ đảm nhiệm vai trò xóa toàn bộ widget bên trong master và dọn file cache.

listenRTP

Hàm listenRTP liên tục nhận về các gói tin RTP sau đó giải mã chúng và hiển thị ra giao diện người dùng thông qua hàm updateMovie và writeFrame.

Trong hàm này, nhóm còn tạo thêm một số biến dùng để phục vụ cho phần mở rộng như packetLoss dùng để tính Loss Rate, packetSlow dùng để tính số gói tin chậm và videoData dùng để tính lưu lượng dữ liệu.

Bên cạnh đó hàm còn được thiết lập để ngùng nhận dữ liệu khi PAUSE hoặc TEARDOWN

sendRtspRequest

Đảm nhiệm chức năng gửi yêu cầu đến server. Các request trước khi gửi phải được kiểm tra xem có đang ở trong trạng thái thích hợp không.

Trong trường hợp SETUP, yêu cầu gửi đi sẽ khác đôi chút so với các yêu cầu còn lại. Dòng cuối cùng của yêu cầu SETUP sẽ chứa giao thức sử dụng và Port của RTP do đây là lần đầu kết nối. Các yêu cầu khác thì dòng cuối sẽ cho biết session hiện tại.

Cuối cùng các yêu cầu sẽ được encode lại và gửi đến server.

recvRtspReply

Có chức năng nhận các phản hồi RTSP từ server, decode các phản hồi và chuyển đến parseRtspReply. Nếu trạng thái hiện tại là TEARDOWN sẽ đóng RTSP socket.

parseRtspReply

Sau khi nhận về phản hồi đã được giải mã từ recvRtspReply, parseRtspReply có nhiệm vụ phân tích các phản hồi, kiểm tra và cập nhật lại trạng thái tùy theo từng phản hồi.

Nếu code là 200 thì sẽ thực hiện các bước kiểm tra trước khi cập nhật lại trạng thái, nếu code là 400 hoặc 500 thì sẽ đưa ra lỗi tương ứng.

Kiểm tra bao gồm kiểm tra xem sequence number từ RTSP có cùng sequence number của yêu cầu hay không. Tiếp theo là kiểm tra xem sessionID có giống nhau không.

Sau khi các bước kiểm tra hoàn tất, phía Client sẽ cập nhật trạng thái tùy theo yêu cầu. Nếu yêu cầu là SETUP thì sẽ đặt sessionID bằng với session trả về sau đó cập nhật lại trạng thái thành READY và mở cổng RTP. Nếu yêu cầu là PLAY trạng thái sẽ được cập nhật thành PLAYING. Nếu yêu cầu là PAUSE sẽ cập nhật trạng thái thành READY. Nếu yêu cầu là TEARDOWN thì sẽ đặt lại trạng thái là INIT và tearDownAcked được đặt về 1.

openRtpPort

Tạo ra RTP socket phía Client để nhận về các gói tin RTP từ server.

handler Dùng để hiện thực cho thao tác đóng ứng dụng. Khi người dùng click vào nút đóng cửa sổ, sẽ hiện lên thông báo: "Quit?", "Are you sure you want to quit?". Nếu người dùng đồng ý thì sẽ xóa toàn bộ file cache, dừng chương trình và xóa toàn bộ giao diện người dùng trong master. Nếu người dùng chọn cancel thì sẽ tiếp tục chiếu video.

2.2 Server

Hàm khởi tạo init

Khởi tạo một dictionary có tên clientInfo để lưu trữ các thuộc tính.

run

Bắt đầu một luồng đã khởi tạo ở init.

recvRtspRequest

Nhận các yêu cầu RTSP từ phía Client sau đó giải mã và chuyển cho processRtspReply.

processRtspReply

Có chức năng tương tự như parseRtspReply của phía Client. Sau khi nhận được các yêu cầu đã được giải mã từ recvRtspReply, processRtspReply sẽ bắt đầu phân tích các yêu cầu đó, xử lí các yêu cầu, cập nhật lại trạng thái tùy theo từng yêu cầu và cuối cùng là gửi lại một phản hồi cho Client thông qua replyRtsp. Nếu yêu cầu là SETUP, trạng thái sẽ được đặt thành READY, sau đó một RTSP session ID sẽ được tạo ngẫu nhiên và Server sẽ nhận được RTP port từ CLient.

Nếu yêu cầu là PLAY, trạng thái sẽ được đặt thành PLAYING, sau đó server sẽ mở một RTP/UDP socket để bắt đầu stream video, Một luồng mới cũng được tạo ra dành riêng cho việc này.

Nếu yêu cầu là PAUSE trạng thái sẽ được đặt về READY. Nếu yêu cầu là TEARDOWN, Server sẽ đóng RTP socket.

sendRtp

Gửi các gói tin RTP có chứa frame của video đang stream mỗi 0.05s.

makeRtp

Có chức năng encode các gói tin RTP thông qua phương thức encode của RtpPacket.

replyRtsp

Gửi phàn hồi về phía Client. Code 200 tương ứng với" OK", code 404 tương ứng "FILE NOT FOUND", code 500 tương ứng "CONNECTION ERROR".

2.3 RtpPacket

encode

Gán các bits vào đúng vị trí trong header field của gói tin RTP. Cuối cùng là là thêm payload vào gói tin.

decode

Dùng để giải mã gói tin RTP.

Các hàm khác

Chỉ đơn giản là các hàm hỗ trợ lấy các giá trị từ header field hoặc lấy giá trị payload.

3 Danh sách các Components

Các component bao gồm:

- Server: tạo ra thread để chạy ServerWorker.
- ClientLauncher: khởi động Client.
- Client: gửi các yêu cầu RTSP đến Server và nhận các dữ liệu liên quan đến video để trình chiếu.
- ServerWorker: gửi các phản hồi RTSP và dữ liệu liên quan đến video thông qua các gói tin RTP cho phía Client.
- RtpPacket: đóng gói frame thành các gói tin RTP để Server gửi cho Client.
- VideoStream: có các hàm hỗ trợ liên quan đến frame number.
- Client3Button: có chức năng tương tự Client nhưng được thiết kế chỉ còn 3 nút.
- ServerWorkerExtend và ClientExtend: có chức năng tương tự như ServerWorker và Client nhưng được thiết kế để xử lí thêm trạng thái SWITCH và một vài chức năng phụ như tua video tiến/lùi 5s và thanh kéo hỗ trợ việc tua.

4 Model and data flow

Dưới đây là quá trình từ khi người dùng bắt đầu stream video cho đến khi kết thúc.

Khởi động ứng dụng - Client

Người dùng mở ứng dụng, trạng thái được đặt thành INIT

Nhấn vào SETUP - Client

Sau khi nhấn vào SETUP, phía Client lần lượt thực hiện các bước:

- Kết nối đến Server.
- Bắt đầu nhận các phản hồi RTSP từ Server.

- Gửi yêu cầu SETUP đến Server.

Xử lí yêu cầu SETUP - Server

Phía Server xử lí yêu cầu SETUP của Client theo các bước:

- Giải mã yêu cầu từ Client.
- Phân tích yêu cầu, xác nhận là yêu cầu SETUP.
- Cập nhật trạng thái thành READY nếu có video hoặc trả về kết quả không tìm thấy file.
- Tọa ra một session ID ngẫu nhiên từ 100000 đến 999999.
- Gửi phản hồi về Client.
- Nhận giá trị rtpPort của Client.

Xử lí phản hồi cho yêu cầu SETUP từ Server - CLient

- Nhận, phân tích và xử lí phản hồi. Nếu Code là 400 sẽ báo lỗi 'File no found'. Nếu Code là 500 sẽ báo lỗi 'Connection error'. Nếu Code là 200, tiếp tục kiểm tra seuquence number. Sau khi kiểm tra thành công sẽ cập nhật lại trạng thái READY, session ID và mở RTP port.
- Một RTP socket sẽ được tạo ra để sẵn sàng nhận các gói tin RTP từ server.

Người dùng nhấn vào PLAY để bắt đầu xem video - CLient

- Client bắt đầu nghe những gói tin RTP từ Server.
- Gửi yêu cầu PLAY đến Server.

Xử lí yêu cầu PLAY - Server

- Gửi Code 200 cho phía Client.
- Tạo một RTP socket ở phía Server.
- Tạo một thread để bắt đầu gửi các gói tin RTP.

Xử lí phản hồi cho yêu cầu SETUP từ Server - Client

- Cập nhật lại trạng thái thành PLAYING.
- Ghi các khung hình nhận được vào một file tạm thời sau đó cập nhật lên giao diện người dùng.

Người dùng nhấn vào PAUSE để dừng video - Client

- Dừng xem video.
- Gửi yêu cầu PAUSE đến Server.

Xử lí yêu cầu PAUSE - Server

- Cập nhật lại trạng thái thành READY.
- Gửi Codoe 200 cho Client.

Người dùng nhấn vào TEARDOWN để kết thúc xem video - Client

- Gửi yêu cầu TEARDOWN đến Server.
- Đóng RTP port, ngừng nhận dữ liệu từ Server.
- Xóa toàn bộ file cache trong ứng dụng.

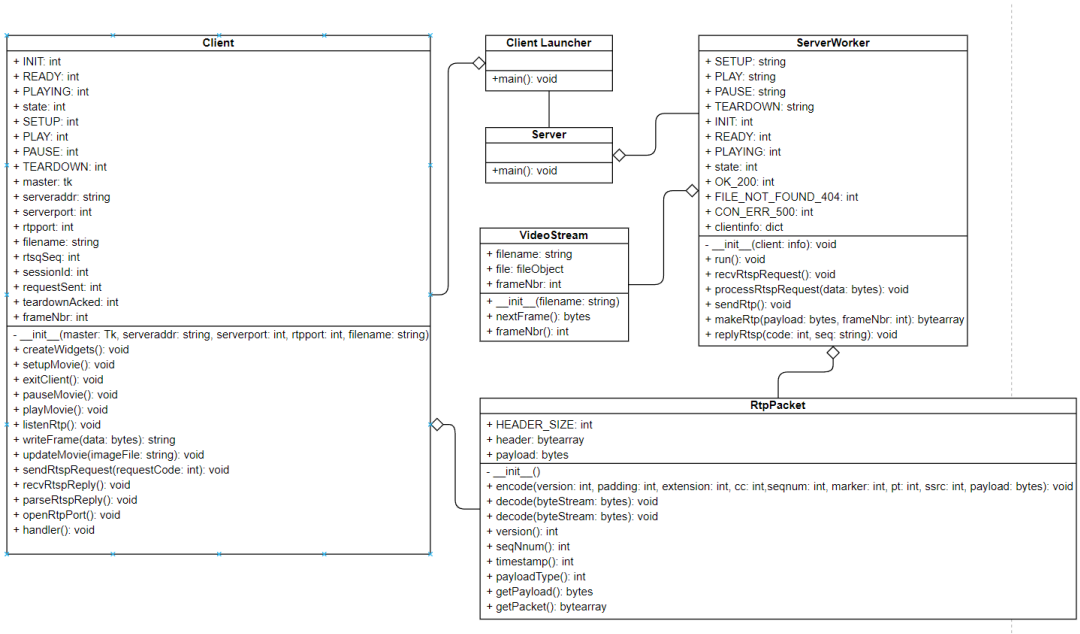
Xử lí yêu cầu PAUSE - Server

- Gửi Codoe 200 cho Client.
- Đóng RTP socket phía Server.

Người dùng nhấn vào nút Close - Client

- Dừng video.
- Hiện bảng thông báo: "Quit?", "Are you sure you want to quit?".
- Nếu người dùng chọn Ok, Client sẽ xóa toàn bộ file cache, thực hiện TEARDOWN và bỏ toàn bộ giao diện người dùng.
- Nếu người dùng chọn Cancel thì sẽ tiếp tục chiếu video.

5 Class diagram



Hình 1: Class diagram

6 Các hàm trong phần mở rộng

6.1 Tính toán thông số

RTP Packet Loss Rate, Slow Rate và Data Rate

Nhóm hiện thực phần tính 3 thông số này trong hàm listenRtp của Client.py.

Đối với việc tính thông số Loss Rate, nhóm sẽ đếm các gói tin bị mất (packetLoss) bằng việc so sánh sequence number của gói tin RTP được nhận với frame number mà nó đáng ra nên được gửi, tức là $framenumber + 1$. Nếu như sequence number lớn hơn, ta sẽ tính số gói tin bị mất bằng cách lấy $sequencenumber - (framenumber + 1)$. Loss Rate sẽ được tính theo công thức:

$$LossRate = (packetLoss - packetSlow)/frameNbr$$

Tương tự với Slow Rate, tuy nhiên ta sẽ tính số gói tin bị chậm(packetSlow). Nếu sequence number bé hơn frame number, tức là gói tin đó bị chậm và ta chỉ cần tăng số gói tin bị chậm lên 1. Slow Rate sẽ được tính theo công thức:

$$SlowRate = packetSlow/frameNbr$$

Cuối cùng với Data Rate, ta sẽ tính lượng dữ liệu được truyền (videoData) thông qua chiều dài của payload cùng với đó là đo tổng thời gian của stream thông qua 2 biến start và end. Data Rate

sẽ được tính theo công thức:

$$DataRate = videoData/(timeEnd - timeStart).$$

```
def listenRtp(self):
  """Listen for RTP packets."""
  packetLoss = 0
  packetSlow = 0
  videoData = 0
  start = time()
  while True:
    try:
      data = self.rtpSocket.recv(20480)
      if data:
        rtpPacket = RtpPacket()
        rtpPacket.decode(data)
        currFrameNbr = rtpPacket.seqNum()
        # Count loss packet
        if currFrameNbr > self.frameNbr + 1:
          packetLoss += currFrameNbr - (self.frameNbr + 1)
        # Count slow packet
        if currFrameNbr < self.frameNbr:
          packetSlow += 1
        # Update frame
        if currFrameNbr > self.frameNbr:
          self.frameNbr = currFrameNbr
          payload = rtpPacket.getPayload()
          self.updateMovie(self.writeFrame(payload))
          # Count video data
          videoData += len(payload)
    except:
      # Stop listening if request is PAUSE or TEARDOWN
      if self.playEvent.isSet():
        break
      if self.teardownAcked:
        self.rtpSocket.shutdown(socket.SHUT_RDWR)
        self.rtpSocket.close()
        break
  end = time()
  # Calc and print data transmission parameters
  print("\n================================")
  print(f"RTP Packet Loss Rate = {packetLoss-packetSlow}/{self.frameNbr} = {100 * (packetLoss-
   packetSlow)/self.frameNbr} %")
  print(f"RTP Packet Slow Rate = {packetSlow}/{self.frameNbr} = {100 * packetSlow /self.frameNbr}
    %")
  print(f"Video data rate = {videoData}/{end - start} = {videoData/(end - start)} bytes/sec")
```

```
41    print("================================\n")
```

6.2 Client3Button

Các hàm trong Client 3 dùng để hỗ trợ việc hiện thực với chỉ 3 nút.

createWidgets

Tương tự như createWidgets của Client, hàm createWidgets của Client3Button chỉ không hiện thực phần tạo nút SETUP và sửa nút TEARDOWN thành nút STOP (chức năng tương tự TEARDOWN).

playMovie

Để xử lí việc không có nút SETUP, nhóm quyết định đưa hàm setupMovie vào trong playMovie. Khi đó, nếu người dùng nhấn PLAY khi đang trong trạng thái INIT (lần đầu mở ứng dụng hoặc sau khi TEARDOWN), ứng dụng sẽ lần lược gửi yêu cầu SETUP và PLAY cho server.

```
1   def playMovie(self):
2     """Play button handler."""
3     # If not yet setup connection, setup it.
4     if self.state == Client.INIT:
5       self.setupMovie()
6       while self.state == Client.INIT:
7         continue
8     # Play movie
9     if self.state == Client.READY:
10      self.playEvent = threading.Event()
11      self.playEvent.clear()
12      threading.Thread(target=self.listenRtp).start()
13      self.sendRtspRequest(Client.PLAY)
```

6.3 Hiện thực chức năng DESCRIBE

Một nút DESCRIBE sẽ được tạo ở phần giao diện người dùng. Khi nhấn vào, client sẽ gửi một request DESCRIBE có định dạng như sau:

```
DESCRIBE <videoname> RTSP/1.0
CSeq: <rtspSeq>
Session: <session ID>
```

Ở phía Server, khi nhận được yêu cầu DESCRIBE, Server sẽ gửi về thông tin của session hiện tại

cho phía Client.

```
    if type == self.DESCRIBE:
      reply += f"\n\nstream=video\n"
      reply += f"type=Mjpeg\n"
      reply += f"NumFrame={int(self.clientInfo['videoStream'].totalTime()*20)}\n"
      reply += f"protocol=RTP\n"
      reply += f"version=2\n"
```

Cuối cùng Client sẽ hiện thị các thông tin đó ra giao diện người dùng.

Hình 2: Giao diện DESCRIBE button

6.4 Hiện thực các chức năng điều khiển thời gian

Hình 3: Giao diện chỉnh sửa thời gian

Các chức năng điều khiển video được hiện thực bao gồm: hiển thị tổng thời gian của video, tua nhanh 5s, tua lùi lại 5s, thanh kéo để tua thời gian (có hiển thị thời gian hiện tại trên thanh kéo cho người dùng dễ sử dụng).

Để hiện thực các chức năng này, ta cần thay đổi các đọc file trong file VideoStream.py, từ cách đọc tuần tự các frame thành đọc tất cả các frame từ đầu để có thông tin về tổng số frame và có thể linh hoạt tua video.

```
def __init__(self, filename):
  self.data = []
  try:
    file = open(filename, 'rb')
    while True:
      data = file.read(5)
      if data:
```

```
        framelength = int(data)
        self.data.append(file.read(framelength))
      else:
        break
  except:
    raise IOError
  self.frameNum = 0
```

Các chức năng này sẽ được tạo và hiện thực trong ClientExtend.py và ServerWorkerExtend.py

Đối với việc hiển thị tổng thời gian của video, phía server sẽ gửi thêm thông tin về tổng thời gian ở phản hồi yêu cầu PLAY:

```
    if type == self.PLAY:
      reply += f"\nTTtime: {int(self.clientInfo['videoStream'].totalTime())}"
```

Sau đó, phía client sẽ nhận thông tin và cài đặt các thông số liên quan

```
def setTotalTimeVideo(self, totalFrame):
  # Display total time
  self.totalTime["text"] = f"{totalFrame/20}"
  self.scroll["to"] = totalFrame/20
  self.totalFrame = totalFrame
```

Đối với các hàm liên quan đến việc thiết lập thời gian như tua nhanh 5s, tua lùi lại 5s hay thanh kéo để tua thời gian, chúng đều có cùng điểm chung là thực hiện theo các bước: kiểm tra trạng thái, thiết lập lại frame number cho phù hợp với thời gian được tua rồi gửi yêu cầu SETTIME đến Server.

SETTIME request được định dạng như sau:

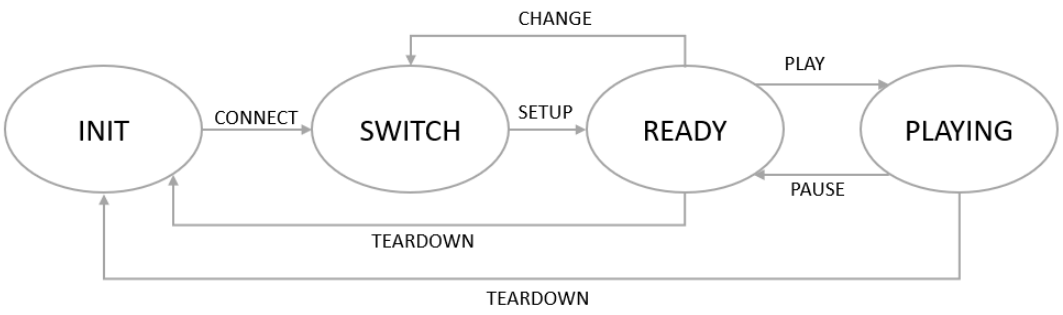
```
SETTIME <file name> RTSP/1.0
CSeq: <rtspSeq>
Session: <sessionID>
FRAME: <frame number>
```

Trong đó: frame number là số thứ tự frame sau khi đã điều chỉnh thời gian.

```
def forwardVideo(self):
  """Forward video button handler."""
  if self.state != self.PLAYING and self.state != self.READY:
    return
  self.frameNbr += 20 * 5 # 20 frame each second -> 5 second = 20 * 5 frame
  # Check if forward exceed total time of video
  if self.frameNbr > self.totalFrame:
```

```
8       self.frameNbr = self.totalFrame
9     # Send request
10    self.sendRtspRequest(ClientExtend.SETTIME)
11
12  def backwardVideo(self):
13    """Backward video button handler."""
14    if self.state != self.PLAYING and self.state != self.READY:
15      return
16    self.frameNbr -= 20 * 5 # 20 frame each second -> 5 second = 20 * 5 frame
17    # Check if backward exceed 0
18    if self.frameNbr < 0:
19      self.frameNbr = 0
20    # Send request
21    self.sendRtspRequest(self.SETTIME)
22
23  def settime(self, value):
24    """Scroll bar handler."""
25    value = int(value)
26    # Only send request if user change time
27    if self.frameNbr > value * 20 + 10 or self.frameNbr < value * 20 - 10:
28      self.frameNbr = int(value) * 20
29      self.sendRtspRequest(self.SETTIME)
```

6.5 Thêm trạng thái SWITCH

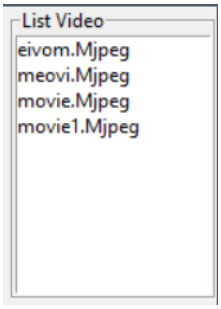


Hình 4: Diagram mới sau khi có thêm trạng thái SWITCH

Để phục vụ việc thêm trạng thái SWITCH, ta sẽ có thêm 1 danh sách các video ở bên phải giao diện người dùng. Các video này sẽ được lấy từ Server thông qua GETLIST request được định dạng như sau:

```
GETLIST / RTSP/1.0
CSeq: <rtspSeq>
Session: <sessionID>
```

```
def getListVideo(self):
  """Get List video request"""
  while self.state == Client.INIT:
    continue
  self.sendRtspRequest(ClientExtend.GETLIST)
```



Hình 5: Giao diện danh sách video

Khi người dùng double click vào một video ở trong danh sách các video, ứng dụng sẽ gọi đến hàm switchVideo. Tại đây, chương trình sẽ kiểm tra trạng thái, tạm dừng video nếu cần sau đó sẽ gửi yêu cầu CHANGE đến Server.

```
CHANGE <file name> RTSP/1.0
CSeq: <rtspSeq>
Session: <sessionID>
```

```
def switchVideo(self, any):
  """Switch video handler."""
  # If video is playing, PAUSE it
  if self.state == ClientExtend.PLAYING:
    self.pauseMovie()
    while self.state == Client.PLAYING:
      continue
  # Send CHANGE video request
  if self.state == ClientExtend.READY or self.state == ClientExtend.SWITCH:
    self.fileName = self.listVideo.get(ACTIVE)
    self.sendRtspRequest(ClientExtend.CHANGE)
```

về phía Server, khi yêu cầu CHANGE được gửi đến, Server sẽ kiểm tra trạng thái, thay đổi video được stream, cập nhật lại trạng thái thành SWITCH và gửi lại phản hồi về phía Client.

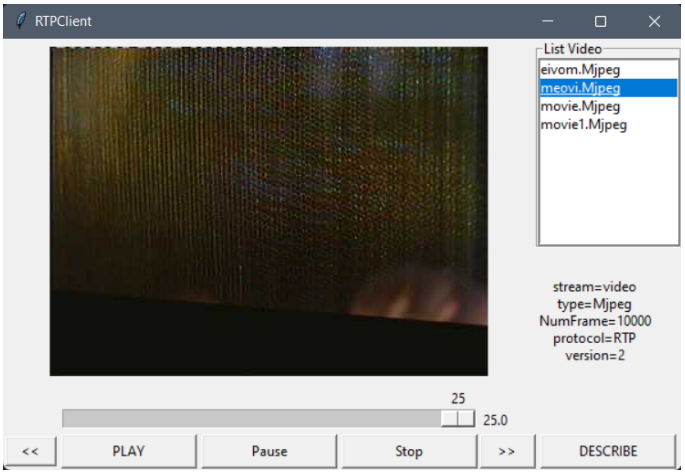
```
  # Process CHANGE request
  elif requestType == self.CHANGE:
    if self.state == self.READY or self.state == self.SWITCH:
      print("processing CHANGE")
      # Check if video exists
      try:
        self.clientInfo['videoStream'] = VideoStream('video/'+filename)
        self.state = self.SWITCH
      except IOError:
        self.replyRtsp(self.FILE_NOT_FOUND_404, seq[1])
      # Send RTSP reply
      self.replyRtsp(self.OK_200, seq[1])
```

Sau khi nhận được phản hồi từ phía Server, Client cũng sẽ cập nhật lại trạng thái thành SWITCH, đổi chữ trên nút PLAY thành RELOAD, khi người dùng nhấn vào RELOAD thì sẽ chiếu video vừa đổi.

```
        elif self.requestSent == ClientExtend.CHANGE:
          self.state = ClientExtend.SWITCH
          self.start['text'] = 'RELOAD'
```

7 Đánh giá kết quả đạt được

- Về các nhiệm vụ được giao, nhóm đã hoàn thành hết tất cả yêu cầu được giao của bài tập lớn (cả phần Requirements lẫn Extend). Tạo ra 3 phiên bản ứng dụng Client gồm Client.py (giải quyết các yêu cầu cơ bản), Client3Button.py (loại bỏ nút Setup), ClientExtend (hiện thực các chức năng mô tả session, điều chỉnh thời gian video, đổi video). Đồng thời cũng tạo ra thêm 1 phiên bản ServerWorkerExtend.py để phục vụ cho các chức năng mà phiên bản ClientExtend.py yêu cầu.



Hình 6: Giao diện của phiên bản cuối cùng của ứng dụng

- Về hiệu năng, khi chạy cả server lẫn client trên cùng 1 localhost, với cấu hình CPU Intel(R) Core(TM) i5-9300H và Ram 16GB:

```
RTP Packet Loss Rate = 0/500 = 0.0 %
RTP Packet Slow Rate = 0/500 = 0.0 %
Video data rate = 4267393/31.493650436401367 = 135500.1068744832 bytes/sec
```

  Tuy nhiên, thời gian thực của video chỉ có 25 giây nhưng thời gian để server phát video tới client ở lần đo trên gần tới 31.5 giây (kể cả thời gian setup). Cho thấy hiệu năng của ứng dụng vẫn chưa được cao, video chậm khoảng: 31.5/25=1.26 lần so với thực tế.

  Ngoài ra, các chức năng của ứng dụng client đều chạy bình thường và ổn định.

- Qua bài tập lớn này, nhóm đã có cơ hội thực hành sử dụng các giao thức ở tầng applicationg cụ thể là RTSP và RTP để tạo ra một ứng dụng mạng đơn giản. Đồng thời nhóm cũng thực hành sử dụng các socket của tầng transport để giao tiếp real-time giữa client-server.

- Ngoài ra, các thành viên trong nhóm học thêm được cách phối hợp làm việc với nhau để đạt hiệu quả công việc tốt nhất, cùng nhau chia sẻ và tìm hiểu kiến thức về đề tài được giao. Sử dụng các ứng dụng/trang web hỗ trợ làm việc nhóm như: source control (github), soạn thảo văn bản (overleaf), giao tiếp/trao đổi từ xa (meta messenger, google meet).

8 Hướng dẫn sử dụng

8.1 Chạy Server

Để chạy Server cho các yêu cầu bình thường, ta sử dụng câu lệnh:

```
python Server.py <server_port>
```

Để chạy Server cho các yêu cầu phần mở rộng, ta sử dụng câu lệnh:

```
python Server.py <server port> 1
```

8.2 Chạy Client

Để chạy Client cho các yêu cầu bình thường, ta dùng câu lệnh:

```
python ClientLauncher.py <server addr> <server port> <RTP port> <video file>
```

Chạy Client cho yêu cầu 3-Button:

```
python ClientLauncher.py <server addr> <server port> <RTP port> <video file> 1
```

Chạy Client cho yêu cầu mở rộng:

```
python ClientLauncher.py <server addr> <server port> <RTP port> <video file> 2
```

9 Mã nguồn

Github: https://github.com/hoale0231/RTSP-RTP-Media_Stream.git.