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)

Bài tập lớn 1

Video Streaming Application

Lớp : L02 - Nhóm : 9

GVHD: Lê Bảo Thịnh

SV thực hiện: Phạm Đình Văn ——- 2015025
Ngô Nhật Thiên ——- 2014567
Nguyễn Đại Tiến —— 2114988

Tp. Hồ Chí Minh, Tháng 4/2023

Mục Lục

1 Danh sách thành viên & Phân chia công việc

STT	Tên	MSSV	Nhiệm vụ
1	Phạm Đình Văn	2015025	Hiện thực code
2	Ngô Nhật Thiên	2014567	Hiện thực code
3	Nguyễn Đại Tiến	2114988	Hiện thực code, làm báo cáo, thuyết trình

2 Analysis of problem requirements

2.1 Yêu cầu chức năng

- Hệ thống có thể hoạt động được, giao tiếp thông qua giao thức RTSP/RTP.
- Đầy đủ các chức năng thiết yếu như pause (tạm dừng video), play(phát video), teardown(dừng video, đóng ứng dụng).
- Có thể có thêm các chức năng phụ như switch(chuyển video), describe(mô tả video), stop(dừng hẳn video về ban đầu).

2.2 Yêu cầu phi chức năng

- Giao diện thân thiện, dễ dàng sử dụng.
- Video được hiển thị trong một khung hình vừa phải, giúp người dùng dễ dàng trong việc xem video.
- Thời gian phản hồi video ngắn.
- Bắt tất cả các ngoại lệ xảy ra trong ứng dụng.

3 Description of different functions

3.1 ClientLauncher

- Xử lý lệnh nhập từ phía user để lấy thông tin.
- Tạo một giao diện người dùng.
- Tạo ra Client để kết nối đến Server.

3.2 Client

- createWidgets() : tạo giao diện chương trình.
- setupMovie() : xử lý cho nút Setup, mở file video.
- exitClient() : xử lý cho nút Teardown, thoát chương trình.
- pauseMovie() : xử lý cho nút Pause, dừng video.
- playMovie() : xử lý cho nút Play, phát video.
- setStop() : xử lý cho nút Stop, dừng video về trạng thái ban đầu.

- setDescribe() : xử lý cho nút Describe, mô tả về video.
- setSwitch() : xử lý cho nút Switch, chuyển sang video tiếp theo.
- listenRtp() : liên tục cập nhật các frame trong suốt quá trình phát video.
- writeFrame(data) : ghi data frame đến file ảnh, trả về file ảnh.
- updateMovie(imageFile) : cập nhật file ảnh trên giao diện GUI.
- connectToServer() : tạo RTSP socket kết nối đến Server.
- sendRtspRequest(requestCode) : xử lý gửi cho từng loại yêu cầu.
- recvRtspReply() : nhận phản hồi RTSP từ Server.
- parseRtspReply(data) : xử lý dữ liệu nhận được từ phản hồi RTSP của Server.
- openRtpPort() : tạo cổng RTP socket để chờ nhận các video frame từ Server.
- handler() : xử lý khi người dùng bấm vào nút [X] ở trên góc phải

3.3 Server

- main() : xử lý nhập lệnh từ Server, tạo RTSP socket để nhận các yêu cầu từ Client

3.4 ServerWorker

- run() : tạo luồng thực thi nhận yêu cầu RTSP từ Client.
- recvRtspRequest() : nhận yêu cầu RTSP từ Client.
- processRtspRequest() : xác định và xử lý từng loại yêu cầu.
- sendRtp() : gửi các video frame thông qua UDP socket.
- makeRtp(payload, frameNbr) : dùng encode để đóng gói header và payload, gửi đến máy khách. Trả về gói RTP vừa được tạo.
- replyRtsp(code, seq) : gửi phản hồi RTSP đến Client.
- replyRtsp_describe(code, seq): gửi phản hồi RTSP cho yêu cầu Describe đến Client.

3.5 RtpPacket

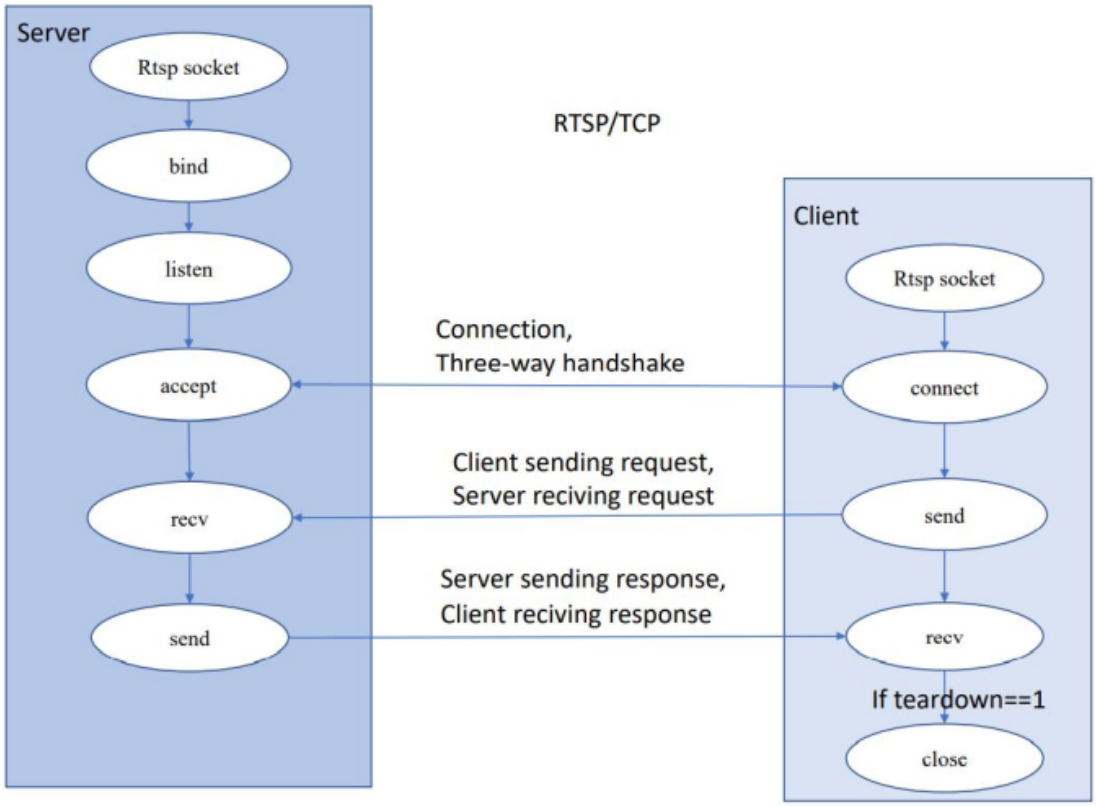
- encode(version, padding, extension, cc, seqnum, marker, pt, ssrc, payload) : encode các thông số vào header theo đúng định dạng, update lại header và payload.
- decode() : tách gói RTP thành 2 phần header và payload.
- version() : trả về version của gói RTP.
- seqNum() : trả về sequence number của gói RTP.
- timestamp() : trả về timestamp của gói RTP.
- payloadType() : trả về kiểu của payload.
- getPayload() : trả về payload trong packet.
- getPacket() : trả về packet (gồm header và payload).

3.6 VideoStream

- nextFrame() : trả về frame tiếp theo của video.
- frameNbr() : trả về frame number.
- getSize() : trả về kích thước của file video.

4 Model and data flow

4.1 RTSP protocol

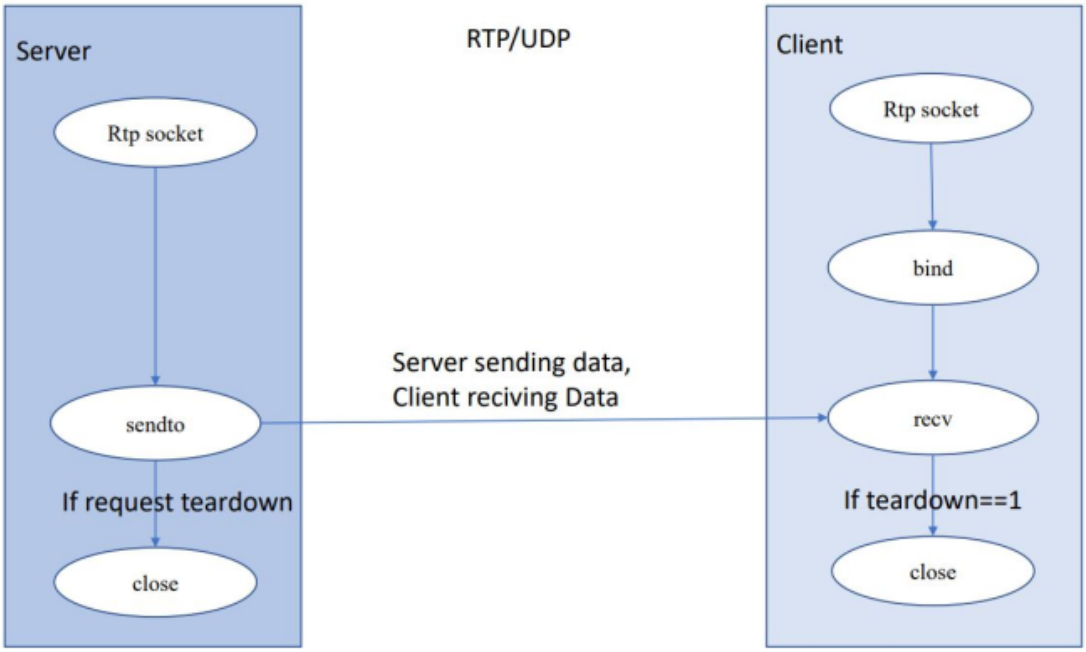


Hình 1: Giao tiếp giữa Client và Server qua giao thức RTSP/TCP

- Server tạo TCP socket lắng nghe các yêu cầu từ Client.
- Client tạo TCP socket gửi yêu cầu thiết lập đến Server.
- Server chấp nhận kết nối từ Client.
- Client gửi yêu cầu đến Server.

- Server tiếp nhận yêu cầu, xử lý yêu cầu và gửi phản hồi trở lại Client.
- Client nhận được phản hồi, đóng cổng TCP socket.

4.2 RTP protocol

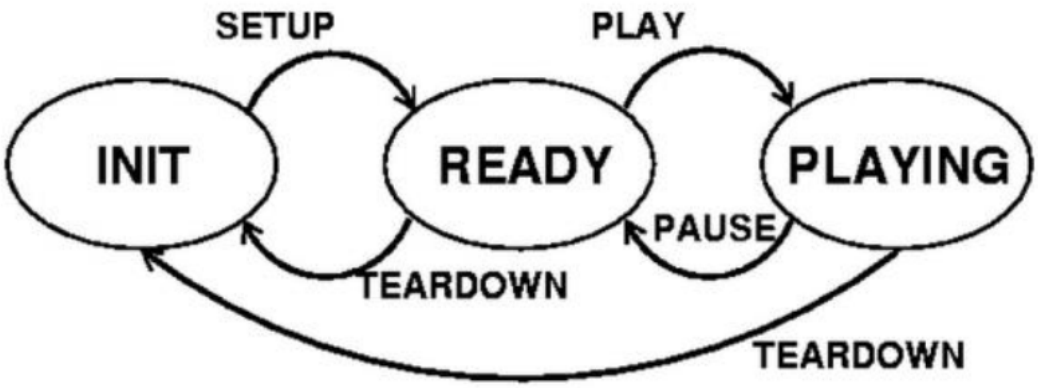


Hình 2: Giao tiếp giữa Client và Server qua giao thức RTP/UDP

- Server tạo UDP socket gửi các video packet đến Client.
- Client tạo UDP socket gửi các video packet từ Server.
- Nếu yêu cầu là Teardown, Client và Server đóng cổng UDP socket.

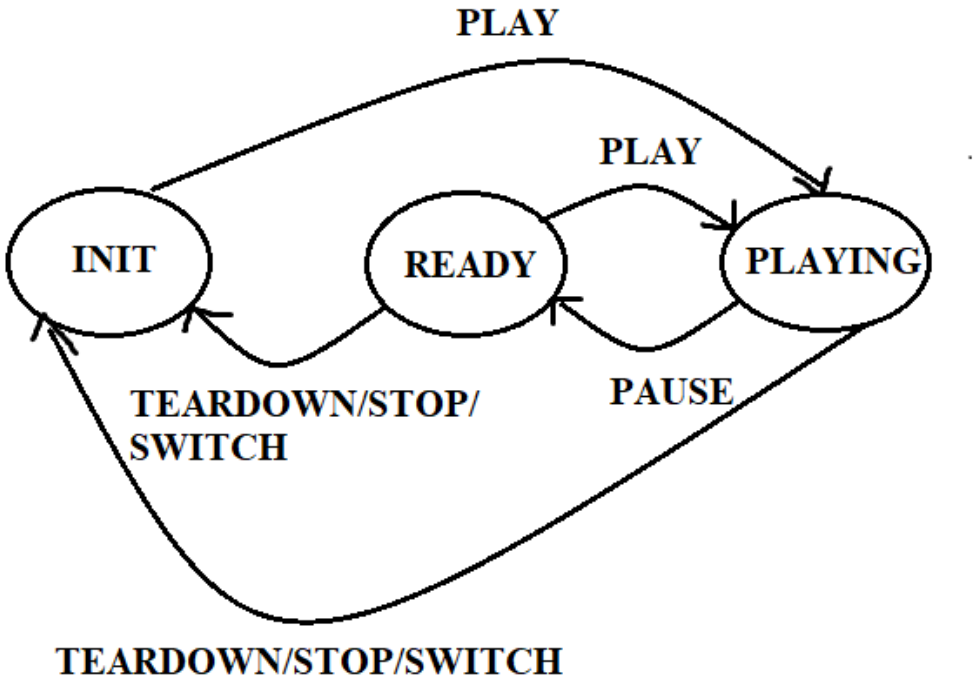
4.3 State diagram

1. Requirement



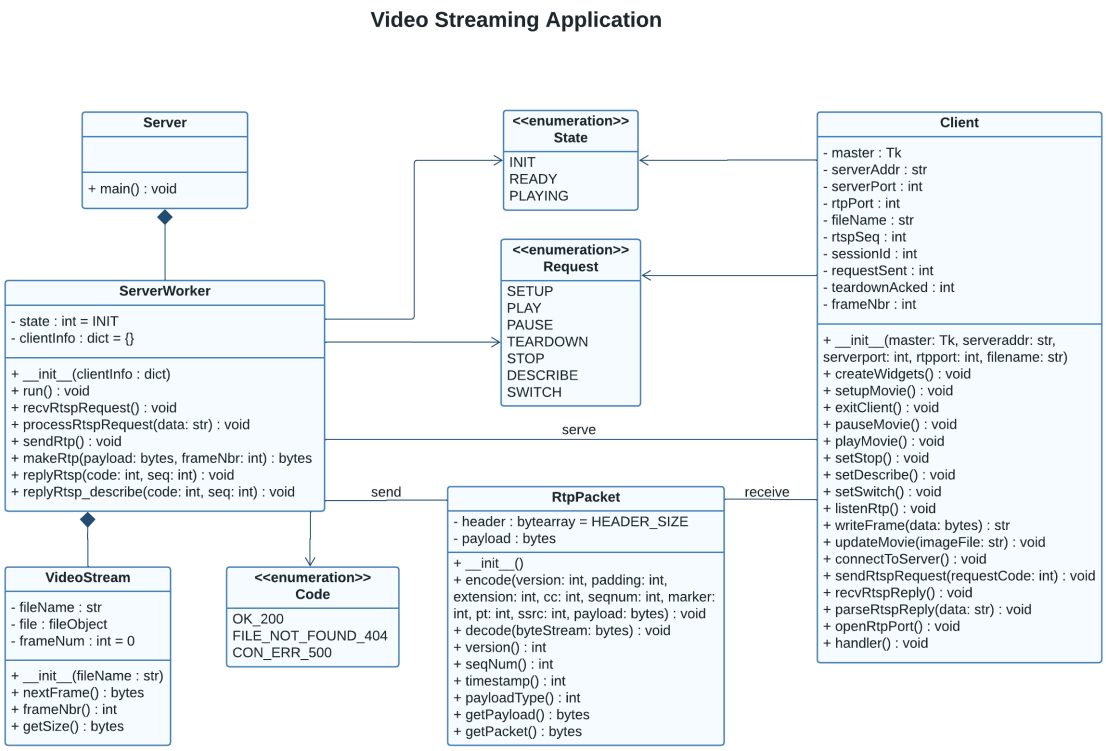
Hình 3: State diagram

2. Extend



Hình 4: State diagram

5 Class diagram



Hình 5: Class diagram

6 Implementation

6.1 Requirement

6.1.1 Tổng quan

1. Server

   - Đầu tiên, Server tạo TCP socket để lắng nghe các yêu cầu từ Client ở Port được cài đặt (hiện thực ở hàm main trong file Server.py)

```
def main(self):
        try:
            SERVER_PORT = int(sys.argv[1])
        except:
            print("[Usage: Server.py Server_port]\n")
        rtspSocket = socket.socket(socket.AF_INET, socket.
            SOCK_STREAM)
        rtspSocket.bind(('', SERVER_PORT))
```

```
        print(SERVER_PORT)
        rtspSocket.listen(5)

        # Receive client info (address,port) through RTSP/TCP
            session
        while True:
            clientInfo = {}
            clientInfo['rtspSocket'] = rtspSocket.accept()
            ServerWorker(clientInfo).run()
```

- Sau khi nhận được yêu cầu từ Client, Server tiến hành xử lý yêu cầu theo như định dạng của đề bài để lấy ra các thông tin quan trọng như tên file, sequence number, loại yêu cầu,... (hiện thực ở hàm recvRtspRequest trong file ServerWorker.py)

```
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"))
```

- Sau khi xác định được loại yêu cầu và trạng thái Client hiện tại, Server tiến hành gửi phản hồi lại cho Client theo định dạng được đề cập trong đề (hiện thực ở hàm replyRSTP trong file ServerWorker.py). Chi tiết xử lý đối với từng loại yêu cầu được trình bày ở các phần tiếp theo.

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

    # Error messages
    elif code == self.FILE_NOT_FOUND_404:
      print("404 NOT FOUND")
    elif code == self.CON_ERR_500:
      print("500 CONNECTION ERROR")
```

2. Client

- Để có thể kết nối đến Server, Client tạo ra TCP socket (hiện thực ở hàm connectToServer trong file Client.py)

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

- Sau khi tạo được kết nối với Server, Client tiến hành gửi yêu cầu thông qua hàm sendRtspRequest (chi tiết hiện thực cho từng loại yêu cầu sẽ được trình bày ở các phần tiếp theo) và chờ phản hồi từ Server thông qua hàm recvRtspReply trong file Client.py

```
def recvRtspReply(self):
        """Receive RTSP reply from the server."""
        while True:
            reply = self.rtspSocket.recv(1024)

            if reply:
                self.parseRtspReply(reply)

            # Close the RTSP socket upon requesting Teardown
            if self.requestSent == self.TEARDOWN:
                self.rtspSocket.shutdown(socket.SHUT_RDWR)
                self.rtspSocket.close()
                break
```

- Cuối cùng, Client nhận được phản hồi từ Server và hiển thị trên giao diện GUI.

```
def createWidgets(self):
        """Build GUI."""
        # 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 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=1, 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=2, padx=2, pady=2)

        # Create Teardown button
        self.teardown = Button(self.master, width=20, padx=3,
            pady=3)
        self.teardown["text"] = "Teardown"
```

```
24          self.teardown["command"] = self.exitClient
25          self.teardown.grid(row=1, column=3, padx=2, pady=2)
26
27          # Create a label to display the movie
28          self.label = Label(self.master, height=19)
29          self.label.grid(row=0, column=0, columnspan=4, sticky=W +
                E + N + S, padx=5, pady=5)
```

6.1.2 Hiện thực nút Setup

1. Client

   - Nếu user nhấn vào nút SETUP, Client tiến hành gọi hàm setupMovie (hiện thực trong file Client.py), nếu trạng thái là INIT, Client tiến hành gửi yêu cầu đến Server thông qua hàm sendRtspRequest.

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

   - Trong file sendRtspRequest, Client gửi yêu cần đến Server theo định dạng được đề cập trong đề đồng thời tạo luồng để nhận phản hồi về từ Server.

```
1  if requestCode == self.SETUP and self.state == self.INIT:
2              threading.Thread(target=self.recvRtspReply).start()
3
4              # Update RTSP sequence number.
5              self.rtspSeq += 1
6
7              # Write the RTSP request to be sent.
8              request = "%s %s %s" % (self.SETUP_STR, self.fileName
                   , self.RTSP_VER)
9              request += "\nCSeq: %d" % self.rtspSeq
10             request += "\nTransport: %s; client_port= %d" % (self
                   .TRANSPORT, self.rtpPort)
11
12             # Keep track of the sent request.
13             self.requestSent = self.SETUP
```

2. Server

   - Nếu loại yêu cầu là SETUP và Client đang ở trạng thái INIT thì Server tiến hành mở file video và đặt trạng thái sang READY (hiện thực ở hàm processRtspRequest trong file ServerWorker.py)

```
1  if requestType == self.SETUP:
2        if self.state == self.INIT:
3          # Update state
4          print("processing SETUP\n")
5
6          try:
7            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]
```

6.1.3 Hiện thực nút Play

1. Client

   - Nếu user nhấn vào nút PLAY, Client tiến hành gọi hàm playMovie (hiện thực trong file Client.py), nếu trạng thái là READY, Client tiến hành tạo UDP socket để chờ nhận các khung video từ Server.

```
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)
```

   - Client gửi yêu cầu đến Server theo định dạng của đề thông qua hàm sendRtspRequest.

```
elif requestCode == self.PLAY and self.state == self.READY:

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

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

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

2. Server

   - Nếu loại yêu cầu là PLAY và Client đang ở trạng thái READY thì Server tiến hành tạo UDP socket để gửi từng khung video về Client thông qua hàm sendRtp và đặt trạng thái sang PLAYING (hiện thực ở hàm processRtspRequest trong file ServerWorker.py)

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

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

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

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

6.1.4 Hiện thực nút Pause

1. Client

   - Nếu user nhấn vào nút PAUSE, Client tiến hành gọi hàm pauseMovie (hiện thực trong file Client.py), nếu trạng thái là PLAYING, Client tiến hành gửi yêu cầu đến Server thông qua hàm sendRtspRequest.

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

   - Client gửi yêu cầu đến Server theo định dạng của đề thông qua hàm sendRtspRequest.

```
elif requestCode == self.PAUSE and self.state == self.PLAYING:

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

            request = "%s %s %s" % (self.PAUSE_STR, self.fileName
                , self.RTSP_VER)
            request += "\nCSeq: %d" % self.rtspSeq
            request += "\nSession: %d" % self.sessionId

            self.requestSent = self.PAUSE
```

2. Server

   - Nếu loại yêu cầu là PAUSE và Client đang ở trạng thái PLAYING thì Server tiến hành đặt luồng thực thi sang trạng thái chờ và chuyển trạng thái Client sang PLAYING (hiện thực ở hàm processRtspRequest trong file ServerWorker.py)

```
elif requestType == self.PAUSE:
      if self.state == self.PLAYING:
        print("processing PAUSE\n")
```

```
        self.state = self.READY

        self.clientInfo['event'].set()

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

6.1.5 Hiện thực nút Teardown

1. Client

   - Nếu user nhấn vào nút TEARDOWN, Client tiến hành gọi hàm exitClient (hiện thực trong file Client.py) để đóng cửa sổ GUI, xóa toàn bộ bộ nhớ đệm và tiến hành gửi yêu cầu đến Server thông qua hàm sendRtspRequest.

```
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
```

   - Client gửi yêu cầu đến Server theo định dạng của đề thông qua hàm sendRtspRequest.

```
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 = "%s %s %s" % (self.TEARDOWN_STR, self.
                fileName, self.RTSP_VER)
            request += "\nCSeq: %d" % self.rtspSeq
            request += "\nSession: %d" % self.sessionId

            self.requestSent = self.TEARDOWN
```

2. Server

   - Nếu loại yêu cầu là TEARDOWN thì Server tiến hành đặt luồng thực thi sang trạng thái chờ và đóng cổng UDP socket (hiện thực ở hàm processRtspRequest trong file ServerWorker.py)

```
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()
```

6.2 Extend

6.2.1 Thống kê

- Khi user nhấn nút Play và trạng thái Client đang ở Ready thì Client sẽ tạo luồng thực thi hàm listenRTP và chuyển sang trạng thái PLAYING. Hàm này sẽ thực hiện các thống kê cho từng frame video được gửi từ Server như size frame, video data rate,... trong suốt quá trình Client ở trạng thái PLAYING.

```
def listenRtp(self):
        """Listen for RTP packets."""
        while True:
            try:
                data = self.rtpSocket.recv(20480)
                if data:
                    rtpPacket = RtpPacket()
                    rtpPacket.decode(data)
                    size_payload = str(sys.getsizeof(rtpPacket.
                        getPayload()))
                    time_tr = time() - rtpPacket.timestamp()

                    print('size frame: ' + str(size_payload) + '\n' +
                          'time: ' + str(time_tr) + '\n' +
                          'video data rate: ' + str(int(size_payload)
                              // time_tr) + ' bytes/s\n')

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

                    # Set time box
                    ...
            except:
                # Stop listening upon requesting PAUSE or TEARDOWN
                ...
```

6.2.2 Hiện thực nút Stop

- Để loại bỏ nút Setup và hiện thực nút Stop, chúng ta hiện thực theo hướng khi user nhấn nút Play lần đầu tiên nếu trạng thái Client là INIT thì gửi yêu cầu Setup đến Server để cập nhật trạng thái sang READY và yêu cầu Play được gửi ngay sau đó.

- Về TEARDOWN và STOP: Khi người dùng nhấn nút Stop sẽ không thích hợp để gửi yêu cầu TEARDOWN. Vì Stop chỉ đơn giản reset lại video về ban đầu(đóng RTP socket), xóa hình ảnh trong bộ nhớ đệm và người dùng có thể bắt đầu xem lại video bằng cách nhấn nút PLAY. Trong khi với TEARDOWN, toàn bộ socket của Client được đóng lại, đối tượng GUI sẽ bị hủy, bộ nhớ đệm bị xóa và ứng dụng phía Client được đóng lại. Hai thành phần này có nhiệm vụ hoàn toàn khác nhau nên không thể thay thế cho nhau.

1. Client

    - Nếu user nhấn vào nút STOP, Client tiến hành gọi hàm setStop (hiện thực trong file Client_extend.py) để xóa bộ nhớ đệm và tiến hành gửi yêu cầu đến Server thông qua hàm sendRtspRequest.

```
def setStop(self):
        self.label.image = None
        self.sendRtspRequest(self.STOP)
        try:
            os.remove(CACHE_FILE_NAME + str(self.sessionId) +
                CACHE_FILE_EXT)
        except:
            pass
```

- Client gửi yêu cầu đến Server thông qua hàm sendRtspRequest.

```
elif requestCode == self.STOP and not self.state == self.INIT:
            self.rtspSeq += 1
            self.state = self.INIT
            self.stop = 1
            # Write the RTSP request to be sent.
            request = 'STOP' + ' ' + self.fileName + ' RTSP/1.0\n
                '
            request = request + ("CSeq: %d\n" % self.rtspSeq)
            request = request + \
                "Transport: RTP/UDP; client_port= %d\n" % (self.
                    rtpPort)
            self.requestSent = self.STOP

            # Update time box
            self.timeBox = "0 : 0"
            self.status = Label(self.master, text="Watched time :
                " + str(self.timeBox), bd=1, relief=SUNKEN,
                                anchor=W)
            self.status.grid(row=2, column=0, columnspan=1,
                sticky=W + E)
```

2. Server

- Nếu Client gửi yêu cầu STOP thì Server tiến hành đặt luồng thực thi sang trạng thái chờ và chuyển trạng thái Client về INIT và đóng cổng UDP socket(hiện thực ở hàm processRtspRequest trong file ServerWorker_extend.py)

```
elif requestType == self.STOP:
            print("processing STOP\n")
            self.clientInfo['event'].set()
            # self.clientInfo['videoStream']=VideoStream(filename
                )
            self.state = self.INIT
            self.replyRtsp(self.OK_200, seq[1])
            self.clientInfo['rtpSocket'].close()
```

6.2.3 Hiện thực nút Describe

1. Client

- Nếu user nhấn vào nút DESCRIBE, Client tiến hành gọi hàm setDescribe (hiện thực trong file Client_extend.py) gửi yêu cầu đến Server thông qua hàm sendRtspRequest.

```
def setDescribe(self):
        self.sendRtspRequest(self.DESCRIBE)
```

- Client gửi yêu cầu DESCRIBE đến Server thông qua hàm sendRtspRequest.

```
elif requestCode == self.DESCRIBE:
            self.rtspSeq += 1
            request = 'DESCRIBE' + ' ' + self.fileName + ' RTSP
                /1.0\n'
            request = request + ("CSeq: %d\n" % self.rtspSeq)
            request = request + \
                "Transport: RTP/UDP; client_port= %d\n" % (self.
                    rtpPort)
            self.requestSent = self.DESCRIBE
```

2. Server

- Nếu Client gửi yêu cầu DESCRIBE thì Server tiến hành gọi đến hàm replyRtsp_describe để gửi trả về Client các thông tin mô tả (hiện thực ở hàm processRtspRequest trong file ServerWorker_extend.py)

```
elif requestType == self.DESCRIBE:
            # print("processing STOP\n")
            self.replyRtsp_describe(self.OK_200, seq[1])
```

- Hàm replyRtsp_describe sẽ gửi trả về các thông tin mô tả về type of stream, encode, kích thước frame cho Client.

```
def replyRtsp_describe(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']) + '\nType of Stream
                    Real-Time Streaming  \nEncode utf-8\n' + str(
                self.clientInfo['videoStream'].getSize())
            connSocket = self.clientInfo['rtspSocket'][0]
            connSocket.send(reply.encode())

        # Error messages
        elif code == self.FILE_NOT_FOUND_404:
            print("404 NOT FOUND")
        elif code == self.CON_ERR_500:
            print("500 CONNECTION ERROR")
```

6.2.4 Hiện thực thêm chức năng

- Tương tự như phần thống kê, khi user nhấn nút Play và trạng thái Client đang ở Ready thì Client sẽ tạo luồng thực thi hàm listenRTP và chuyển sang trạng thái PLAYING. Hàm này sẽ liên tục cập nhật thời gian xem video trong suốt quá trình Client ở trạng thái PLAYING.

```
def listenRtp(self):
        """Listen for RTP packets."""
        while True:
            try:
                data = self.rtpSocket.recv(20480)
                if data:
                    ...
                    # Set time box
                    currentTime = int(currFrameNbr * 0.05)
                    self.timeBox = str(currentTime // 60) + \
                        " : " + str(currentTime % 60)
                    self.status = Label(self.master, text="Watched
                        time : " + str(self.timeBox), bd=1, relief=
                        SUNKEN, anchor=W)
                    self.status.grid(
                        row=2, column=0, columnspan=1, sticky=W + E)
                    ...
            except:
                # Stop listening upon requesting PAUSE or TEARDOWN
                ...
```

6.2.5 Hiện thực nút Switch

1. Client

- Để dễ dàng hiện thực nút Switch, chúng ta sẽ thay đổi cách nhập dữ liệu bên phía Client (hiện thực ở file Client_extend) : python ClientLauncher.py server_host server_port RTP_port [video_file1,video_file2,...]. Video_file1 sẽ được chạy khi user nhấn nút Play lần đầu, các video file sau sẽ được chạy tuần từ mỗi khi user nhấn nút Switch.

```
def __init__(self, master, serveraddr, serverport, rtpport,
    fileNameList):
        self.video_list = fileNameList[1:len(fileNameList) - 1]
        self.video_list = self.video_list.split(',')
        print(self.video_list)
        self.video_list_index = 0
        self.fileName = self.video_list[0]
```

- Nếu user nhấn vào nút SWITCH, Client tiến hành gọi hàm setSwitch (hiện thực trong file Client_extend.py) để xóa bộ nhớ đệm và tiến hành gửi yêu cầu đến Server thông qua hàm sendRtspRequest.

```
def setSwitch(self):
        self.label.image = None
        self.sendRtspRequest(self.SWITCH)
        try:
            os.remove(CACHE_FILE_NAME + str(self.sessionId) +
                CACHE_FILE_EXT)
        except:
            pass
```

- Ở hàm sendRtspRequest, Client tiến hành chuyển sang video tiếp theo trong danh sách và cập nhật lại time box.

```
elif requestCode == self.SWITCH:
            self.rtspSeq += 1
            self.state = self.INIT
            self.stop = 1

            # Switch video
            if self.video_list_index < len(self.video_list) - 1:
                self.video_list_index += 1
            else:
                self.video_list_index = 0
            self.fileName = self.video_list[self.video_list_index
                ]

            request = 'SWITCH' + ' ' + self.fileName + ' RTSP
                /1.0\n'
            request = request + ("CSeq: %d\n" % self.rtspSeq)
            request = request + \
                "Transport: RTP/UDP; client_port= %d\n" % (self.
                    rtpPort)
            self.requestSent = self.SWITCH

            # Update time box
            self.timeBox = "0 : 0"
            self.status = Label(self.master, text="Watched time :
                 " + str(self.timeBox), bd=1, relief=SUNKEN,
                                 anchor=W)
            self.status.grid(row=2, column=0, columnspan=1,
                sticky=W + E)
```

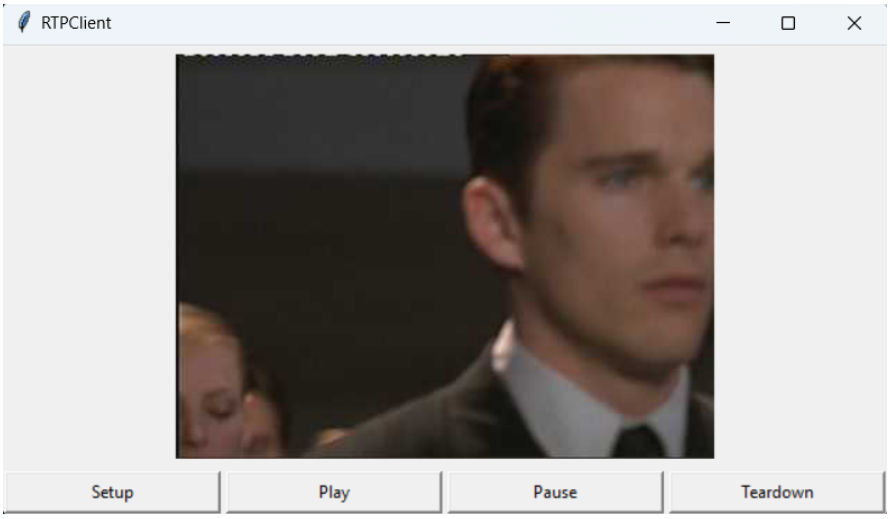
2. Server

   - Nếu Client gửi yêu cầu SWITCH thì Server tiến hành đặt luồng thực thi sang trạng thái chờ và chuyển trạng thái Client về INIT và đóng cổng UDP socket(hiện thực ở hàm processRtspRequest trong file ServerWorker_extend.py)

```
elif requestType == self.SWITCH:
            print("processing SWITCH\n")
            self.clientInfo['event'].set()
            self.state = self.INIT
            self.replyRtsp(self.OK_200, seq[1])
            self.clientInfo['rtpSocket'].close()
```

7 A summative evaluation of achieved results

- Chất lượng hình ảnh còn kém.



Hình 6: Giao diện video

- Packet loss rate xấp xỉ 0% (số packet gửi đi xấp xỉ số packet nhận được)

```
Current Seq Num: 499
size frame: 7320
time: 0.8938333988189697
video data rate: 8189.0 bytes/s

Current Seq Num: 500
```

Hình 7: Tổng số packet nhận được

- Video data rate $\leq$ 400 KBps

```
Current Seq Num: 485
size frame: 7082
time: 0.01971149444580078
video data rate: 359282.0 bytes/s

Current Seq Num: 486
size frame: 7046
time: 0.08224821090698242
video data rate: 85667.0 bytes/s
```

Hình 8: Video data rate

- Mức sử dụng tài nguyên thấp

Name	Status	6% CPU	90% Memory	1% Disk	0% Network
Visual Studio Code		0%	856.7 MB	0 MB/s	0 Mbps
Python		0%	22.0 MB	0 MB/s	0 Mbps

Hình 9: Ban đầu khi chưa chạy chương trình

Name	Status	80% CPU	89% Memory	16% Disk	0% Network
Visual Studio Code (2)		13.1%	712.6 MB	0.3 MB/s	0 Mbps
Python		0%	9.3 MB	0 MB/s	0 Mbps
Python		0%	5.0 MB	0.3 MB/s	0 Mbps

Hình 10: Sau khi khởi tạo Server

Name	Status	50% CPU	87% Memory	1% Disk	0% Network
Visual Studio Code (2)		17.0%	695.5 MB	0.2 MB/s	0 Mbps
Python		0%	9.3 MB	0 MB/s	0 Mbps
Python		0%	5.1 MB	0.1 MB/s	0 Mbps
Python		3.4%	14.9 MB	0.2 MB/s	0 Mbps
RTPClient					

Hình 11: Tạo 1 Client kết nối đến Server

8 User manual

8.1 Requirement

- Đầu tiên, ta đến thư mục Server, khởi tạo Server bằng lệnh : python Server.py server_port (với server_port > 1024)
  Ví dụ : python Server.py 2000

```
PS D:\Download\Assignment 1> cd ./Server
PS D:\Download\Assignment 1\Server> python Server.py 2000
2000
```

Hình 12: Chạy lệnh khởi tạo Server trên terminal

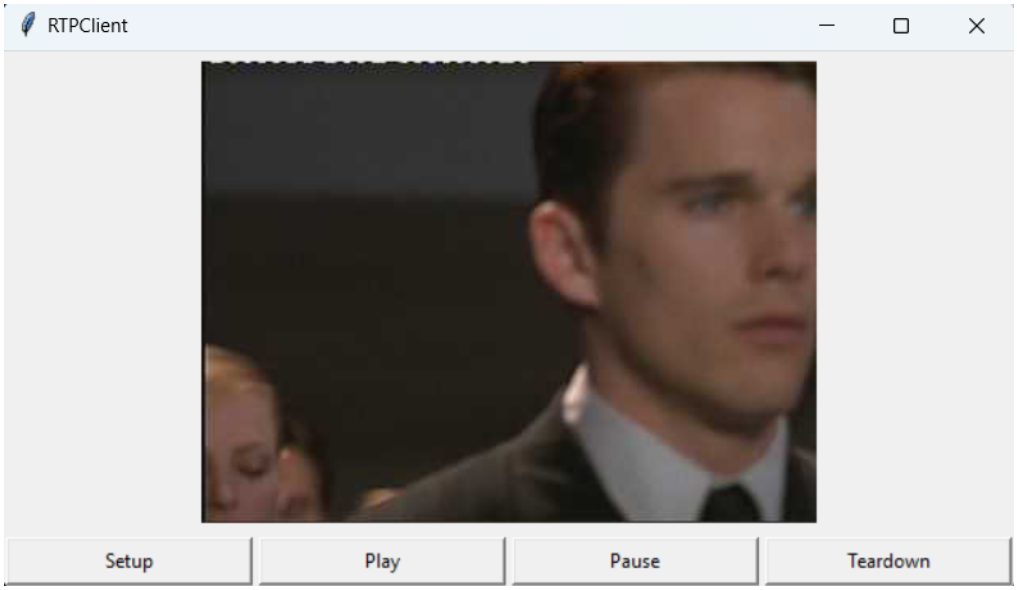
- Sau khi đã tạo thành công Server, ta sử dụng 1 máy khác kết nối đến cùng điểm truy cập cá nhân, nhảy đến thư mục Client, tạo Client kết nối đến Server bằng lệnh : python ClientLauncher.py server_host server_port RTP_port video_file. Với :
  - server_host là địa chỉ IP Server đang chạy.
  - server_port là cổng Server đang chờ yêu cầu từ Client.
  - RTP_port là cổng nhận các gói RTP.
  - video_file là tên video mà user muốn xem.

  Ví dụ : python ClientLauncher.py 192.168.43.69 2000 3000 movie.Mjpeg

```
PS D:\Download\Assignment 1> cd ./Client
PS D:\Download\Assignment 1\Client> python ClientLauncher.py localhost 200
0 3000 movie.Mjpeg
```

Hình 13: Chạy lệnh khởi tạo Client kết nối đến Server trên terminal

- Sau khi chạy lệnh kết nối đến Server, bên miền Client sẽ hiển thị giao diện GUI để trình chiếu và tương tác với video gồm 4 nút :
  - Setup : khởi tạo đường truyền video.
  - Play : xem video.
  - Pause : tạm dừng video.
  - Teardown : dừng hẳn video, ngắt kết nối.



Hình 14: Hiển thị giao diện tương tác với video

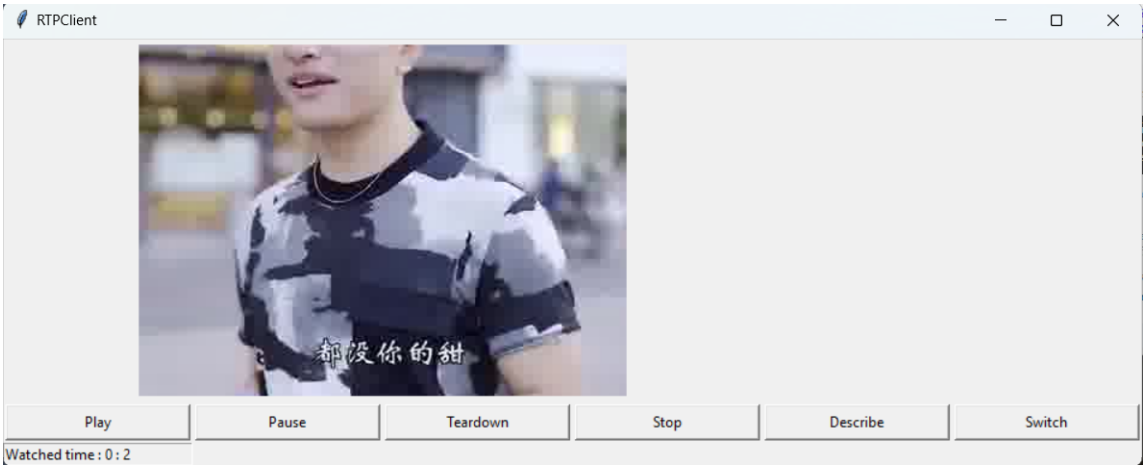
8.2 Extend

- Việc chạy lệnh tương tác giữa Client và Server tương tự như phần Requirement ở trên chỉ thay đổi một chút ở phần video file trong lệnh tạo Client : python ClientLauncher.py server_host server_port RTP_port [video_file1,video_file2,..] (một danh sách các video sẽ được đặt trong dấu [] cách nhau bởi dấu phẩy không khoảng trắng, video_file1 sẽ mặc định được chạy đầu tiên)
  Ví dụ : python ClientLauncher.py 192.168.43.69 2000 3000 [movie.Mjpeg1,movie.Mjpeg2]

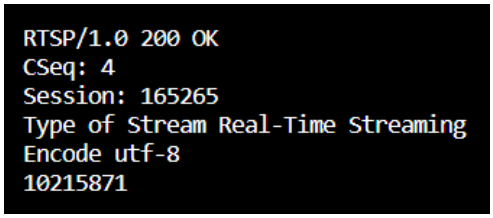
```
PS D:\Download\Assignment 1\Client> python ClientLauncher.py localhost 200
0 3000 [movie.Mjpeg,movie1.Mjpeg]
['movie.Mjpeg', 'movie1.Mjpeg']
```

Hình 15: Chạy lệnh khởi tạo Client kết nối đến Server trên terminal

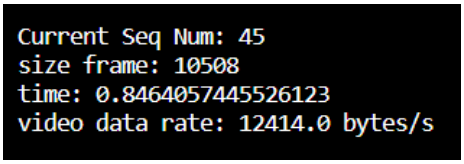
- Khác với phần Requirement, bên miền Client sẽ hiển thị giao diện GUI để trình chiếu và tương tác với video gồm 6 nút, bỏ nút Setup và thêm các nút Stop, Describe, Switch.
  - Play, Pause, Teardown : tương tự như mô tả ở phần Requirement.
  - Stop : dừng video và reset về lại ban đầu, tiếp tục chạy lại từ đầu video khi user nhấn nút Play.
  - Describe : hiển thị các thông tin mô tả về video như type of stream, encode, kích thước video trên terminal.
  - Switch : chuyển đến video tiếp theo trong danh sách video nếu đã là video cuối cùng thì chuyển lại về video đầu tiên.
  - Ngoài ra, GUI cũng hiển thị thông tin về thời gian video ở góc dưới bên trái giao diện GUI và thông kê từng khung video trong suốt quá trình Play trên terminal.



Hình 16: Hiển thị giao diện tương tác với video



Hình 17: Hiển thị mô tả trên terminal khi nhấn nút Describe



Hình 18: Hiển thị thống kê trên terminal

9 Full source code

Link source code : https://github.com/tien2114988/Computer-network.git