

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



**MẠNG MÁY TÍNH (CO3003)**

---

**Bài tập lớn 1**

# **Video streaming using RTSP and RTP**

---

GV ra đề và HD: Bùi Xuân Giang

Nhóm SV thực hiện :

Phan Anh Tú - 1915822 (Nhóm trưởng)

Huỳnh Tuấn Đạt - 1913026

Trần Quốc Việt - 1915919

Nguyễn Văn Quốc - 1914864

Nguyễn Việt Đức - 1913167

Thành phố Hồ Chí Minh, 11/2021



## Mục lục

<b>1</b>	<b>Phân tích yêu cầu</b>	<b>2</b>
1.1	Thực thi giao thức RTSP trên Client . . . . .	2
1.2	Thực thi giao thức RTP trên Server . . . . .	2
<b>2</b>	<b>Mô tả chức năng</b>	<b>2</b>
2.1	ClientLauncher . . . . .	2
2.2	Client . . . . .	3
2.3	ServerWorker . . . . .	3
2.4	Server . . . . .	4
2.5	RtpPacket . . . . .	4
2.6	VideoStream . . . . .	4
<b>3</b>	<b>Class diagram</b>	<b>5</b>
<b>4</b>	<b>Hướng dẫn chạy chương trình</b>	<b>5</b>
<b>5</b>	<b>Source code</b>	<b>8</b>
5.1	ClientLaucher . . . . .	8
5.2	Client . . . . .	8
5.3	ServerWorker . . . . .	13
5.4	Server . . . . .	16
5.5	RTP packet . . . . .	17
5.6	VideoStream . . . . .	19
<b>6</b>	<b>Extend</b>	<b>19</b>
6.1	Extend 1 . . . . .	19
6.1.1	Tỷ lệ mất gói . . . . .	19
6.1.2	Tốc độ truyền dữ liệu . . . . .	19
6.1.3	Code Extend 1 . . . . .	20
6.1.4	Kết quả minh họa . . . . .	20
6.2	Extend 2 . . . . .	20
6.3	Extend 3 . . . . .	22

## 1 Phân tích yêu cầu

### 1.1 Thực thi giao thức RTSP trên Client

Giao diện người dùng được thiết kế với 4 nút: SETUP, PLAY, PAUSE, TEARDOWN. Khi người dùng nhấp vào nút thích hợp, các hành động sẽ tuân theo từng loại yêu cầu. Ví dụ: Khi người dùng ấn nút Play, yêu cầu PLAY được gửi tới máy chủ. Máy chủ sẽ nhận tin nhắn, bắt đầu quá trình chuyển đổi từng gói RTP và cập nhật lại trạng thái của client là PLAYING. Trong giai đoạn này chúng ta sẽ tạo yêu cầu PLAY cho server, nhập tiêu đề Transport và dùng session ID được trả về trong SETUP.

- **SETUP:** khi người dùng nhấn nút SETUP, client sẽ gửi yêu cầu đến Server để bắt đầu đóng gói các dữ liệu cần truyền thành các gói RTP và chuyển trạng thái của Client từ INIT sang READY.
- **PAUSE:** Khi người dùng nhấn nút PAUSE, yêu cầu PAUSE sẽ được gửi tới server. Máy chủ sẽ nhận lệnh PAUSE sau đó dừng chuyển đổi từng gói RTP và cập nhật lại trạng thái của client từ PLAYING thành READY. Trong phần này chúng ta sẽ tạo yêu cầu PAUSE cho máy chủ và chèn các tiêu đề Transport và dùng session ID trả về từ phản hồi SETUP.
- **TEARDOWN:** Khi người dùng nhấn nút TEARDOWN, yêu cầu teardown sẽ được gửi tới máy chủ. Một khi máy chủ nhận được thông điệp sẽ cập nhật trạng thái của INIT. Trong phần này, chúng ta sẽ tạo yêu cầu TEARDOWN cho máy chủ, chèn tiêu đề Transport và sử dụng session ID trả về trong phản hồi SETUP.

### 1.2 Thực thi giao thức RTP trên Server

Server sẽ thực hiện đóng gói các dữ liệu thành các gói RTP, thiết lập vùng trong tiêu đề gói và sao chép payload vào gói. Khi máy chủ nhận được yêu cầu PLAY từ client, máy chủ sẽ đọc video frame từ file và tạo RtpPacket-object. Sau đó máy chủ sẽ gửi các frame đến client thông qua giao thức UDP mỗi 50 milliseconds. Để thực hiện đóng gói, server sẽ gọi hàm mã hóa trong RtpPacket class. Rtp header bao gồm:

RTP packet header							
Bit offset <sup>[b]</sup>	0-1	2	3	4-7	8	9-15	16-31
0	Version	P	X	CC	M	PT	Sequence number
32	Timestamp						
64	SSRC identifier						

## 2 Mô tả chức năng

Chương trình bao gồm 6 phần

### 2.1 ClientLauncher

ClientLauncher sẽ chịu trách nhiệm khởi tạo máy khách với server address, serverport, RTP port và file .mjpeg sẽ được truyền.

## 2.2 Client

Client thiết kế giao diện người dùng với 4 nút: SETUP, PLAY, PAUSE, TEARDOWN và hiện thực các hành động khi nhấn nút tương ứng. Trong client bao gồm các hàm:

- def \_\_init\_\_(self, master, serveraddr, serverport, rtpport, filename): khởi tạo client với các tham số mặc định
- def createWidgets(self): tạo giao diện người dùng với 4 buttons: SETUP, PLAY, PAUSE, TEARDOWN và giao diện để hiển thị video được truyền.
- def setupMovie(self): thiết lập đường truyền khi người dùng nhấp vào nút SETUP
- def exitClient(self): thoát giao diện khi người dùng nhấp vào nút TEARDOWN
- def pauseMovie(self): tạm dừng video đang phát khi người dùng nhấp vào nút PAUSE
- def playMovie(self): bắt đầu phát video khi người dùng nhấp vào nút PLAY
- def listenRtp(self): nhận các gói RTP được gửi từ Server
- def writeFrame(self, data): ghi khung nhận được vào file hình ảnh tạm thời và trả về file hình ảnh.
- def updateMovie(self, imageFile): cập nhật file hình ảnh dưới dạng khung video trong giao diện người dùng.
- def connectToServer(self): thiết lập kết nối với server bằng cách bắt đầu một phiên RTSP / TCP mới.
- def sendRtspRequest(self, requestCode): gửi yêu cầu đến máy chủ khi người dùng nhấp vào các nút trong giao diện người dùng. Chúng ta cũng có thể sử dụng chức năng này để in ra thông tin của tin nhắn được gửi từ client.
- def recvRtspReply(self): nhận RTSP trả lời từ máy chủ
- def parseRtspReply(self, data): phân tích cú pháp RTSP trả lời từ server để thiết lập trạng thái của client hoặc quyết định đóng socket.
- def openRtpPort(self): mở socket RTP được liên kết với một port được chỉ định.
- def handler(self): xử lý đóng giao diện khi người dùng nhấp vào nút X.

## 2.3 ServerWorker

ServerWorker xác định trạng thái của server (INIT, READY, PLAYING) và xử lý yêu cầu từ client.

- def \_\_init\_\_(self, clientInfo): xác định thông tin của client.
- def run(self): chạy một thread để nhận các yêu cầu RTSP từ client.
- def processRtspRequest(self, data): xử lý các yêu cầu RTSP đã gửi
- def sendRtp(self): gửi các gói RTP qua UDP.
- def replyRtsp(self, code, seq): gửi RTSP trả lời cho client.

## 2.4 Server

Server sẽ phụ trách việc khởi Server với `server_port` và nhận thông tin từ client (`address,port`) thông qua phiên RTSP / TCP.

## 2.5 RtpPacket

RtpPacket sẽ xử lý các gói RTP. Nó cũng có phương pháp mã hóa, giải mã packet RTP và nhận thông tin từ packet.

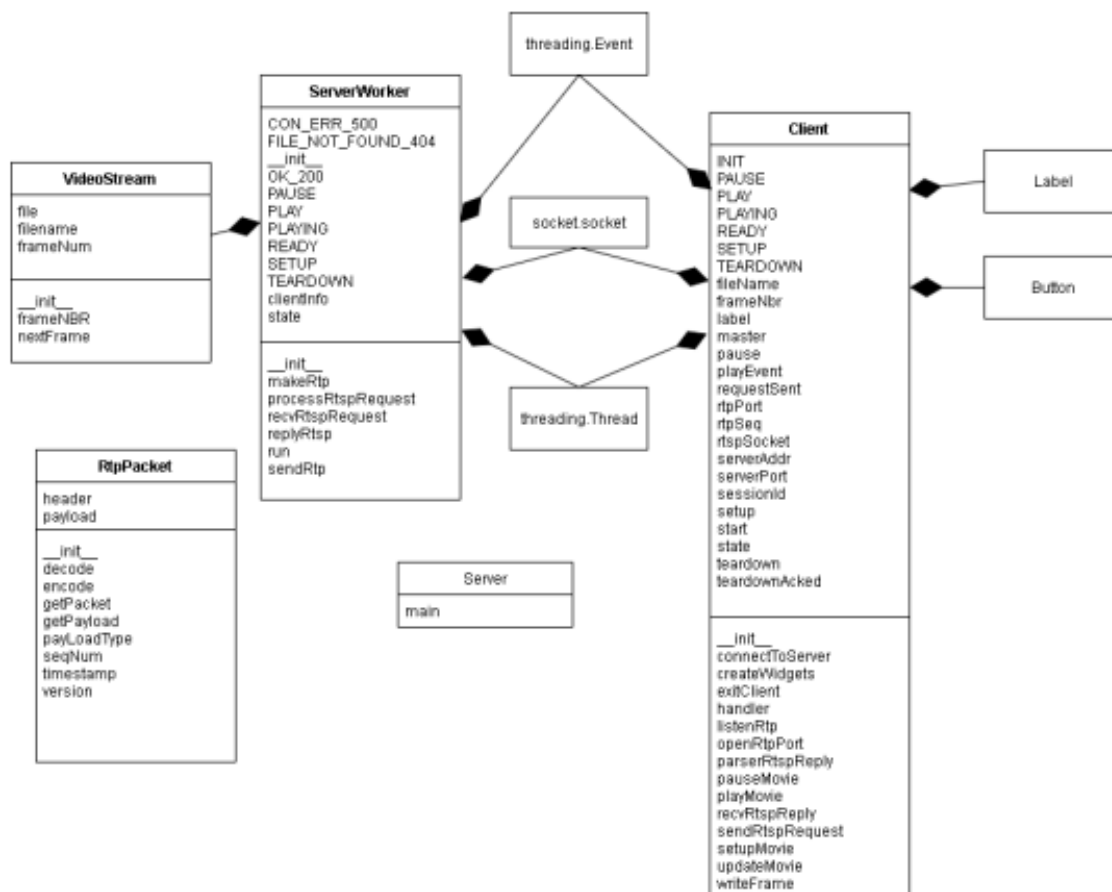
- `def encode(self, version, padding, extension, cc, seqnum, marker, pt, ssrc, payload)`: mã hóa gói RTP với các trường tiêu đề và tải trọng.
- `def decode(self, byteStream)`: giải mã gói RTP
- `def version(self)`: trả về phiên bản RTP
- `def seqNum(self)`: trả về seqNum
- `def timestamp(self)`: trả về timestamp
- `def payloadType(self)`: trả về payloadType
- `def getPayload(self)`: trả về payload
- `def getPacket(self)`: trả về packet RTP

## 2.6 VideoStream

VideoStream sẽ phụ trách việc đọc dữ liệu video từ tệp trên đĩa.

- `def __init__(self, filename)`: mở tệp trong đĩa
- `def nextFrame(self)`: lấy khung dữ liệu video tiếp theo
- `def frameNbr(self)`: lấy số khung

### 3 Class diagram



### 4 Hướng dẫn chạy chương trình

Đầu tiên, trong Command line nhập lệnh: `python Server.py server_port`. Với `server_port` là ID của 1 port bất kỳ, tuy nhiên ta nên chọn con số port lớn hơn 1024 để không trùng port với các giao thức khác, tránh phát sinh lỗi( HTTP:80, DNS:53, TCP :45 ...)

Lệnh này sẽ setup 1 server

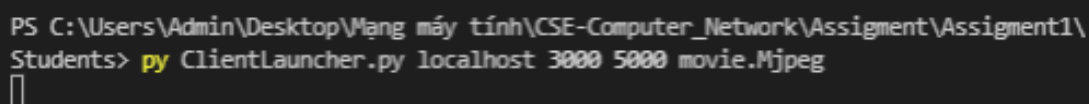
```

PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE
PS C:\Users\Admin\Desktop\ Mạng máy tính\CSE-Computer_Network\Assigment\Assignment1\Students> py Server.py 3000

```

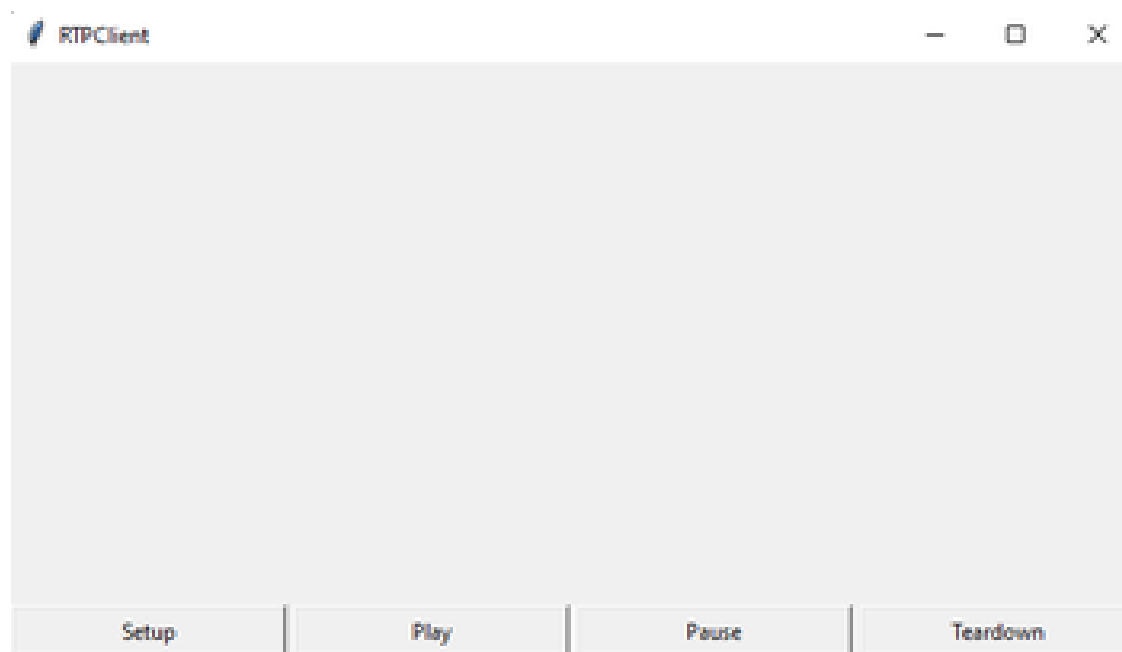
Sau đó, ta nhập lệnh: `python ClientLauncher.py ip_add server_port client_port filename`.  
Với:

- *ip\_add*: địa chỉ ip của máy tính
- *server\_port*: số port của server
- *client\_port*: số port của client (ở đây là 1 con số bất kỳ khác *server\_port* và lớn hơn 1024)
- *filename*: tên file cần phát



```
PS C:\Users\Admin\Desktop\Mạng máy tính\CSE-Computer_Network\Assignment\Assignment1\Students> py ClientLauncher.py localhost 3000 5000 movie.Mjpeg
█
```

Sau lệnh này, một giao diện sẽ xuất hiện như hình dưới



Lúc này ta bấm Setup để tiến hành thiết lập kết nối giữa client và server. Từ lúc này, ta có thể bấm Play hoặc Pause để phát hoặc dừng video.

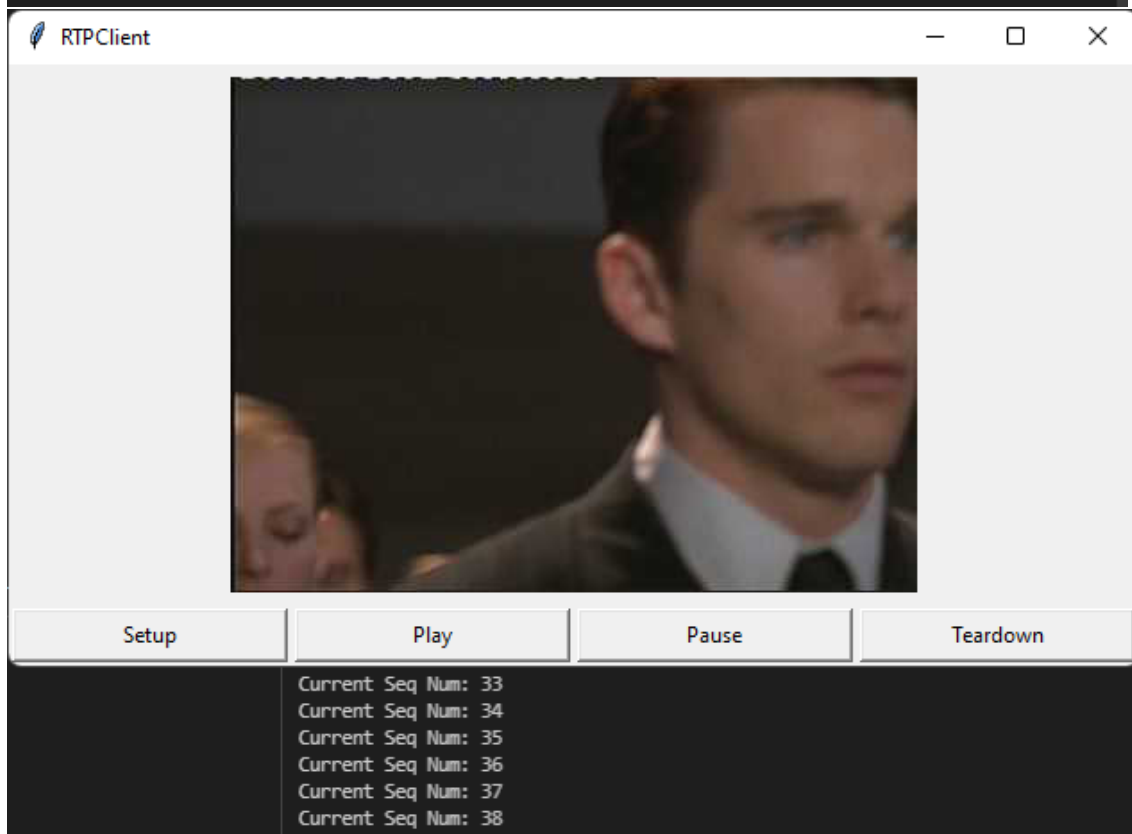
Khi muốn đóng chương trình, ta bấm vào nút Teardown



```
PS C:\Users\Admin\Desktop\Mạng máy tính\CSE-Computer_Network\Assignment\Assignment1\
Students> py ClientLauncher.py localhost 3000 5000 movie.Mjpeg

Data sent:
SETUP movie.Mjpeg RTSP/1.0
CSeq: 1
Transport: RTP/UDP; client_port= 5000
[]

PS C:\Users\Admin\Desktop\Mạng máy tính\CSE-Computer_Network\Assignment\Assignment1\
Students> py Server.py 3000
Data received:
SETUP movie.Mjpeg RTSP/1.0
CSeq: 1
Transport: RTP/UDP; client_port= 5000
processing SETUP
```





## 5 Source code

### 5.1 ClientLaucher

---

```
import sys
from tkinter import Tk
from Client import Client

if __name__ == "__main__":
    try:
        serverAddr = sys.argv[1]
        serverPort = sys.argv[2]
        rtpPort = sys.argv[3]
        fileName = sys.argv[4]
    except:
        print("[Usage: ClientLauncher.py Server_name Server_port RTP_port Video_file]\n")

    root = Tk()

    # Create a new client
    app = Client(root, serverAddr, serverPort, rtpPort, fileName)
    app.master.title("RTPClient")
    root.mainloop()
```

---

### 5.2 Client

---

```
from tkinter import *
import tkinter.messagebox
from PIL import Image, ImageTk
import socket
import threading
import sys
import traceback
import os

from RtpPacket import RtpPacket

CACHE_FILE_NAME = "cache-"
CACHE_FILE_EXT = ".jpg"

class Client:
    INIT = 0
    READY = 1
    PLAYING = 2
    state = INIT

    SETUP = 0
    PLAY = 1
    PAUSE = 2
    TEARDOWN = 3
```

```
# Initiation..
def __init__(self, master, serveraddr, serverport, rtpport, filename):
    self.master = master
    self.master.protocol("WM_DELETE_WINDOW", self.handler)
    self.createWidgets()
    self.serverAddr = serveraddr
    self.serverPort = int(serverport)
    self.rtpPort = int(rtpport)
    self.fileName = filename
    self.rtspSeq = 0
    self.sessionId = 0
    self.requestSent = -1
    self.teardownAcked = 0
    self.connectToServer()
    self.frameNbr = 0

# THIS GUI IS JUST FOR REFERENCE ONLY, STUDENTS HAVE TO CREATE THEIR OWN 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"
    self.teardown["command"] = self.exitClient
    self.teardown.grid(row=1, column=3, 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=4,
                    sticky=W+E+N+S, padx=5, pady=5)

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

```
# TODO

def exitClient(self):
    """Teardown button handler."""
    self.sendRtspRequest(self.TEARDOWN)
    self.master.destroy() # Close the gui window
    # Delete the cache image from video
    os.remove(CACHE_FILE_NAME + str(self.sessionId) + CACHE_FILE_EXT)
# TODO

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

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)
# TODO

def listenRtp(self):
    """Listen for RTP packets."""
    while True:
        try:
            data = self.rtpSocket.recv(20480)
            if data:
                rtpPacket = RtpPacket()
                rtpPacket.decode(data)
                currFrameNbr = rtpPacket.seqNum()
                print("Current Seq Num: " + str(currFrameNbr))
                if currFrameNbr > self.frameNbr: # Discard the late packet
                    self.frameNbr = currFrameNbr
                    self.updateMovie(self.writeFrame(
                        rtpPacket.getPayload()))
            except:
                # Stop listening upon requesting PAUSE or TEARDOWN
                if self.playEvent.isSet():
                    break

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

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

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

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)

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:
        self.rtspSeq += 1
        request = 'PLAY ' + self.fileName + ' RTSP/1.0\nCSeq: ' + \
            str(self.rtspSeq) + '\nSession: ' + str(self.sessionId)
        self.requestSent = self.PLAY

    # Pause request
    elif requestCode == self.PAUSE and self.state == self.PLAYING:
        self.rtspSeq += 1
        request = 'PAUSE ' + self.fileName + ' RTSP/1.0\nCSeq: ' + \
            str(self.rtspSeq) + '\nSession: ' + str(self.sessionId)
```

```
self.requestSent = self.PAUSE

# Teardown request
elif requestCode == self.TEARDOWN and not self.state == self.INIT:
    self.rtspSeq += 1
    request = 'TEARDOWN ' + self.fileName + ' RTSP/1.0\nCSeq: ' + \
        str(self.rtspSeq) + '\nSession: ' + str(self.sessionId)
    self.requestSent = self.TEARDOWN
else:
    return

# Send the RTSP request using rtspSocket.
self.rtspSocket.send(request.encode())

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

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

        if reply:
            self.parseRtspReply(reply.decode())

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

def parseRtspReply(self, data):
    """Parse the RTSP reply from the server."""
    lines = data.split('\n')
    seqNum = int(lines[1].split(' ')[1])
    if seqNum == self.rtspSeq:
        session = int(lines[2].split(' ')[1])
        # New RTSP session ID
        if self.sessionId == 0:
            self.sessionId = session

        # Process only if the session ID is the same
        if self.sessionId == session:
            if int(lines[0].split(' ')[1]) == 200:
                if self.requestSent == self.SETUP:
                    # Update RTSP state.
                    self.state = self.READY
                    # Open RTP port.
                    self.openRtpPort()
                elif self.requestSent == self.PLAY:
                    self.state = self.PLAYING
                elif self.requestSent == self.PAUSE:
                    self.state = self.READY
                    # The play thread exits. A new thread is created on resume.
```

```
        self.playEvent.set()
    elif self.requestSent == self.TEARDOWN:
        self.state = self.INIT
        # Flag the teardownAcked to close the socket.
        self.teardownAcked = 1

def openRtpPort(self):
    """Open RTP socket binded to a specified port."""
    # Create a new datagram socket to receive RTP packets from the server
    self.rtpSocket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

    # Set the timeout value of the socket to 0.5sec
    self.rtpSocket.settimeout(0.5)

    try:
        # Bind the socket to the address using the RTP port given by the client user
        self.rtpSocket.bind(("", self.rtpPort))
    except:
        tkinter.messagebox.showwarning(
            'Unable to Bind', 'Unable to bind PORT=%d' % self.rtpPort)

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

---

### 5.3 ServerWorker

---

```
from random import randint
import sys, traceback, threading, socket

from VideoStream import VideoStream
from RtpPacket import RtpPacket

class ServerWorker:
    SETUP = 'SETUP'
    PLAY = 'PLAY'
    PAUSE = 'PAUSE'
    TEARDOWN = 'TEARDOWN'

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

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

```
clientInfo = {}

def __init__(self, clientInfo):
    self.clientInfo = clientInfo

def run(self):
    threading.Thread(target=self.recvRtspRequest).start()

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"))

def processRtspRequest(self, data):
    """Process RTSP request sent from the client."""
    # Get the request type
    request = data.split('\n')
    line1 = request[0].split(' ')
    requestType = line1[0]

    # Get the media file name
    filename = line1[1]

    # Get the RTSP sequence number
    seq = request[1].split(' ')

    # Process SETUP request
    if requestType == self.SETUP:
        if self.state == self.INIT:
            # Update state
            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]

    # Process PLAY request
```

```
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()

# Process PAUSE request
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])

# Process TEARDOWN request
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()

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['rtspSocket'].sendto(self.makeRtp(data,
            frameNumber),(address,port))
    except:
        print("Connection Error")
        #print('-'*60)
        #traceback.print_exc(file=sys.stdout)
        #print('-'*60)

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()

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")
```

---

## 5.4 Server

---

```
import sys, socket

from ServerWorker import ServerWorker

class Server:

    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))
rtspSocket.listen(5)

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

if __name__ == "__main__":
    (Server()).main()
```

---

## 5.5 RTP packet

---

```
import sys
from time import time
HEADER_SIZE = 12

class RtpPacket:
    header = bytearray(HEADER_SIZE)

    def __init__(self):
        pass

    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] = (header[0] | version << 6) & 0xC0 # 2 bits
        header[0] = (header[0] | padding << 5) # 1 bit
        header[0] = (header[0] | extension << 4) # 1 bit
        header[0] = (header[0] | (cc & 0x0F)) # 4 bits
        header[1] = (header[1] | marker << 7) # 1 bit
        header[1] = (header[1] | (pt & 0x7f)) # 7 bits
        header[2] = (seqnum & 0xFF00) >> 8 # 16 bits total, this is first 8
        header[3] = (seqnum & 0xFF) # second 8
        header[4] = (timestamp >> 24) # 32 bit timestamp
```

```
header[5] = (timestamp >> 16) & 0xFF
header[6] = (timestamp >> 8) & 0xFF
header[7] = (timestamp & 0xFF)
header[8] = (ssrc >> 24) # 32 bit ssrc
header[9] = (ssrc >> 16) & 0xFF
header[10] = (ssrc >> 8) & 0xFF
header[11] = ssrc & 0xFF

# Set RtpPacket's header and payload.
self.header = header
self.payload = payload

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

def decode(self, byteStream):
    """Decode the RTP packet."""
    self.header = bytearray(byteStream[:HEADER_SIZE])
    self.payload = byteStream[HEADER_SIZE:]

def version(self):
    """Return RTP version."""
    return int(self.header[0] >> 6)

def seqNum(self):
    """Return sequence (frame) number."""
    seqNum = self.header[2] << 8 | self.header[3]
    return int(seqNum)

def timestamp(self):
    """Return timestamp."""
    timestamp = self.header[4] << 24 | self.header[5] << 16 | self.header[6] << 8 |
        self.header[7]
    return int(timestamp)

def payloadType(self):
    """Return payload type."""
    pt = self.header[1] & 127
    return int(pt)

def getPayload(self):
    """Return payload."""
    return self.payload

def getPacket(self):
    """Return RTP packet."""
    return self.header + self.payload
```

---

## 5.6 VideoStream

---

```
class VideoStream:
    def __init__(self, filename):
        self.filename = filename
        try:
            self.file = open(filename, 'rb')
        except:
            raise IOError
        self.frameNum = 0

    def nextFrame(self):
        """Get next frame."""
        data = self.file.read(5) # Get the framelength from the first 5 bits
        if data:
            framelength = int(data)

            # Read the current frame
            data = self.file.read(framelength)
            self.frameNum += 1
        return data

    def frameNbr(self):
        """Get frame number."""
        return self.frameNum
```

---

## 6 Extend

### 6.1 Extend 1

#### 6.1.1 Tỷ lệ mất gói

Ta có, tỷ lệ mất gói được tính theo công thức:  $\frac{N_s - N_r}{N_s}$

Trong đó:

- $N_s$  là số gói được gửi từ Server đến Client.
- $N_r$  là số gói Client nhận được từ Server.

#### 6.1.2 Tốc độ truyền dữ liệu

Ta có, tốc độ truyền dữ liệu của video được tính bằng công thức  $\frac{S}{T}$

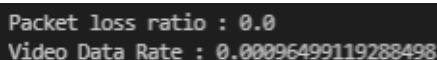
Trong đó:

- $S$  là dung lượng của video
- $T$  là thời gian thực hiện truyền tin từ Server đến client

### 6.1.3 Code Extend 1

```
class Client:
    def exitClient(self):
        """Teardown button handler."""
        self.sendRtspRequest(self.TEARDOWN)
        print("Packet loss ratio : " +
              str(float(self.numLostPackets)/float(self.numSentPackets)))
        print("Video Data Rate : " +
              str(float(self.totalLen)/float(self.sumOfDeltaTime)))
        self.master.destroy() # Close the gui window
        # Delete the cache image from video
        os.remove(CACHE_FILE_NAME + str(self.sessionId) + CACHE_FILE_EXT)
        # TODO
```

### 6.1.4 Kết quả minh họa



```
Packet loss ratio : 0.0
Video Data Rate : 0.00096499119288498
```

## 6.2 Extend 2

Trong yêu cầu này, giao diện người dùng chỉ có 3 nút là PLAY, PAUSE, STOP( không có nút SETUP) Để xóa nút SETUP ta thực hiện theo những bước sau:

- Thiết lập GUI với 3 nút: PLAY, PAUSE, STOP. Vì không có nút SETUP để bắt đầu phân tích dữ liệu thành các gói RTP, nên khi module ClientLauncher tạo một client, nó cũng sẽ đặt cho ta phim và thiết lập kết nối RTSP.

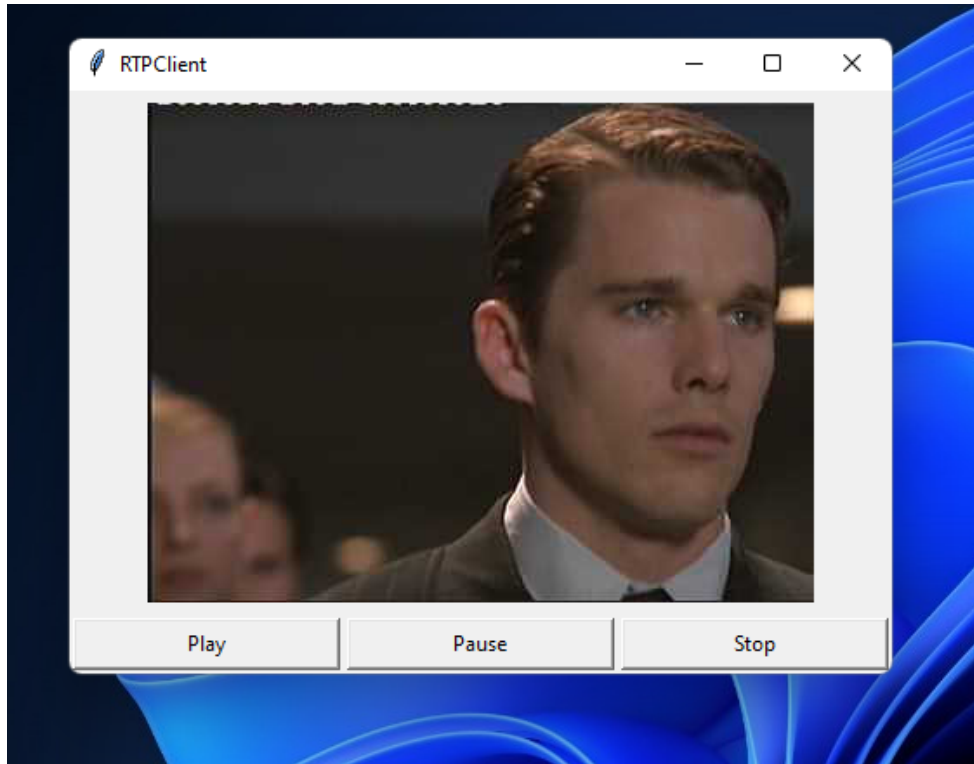
```
def __init__(self, master, serveraddr, serverport, rtpport, filename):
    self.master = master
    self.master.protocol("WM_DELETE_WINDOW", self.handler)
    self.createWidgets()
    self.serverAddr = serveraddr
    self.serverPort = int(serverport)
    self.rtpPort = int(rtpport)
    self.fileName = filename
    self.rtspSeq = 0
    self.sessionId = 0
    self.requestSent = -1
    self.teardownAked = 0
    self.connectToServer()
    self.frameNbr = 0
    self.setupMovie()
```

- Nút STOP đóng vai trò như một nút reset. Khi người dùng nhấp vào nút STOP, kết nối giữa máy khách và máy chủ đã được thiết lập lại. CSeq được đặt thành 1. Ta sẽ viết hàm resetMovie trong client module để xử lý hành động này. Ta đặt thời gian chờ là 1 giây khi người dùng nhấp vào nút STOP.

```
def reset(self):  
    # if self.state == self.TEARDOWN:  
    self.pauseMovie()  
    try:  
        os.remove(CACHE_FILE_NAME +  
                  str(self.sessionId) + CACHE_FILE_EXT)  
    except:  
        pass  
    time.sleep(1)  
    self.state = self.INIT  
    self.rtspSeq = 0  
    self.sessionId = 0  
    self.requestSent = -1  
    self.teardownAcked = 0  
    self.frameNbr = 0  
    self.counter = 0  
    self.check = False  
    self.connectToServer()  
    self.rtpSocket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)  
    self.setupMovie()
```

- Ta cần xử lý khi người dùng nhấn vào nút STOP bằng cách cài đặt cho lệnh resetMovie.

Giao diện người dùng trên Client



### 6.3 Extend 3

Trong yêu cầu này, ta sẽ xử lý phương thức DESCRIBE được sử dụng để đưa thông về media stream. Khi máy chủ nhận được yêu cầu DESCRIBE, nó sẽ gửi lại thông tin về video stream.

Giao diện người dùng:



Để hiện thực phương pháp DESCRIBE, ta thực hiện theo những bước sau:

- Tạo phương thức DESCRIBE từ Client được gửi đến Server. Yêu cầu được thực hiện chỉ khi trạng thái của client là READY. Yêu cầu được tạo với cấu trúc: DESCRIBE + file name + Transport protocol + RTSP Sequence number
- Sau khi nhận được yêu cầu DESCRIBE. Server sẽ xử lý yêu cầu và trả lời Client.
- Sau khi Client nhận được thông tin từ Server, Client sẽ cập nhật trạng thái là READY

Kết quả:

```
SDF-863223.txt u x
CSE-Computer_Network > Assignent > Assignent1 > Students_Extends > SDF-863223.txt
1 |type_of_stream RTSP
2 |version 1.0
3 |session_id 863223
4 |session_starttime Sat
5 |pkt_encoding rtpFormat
6 |server_addr localhost
7 |server_port 3000
8 |client_port 5000
```