

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

# Assignment 1

Mã môn học: CO3093  
GVHD: Nguyễn Quang Sang  
Nhóm SV thực hiện: Đào Xuân Đạt 1911000 - L01  
Nguyễn Thế Huy 1812404 - L01  
Lê Bảo Khánh 1911363 - L01  
Vũ Quang Trung 1912330 - L01

TP. HỒ CHÍ MINH, THÁNG 11/2021



## Contents

<b>1</b>	<b>Danh sách các thành viên và phân công công việc</b>	<b>1</b>
<b>2</b>	<b>Analysis of problem requirements</b>	<b>1</b>
2.1	Yêu cầu chức năng . . . . .	1
2.2	Yêu cầu phi chức năng . . . . .	1
<b>3</b>	<b>Description functions</b>	<b>2</b>
3.1	ClientLauncher . . . . .	2
3.2	Client . . . . .	2
3.3	Server . . . . .	3
3.4	ServerWorker . . . . .	4
3.5	RtpPacket . . . . .	4
3.6	VideoStream . . . . .	5
<b>4</b>	<b>Class diagram</b>	<b>6</b>
<b>5</b>	<b>A summative evaluation of achieved results</b>	<b>6</b>
5.1	Về lập trình . . . . .	6
5.2	Kết quả . . . . .	7
<b>6</b>	<b>User manual</b>	<b>9</b>
<b>7</b>	<b>Extend</b>	<b>10</b>
7.1	Statistics . . . . .	10
7.2	Hiện thực PLAY, PAUSE, STOP . . . . .	10
7.3	Hiện thực DESCRIBE . . . . .	11
7.4	Hiện thị thời gian . . . . .	11
7.5	Hiện thực nút Switch . . . . .	12
<b>8</b>	<b>Source code</b>	<b>12</b>



## 1 Danh sách các thành viên và phân công công việc

Thành viên	MSSV	Lớp	Công việc
Đào Xuân Đạt	1911000	L01	Hiện thực hàm parseRtspReply trong file Client.py và báo cáo
Nguyễn Thế Huy	1812404	L01	Hiện thực hàm sendRtspRequest trong file Client.py và báo cáo
Lê Bảo Khánh	1911363	L01	Thực hiện Extend và báo cáo
Vũ Quang Trung	1912330	L01	Hiện thực hàm encode trong file RtpPacket.py, Extend và báo cáo

Table 1: Bảng phân công

## 2 Analysis of problem requirements

### 2.1 Yêu cầu chức năng

- Hệ thống có thể hoạt động (stream video) được, giao diện dễ sử dụng, có các chức năng thiết yếu như pause (tạm dừng video), play(phát video), teardown(dừng video, đóng ứng dụng).
- Hệ thống có thể giao tiếp với người dùng qua giao thức RTSP/RTP.
- Có thể kết nối được đến server qua terminal.
- Người dùng có thể xem các video từ server liên tục mà không bị gián đoạn.

### 2.2 Yêu cầu phi chức năng

- Thời gian phản hồi từ server ngắn, nhỏ hơn 0,5s.
- Các video phải có định dạng .Mjpeg

### 3 Description functions

Class Name	Function	Parameter	Description
SeverWorker	__init__(self, clientInfo)	self, clientInfo	Constructor
	run(self)	self	Run the server
	processRtpRequest(self, data)	self, data	Process the Rtp request
	sendRtp(self)	self	Send RTP packets over UDP
	makeRtp(self, payload, frameNbr)	self, payload, frameNbr	RTP-packetize the video data
	replyRtsp(self, code, seq)	self, code, seq	Send RTSP reply to the Client
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
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, buftime)	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
RtpPacket	__init__(self)	self	constructor
	encode(self, version, padding, extension, cc, seqnum, marker, pt, ssrc, payload)	self, version, padding, extension, cc, seqnum, marker, pt, ssrc, payload	Encode the RTP packet with header fields and 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

Figure 1: Bảng mô tả chức năng của từng hàm tương ứng với các Class

#### 3.1 ClientLauncher

- Tạo một giao diện người dùng sử dụng thư viện tkinter của ngôn ngữ python.
- Thông qua giao diện người dùng có thể gửi yêu cầu với giao thức RTSP thông qua việc kích hoạt các nút được tạo trên giao diện và hiện thị video mà người dùng yêu cầu.

#### 3.2 Client

- Tạo các Widget trên giao diện người dùng và thiết lập chức năng cho từng Widget.
- Tạo một RTSP socket cho client và kết nối nó với RTSP socket mà server đã tạo, thông qua public host và địa chỉ port mà socket Server đã liên kết. Khi người dùng click các nút trên khung giao diện thì client sẽ gửi yêu cầu đến server qua hai RTSP socket này.
- Nút Setup: gửi yêu cầu Setup đến Server, cung cấp tên file video mà người dùng yêu cầu, cùng thông tin giao thức RTP/UDP và port RTP mà client sẽ tạo để Server có thể kết nối và gửi các segment chứa các frame hình ảnh video đến Client. Khi nhận được hồi đáp từ

Server là "OK 200" thì chuyển trạng thái Client sang READY nhằm cho biết hệ thống đã sẵn sàng để phát video trên ứng dụng. Ngoài ra khi nhận được hồi đáp thành công từ server thì Client sẽ mở một RTP socket chạy trên giao thức UDP để nhận các segment chứa frame hình ảnh mà server gửi về khi nhận được yêu cầu Play.

- Nút Play: Gửi yêu cầu phát video đến Server. Người dùng click Play, nếu trạng thái hiện tại đang là READY thì yêu cầu play sẽ được Client gửi đến Server. Khi nhận được yêu cầu play, Server sẽ tạo một RTP socket gửi segment chứa frame hình ảnh video đến client bằng cách gửi dữ liệu đến RTP socket của Client, dữ liệu được gửi một cách liên tục theo thời gian thực, ngoài ra Server sẽ gửi hồi đáp thành công cho client qua RTSP socket. Khi nhận được hồi đáp thành công, trạng thái client được chuyển sang PLAYING.
- Khi nhận được dữ liệu mà Server gửi qua RTP socket thì tại Client chúng được giải mã và trích xuất các frame là dữ liệu hình ảnh được gửi về từ Server. Các frame được lưu trong bộ nhớ đệm dưới dạng jpeg và hiện thị trên giao diện người dùng.
- Nút Pause: Gửi yêu cầu tạm dừng video đến server. Người dùng click Pause, nếu client đang ở trạng thái PLAYING, yêu cầu pause được gửi đến server. Nhận được yêu cầu này server dừng gửi dữ liệu. Hồi đáp thành công đến client và video dừng lại, trạng thái client chuyển sang READY. Dữ liệu tiếp tục được gửi nếu người dùng nhấn Play.
- Nút TearDown: Gửi yêu cầu dừng video, các socket trên Client được đóng. Người dùng click TearDown, yêu cầu teardown được gửi đến Server. Nhận được yêu cầu này, Server sẽ dừng gửi dữ liệu đến Client, gửi hồi đáp thành công đến Client. Khi nhận được hồi đáp thành công từ Server, Client chuyển trạng thái sang INIT, đóng toàn bộ socket của Client hủy đối tượng giao diện và xóa file hình ảnh tại bộ nhớ đệm.

### 3.3 Server

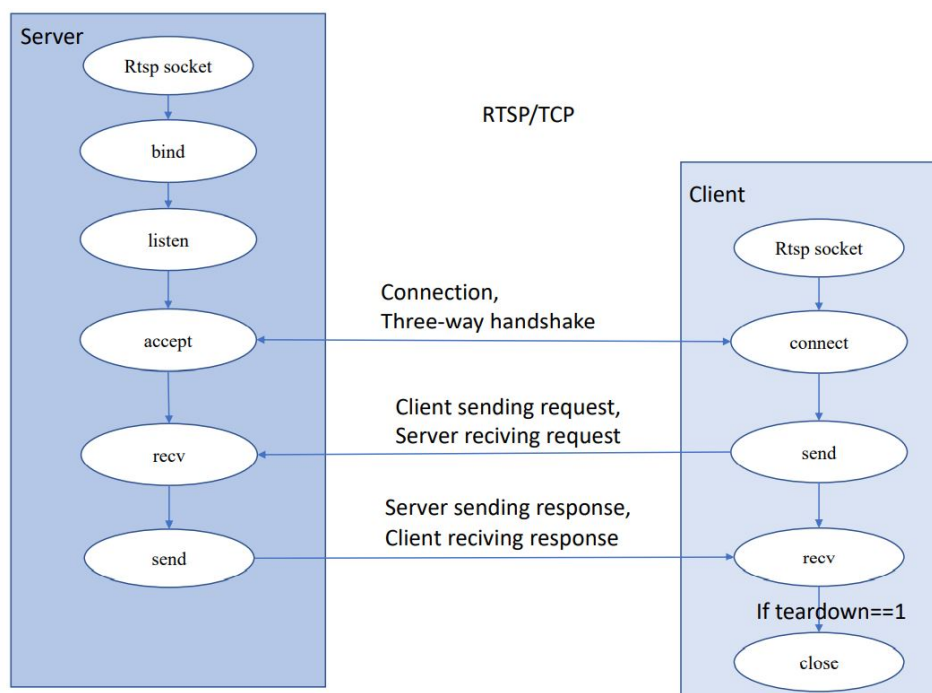


Figure 2: Giao tiếp giữa Client và Server bằng phương thức RTSP/TCP

- Tạo RTSP Socket và mở kết nối đến public host và địa chỉ port đã cung cấp, Client và Server sẽ giao tiếp với nhau qua các RTSP socket trên mỗi thành phần - nhận và phản hồi yêu cầu từ Client.
- Biến ClientInfo được tạo ra để chứa dữ liệu mà client gửi cho server và chuyển qua Server-Worker để xử lý yêu cầu.

### 3.4 *ServerWorker*

- Xử lý Request(Setup, Play, Pause, Teardown) được gửi từ Client thông qua giao thức RTSP.
- Với mỗi client, một luồng xử lý Request được tạo ra. Trong mỗi luồng, một rtp socket được khởi tạo để gửi data đến rtp socket của Client, mỗi luồng có các phương thức nhận, xử lý yêu cầu và phản hồi yêu cầu từ Client.
- Khi nhận được yêu cầu từ Client, Server trích xuất dữ liệu của yêu cầu xác định kiểu yêu cầu và các thông tin cần thiết.
- Khi nhận được yêu cầu setup từ Client, tên file video cần phát được cung cấp qua yêu cầu từ Client và lưu vào ClientInfo đối tượng VideoStream của file video nhằm phục vụ cho quá trình tải và gửi dữ liệu video. Ngoài ra thông tin về Client port cũng được xác định và lưu lại. Một sessionId được tạo ra một cách ngẫu nhiên, con số này giúp cho Client nhận đúng phản hồi mà Server gửi đến. Server gửi hồi đáp nhận yêu cầu thành công đến Client và chuyển trạng thái server sang READY.
- Khi nhận được yêu cầu play từ Client, kiểm tra trạng thái hiện tại của Server, nếu trạng thái hiện tại là READY, chuyển trạng thái Server sang PLAYING, tạo một RTP socket để gửi dữ liệu hình ảnh đến Client. Server sẽ tạo ra một luồng để đọc dữ liệu từ video được yêu cầu, sử dụng phương thức nextFrame của Videostream để đọc các frame tương ứng với từng hình ảnh trong video, mã hóa, đóng gói thành packet, đóng gói packet thành các segment UDP và gửi đến Client theo giao thức RTP/UDP, luồng đọc lặp và vòng lặp while kết hợp gửi giao thức UDP cho phép dữ liệu được đọc và gửi một cách liên tục. Gửi phản hồi yêu cầu đến Client.
- Khi nhận được yêu cầu pause, Server kiểm tra trạng thái hiện tại của mình, nếu trạng thái hiện tại là PLAYING, dùng threading.Event().set() đặt cờ thành true. Tại luồng đọc dữ liệu, nếu cờ bằng true dừng đọc dữ liệu. Chuyển trạng thái server sang READY và gửi phản hồi thành công đến Client.
- Khi nhận được yêu cầu teardown, dừng video như xử lý yêu cầu pause, gửi phản hồi về Client và đóng RTP socket.

### 3.5 *RtpPacket*

- Hàm encode: Thiết lập header cho đối tượng, gán header và frame dữ liệu và các biến header và payload của đối tượng.
- Hàm decode: xác định thành phần header và payload trong một packet, và gán chúng cho các biến tương ứng.
- Các hàm version, seqNum, timestamp, payloadType xuất các thông số được lưu ở phần header.
- Hàm getPayload: trả về payload trong packet.

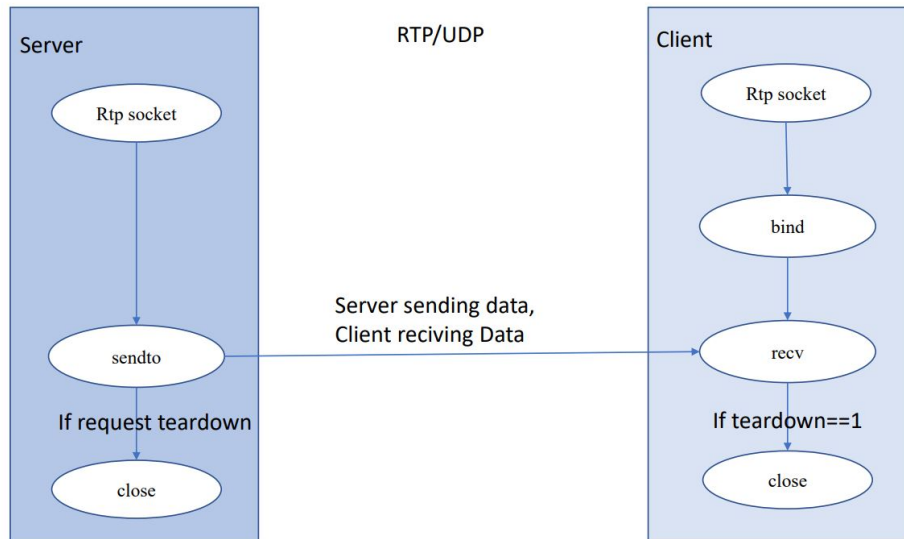


Figure 3: Giao tiếp giữa Client và Server bằng phương thức RTP/TCP

- Hàm `getPacket`: Đóng gói frame hình ảnh thành packet.

### 3.6 *VideoStream*

Mở file video cần đọc, đọc từng frame trong dữ liệu video, mỗi frame tương ứng với một hình ảnh xuất hiện trong video. Hàm `nextFrame` là phương thức có chức năng đọc một frame dữ liệu trong file video, khi kết hợp vòng lặp `while` toàn bộ frame trong file video sẽ được đọc.

## 4 Class diagram

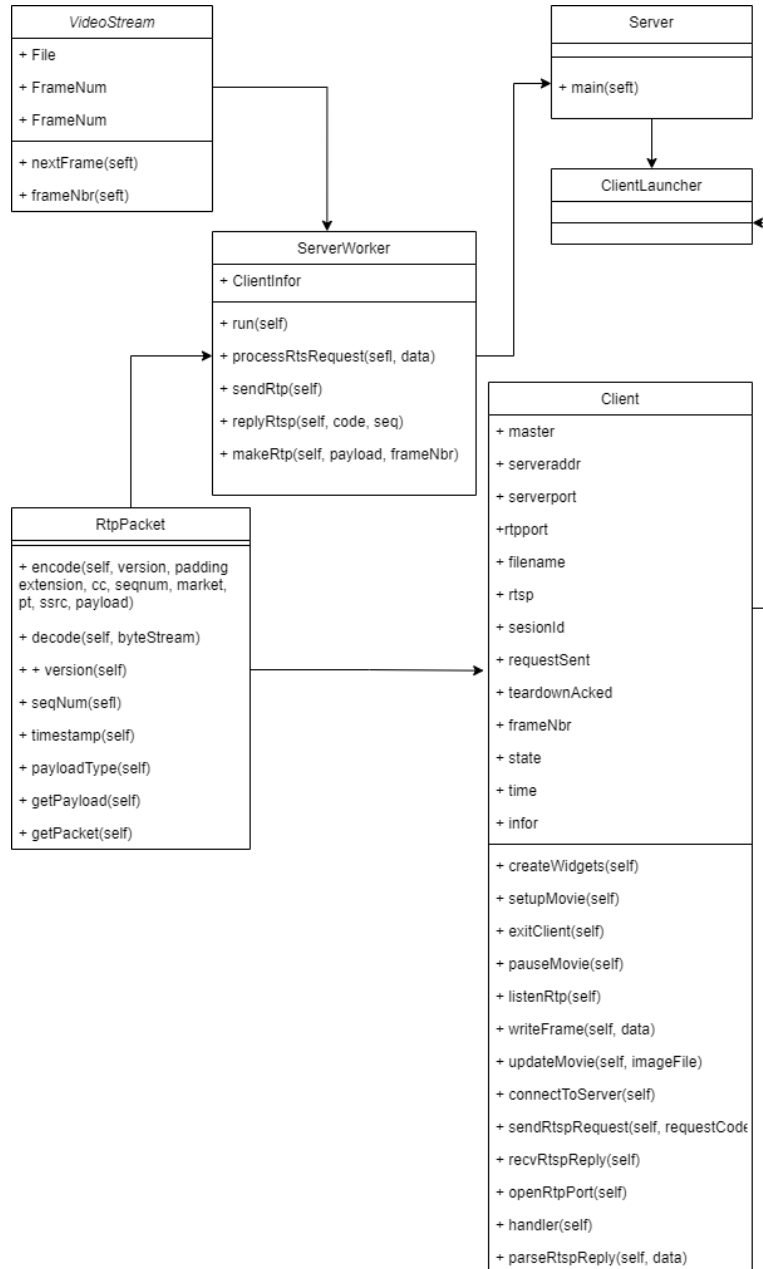


Figure 4: Bảng mô tả chức năng của từng hàm tương ứng với các Class

## 5 A summative evaluation of achieved results

### 5.1 Về lập trình

- RtpPacket: Hoàn thành được thiết lập header cho rtp packet, hiểu cách sử dụng phương thức được xây dựng trong các hàm, class.
- Về Client: Hoàn thành phần viết cấu trúc yêu cầu cho Client và gửi nó đến Server.



## 5.2 Kết quả

- Khởi động được Server. Server nhận được các yêu cầu của Client khi chúng được gửi đến, quá trình trích xuất yêu cầu tại Server hoạt động và cho ra kết quả như mong muốn.
- Client nhận được các segment chứa dữ liệu hình ảnh mà Server gửi về và phát video thành công. Tuy nhiên, chúng tôi nhận chất lượng hình ảnh còn kém. Figure 5 minh họa cho sự thành công của chúng tôi trong việc phát video ở ứng dụng Client. Các button trên khung giao diện hoạt động tốt và đúng chức năng đã thiết lập. Nhìn chung, chúng tôi đã giải quyết các vấn đề được yêu cầu ở bài tập này và hoàn thành bài tập.
- Với phần mở rộng của đề tài, chúng tôi đã sử dụng WireShark để bắt các gói tin trao đổi giữa Client và Server để tính toán các tỉ lệ được yêu cầu kết quả cụ thể sẽ được trình bày ở phần sau. Về việc hiện thực ứng dụng và không có sử dụng nút "Setup" chúng tôi đã hoàn thành, ngoài ra chúng tôi đã thiết lập thêm nút "Stop" và "Describe" cho ứng dụng. Code hiện thực được lưu ở file Client1.py, ServerWork1.py .

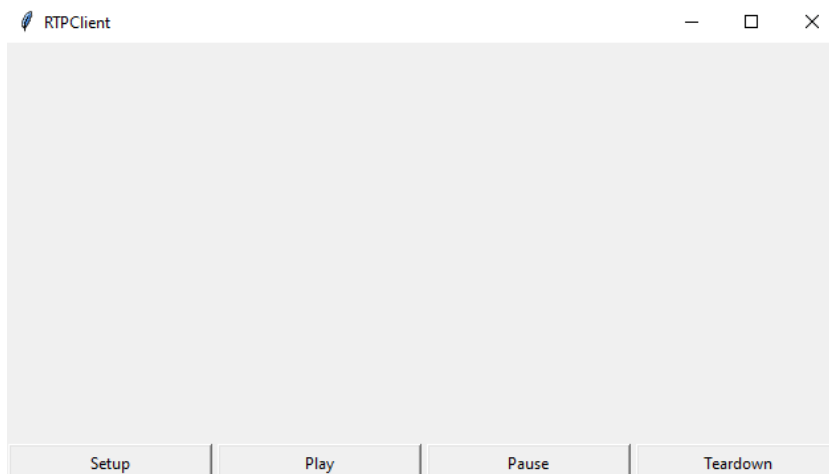


Figure 5: Giao diện của ứng dụng khi khởi động

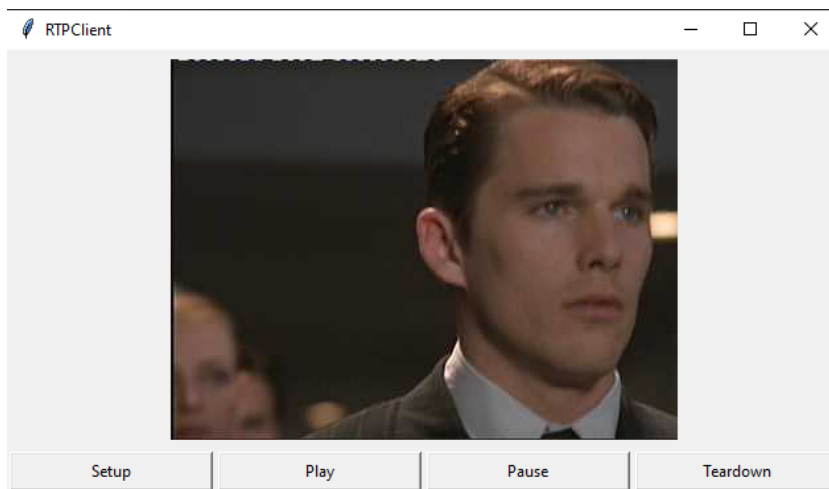


Figure 6: Giao diện của ứng dụng khi bấm Play



Figure 7: Giao diện của ứng dụng khi bấm Pause

```
Current Seq Num: 152
size frame: 8368
time: 0.33721351623535156
video data rate: 24815.0 bytes/s

Current Seq Num: 153
size frame: 8364
time: 0.40004587173461914
video data rate: 20907.0 bytes/s

Current Seq Num: 154
size frame: 8355
time: 0.4628770351409912
video data rate: 18050.0 bytes/s

Current Seq Num: 155
size frame: 8336
time: 0.5247128009796143
video data rate: 15886.0 bytes/s
```

Figure 8: Chi tiết Session

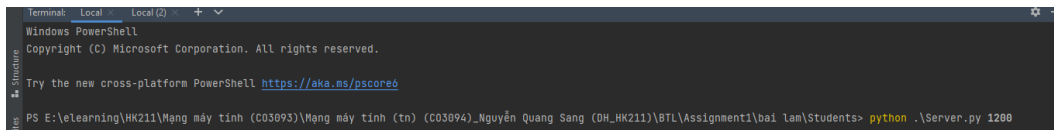
9 19:54:35.002825	192.168.1.10	40.77.226.250	TCP	66 63075 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
11 19:54:35.517985	40.77.226.250	192.168.1.10	TCP	66 443 → 63075 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1420 WS=256 SACK_PERM=1
14 19:54:35.518371	192.168.1.10	40.77.226.250	TCP	54 63075 → 443 [ACK] Seq=1 Ack=1 Win=66560 Len=0
16 19:54:35.518941	192.168.1.10	40.77.226.250	TLSv1.2	571 Client Hello
20 19:54:35.830476	40.77.226.250	192.168.1.10	TCP	1506 443 → 63075 [ACK] Seq=1 Ack=518 Win=262144 Len=1452 [TCP segment of a reassembled PDU]
21 19:54:35.830476	40.77.226.250	192.168.1.10	TCP	1506 443 → 63075 [ACK] Seq=1453 Ack=518 Win=262144 Len=1452 [TCP segment of a reassembled PDU]
22 19:54:35.830476	40.77.226.250	192.168.1.10	TLSv1.2	1045 Server Hello, Certificate, Server Key Exchange, Server Hello Done
23 19:54:35.830649	192.168.1.10	40.77.226.250	TCP	54 63075 → 443 [ACK] Seq=518 Ack=3896 Win=66560 Len=0
24 19:54:35.832883	192.168.1.10	40.77.226.250	TLSv1.2	147 Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
25 19:54:36.249047	40.77.226.250	192.168.1.10	TLSv1.2	105 Change Cipher Spec, Encrypted Handshake Message
26 19:54:36.251841	192.168.1.10	40.77.226.250	TLSv1.2	1470 Application Data
27 19:54:36.743601	40.77.226.250	192.168.1.10	TCP	54 443 → 63075 [ACK] Seq=3947 Ack=2027 Win=262656 Len=0
29 19:54:36.986886	40.77.226.250	192.168.1.10	TLSv1.2	368 Application Data
30 19:54:36.986191	192.168.1.10	40.77.226.250	TCP	54 63075 → 443 [ACK] Seq=2027 Ack=4262 Win=66304 Len=0
31 19:54:36.987402	192.168.1.10	40.77.226.250	TLSv1.2	85 Encrypted Alert
32 19:54:36.987609	192.168.1.10	40.77.226.250	TCP	54 63075 → 443 [FIN, ACK] Seq=2058 Ack=4262 Win=66304 Len=0
33 19:54:37.266833	40.77.226.250	192.168.1.10	TCP	54 [TCP Dup ACK 27#1] 443 → 63075 [ACK] Seq=4262 Ack=2027 Win=262656 Len=0
34 19:54:37.266833	40.77.226.250	192.168.1.10	TCP	54 443 → 63075 [ACK] Seq=4262 Ack=2059 Win=262656 Len=0
73 19:54:55.316185	192.168.1.10	40.77.226.250	TCP	66 63077 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
74 19:54:55.791184	40.77.226.250	192.168.1.10	TCP	66 443 → 63077 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1420 WS=256 SACK_PERM=1

Figure 9: Thông tin khi mở WireShark

## 6 User manual

- Bước 1: Khởi động Server: chạy Terminal trong thư mục chứa file Server.py.  
Gõ lệnh: `python .\Server.py "port-server"`

- Trong đó "port-server" là một số bất kì lớn hơn 1024.
- Ví dụ: `python .\Server.py 1200`



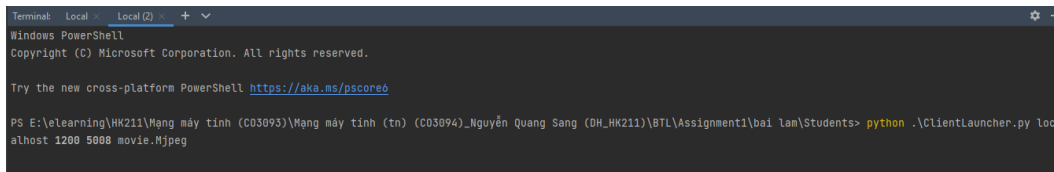
```
Terminal: Local - Local (2) + -
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS E:\elearning\HK211\Mạng máy tính (C03093)\Mạng máy tính (tn) (C03094)_Nguyễn Quang Sang (DH_HK211)\BTL\Assignment1\bai lam\Students> python .\Server.py 1200
```

- Bước 2: Khởi động ClientLauncher: mở Terminal mới trong thư mục chứa file ClientLauncher.py, đây được coi là 1 client kết nối với Server chúng ta mới khởi động.  
Gõ lệnh: `python .\ClientLauncher.py "IP-server" "port-server" "port-RTP" "name-video"`

- "name-server" là tên của server.
- "port-server" là cổng mà ta đã chọn ở bước 1.
- "port-RTP" là số cổng bất kì để nhận RTP packet.
- "name-video" là tên file video ta muốn xem.
- Ví dụ: `python .\ClientLauncher.py localhost 1200 5008 movie.Mjpeg`



```
Terminal: Local - Local (2) + -
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS E:\elearning\HK211\Mạng máy tính (C03093)\Mạng máy tính (tn) (C03094)_Nguyễn Quang Sang (DH_HK211)\BTL\Assignment1\bai lam\Students> python .\ClientLauncher.py localhost 1200 5008 movie.Mjpeg
```

- Bước 3: Các thao tác trên giao diện:
  - Nhấn nút "Setup" để tạo đường truyền RTP.
  - Nhấn "Play" để xem video.
  - Nhấn "Pause" để tạm dừng
  - Nhấn "Teardown" để dừng hẳn video và ngắt kết nối RTP.

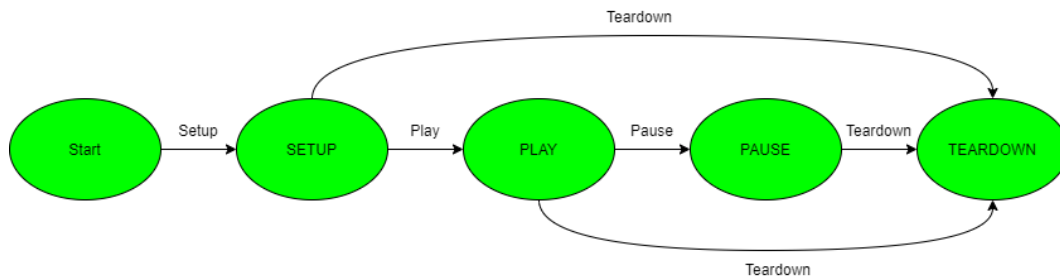


Figure 10: Tuần tự các nút nên bấm

## 7 Extend

Phần extend tương ứng với Clie1.py và ServerWorker1.py trong source code

### 7.1 Statistics

- Chạy Server, ClientLauncher và sử dụng WireShark để bắt các gói tin trao đổi trên localhost, chúng tôi thu được kết quả như Figure 8. Theo quan sát, khi triển khai cả Server và Client trên cùng một localhost của máy, tỉ lệ RTP packet bị lạc gần như bằng 0% (số packet gửi đi 500 và số packet nhận được là bằng nhau). Để theo dõi tốc độ truyền dữ liệu video: xuất ra kích thước payload (phần chứa dữ liệu video trong packet), xuất thời gian chuyển packet từ Server đến Client của từng packet và sử dụng công thức tính tốc độ truyền dữ liệu video như sau: kích thước payload/time.

```
RTSP/1.0 200 OK
CSeq: 3
Session: 826192
Type of Stream Real-Time Streaming
Encode utf-8
4269893
```

Figure 11: Chi tiết Session

### 7.2 Hiện thực *PLAY, PAUSE, STOP*

- Trong Media Player, yêu cầu SETUP là một yêu cầu bắt buộc. Tuy nhiên, người dùng không nhìn thấy nút "Setup" trên khung giao diện, chúng tôi sẽ triển khai nó bằng cách hiện thực theo hướng: khi người dùng click 'Play' lần đầu tiên sau khi mở ứng dụng, xét trạng thái của Client, nếu trạng thái hiện tại là INIT thì gửi thông điệp Setup đến Server. Khi nhận được phản hồi thành công từ Server, Client cập nhật trạng thái sang READY và yêu cầu Play được gửi ngay sau đó. Về TEARDOWN và STOP: Khi người dùng Click "STOP" sẽ không thích hợp để gửi yêu cầu TEARDOWN. Vì STOP chỉ đơn giản là ứng dụng sẽ "reset" lại video (đóng RTP socket), xóa hình ảnh trong bộ nhớ đệm và người dùng có thể tiếp tục xem video bằng cách Click PLAY. Tuy nhiên, lúc này video sẽ được bắt đầu lại từ đầu. Còn 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.

### 7.3 Hiện thực *DESCRIBE*

- DESCRIBE: thêm một nút 'Describe' mà chức năng của nó là khi được click, Client sẽ gửi yêu cầu DESCRIBE đến Server. Khi Server nhận được yêu cầu DESCRIBE, Server sẽ phản hồi lại Client với dữ liệu được yêu cầu trong file đặc tả như: "type of Stream", "encode", kích thước của video. Khi nhận được phản hồi từ Server, Client sẽ trích xuất phản hồi đó và xuất ra terminal.

### 7.4 Hiện thị thời gian

- Thực hiện một số chức năng bổ sung cho giao diện người dùng như: hiển thị tổng thời gian video và thời gian còn lại, tua đi hoặc tua lại video.
- Để thực hiện việc hiển thị thời gian cho video, chúng ta thêm vào class Client:

```
1 def __init__(self, master, serveraddr, serverport, rtpport, fileNameList):
2     self.timeBox = "0 : 0"
3
```

Thêm thanh thời gian vào GUI:

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

Cập nhật time box sau mỗi frame:

```
1 # Set time box
2 currentTime = int(currFrameNbr * 0.05)
3 self.timeBox = str(currentTime // 60) + " : " + str(currentTime % 60)
4 self.status = Label(self.master, text="Watched time : " + str(self.timeBox),
5 bd=1, relief=SUNKEN, anchor=W)
6 self.status.grid(row=2, column=0, columnspan=1, sticky=W + E)
```

Giải thích: Do tốc độ truyền của 1 frame là 0.05s nên ta nhân số frame hiện tại với 0.05s sẽ ra thời điểm tương ứng

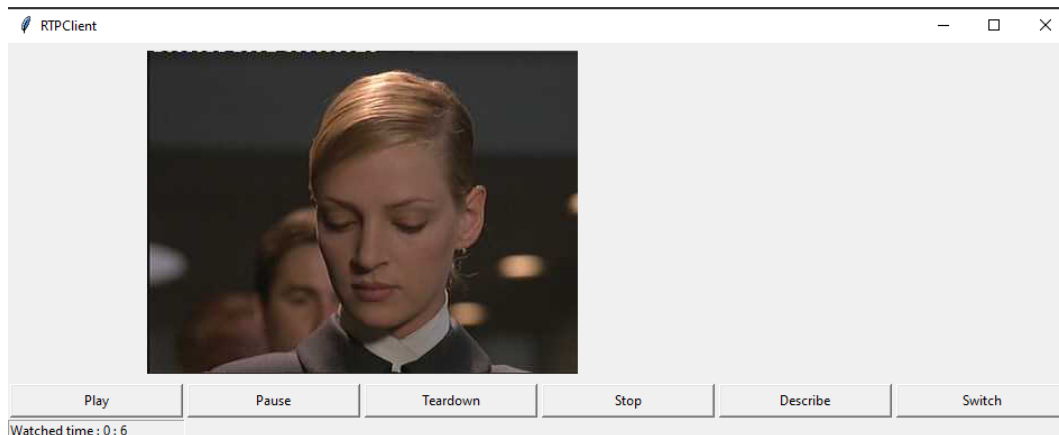


Figure 12: Giao diện có thêm hiển thị thời gian

## 7.5 Hiện thực nút Switch

- Khi người dùng bấm vào nút Switch thì GUI sẽ chuyển sang một video khác với tên do người dùng nhập vào lúc đầu:  
python ClientLauncher.py localhost 1200 5008 [movie1.Mjpeg,movie2.Mjpeg]  
Với danh sách video được nhập vào có định dạng [movie1,movie2,..]

- Thay đổi cách nhập dữ liệu trong Client.py

```
1 def __init__(self, master, serveraddr, serverport, rtpport, fileNameList):
2     self.video_list = fileNameList[1:len(fileNameList) - 1]
3     self.video_list = self.video_list.split(',')
4     print(self.video_list)
5     self.video_list_index = 0
6     self.fileName = self.video_list[0]
7
```

- Chúng ta tiếp tục thêm nút "Switch" tương tự như các nút ở trên. Nút "Switch" chúng ta có thể tách thành 2 bước "Stop" và "Start" sau khi cập nhật lại fileName ứng với video mong muốn

```
1     # Switch movie
2     if self.video_list_index < len(self.video_list) - 1:
3         self.video_list_index += 1
4     else:
5         self.video_list_index = 0
6     self.fileName = self.video_list[self.video_list_index]
7
```

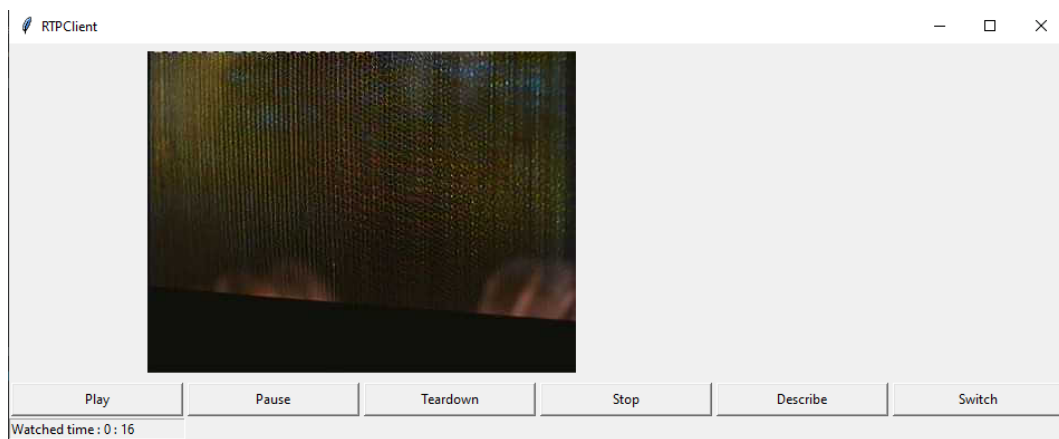


Figure 13: Giao diện trước khi nhấn Switch

## 8 Source code

Link source code:

[https://github.com/baokhanhle123/ComputerNetwork\\_HK211\\_Assignment1](https://github.com/baokhanhle123/ComputerNetwork_HK211_Assignment1)

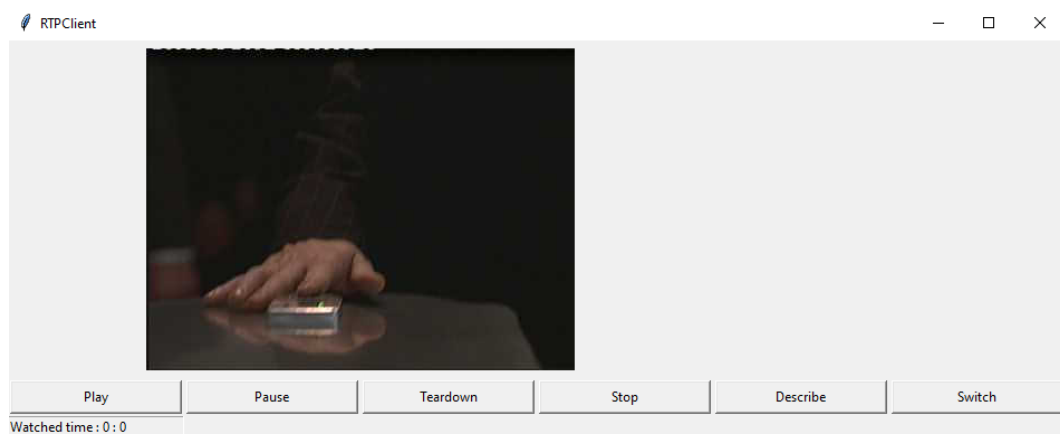


Figure 14: Giao diện sau khi nhấn Switch