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)

Báo Cáo Bài Tập Lớn 1

Video Streaming

GVHD: Trần Huy

Lớp: L06

Thành viên:

Phạm Minh Hiếu – 1913356

Nguyễn Hữu Lợi

Lộc Minh Hiếu - 1913336

Ngô Tấn Phát Đạt - 1913045

Tp. Hồ Chí Minh, Tháng 10/ 2021

Mục lục

1. Phân tích yêu cầu

a. Yêu cầu chức năng

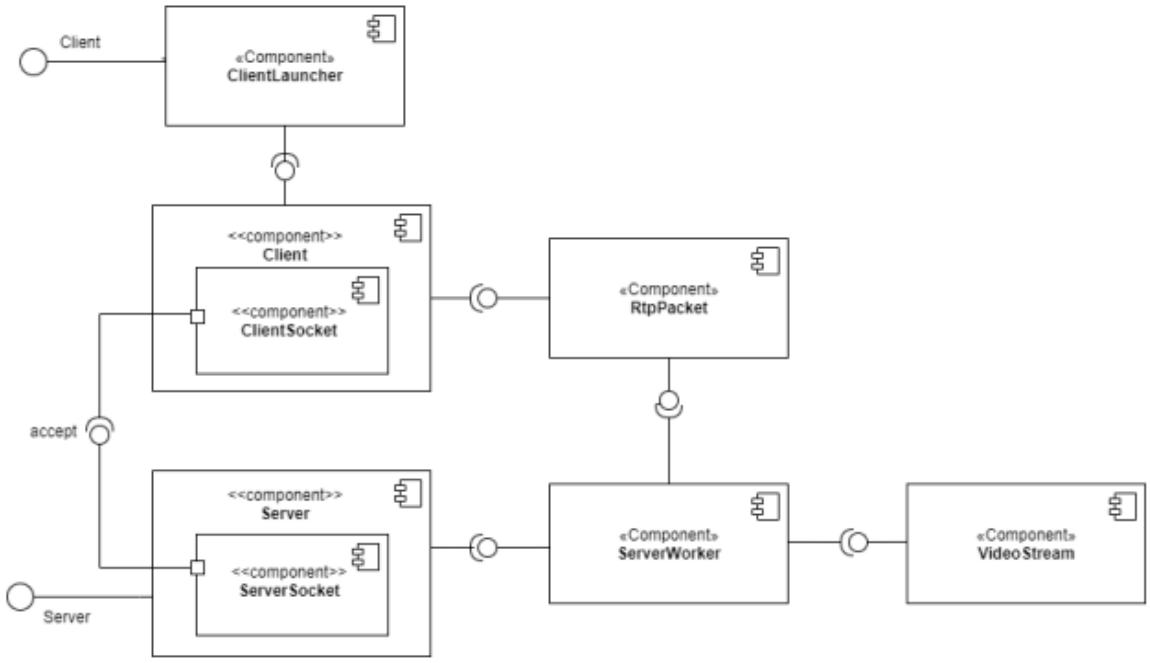
- SETUP: Thiết lập kết nối giữa Client và Server
- PLAY: Client có thể xem video được hiển thị trên màn hình qua những frame được thay đổi liên tục.
- PAUSE: Video streaming ngừng chạy
- TEARDOWN: Ngừng video streaming và đóng client
- CLOSE WINDOW: Video streaming ngừng chạy và client sẽ lựa chọn giữa đóng ứng dụng hoặc tiếp tục xem video

b. Yêu cầu phi chức năng

- Hỗ trợ video ở định dạng MJPEG.
- Hỗ trợ video frame lên đến 20 000 bytes.
- Thời gian để gửi yêu cầu và nhận phản hồi từ server nhỏ hơn 0,05s
- Thời gian chuyển video không quá 0,5s
- Tỉ lệ mất dữ liệu nhỏ hơn 0,1

2. Danh sách component và các hàm tương ứng

- **Đồ thị component:**

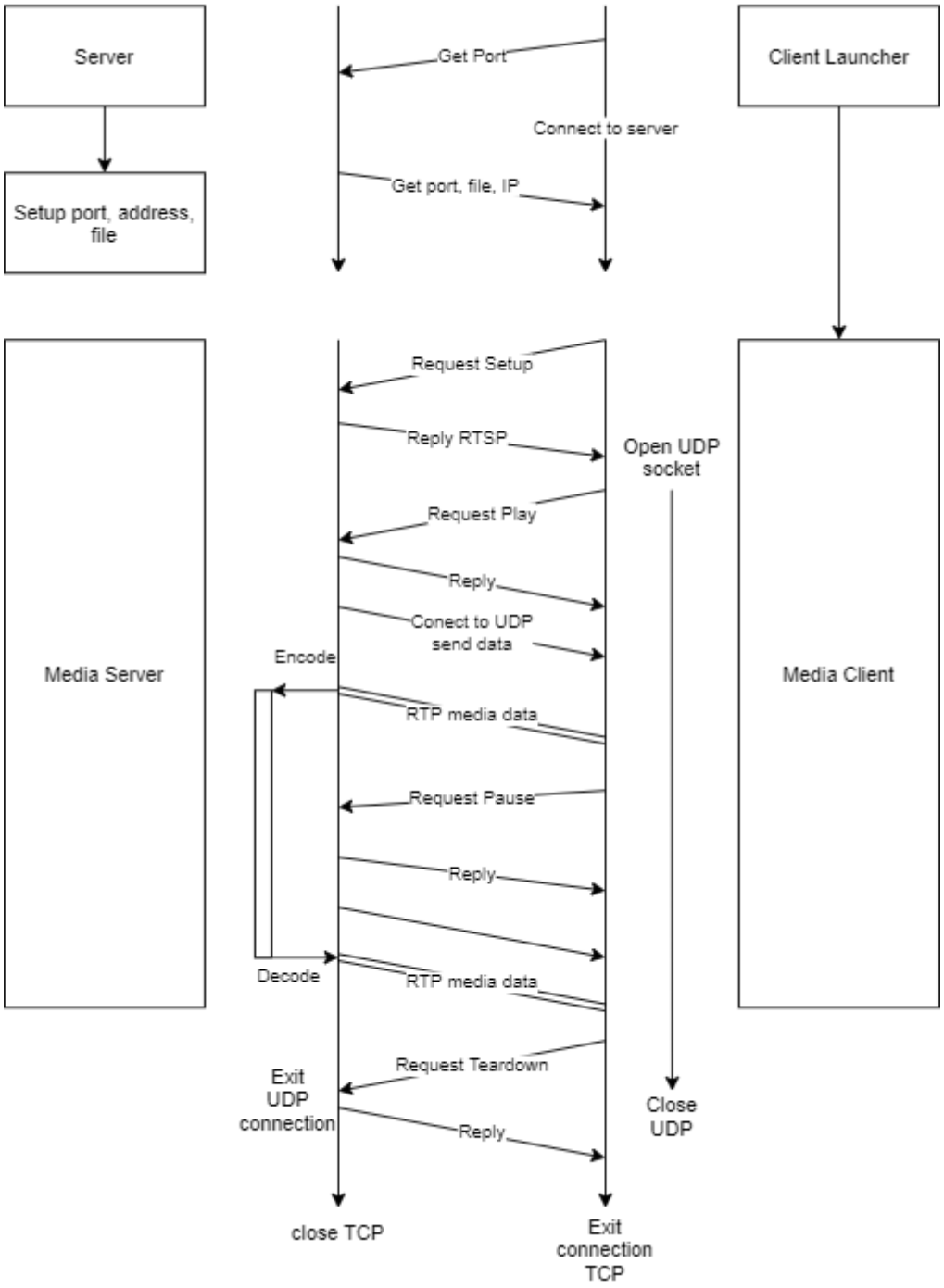


- **Danh sách các hàm**

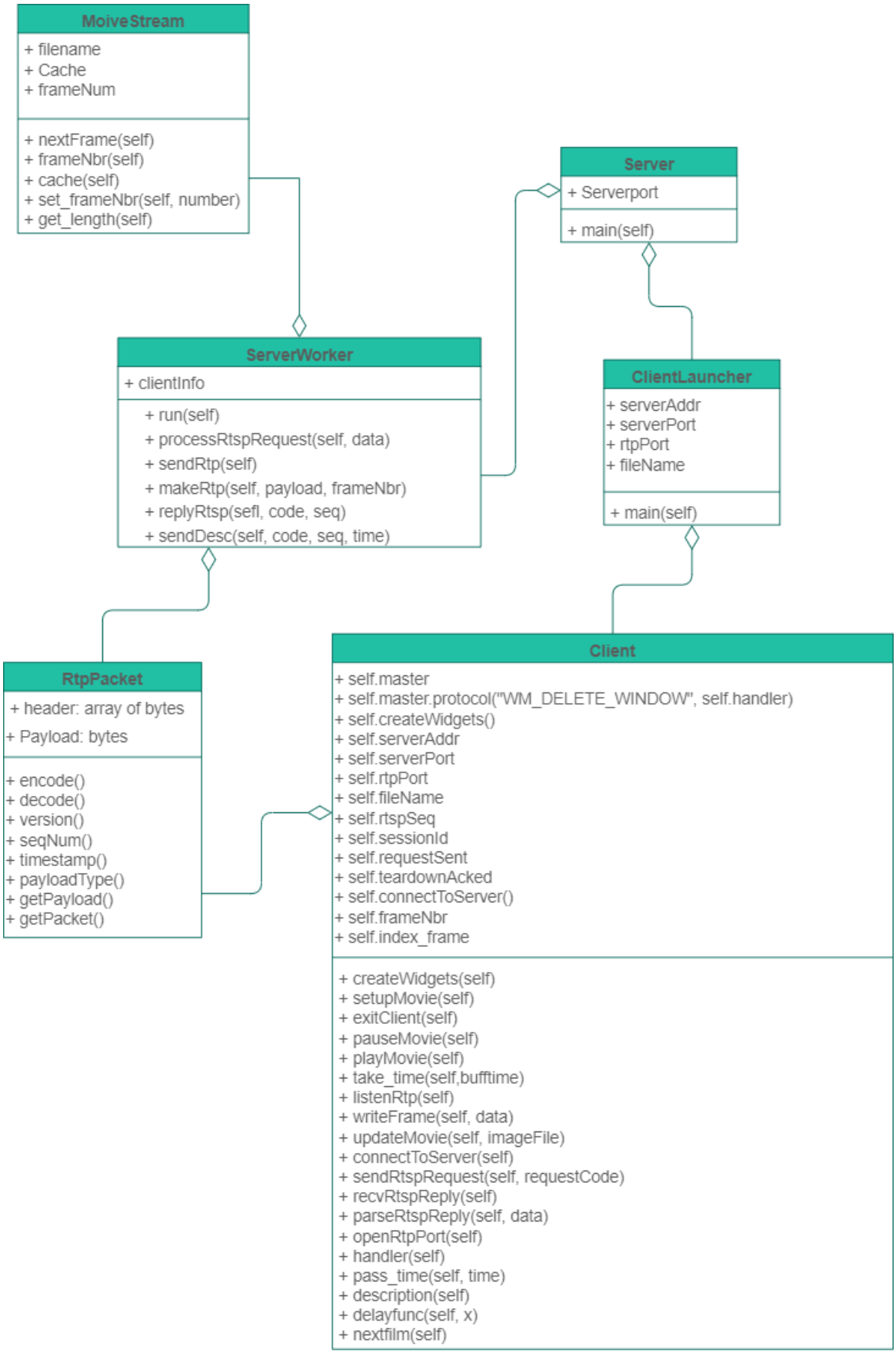
Class Name	Function	Parameter	Description
SeverWorker	__init__(self, clientInfo)	self, clientInfo	Constructor
	run(self)	self	Run the server
	processRtspRequest(self, data)	self, data	Process the Rtsp request
	sendRtp(self)	self	Send RTP packets over UDP
	makeRtp(self, payload, frameNbr)	self, payload, frameNbr	RPT-packetize the video data
	replyRtsp(sefl, code, seq)	self, code, seq	Send RTSP reply to the Client
	sendDesc(self, code, seq, time =0)	self, code, seq, time	send description of RTP packet
Sever	main(self)	self	Main function to run the whole program.
VideoStream	__init__(self, filename)	self, filename	Constructor
	nextFrame(self)	self	Get next frame
	frameNbr(self)	self	Get frame number
	cache(self)	self	Create list data
	set_frameNbr(self, number)	self, number	Set frame number
	get_length(self)	self	Get length of list data
Client	__init__(self, master, serveraddr, serverport, rtpport, filename)	self, master, serveraddr, serverport, rtpport, filename	Constructor
	createWidgets(self)	self	Build GUI
	setupMovie(self)	self	Setup button handler
	exitClient(self)	self	Teardown button handler
	pauseMovie(self)	self	Pause button handler
	playMovie(self)	self	Play button handler
	take_time(self,bufftime)	self,buftime	change time to format minutes : seconds
	listenRtp(self)	self	Listen for RTP packets and analysis somethings
	writeFrame(self, data)	self, data	Write the received frame to a temp image file
	updateMovie(self, imageFile)	self, imageFile	Update the image file as video frame in the GUI
	connectToServer(self)	self	Connect to the Server. Start a new RTSP/TCP session
	sendRtspRequest(self, requestCode)	self, requestCode	Send RTSP request to the server
	recvRtspReply(self)	self	Receive RTSP reply from the server
	parseRtspReply(self, data)	self, data	Parse the RTSP reply from the server
	openRtpPort(self)	self	Open RTP socket binded to a specified port
	handler(self)	self	Handler on explicitly closing the GUI window
	pass_time(self, time)	self, time	Backward button or forward button handler
	description(self)	self	Description button handler
	delayfunc(self, x)	self, x	Delay x seconds
	nextfilm(self)	self	Select film
RtpPacket	__init__(self)	self	constructor
	encode(self, version, padding, extension, cc, seqnum, marker, pt, ssrc,	self, version, padding, extension, cc, seqnum,	Encode the RTP packet with header fields and payload

	payload)	marker, pt, ssrc, payload	
	decode(self, byteStream)	self, byteStream	Decode the RTP packet
	version(self)	self	Return RTP version
	seqNum(self)	self	Return sequence (frame) number
	timestamp(self)	self	Return timestamp
	payloadType(self)	self	Return payload type
	getPayload(self)	self	Return payload
	getPacket(self)	self	Return RTP packet

3. Model và quy trình truyền tải dữ liệu



4. Class diagram



5. Thực hiện code

a. Requirements

Với bài tập lớn này, ta sẽ hiện thực 4 câu lệnh: SETUP, PLAY, PAUSE, TEARDOWN. Những câu lệnh này được gửi từ client đến server thông qua RTSP. Bên cạnh đó, ta cũng có 3 trạng thái cho client và server: INIT, READY, PLAYING

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

    SETUP = 0
    PLAY = 1
    PAUSE = 2
    TEARDOWN = 3
    SKIP = 4
    DESCRIPTION = 5

    RTSP_VER = "RTSP/1.0"
    TRANSPORT = "RTP/UDP"
```

Server luôn luôn phản hồi lại tất cả các yêu cầu mà client gửi đến. Trong bài tập lớn này, ta có 3 loại mã:

• Code 200: Successful request
• Code 404: FILE_NOT_FOUND error
• Code 500: Connection error

```
class ServerWorker:
    SETUP = 'SETUP'
    PLAY = 'PLAY'
    PAUSE = 'PAUSE'
    TEARDOWN = 'TEARDOWN'
    DESCRIPTION = "DESCRIPTION"
    SKIP = 'SKIP'

    INIT = 0
    READY = 1
    PLAYING = 2
    state = INIT


    OK_200 = 0
    FILE_NOT_FOUND_404 = 1
    CON_ERR_500 = 2
```

❖ **Client:**

- SETUP

- Gửi SETUP request tới server. SETUP RTSP packet bao gồm: câu lệnh SETUP, tên file video, loại giao thức: RTSP/1.0 RTP, RTSP Packet Sequence Number bắt đầu từ 1 (sau khi cộng 1, sequence number ban đầu là 0), Transmission Protocol: UDP, RTP Port cho việc truyền video stream từ client.

```
def sendRtspRequest(self, requestCode):

    """Send RTSP request to the server."""
    # Setup request
    request = ""
    if requestCode == self.SKIP and self.state != self.INIT:···

    elif  requestCode == self.SETUP and self.state == self.INIT:
        self.threadrecv = threading.Thread(target=self.recvRtspReply)
        self.threadrecv.start()
        # Update RTSP sequence number.
        self.rtspSeq +=1

        # Write the RTSP request to be sent.
        request = "%s %s %s" % (self.SETUP_STR, self.fileName, self.RTSP_VER)
        request+="\nCSeq: %d" % self.rtspSeq
        request+="\nTransport: %s; client_port= %d" % (self.TRANSPORT,self.rtpPort)
        self.requestSent = self.SETUP
    # Play request
```

- Đọc phản hồi từ server và phân tích Session header để lấy RTSP session ID.

```
def parseRtspReply(self, data):
    """Parse the RTSP reply from the server."""
    lines = data.decode().split('\n')
    seqNum = int(lines[1].split(' ')[1])
    totaltime= float(lines[3].split(' ')[1])
    #Chỉ xử lý nếu số thứ tự của câu trả lời của máy chủ giống với số thứ tự của yêu cầu
    if seqNum == self.rtspSeq:
        session = int(lines[2].split(' ')[1])
        # New RTSP session ID
        if self.sessionId == 0:
            self.sessionId = session
        # Chỉ xử lý nếu ID phiên giống nhau
        if self.sessionId == session:
            if int(lines[0].split(' ')[1]) == 200:
                if self.requestSent == self.SETUP:
                    if self.state == self.INIT:
                        self.rate = 0
                        self.loss = 0
                        self.end_time["text"] = str(datetime.timedelta(seconds=totaltime))
                        v = datetime.timedelta()
                        self.my_slider=Scale(self.master,variable = v,from_=0,to=totaltime,orient=HORIZONTAL)

                        self.my_slider.grid(row=2,column=0,columnspan=4,sticky='ew')

                        self.slider_label=Label(self.master,text='0')
                        self.slider_label.grid(row=3, columnspan=4, sticky='ew')
                        # Open RTP port.
                        self.openRtpPort()

                        # Liên kết ổ cắm với địa chỉ bằng cách sử dụng cổng RTP do người dùng máy khách cung cấp.
                        self.state = self.READY
```

- Tạo một datagram socket để nhận RTP data và đặt thời gian timeout đến 0.5s.

```
def openRtpPort(self):
    """Open RTP socket binded to a specified port."""
    # Tạo một ổ cắm datagram mới để nhận các gói RTP từ máy chủ
    self.rtpSocket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

    try:
        self.rtpSocket.bind(('',self.rtpPort))
    except:
        tkinter.messagebox.showwarning("UDP warning", "Can't blind to Port: " + str(self.rtpPort))
        self.rtpSocket.shutdown(socket.SHUT_RDWR)
        self.state = self.INIT
        self.rtpSocket.close()
    # Đặt giá trị thời gian chờ của ổ cắm thành 0,5 giây
    self.rtpSocket.settimeout(0.5)
```

- PLAY

- Gửi PLAY request. Trong yêu cầu này, ta phải thêm Session header và sử dụng session ID được trả về trong phản hồi của server ở câu lệnh SETUP, nhưng ta không đặt Transport header vào.

```
    # 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 = "%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
        self.played = 1
```

- Đọc phản hồi từ server

```
elif self.requestSent == self.PLAY and self.state == self.READY:
    self.state = self.PLAYING
    self.playEvent = threading.Event()
    self.threadlisten = threading.Thread(target=self.listenRtp)
    self.threadlisten.start()
```

- PAUSE, TERRDOWN
  Gửi PAUSE, TEARDOWN request và phân tích phản hồi từ server tương tự như SETUP và PLAY.

```
# Pause request
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
    # The play thread exits. A new thread is created on resume.
    self.playEvent.set()

# 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 = "%s %s %s" % (self.TEARDOWN_STR, self.fileName, self.RTSP_VER)
    request+="\nCSeq: %d" % self.rtspSeq
    request+="\nSession: %d" % self.sessionId
    # The play thread exits. A new thread is created on resume.

    self.requestSent = self.TEARDOWN
```

```
elif self.requestSent == self.PAUSE and self.state == self.PLAYING:
    self.state = self.READY
elif self.requestSent == self.TEARDOWN and self.state != self.INIT:
    self.state = self.INIT
    self.teardownAcked = 1
```

❖ **RtpPacket**
Ta cần điền vào RTP packet header với các trường theo yêu cầu:
• RTP-version: V = 2
• P = X = CC = M (in this lab)
• Payload type: PT = 26 corresspond to MJPEG type
• Sequence number = frameNbr, frameNbr được cho bởi server
• Timestamp: sử dụng thời gian trong module của Python
• SSRC là một số nguyên bất kỳ
• CC = 0 (ta không có nguồn phân phát nào khác, nên CRSC không tồn tại)

Chiều dài tổng của RTP packet header là 12 bytes

```
def encode(self, version, padding, extension, cc, seqnum, marker, pt, ssrc, payload):
    """Encode the RTP packet with header fields and payload."""
    timestamp = int(time())
    header = bytearray(HEADER_SIZE)
    #--------------
    # TO COMPLETE
    #--------------
    # Fill the header bytearray with RTP header fields
    header[0] = (header[0] |version << 6) & 0xC0 # V = 2 & 11000000
    header[0] = header[0] | padding << 5         # P = 1
    header[0] = header[0] | extension << 4       # X = 1
    header[0] = header[0] | cc << 3              # CC = 4

    header[1] = (header[1] | marker << 7)        # M = 1
    header[1] = header[1] | pt & 0x7F            # pt = 7

    header[2] = (seqnum >> 8) & 0xFF             # sequence 1 = 8 01111111
    header[3] = seqnum & 0xFF                    # sequence 2 = 8

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

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

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

b. Extend

- Câu 1: Trong quá trình xử lí reply từ server ở quá trình SETUP thì nhóm đã khởi tạo 1 biến self.rate để đo tốc độ truyền tải film và self.lost để tính toán số gói dữ liệu bị mất. Thì sau cuối chương trình nhận được kết quả: self.rate = 159240bytes/sec và self.lost = 0

```
Video data rate: 159240bytes/sec
Data lost: 0
```

- Câu 2: Vì ở câu 5 có yêu cầu người dùng lựa chọn film nên nhóm quyết định chọn phương án sẽ thực hiện SETUP ngày sau khi người dùng lựa chọn xong film mình muốn xem. Và gửi TEARDOWN khi người dùng nhấp vào nút STOP là thích hợp vì gửi TERRDOWN về thì sẽ giúp cho Server biết khi nào dừng gửi RtpPacket và dừng kết nối UDP với Client.

- Câu 3: Cũng giống như những chức năng SETUP, PLAY, PAUSE, TERRDOWN thì cũng sẽ yêu cầu và phân tích phản hồi từ Server

```
    elif requestCode == self.DESCRIPTION and self.state != self.INIT:
        self.requestSent = self.DESCRIPTION
        #message request
        request = "DESCRIPTION " + str(self.fileName) + " RTSP/1.0\n"
        request += "CSeq: " + str(self.rtspSeq)
        request += "\nSession: " + str(self.sessionId)
    else:...

    # Gửi yêu cầu RTSP bằng rtspSocket.
    print ('\nData Sent:' + request)
    self.rtspSocket.send(request.encode())#create in connectToServer
```

```
    elif self.requestSent == self.DESCRIPTION:
        if self.state != self.INIT:
            print("Description session: " + str(self.sessionId))
            for x in lines[4:]:
                print(x)
```

Và bên file ServerWorker.py sẽ thêm phần xử lí phản hồi description và hàm gửi về thông tin RtpPacket

```
        # Process DESCRIPTION request
        elif requestType ==  self.DESCRIPTION and self.state != self.INIT:
            print("processing DESCRIPTION\n")

            self.sendDesc(self.OK_200, seq[1], self.totaltime)

    def sendDesc(self, code, seq, time = 0):
        if code == self.OK_200:
            reply = 'RTSP/1.0 200 OK\nCSeq: ' + seq + '\nSession: ' + str(self.clientInfo['session']) + '\ntotaltime: '+ str(time)
            packet = RtpPacket()
            packet.decode(self.RTPpacket)
            reply += '\nVersion: ' + str(packet.version())
            reply += '\nSequence number: ' + str(packet.seqNum())
            reply += '\nPayload type: ' + str(packet.payloadType())
            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")

    def sendRtp(self):...
```

- Câu 4: Vì bên phía Server không có thông tin về tổng thời gian của film mà chỉ đọc từng dữ liệu trong film và lập tức gửi đến client nên nhóm đã viết lại file VideoStream.py để lưu trữ các dữ liệu đầy đủ và chứa thông tin thời gian về film cho Server. Nhóm đã dùng 1 yêu cầu SKIP đến Server để điều chỉnh tua thời gian film đi 2 giây hoặc lùi 2 giây và đồng thời thể

hiện thời gian chạy trong qua thanh scroll.

```python
import  copy
class VideoStream:
    def __init__(self, filename):
        self.filename = filename
        self.Cache= self.cache()

        try:
            self.file = open(filename, 'rb')
            #self.Cache=self.cache()
        except:
            raise IOError
        self.frameNum = 0
    def nextFrame(self):...
    def cache(self):...
    def frameNbr(self):...
    def set_frameNbr(self,number):...
    def get_length(self):...
```

File MovieStream.py

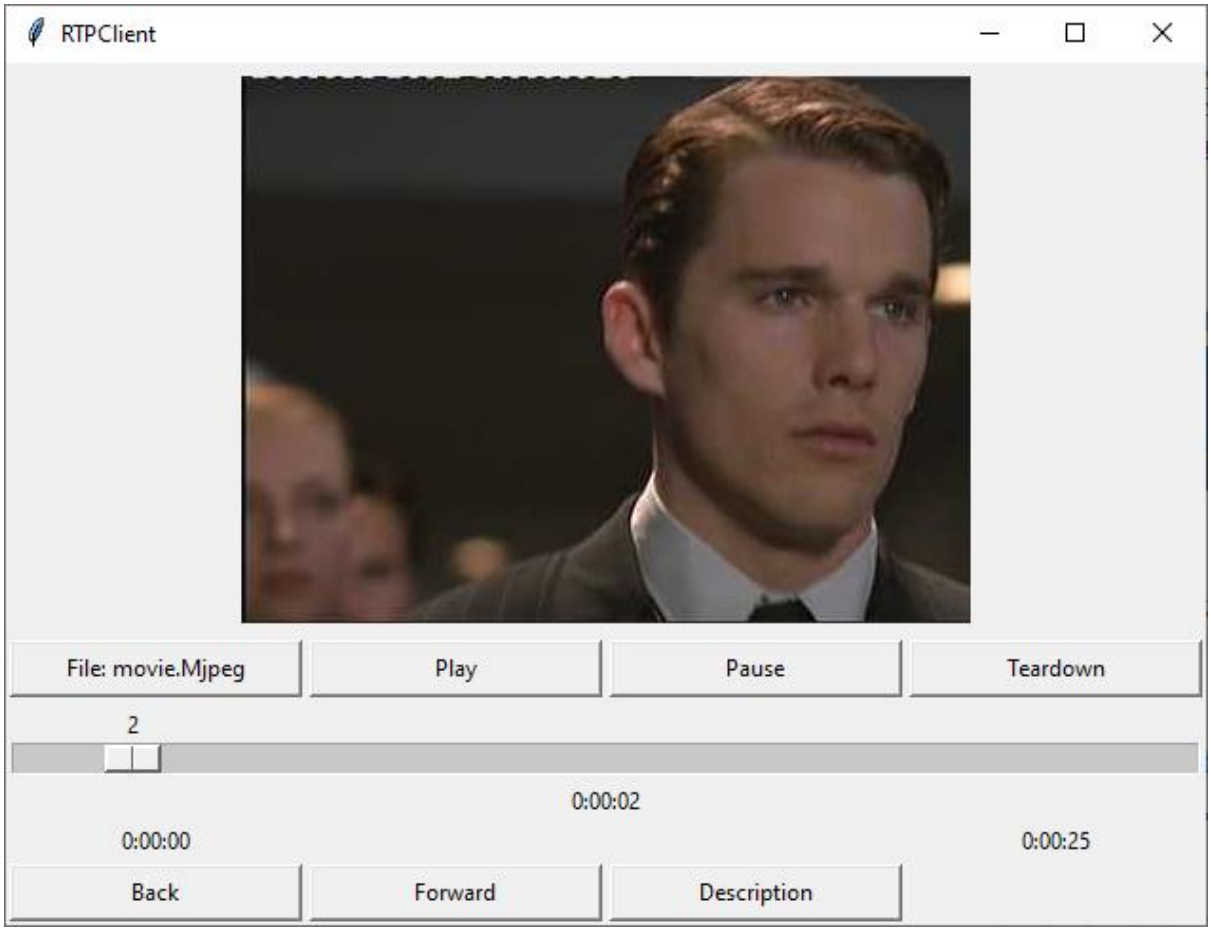
```python
    def sendRtspRequest(self, requestCode):

        """Send RTSP request to the server."""
        # Setup request
        request = ""
        if requestCode == self.SKIP and self.state != self.INIT:
            self.rtspSeq+=1

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

            self.requestSent = self.SKIP
            self.state = self.READY
```

Yêu cầu Skip



Thanh scroll thể hiện thời gian chạy của film

- Câu 5: Nhóm đã thêm trạng thái SWITCH để lựa chọn film ban đầu hoặc ngày sau khi Pause, Terrdown. Tên film sẽ được hiển thị ở nút 1. Người dùng cần đợi 1 giây để film load lại. Khi

nhấn lựa chọn film sẽ kích hoạt hàm nextfilm trong Client.

```
def nextfilm(self):
    if self.state != self.PLAYING:
        if self.played == 0:
            self.delayfunc(float(0.1))
            self.playMovie()
        if self.played == 1:
            self.breakpoint = 1
            self.played = 0

            if self.teardownAcked == 0:
                self.delayfunc(float(0.1))
                self.exitClient()

            self.delayfunc(float(1))
            self.connectToServer() #TCP close()
            self.breakpoint = 0

        self.index += 1
        if self.index == len(self.listfilm):
            self.index = 0
        self.fileName = self.listfilm[self.index]
        self.state = self.INIT
        self.choose["text"] = "File: " + str(self.fileName)
        self.setupMovie()
```

6. **Kết quả đạt được**

Gửi RTSP request tới server: thành công và không bị mất các yêu cầu từ Client.

Mở RTP socket kết nối tới port được định sẵn: không có các gói dữ liệu bị mất và server kết nối được với UDP socket.

Encode RTP packet với các trường trong header và payload: encode và decode không bị lỗi.

Quá trình chọn film, setup, tua film đều có khoảng thời gian chờ nhất định.

7. Hướng dẫn sử dụng

- Khởi động 1 commandline để tạo Server với cú pháp: python Server.py server_port Trong đó: server_port là số bất kỳ

  Ví dụ: python Server.py 2021

```
PS D:\Document for study\Networks\Assignment\1\Initcode> python Server.py 1200
```

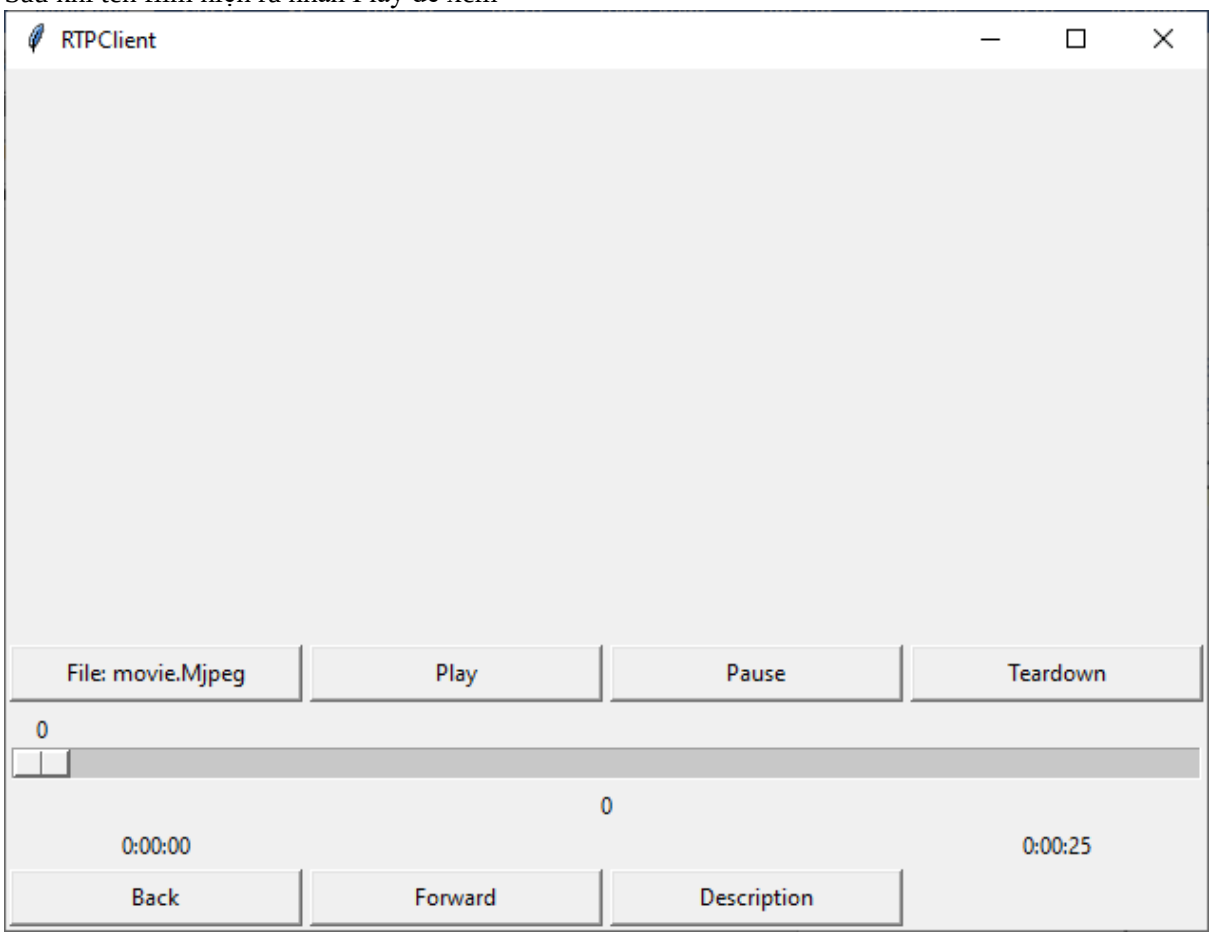
- Khởi động commandline thứ 2 để tạo client với cú pháp: python ClientLauncher.py host_name server_port RPT_port name_movie1 name_movie2 …
  Trong đó:
  - host_name: tên Server đang dùng. Ở đây là localhost
  - server_port: port đã tạo
  - RPT_port: port bất kỳ
  - name_movie1, name_movie2 …: danh sách file movie đưa vào làm dữ liệu cho Server gửi dữ liệu về Client.

  Ví dụ: python ClientLauncher.py localhost 2021 8000 movie.Mjpeg movie1.Mjpeg movie2.Mjpeg

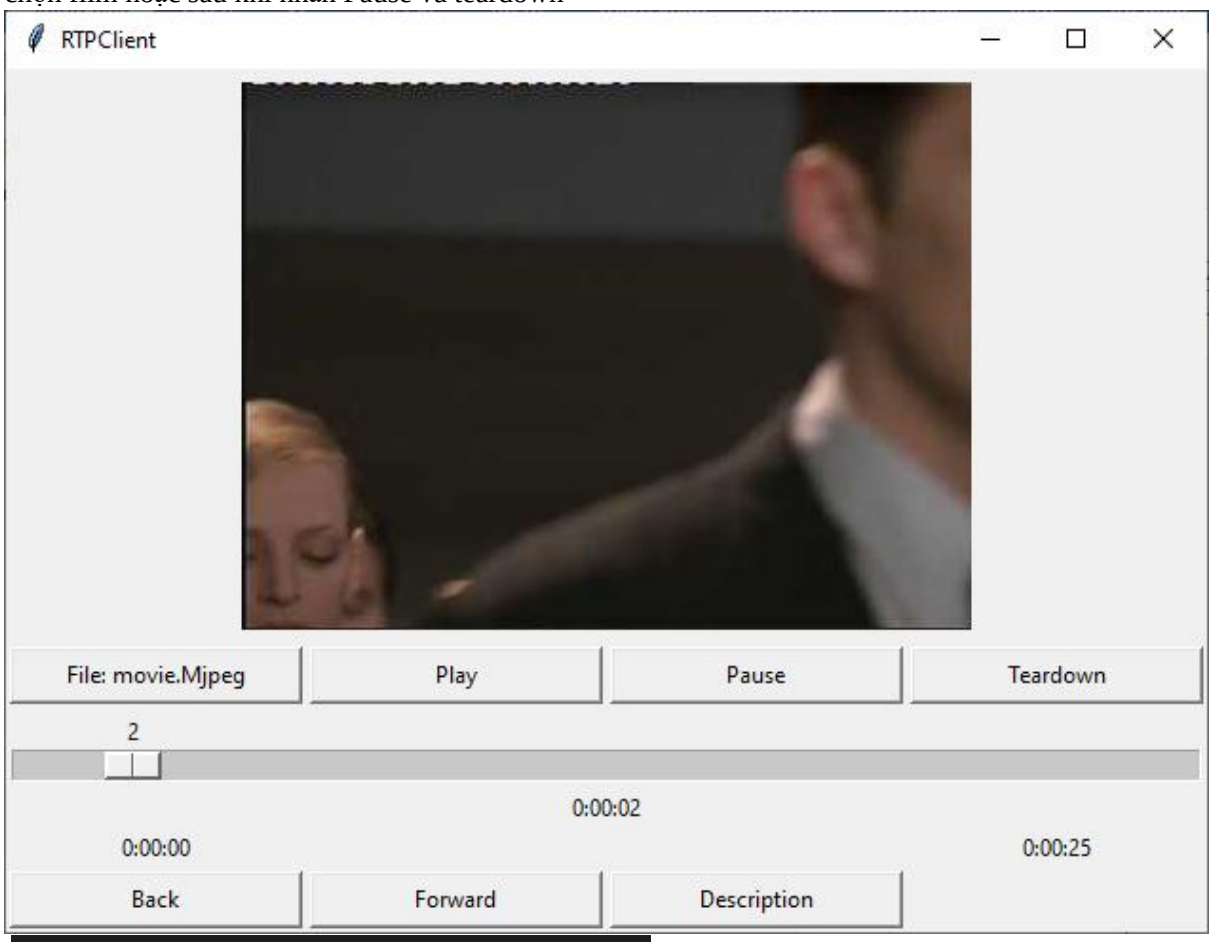
```
python ClientLauncher.py localhost 1200 800 movie.Mjpeg movie1.Mjpeg movie2.Mjpeg
```

- Nhấn vào Select film để tạo chọn film cần xem

- Sau khi tên film hiện ra nhấn Play để xem

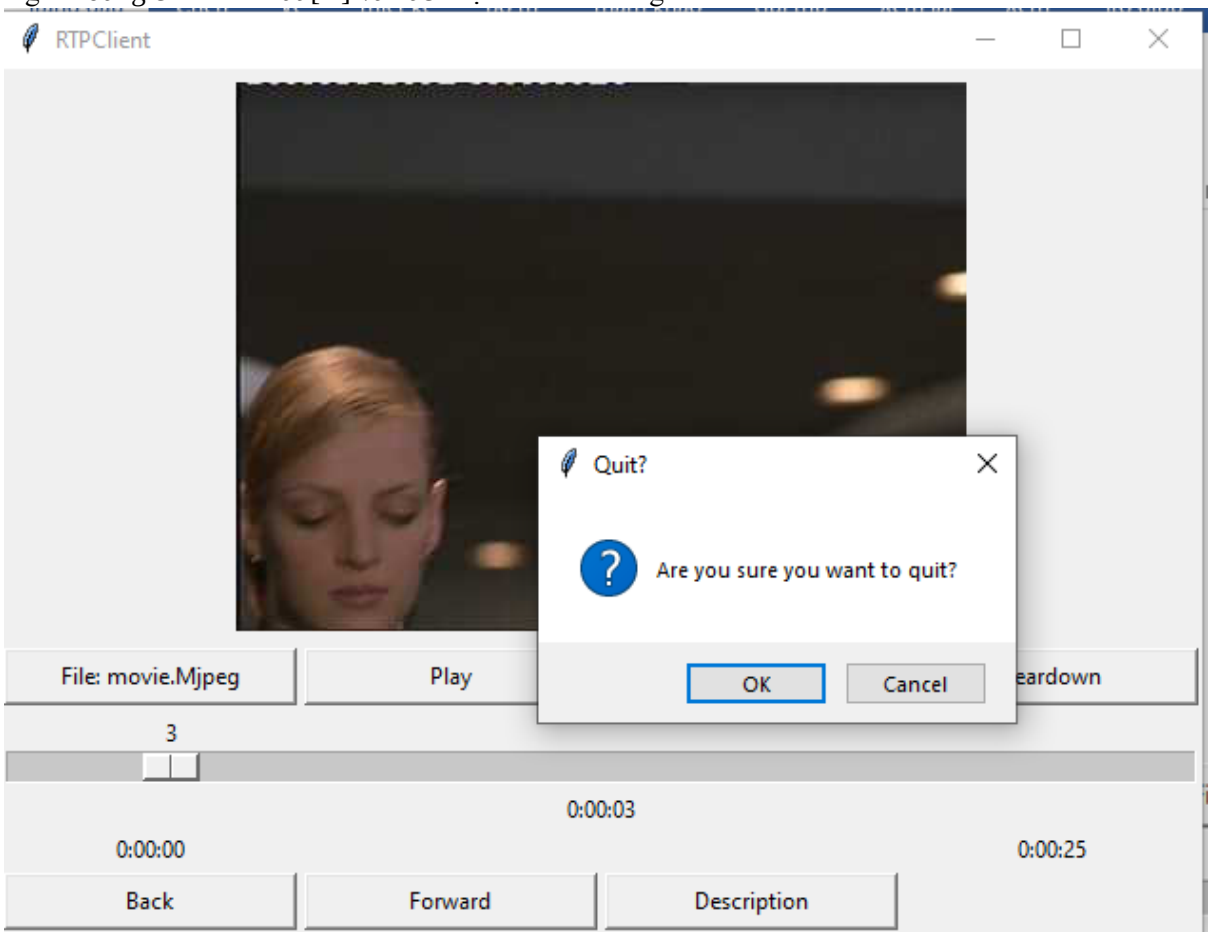


- Film sẽ chiếu. Lúc này người dùng có thể Pause khi muốn dừng, Teardown khi đóng film, back lùi 2 giây, forward tua đi 2 giây, Description để xem thông tin dữ liệu Server gửi về ở terminal. Người dùng có thể chọn film từ danh sách đã nhập khi nhấn lại nút 1 ngay sau khi chọn film hoặc sau khi nhấn Pause và teardown




```
Data Sent:DESCRIPTION movie.Mjpeg RTSP/1.0
CSeq: 3
Session: 429817
Description session: 429817
Version: 2
Sequence number: 90
Payload type: 26
```

- Người dùng cần nhấn nút [X] và xác nhận form để đóng màn hình .



8. Source code

Folder Innitcode đính kèm theo file.
Hoặc có thể truy cập vào github để clone file với đường link:
https://github.com/hieugc/Networks.git