

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



MẠNG MÁY TÍNH (CO3094)

Bài tập lớn 1

Video Streaming Application

GV ra đề & hướng dẫn: Lê Bảo Thịnh

Lớp: L02

Nhóm: Akatsuki

Sinh viên thực hiện: Võ Minh Toàn 1915570
Đặng Hùng Cường 1912817
Nguyễn Hải Linh 1913944
Nguyễn Đình Hiếu 1913341



Mục lục

1	Requirements Analysis	3
1.1	Yêu cầu chức năng	3
1.2	Yêu cầu phi chức năng	3
2	Function Description	3
2.1	Client.py	3
2.2	Client2.py	6
2.3	Server.py	11
2.4	RtpPacket.py	11
2.5	ServerWorker.py	12
2.6	VideoStream.py	14
2.7	ClientLauncher.py	15
3	Class Diagram	16
4	A summative Evaluation of Results Achieved	18
5	User Manual	20
5.1	Chạy chương trình	20
5.2	Cách sử dụng sau khi chạy chương trình	21
6	Extend	23
6.1	Calculate the statistics about the session	23
6.1.1	RTP Packet lost rate	23
6.1.2	Video data rate	23
6.2	Implement PLAY, PAUSE and STOP buttons	24
6.3	Implement the DESCRIBE method	27
6.4	Implement some additional functions for user interface such as: display video total time and remaining time, fast forward or backward video	30
6.4.1	Label total time and remaining time	30
6.4.2	Fast forward and backward video	31

Danh sách hình vẽ

1	Class Diagram	16
2	Class Diagram cho Extend	17
3	Giao diện cơ bản của người dùng ở normal flow	18
4	Giao diện của người dùng ở extend flow	19
5	RTP packet header	19
6	Giao diện của người dùng	21
7	Giao diện của người dùng khi nhấn nút Play	21
8	Giao diện của người dùng khi nhấn nút Pause	22
9	Giao diện của người dùng khi nhấn nút X	22
10	Khi người dùng nhấn vào nút Play để xem video	25
11	Khi người dùng nhấn vào nút Pause để dừng video	26
12	Khi người dùng nhấn vào nút Stop để reset video	26
13	Khi người dùng nhấn vào nút X để thoát	27



14	Nội dung về Stream hiển thị ra cho người dùng khi người dùng gửi yêu cầu DESCRIBE	29
15	Nút "«" và "»" trong giao diện người dùng	32

Python Code

1	Hàm displayStats() hiện thực trong Client2.py	23
2	Hàm describeMovie() dùng để gửi cho server yêu cầu DESCRIBE từ phía Client	27
3	Đoạn code dùng để xử lý yêu cầu DESCRIBE từ phía Client bên Server	27
4	Hàm replyDescribe() dùng để gửi thông tin về Stream từ Server về Client	28
5	Một đoạn code trong hàm parseRtspReply() phía Client2.py để xử lý reply từ Server	28
6	Hàm displayDescription() dùng để in ra form gồm những tin về Stream cho người dùng	29
7	Hàm replySetup để trả về thông tin sau khi người dùng SETUP	30
8	Hàm get_total_time() dùng để lấy tổng thời gian của video	31
9	Hàm forwardMovies() hiện thực trong Client2.py	32
10	Hàm nextFrame() hiện thực trong Client2.py	33
11	Đoạn code xử lý yêu cầu BACKWARD từ Client trong Server	33
12	Hàm prevFrame() hiện thực trong VideoStream.py	34



1 Requirements Analysis

1.1 Yêu cầu chức năng

- Triển khai một server video trực tuyến và client giao tiếp bằng giao thức RTSP và gửi dữ liệu bằng giao thức RTP
- Client khởi động sẽ mở RTSP socket đến server để gửi yêu cầu cho server
- Trạng thái của Client liên tục cập nhật khi nhận phản hồi từ server
- Có giao diện đồ họa để client thao tác SETUP, PLAY, PAUSE, TEARDOWN, DESCRIBE

1.2 Yêu cầu phi chức năng

- Thời gian chờ của datagram socket để nhận RTP data từ server là 0.5s
- Server gửi gói RTP cho client bằng UDP mỗi 0.05s

2 Function Description

2.1 Client.py

- Class variable:
 - + INIT lưu giá trị 0, READY lưu giá trị 1, PLAYING lưu giá trị 2
 - + state dùng để chỉ ra trạng thái hiện tại của class Client, trong quá trình sử dụng phương thức sẽ gán 3 giá trị INIT, READY, PLAYING
 - + SETUP lưu giá trị 0, PLAY lưu giá trị 1, PAUSE lưu giá trị 2, TEARDOWN lưu giá trị 3.
 - + checkSocketIsOpen (giá trị khởi tạo là False) dùng để kiểm tra Rtp socket đã được mở chưa.
 - + checkPlay (giá trị khởi tạo là False) dùng để kiểm tra video đã đang chạy hay không.
 - + counter dùng để đếm số gói dữ liệu thất lạc, có giá trị là 0.
- Phương thức Client._init_(self, master, serveraddr, serverport, rtpport, filename): Phương thức dùng để khởi tạo các Instance variable, kết nối tới server và khởi tạo giao diện người dùng:
 - + Field master: đối tượng tk được truyền vào, nhiệm vụ thiết kế giao diện đồ họa cho người dùng
 - + Bốn field serverAddr, serverPort, rtpPort, fileName tương ứng với bốn tham số truyền vào khi người dùng gõ trong cmd.
 - + Field rtspSeq (giá trị khởi tạo = 0): dùng để đếm số lần chuyển trạng thái khi người dùng thao tác với các nút bấm
 - + Field sessionId (giá trị khởi tạo = 0) là 1 mã định danh riêng được gán cho mỗi máy khách khi kết nối đến máy chủ
 - + Field requestSent (giá trị khởi tạo = -1): lưu giữ các giá trị nguyên ứng với từng thao tác trên nút bấm

- + Field `teardownAked` (giá trị khởi tạo = 0) là cờ đánh dấu việc kết thúc kết nối đến máy chủ và đóng socket
- + Field `frameNbr` (giá trị khởi tạo = 0) hỗ trợ việc đếm số packet bị mất trong quá trình truyền thông tin giữa máy chủ và máy khách, ngoài ra còn giúp cho việc đồng bộ các frame khi truyền dữ liệu.
- + Phương thức này gọi phương thức `createWidgets(self)` dùng để khởi tạo giao diện của người dùng
- + Phương thức này gọi phương thức `master.protocol("WM_DELETE_WINDOW", self.handler)` dùng để xử lý sự kiện khi người dùng bấm vào dấu "X".
- + Phương thức này gọi phương thức `connectToServer(self)` để kết nối tới máy chủ.
- Phương thức `Client.createWidgets(self)`: Phương thức này có nhiệm vụ khởi tạo giao diện đồ họa trong luồng chạy cơ bản của đề bài
 - + Một nút có tên "Setup", khi bấm vào sẽ chạy phương thức `Client.setupMovie()`.
 - + Một nút có tên "Play", khi bấm vào sẽ chạy phương thức `Client.playMovie()`.
 - + Một nút có tên "Pause", khi bấm vào sẽ chạy phương thức `Client.pauseMovie()`.
 - + Một nút có tên "Teardown", khi bấm vào sẽ chạy phương thức `Client.resetMovie()`.
- Phương thức `Client.setupMovie(self)`:
 - + Phương thức sẽ kiểm tra xem miền state hiện tại có giá trị là INIT (có giá trị 0) hoặc đang nhận giá trị khác (READY, PLAYING).
 - + Nếu state hiện tại đang nhận giá trị là READY (có giá trị 1) hoặc PLAYING (có giá trị 2) thì phương thức kết thúc.
 - + Nếu state hiện tại đang nhận giá trị là INIT (có giá trị 0) thì sẽ gọi phương thức `sendRtspRequest(self, requestCode)` với tham số `requestCode = SETUP`.
- Phương thức `Client.resetMovie(self)`:
 - + Phương thức này đầu tiên sẽ kiểm tra cờ `checkPlay`. Nếu `checkPlay` được mở (có giá trị True) thì phương thức sẽ gọi phương thức `sendRtspRequest()` với tham số là `self.SETUP` để tiến hành việc reset chuyển state của Client về INIT và tiến hành xóa hết tất cả các cache được hệ thống lưu trước đó. Sau đó, sẽ tiến hành reset lại những thông số mặc định và tiến hành kết nối lại với server, mở cổng rtp.
 - + Nếu cờ `checkPlay` không được mở (giá trị False) thì phương thức kết thúc.
- Phương thức `Client.pauseMovie(self)`:
 - + Phương thức sẽ kiểm tra xem field state hiện tại có giá trị là PLAYING (có giá trị 2) hoặc đang nhận giá trị khác (INIT, READY).
 - + Nếu state hiện tại đang nhận giá trị là INIT (có giá trị 0) hoặc READY (có giá trị 1) thì phương thức kết thúc.
 - + Nếu state hiện tại đang nhận giá trị là PLAYING (có giá trị 2) thì sẽ gọi phương thức `sendRtspRequest(self, requestCode)` với tham số `requestCode = PAUSE`.
- Phương thức `Client.playMovie(self)`:

- + Phương thức sẽ kiểm tra field state hiện tại có giá trị là READY (có giá trị 1) hoặc đang nhận giá trị khác (INIT, PLAYING).
- + Nếu state hiện tại đang nhận giá trị là INIT (có giá trị 0) hoặc PLAYING (có giá trị 2) thì phương thức kết thúc.
- + Nếu state hiện tại đang nhận giá trị là READY (có giá trị 1) thì sẽ gọi phương thức này sẽ:
 1. Set cờ checkPlay lên True.
 2. Phương thức này gọi `threading.Thread(target=self.listenRtp).start()` để tạo ra một thread chạy (ở phương thức `listenRtp(self)`) song song với thread chính hiện tại.
 3. Tạo ra một cờ `playEvent` (với giá trị khởi tạo là 0) có nhiệm vụ để kết thúc các luồng chạy song song với luồng chính hiện tại (khi `playEvent` được set 1).
 4. Phương thức này sẽ gọi phương thức `sendRtspRequest(self, requestCode)` với tham số `requestCode = PLAY`.
- Phương thức `Client.listenRtp(self)`:
 - + Phương thức này dùng để khởi tạo đối tượng `RtpPacket`, đọc và giải mã các gói dữ liệu nhận vào với bộ đệm được cấp là 20 byte.
 - + In ra terminal thứ tự của chuỗi các gói dữ liệu đã được nhận.
 - + Nếu phát hiện bộ đếm gói dữ liệu có giá trị đếm khác số thứ tự của gói dữ liệu đó, in ra terminal báo hiệu bị mất gói và lưu số gói đã bị mất, tiếp tục đọc các gói kế đó.
 - + Nếu phát hiện bộ đếm gói dữ liệu nhận vào lớn hơn số thứ tự của gói dữ liệu vừa nhận (gói đến trễ) thì bỏ qua gói đó.
 - + Khi nhận tín hiệu PAUSE thì ngừng theo dõi gói.
 - + Khi nhận tín hiệu TEARDOWN thì sẽ tiến hành set cờ `checkSocketIsOpen` thành False, ngừng theo dõi gói và hủy cổng kết nối.
- Phương thức `Client.updateMovie(self, imageFile)`:
 - + Phương thức này nhận tham số `imageFile` là một chuỗi tên của tập tin .jpg (là một khung video).
 - + Sau đó sẽ mở tập tin lên và cập nhật khung hình lên giao diện đồ họa.
- Phương thức `Client.writeFrame(self, data)`: phương thức này đọc dữ liệu nhận được của gói vừa nhận đã được giải mã rồi lưu vào một tập tin tạm tên "cache-x.jpg" với x chính là số thứ tự của gói.
- Phương thức `Client.connectToServer(self)`:
 - + Phương thức này tạo ra field `rtspSocket` và gán với một đối tượng socket có internet protocol là IPv4 và có giao thức TCP. Sau đó socket kết nối tới `serverAddr` của server và port của server. Nếu kết nối không thành công thì sẽ hiện ra một biểu mẫu báo lỗi.
 - + Sau khi kết nối thành công, sẽ mở tập tin lên và cập nhật khung hình lên giao diện đồ họa.
- Phương thức `Client.sendRtspRequest(self)`:



- + Phương thức này có 5 trường hợp để xử lý yêu cầu của Client gửi tới Server tương ứng với 5 mã SETUP, PLAY, PAUSE, TEARDOWN và mã không hợp lệ.
- + Đối với yêu cầu SETUP từ Client tới Server thì phương thức sẽ tạo ra một thread (ở phương thức `recvRtspReply(self)`) song song với thread hiện đang chạy.
- + Phương thức sẽ gửi thông điệp tới Server qua socket.
- Phương thức `Client.recvRtspReply(self)`:
 - + Phương thức này liên tục nhận dữ liệu trả về từ máy chủ là các gói dữ liệu với độ rộng tối đa 1 byte và gọi phương thức `parseRtspReply(self, data)` để xử lý thông điệp.
 - + Điều kiện dừng của phương thức này là khi thông điệp gửi là TEARDOWN.
- Phương thức `Client.parseRtspReply(self, data)`:
 - + Phương thức này sẽ kiểm tra và xử lý các gói dữ liệu trả về từ máy chủ và thực hiện lệnh tùy vào yêu cầu phía Client đã gửi cho Server là SETUP, PLAY, PAUSE, TEARDOWN.
 - + Đối với trường hợp `requestSent` bởi Client là SETUP thì phương thức này sẽ chuyển state của Client sang SETUP và gọi phương thức `openRtpPort()`.
 - + Đối với trường hợp `requestSent` bởi Client là PLAY thì phương thức này sẽ chuyển state của Client sang PLAYING.
 - + Đối với trường hợp `requestSent` bởi Client là PAUSE thì phương thức này sẽ chuyển state của Client sang READY, rồi set cờ `playEvent` lên 1 để tiến hành hủy các thread đang chạy song song với thread chạy chính.
 - + Đối với trường hợp `requestSent` bởi Client là TEARDOWN thì phương thức này sẽ chuyển state của Client sang INIT, set giá trị của field `teardownAked` sang 1 để đóng socket.
- Phương thức `Client.openRtspPort(self)`: phương thức này đầu tiên sẽ set cờ `checkSocketIsOpen` thành True và tạo ra field `rtpSocket` và gán với một đối tượng socket có internet protocol là IPv4 có giao thức UDP và timeout là 0.5s. Sau đó socket sẽ kết nối đến IP của Client và port của Client. Nếu kết nối không thành công thì sẽ hiện lên một biểu mẫu thông báo kết nối thất bại cho Client.
- Phương thức `Client.handler(self)`:
 - + Phương thức này xử lý trường hợp người dùng bấm vào dấu "X" với mong muốn kết thúc chương trình.
 - + Tạo một biểu mẫu hỏi người dùng muốn thoát hay không, nếu có thì kết thúc chương trình.

2.2 Client2.py

- Class variable:
 - + INIT lưu giá trị 0, READY lưu giá trị 1, PLAYING lưu giá trị 2
 - + state dùng để chỉ ra trạng thái hiện tại của class Client, trong quá trình sử dụng phương thức sẽ gán 3 giá trị INIT, READY, PLAYING



- + SETUP lưu giá trị 0, PLAY lưu giá trị 1, PAUSE lưu giá trị 2, TEARDOWN lưu giá trị 3, DESCRIBE lưu giá trị 4, FORWARD lưu giá trị 5, PREV lưu giá trị 6.
 - + checkSocketIsOpen (giá trị khởi tạo là False) dùng để kiểm tra Rtp socket đã được mở chưa.
 - + checkPlay (giá trị khởi tạo là False) dùng để kiểm tra video đã đang chạy hay không.
 - + isFirstPlay (có giá trị khởi tạo là True) là cờ dùng để setup video trong lần chạy đầu tiên.
 - + counter dùng để đếm số gói dữ liệu thất lạc, có giá trị là 0.
- Phương thức Client2._init_(self, master, serveraddr, serverport, rtpport, filename): Phương thức dùng để khởi tạo các Instance variable, kết nối tới server và khởi tạo giao diện người dùng.
- + Field master: đối tượng tk được truyền vào, nhiệm vụ thiết kế giao diện đồ họa cho người dùng
 - + Bốn field serverAddr, serverPort, rtpPort, fileName tương ứng với bốn tham số truyền vào khi người dùng gõ trong cmd.
 - + Field rtspSeq (giá trị khởi tạo = 0): đếm số lần chuyển trạng thái khi người dùng thao tác với các nút bấm
 - + Field sessionID (giá trị khởi tạo = 0): là 1 mã định danh riêng được gán cho mỗi máy khách khi kết nối đến máy chủ
 - + Field requestSent (giá trị khởi tạo = -1): lưu giữ các giá trị nguyên ứng với từng thao tác trên nút bấm
 - + Field teardownAked (giá trị khởi tạo = 0): là cờ đánh dấu việc kết thúc kết nối đến máy chủ và đóng socket
 - + Field frameNbr (giá trị khởi tạo = 0): hỗ trợ việc đếm số packet bị mất trong quá trình truyền thông tin giữa máy chủ và máy khách, ngoài ra còn giúp cho việc đồng bộ các frame khi truyền dữ liệu
 - + Phương thức này sẽ gọi phương thức createWidgets() dùng để khởi tạo giao diện của người dùng.
 - + Phương thức này sẽ gọi phương thức master.protocol("WM_DELETE_WINDOW", self.handler) dùng để xử lý sự kiện khi người dùng bấm vào dấu "X".
 - + Field currentTime (giá trị khởi tạo = 0): dùng để hỗ trợ việc hiển thị Remaining time label trong GUI.
 - + Field totalTime dùng để hiển thị tổng số thời gian để video chạy.
 - + Field isForward (giá trị khởi tạo = 0): là cờ dùng để kiểm tra sự kiện người dùng tua video tới.
 - + Field isBackward (giá trị khởi tạo = 0): là cờ dùng để kiểm tra sự kiện người dùng tua video lùi.
 - + Các Field countTotalPacket, timerBegin, timerEnd, timer, bytes, packets, packetsLost, lastSequence, totalJitter, arrivalTimeofPreviousPacket, lastPacketSpacing được dùng để tiến hành thống kê dữ liệu đường truyền và được khởi tạo giá trị mặc định bằng 0.
- Phương thức Client2.createWidgets(self): Phương thức này có nhiệm vụ khởi tạo giao diện đồ họa trong luồng chạy cơ bản của đề bài

- + Một nút có tên "Play", khi bấm vào sẽ chạy phương thức Client2.playMovie().
- + Một nút có tên "Pause", khi bấm vào sẽ chạy phương thức Client2.pauseMovie().
- + Một nút có tên "Stop", khi bấm vào sẽ chạy phương thức Client2.resetMovie().
- + Một nút có tên "Describe", khi bấm vào sẽ chạy phương thức Client2.describeMovie(), tuy nhiên do trong giai đoạn mới vừa khởi tạo lên GUI thì nút này sẽ bị disable vì chưa có dữ liệu về video để gửi về cho người dùng.
- + Một nút có tên "«", khi bấm vào sẽ chạy phương thức Client2.forwardMovies(), tuy nhiên do trong giai đoạn mới vừa khởi tạo lên GUI thì nút này sẽ bị disable vì chưa có dữ liệu về video để người dùng tua tới.
- + Một nút có tên "»", khi bấm vào sẽ chạy phương thức Client2.prevMovie(), tuy nhiên do trong giai đoạn mới vừa khởi tạo lên GUI thì nút này sẽ bị disable vì chưa có dữ liệu về video để người dùng tua về.
- + Một label có tên "Remaining time", hiển thị thời gian còn lại của video khi đang chạy.
- + Một label có tên "Total time", hiển thị tổng thời gian của video.
- Phương thức Client2.describeMovie(self): phương thức này sẽ gọi phương thức sendRtspRequest(self, requestCode) với tham số requestCode = DESCRIBE.
- Phương thức Client2.setupMovie(self):
 - + Phương thức sẽ kiểm tra xem miền state hiện tại có giá trị là INIT (0) hoặc đang nhận giá trị khác (READY, PLAYING).
 - + Nếu state hiện tại đang nhận giá trị là READY (1) hoặc PLAYING (2) thì phương thức kết thúc.
 - + Nếu state hiện tại đang nhận giá trị là INIT (0) thì sẽ gọi phương thức sendRtspRequest(self, requestCode) với tham số requestCode = SETUP.
- Phương thức Client2.resetMovie(self):
 - + Phương thức này đầu tiên sẽ kiểm tra cờ checkPlay. Nếu checkPlay được mở (có giá trị True) thì phương thức sẽ gọi phương thức sendRtspRequest(requestCode) với tham số là SETUP để tiến hành việc reset chuyển state của Client về INIT và tiến hành xóa hết tất cả các cache được hệ thống lưu trước đó. Sau đó, sẽ tiến hành reset lại những thông số mặc định, chuyển state của Client lại thành INIT và tiến hành kết nối lại với server, khởi tạo đối tượng socket (IPv4 với giao thức UDP).
 - + Nếu cờ checkPlay không được mở (giá trị False) thì phương thức kết thúc.
- Phương thức Client2.pauseMovie(self):
 - + Đầu tiên, kiểm tra state hiện tại đang có phải là PLAYING hay không. Nếu là PLAYING thì sẽ tiến hành disable 2 nút "»" (FORWARD) và "«" (BACKWARD) và gọi phương thức sendRtspRequest(self, requestCode) với requestCode nhận giá trị PAUSE.
 - + Nếu state hiện tại không phải là PLAYING thì phương thức kết thúc.
- Phương thức Client2.playMovie(self):
 - + Phương thức này đầu tiên sẽ setupMovie nếu như Client chưa gửi yêu cầu setup tới server, set cờ isFirstPlay thành False, cờ checkPlay thành True, frameNbr thành 0 và set những field thống kê thành 0.

- + Phương thức sẽ enable 2 nút "»" (FORWARD), "«" (BACKWARD)
- + Sau khi setup thành công, phương thức kiểm tra xem miền state hiện tại có giá trị là READY (có giá trị 1) hoặc đang nhận giá trị khác (INIT, PLAYING).
- + Nếu state hiện tại đang nhận giá trị là INIT (có giá trị 0) hoặc PLAYING (có giá trị 2) thì phương thức kết thúc.
- + Nếu state hiện tại đang nhận giá trị là READY (có giá trị 1) thì sẽ gọi phương thức này sẽ:
 1. Phương thức này sẽ gọi `threading.Thread(target=self.listenRtp).start()` để tạo ra một luồng chạy (ở phương thức `listenRtp()`) song song với luồng chính hiện tại.
 2. Tạo ra một cờ `playEvent` (với giá trị khởi tạo là 0) dùng để kết thúc các luồng chạy song song với luồng chính hiện tại (khi `playEvent` được set 1).
 3. Phương thức này sẽ gọi phương thức `sendRtspRequest(self, requestCode)` với tham số `requestCode = PLAY`.
- Phương thức `Client2.forwardMovies(self)`: phương thức này sẽ gọi phương thức `sendRtspRequest(self, requestCode)` với `requestCode` nhận giá trị FORWARD và sau đó set cờ `isForward` lên 1.
- Phương thức `Client2.prevMovie(self)`: phương thức này sẽ gọi phương thức `sendRtspRequest(self, requestCode)` với `requestCode` nhận giá trị BACKWARD, set cờ `isBackward` lên 1 và sau đó tiến hành chỉnh sửa lại field `frameNbr`.
- Phương thức `Client2.listenRtp(self)`:
 - + Phương thức này dùng để khởi tạo đối tượng `RtpPacket`, đọc và giải mã các gói dữ liệu nhận vào với bộ đệm được cấp là 20 byte.
 - + In ra terminal thứ tự của chuỗi các gói dữ liệu đã được nhận.
 - + Nếu phát hiện bộ đếm gói dữ liệu có giá trị đếm khác số thứ tự của gói dữ liệu đó, in ra terminal báo hiệu bị mất gói và lưu số gói đã bị mất, tiếp tục đọc các gói kế đó.
 - + Nếu phát hiện bộ đếm gói dữ liệu nhận vào lớn hơn số thứ tự của gói dữ liệu vừa nhận (gói đến trễ) thì bỏ qua gói đó.
 - + Đồng thời, phương thức này sẽ tiến hành thống kê các giá trị về đường truyền từ server về client.
 - + Khi nhận tín hiệu PAUSE thì ngừng theo dõi gói và hiện ra biểu mẫu cho phép người dùng thấy được các giá trị thống kê.
 - + Khi nhận tín hiệu TEARDOWN thì sẽ tiến hành set cờ `checkSocketIsOpen` thành False, ngừng theo dõi gói, hủy cổng kết nối và hiện ra biểu mẫu cho phép người dùng thấy được các giá trị thống kê.
- Phương thức `Client2.updateMovie(self, imageFile)`:
 - + Phương thức này nhận tham số `imageFile` là một chuỗi tên của tập tin .jpg (là một khung video).
 - + Sau đó sẽ mở tập tin lên và cập nhật khung hình lên giao diện đồ họa.
- Phương thức `Client2.writeFrame(self, data)`: phương thức này đọc dữ liệu nhận được của gói vừa nhận đã được giải mã rồi lưu vào một tập tin tạm tên "cache-x.jpg" với x chính là số thứ tự của gói

- Phương thức `Client2.connectToServer(self)`:
 - + Phương thức này đầu tiên sẽ set cờ `checkSocketIsOpen` thành `True` và sau đó tạo ra field `rtspSocket`, gán với một đối tượng socket có internet protocol là `IPv4` và có giao thức `TCP`. Sau đó socket kết nối tới `serverAddr` của server và `port` của server. Nếu kết nối không thành công thì sẽ hiện ra một biểu mẫu báo lỗi.
 - + Sau khi kết nối thành công, sẽ mở tập tin lên và cập nhật khung hình lên giao diện đồ họa.
- Phương thức `Client2.sendRtspRequest(self)`:
 - + Phương thức này có 8 trường hợp để xử lý yêu cầu của Client gửi tới Server tương ứng với 5 mã `SETUP`, `PLAY`, `PAUSE`, `TEARDOWN`, `DESCRIBE`, `FORWARD`, `PREV` và mã không hợp lệ.
 - + Đối với yêu cầu `SETUP` từ Client tới Server thì phương thức sẽ tạo ra một thread (ở phương thức `recvRtspReply(self)`) song song với thread hiện đang chạy.
 - + Đối với yêu cầu `DESCRIBE` thì sẽ tạo ra một biểu mẫu hiển thị các thông tin về video cho người dùng.
 - + Phương thức sẽ gửi thông điệp tới Server qua socket.
- Phương thức `Client2.recvRtspReply(self)`:
 - + Phương thức này liên tục nhận dữ liệu trả về từ máy chủ là các gói dữ liệu với độ rộng tối đa 1 byte và gọi phương thức `parseRtspReply(self, data)` để xử lý thông điệp.
 - + Điều kiện dừng của phương thức này là khi thông điệp gửi là `TEARDOWN`.
- Phương thức `Client2.parseRtspReply(self, data)`:
 - + Phương thức này sẽ kiểm tra và xử lý các gói dữ liệu trả về từ máy chủ và thực hiện lệnh tùy vào yêu cầu phía Client đã gửi cho Server là `DESCRIBE`, `SETUP`, `PLAY`, `PAUSE`, `TEARDOWN`.
 - + Đối với trường hợp `requestSent` bởi Client là `SETUP` thì phương thức này sẽ chuyển state của Client sang `SETUP` và gọi phương thức `openRtpPort(self)`.
 - + Đối với trường hợp `requestSent` bởi Client là `PLAY` thì phương thức này sẽ chuyển state của Client sang `PLAYING` và tiến hành lưu thời gian lần đầu tiên video chạy.
 - + Đối với trường hợp `requestSent` bởi Client là `PAUSE` thì phương thức này sẽ chuyển state của Client sang `READY`, lưu thời điểm video dừng lại và set cờ `playEvent` lên 1 để tiến hành hủy các thread đang chạy song song với thread chạy chính.
 - + Đối với trường hợp `requestSent` bởi Client là `TEARDOWN` thì phương thức này sẽ chuyển state của Client sang `INIT`, lưu thời gian hủy chạy video và set giá trị của field `teardownAacked` sang 1 để đóng socket.
 - + Đối với trường hợp `requestSent` bởi Client là `DESCRIBE` thì phương thức này sẽ gọi phương thức `displayDescription(self, lines)` với `lines` là các thông tin Client muốn.
- Phương thức `Client2.openRtpPort(self)`:
 - + Phương thức này sẽ tạo ra field `rtpSocket` và gán với một đối tượng socket có internet protocol là `IPv4` có giao thức `UDP` và `timeout` là `0.5s`. Sau đó socket sẽ kết nối đến IP của Client và `port` của Client. Nếu kết nối không thành công thì sẽ hiện lên một biểu mẫu thông báo kết nối thất bại cho Client.



- Phương thức `Client2.handler(self)`:
 - + Phương thức này xử lý trường hợp người dùng bấm vào dấu "X" với mong muốn kết thúc chương trình.
 - + Tạo một biểu mẫu hỏi người dùng muốn thoát hay không, nếu có thì kết thúc chương trình.
- Phương thức `Client2.displayDescription(self, lines)`: phương thức này sẽ hiện ra một biểu mẫu gồm những thông tin về streams cũng như những phương thức encoding đang sử dụng
- Phương thức `Client2.displayStats(self)`: phương thức này sẽ hiện ra một biểu mẫu gồm những thông số thống kê trong quá trình Streaming video từ server tới phía Client.

2.3 Server.py

Phương thức `Server.main(self)`:

- Phương thức tạo 1 biến `SERVER_PORT` là port của máy chủ được truyền vào từ cmd.
- Phương thức sẽ khởi tạo 1 socket và gán socket này với port vừa truyền vào.
- Sau đó, phương thức liên tục nhận thông điệp đến từ các nguồn khác nhau (tối đa 5).

2.4 RtpPacket.py

- Global Variable: `HEADER_SIZE` (giá trị 12): quy định số lượng phần tử của mảng header trong class `RtpPacket`.
- Class Variable: `header`: một mảng có 12 byte chứa siêu dữ liệu (thông tin của 1 gói RTP).
- Phương thức `RtpPacket.encode(self, version, padding, extension, cc, seqnum, marker, pt, ssrc, payload)`: Phương thức dùng để mã hóa các gói rtp.
 - + Tham số `version` (2 bit) là phiên bản của giao thức.
 - + Tham số `padding` (1 bit) là tín hiệu báo trong gói sẽ có 1 (0) hoặc nhiều padding trong gói (1), thường dùng để lấp các khoảng trống trong một khối dữ liệu.
 - + Tham số `extension` (1 bit) là tín hiệu báo có header mở rộng (1) hay không (0).
 - + Tham số `cc` (CSRC count) (4 bit) chứa các giá trị của trường CSRC ID trong header cố định.
 - + Tham số `marker` (1 bit) dùng ở lớp ứng dụng, xác định một profile.
 - + Tham số `pt` (payload type) (chiếm 7 bit) xác định và nêu ý nghĩa các dạng payload của RTP.
 - + Tham số `seqnum` (chiếm 16 bit) mang số thứ tự của gói RTP.
 - + Tham số `timestamp` (chiếm 32 bit) xác định thời điểm lấy mẫu của byte đầu tiên trong gói.
 - + Tham số `ssrc` (synchronization source identifier) (chiếm 32 bit) chỉ ra nguồn đồng bộ (nguồn phát gói tin RTP từ micro, camera hay bộ ngẫu phối RTP) của gói tin.
- Phương thức `RtpPacket.decode(self, byteStream)`: Phương thức dùng để giải mã lấy header và payload của gói rtp.

- Phương thức `RtpPacket.version(self)`: Phương thức để trả về phiên bản của giao thức.
- Phương thức `RtpPacket.seqNum(self)`: Phương thức dùng để trả về số thứ tự của gói.
- Phương thức `RtpPacket.timestamp(self)`: Phương thức để trả về mốc thời gian của gói.
- Phương thức `RtpPacket.payloadType(self)`: Phương thức để trả về định dạng của gói.
- Phương thức `RtpPacket.getPayload(self)`: Phương thức dùng để lấy nội dung của gói.
- Phương thức `RtpPacket.getPacket(self)`: : Phương thức này dùng để trả về header và payload của gói.

2.5 ServerWorker.py

Class có nhiệm vụ quản lý các công việc phía máy chủ bao gồm tạo socket kết nối, truyền/nhận và mã hóa/giải mã dữ liệu với máy khách.

- Class variables:
 1. SETUP (giá trị= 'SETUP'): là mã xử lý request SETUP từ phía Client.
 2. PLAY (giá trị= 'PLAY'): là mã dùng để xử lý request PLAY từ phía Client.
 3. PAUSE (giá trị= 'PAUSE'): là mã xử lý request PAUSE từ phía Client.
 4. TEARDOWN (giá trị= 'TEARDOWN'): là mã dùng để xử lý request TEARDOWN từ phía Client.
 5. FORWARD (giá trị= 'FORWARD'): là mã dùng để xử lý request FORWARD từ phía Client
 6. PREV (giá trị= 'PREV'): là mã dùng để xử lý request PREV từ phía Client.
 7. INIT (giá trị= 0): là mã thể hiện trạng thái khởi tạo.
 8. READY (giá trị= 1): là mã thể hiện trạng thái sẵn sàng.
 9. PLAYING (giá trị= 2): là mã thể hiện trạng thái đang chạy.
 10. state (giá trị khởi tạo là INIT): là field lưu trạng thái của ServerWorker.
 11. OK_200 (giá trị= 0): là mã dùng để hiển thị giá trị OK gửi về từ phía Server đến Client.
 12. FILE_NOT_FOUND_404 (giá trị= 1): là mã dùng để hiển thị giá trị không tìm thấy gửi về từ phía Server tới Client.
 13. CON_ERR_500 (giá trị= 2): là mã hiển thị giá trị máy chủ không thể xử lý request gửi về từ phía Server tới Client.
 14. clientInfo: chứa các info của phía client.
- Phương thức `ServerWorker.__init__(self, clientInfo)`:
 - + Gán cho field `clientInfo` đối tượng `clientInfo` nhận vào.
 - + Khởi tạo cờ `opt` với giá trị 0 dùng để xử lý trường hợp người yêu cầu PLAY, FORWARD, BACKWARD.
- Phương thức `ServerWorker.run(self)`: Tạo ra một thread mới chạy song song với thread chính để quản lý các request RTP/RTSP đối với mỗi máy khách.



- Phương thức `ServerWorker.recvRtspRequest(self)`: Mở socket để truyền/nhận dữ liệu và mã hóa/giải mã trước khi gửi đi.
- Phương thức `ServerWorker.processRtspRequest(self, data)`:
 - + Tách dữ liệu truyền từ máy khách thành các phần tử để nhận dạng yêu cầu, bao gồm kiểu request (requestType), tên file muốn truyền tải (filename), thông tin chuỗi RTSP (seq).
 - + Căn cứ trên kiểu request, máy chủ sẽ có thao tác xử lý khác nhau đối với file được yêu cầu.
 - + Đối với yêu cầu SETUP từ Client thì đầu tiên sẽ tiến hành thiết lập video Client mong muốn xem, trong trường hợp không thể mở được video thì sẽ gọi phương thức `replyRtsp(self, code, seq)` với `code = FILE_NOT_FOUND_404`. Sau khi đã tiến hành thiết lập video thì sẽ tiến hành thiết lập sessionID và gọi phương thức `replyRtsp(self, code, seq)` với `code = OK_200`.
 - + Đối với yêu cầu PLAY từ Client thì server sẽ tiến hành tạo socket (IPV4 và giao thức UDP), sau đó gọi giao thức `replyRtsp(self, code, seq)` với `code = OK_200` và tạo ra một thread chạy song song để gửi các gói tin Rtp, tạo ra cờ `clientInfo['event']` (giá trị khởi tạo 0) để có thể kiểm soát quá trình gửi gói tin Rtp cho phía Client.
 - + Đối với yêu cầu FORWARD từ Client thì sẽ gọi phương thức `forwardStream(self)`.
 - + Đối với yêu cầu BACKWARD từ Client thì sẽ set cờ `opt` lên 1.
 - + Đối với yêu cầu PAUSE từ Client thì sẽ set cờ `clientInfo['event']` lên 1 để tiến hành hủy việc gửi gói tin Rtp tới Client và gọi phương thức `replyRtsp(self, code, seq)` với mã `code = OK_200`.
 - + Đối với yêu cầu TEARDOWN từ Client thì sẽ set cờ `clientInfo['event']` lên 1 để tiến hành hủy việc gửi gói tin Rtp tới Client và gọi phương thức `replyRtsp(self, code, seq)` với mã `code = OK_200`.
 - + Đối với yêu cầu DESCRIBE từ Client thì sẽ gọi phương thức `replyDescribe(self, code, seq, filename)`.
- Phương thức `ServerWorker.sendRtp(self, data)`:
 - + Hàm có chức năng gửi các gói tin RTP thông qua giao thức UDP ở tầng transport. Các gói tin trước khi gửi sẽ có 1 khoảng thời gian chờ, mặc định là 0.05s.
 - + Trường hợp request gửi tới là STOP hoặc TEARDOWN, sẽ dừng việc truyền dữ liệu về video cho máy khách.
 - + Trường hợp request gửi tới là PREV, máy chủ sẽ lấy dữ liệu của frame cách 10% tổng số frame trước frame hiện tại và gán vào biến data và gửi về cho Client.
 - + Nếu request ko nằm trong 2 trường hợp ở trên, máy chủ sẽ lấy dữ liệu từng frame video và gán vào biến data. Trường hợp data có dữ liệu: lấy số frame hiện tại, gửi về máy khách cùng với frame video thông qua hàm `makeRtp`, ngược lại sẽ xuất thông báo lỗi ra màn hình.
- Phương thức `ServerWorker.forwardStream(self)`: Phương thức này sẽ set cờ `isNext` của thực thể `clientInfo['videoStream']` lên 1 để xử lý sự kiện Client gửi yêu cầu FORWARD.
- Phương thức `ServerWorker.makeRtp(self, payload, frameNbr)`: Có nhiệm vụ nén và chuyển đổi định dạng dữ liệu nhận từ biến data trong hàm `sendRtp`, các thông số đi kèm khác thành các gói tin.

- Phương thức `ServerWorker.replyRtsp(self, code, seq)`: Hàm có chức năng gửi trả gói tin đến máy khách dựa vào mã code được truyền vào hoặc báo lỗi nếu kết nối không thành công hoặc không tìm thấy file.
- Phương thức `ServerWorker.replyDescribe(self, code, seq, filename)`: Tiến hành kiểm tra tham số code nhận vào. Nếu code = OK_200 thì sẽ tiến hành gửi các thông tin về DESCRIBE mà Client yêu cầu qua Socket cho phía Client.

2.6 VideoStream.py

- Phương thức `VideoStream.__init__(self, filename)`:
 - + Phương thức này nhận một tham số là tên tập tin filename cần truyền.
 - + Phương thức này cũng khởi tạo các field:
 1. `frameNum` (khởi tạo với giá trị 0) lưu số thứ tự của khung hình hiện tại.
 2. `filename` lưu tên của tập tin cần truyền.
 3. `isNext` là cờ dùng để xử lý cho trường hợp Client gửi yêu cầu FORWARD.
 4. `totalFrame` là tổng số frame của video Client yêu cầu xem, field này được gán giá trị bằng cách phương thức `__init__()` gọi phương thức `get_total_time_video()`.
 - + Phương thức này sẽ mở tập tin có tên như filename để đọc.
- Phương thức `VideoStream.get_total_time_video(self)`: phương thức này sẽ trả về tổng số thời gian tính theo frame và đồng thời gán giá trị cho field `totalFrame` bằng tổng số frame của video.
- Phương thức `VideoStream.setIsNext(self)`: Phương thức này dùng để set cờ `isNext` lên 1.
- Phương thức `VideoStream.nextFrame(self)`:
 - + Phương thức này đầu tiên sẽ kiểm tra cờ `isNext`.
 - + Nếu cờ `isNext` là 1 tức là Client đang yêu cầu tua lên video nên phương thức này sẽ trả về frame cách sau frame hiện tại 10% tổng số frame, set cờ `isNext` bằng 0, cộng tích lũy field `frameNum` thêm 10% tổng số frame đơn vị. Trong trường hợp số frame yêu cầu tua vượt quá `totalFrame` thì số frame tua sẽ bằng `totalFrame - frameNum`.
 - + Nếu cờ `isNext` là 0 thì phương thức này sẽ trả về frame tiếp theo của frame hiện tại và cộng tích lũy field `frameNum` thêm 1 đơn vị.
- Phương thức `VideoStream.prevFrame(self)`:
 - + Phương thức này trả về frame trước frame hiện tại 10% tổng số frame, lùi `frameNum` lại 10% tổng số frame.
 - + Trong trường hợp việc tua về dẫn đến đọc phải frame không tồn tại (đứng trước frame đầu tiên) thì sẽ trả về frame đầu tiên của video và gán `frameNum = 0`.
- Phương thức `VideoStream.frameNbr(self)`: Phương thức này trả về số thứ tự của frame vừa lấy.



2.7 ClientLauncher.py

Phương thức `__main__` :

- Nhận 4 tham số từ người dùng trong cmd:
 1. Địa chỉ IP của máy chủ
 2. Mã port với máy chủ của "máy khách" (Ở đây là lớp Client được khởi tạo)
 3. Mã port các gói dữ liệu qua giao thức vận chuyển thời gian thực (Real-time Protocol)
 4. Tên của tập tin cần đọc.
- Hàm có nhiệm vụ khởi tạo đối tượng tk (giúp điều khiển, thiết kế giao diện winform) vào thực thể Client được khởi tạo và tạo vòng lặp vô tận cho biểu mẫu Windows (winform)

3 Class Diagram

Link ảnh Class Diagram:

<https://drive.google.com/drive/folders/1t0JqysIcfrEmkvy2Xx4lW3DlXeEHvVCX?usp=sharing>

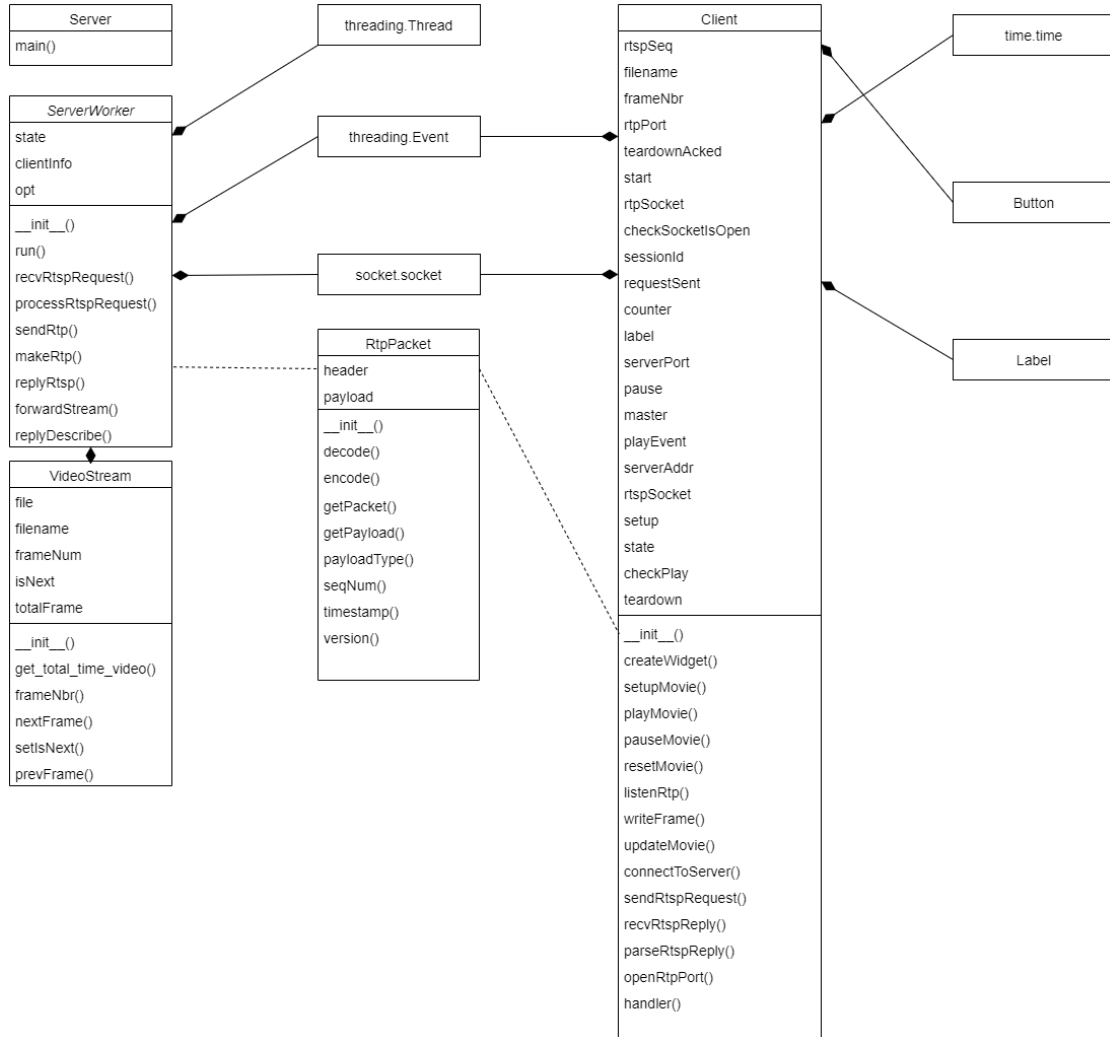


Figure 1: Class Diagram

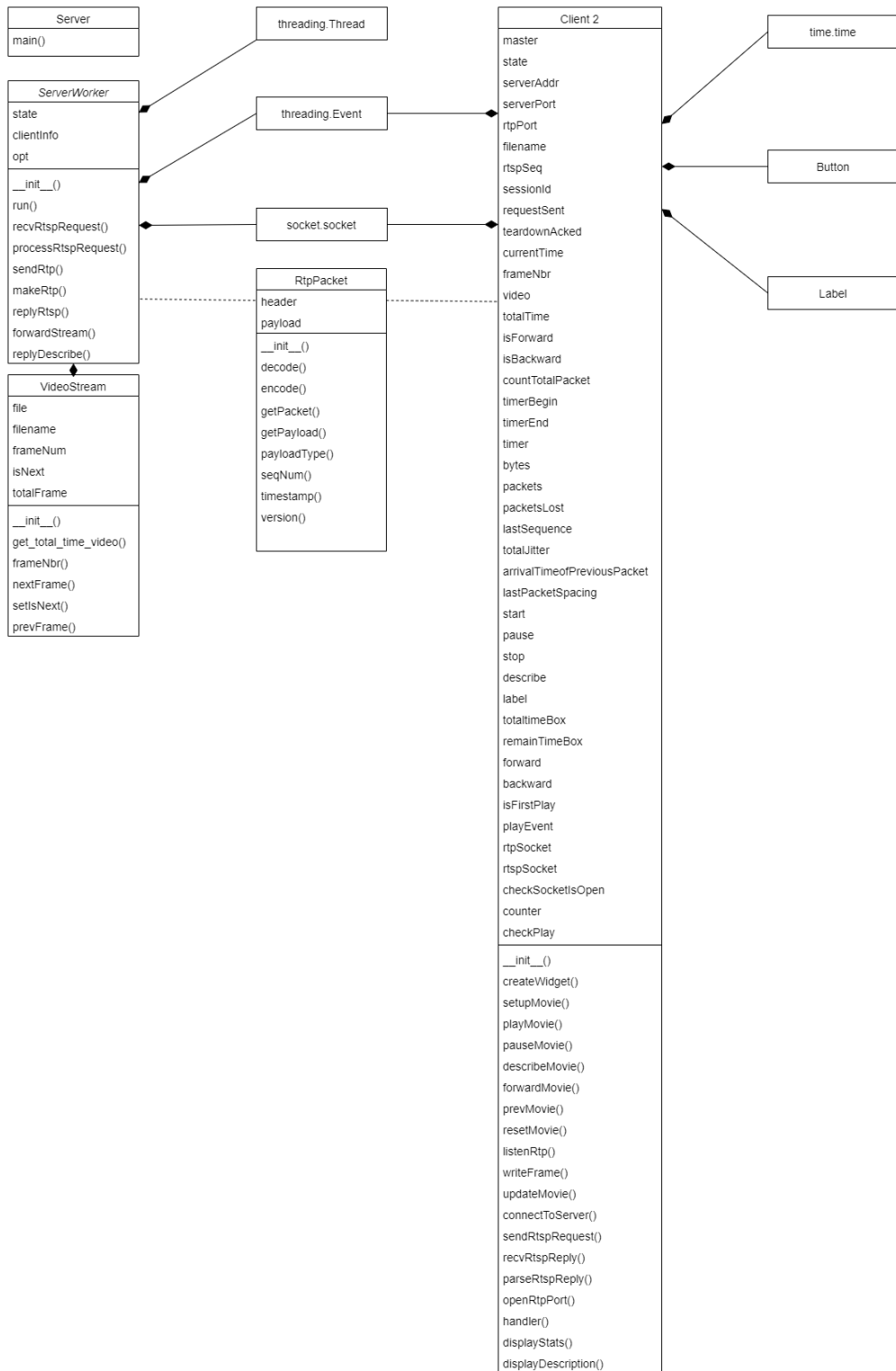


Figure 2: Class Diagram cho Extend

4 A summative Evaluation of Results Achieved

Thông qua Bài tập lớn "Video Streaming Application" này, nhóm đã có thể:

- Thiết kế được GUI cho người dùng để có thể xem được video thông qua thư viện tkinter của Python. Đối với normal flow thì nhóm đã có thể thiết kế được một GUI cơ bản với 4 nút "SETUP", "PLAY", "PAUSE", "TEARDOWN" video theo yêu cầu của bài tập lớn. Đối với extend flow thì nhóm đã thiết kế lại GUI gồm 6 nút để người dùng có thể tương tác: "PLAY", "PAUSE", "STOP", "DESCRIBE", "FORWARD", "BACKWARD" và 2 label "Total time" dùng để hiển thị thời gian tổng của video, "Remaining time" thể hiện số thời gian còn cho tới khi video kết thúc và đồng thời thiết kế màu sắc cũng như giao diện để cho GUI trở nên bắt mắt hơn.



Figure 3: Giao diện cơ bản của người dùng ở normal flow

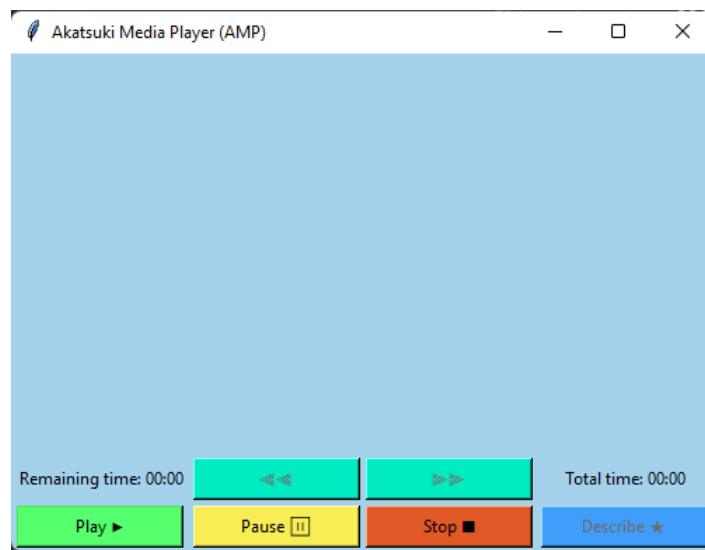


Figure 4: Giao diện của người dùng ở extend flow

- Thiết kế được một mô hình client-server, với server online ở 1 port cố định (lớn hơn 1024) và được client kết nối để xem video trên 1 port bất kì của clien. Nhóm đã có thể tự biết cách nghiên cứu SPEC của các protocol thông qua các tài liệu RFC, như trong bài tập lớn này là RFC 2326 (RTSP), RFC 1889 (RTP) để biết cú pháp gửi những thông tin cho RTSP Socket vì phải truyền đúng cú pháp thì bên máy khách mới biết dùng protocol gì để giải mã video, chẳng hạn như bài này phần giải mã chúng ta đã mặc định header UDP là 12 bytes, nên chúng ta chỉ cần lấy payload từ byte thứ 12 trở đi và xử lí video là có thể xem được rồi, nhưng với xu hướng phát triển vũ bão của công nghệ sự phát triển của hàng triệu protocol, nếu chúng ta không có thông tin về protocol thì sẽ không biết cần lấy từ byte thứ bao nhiêu.

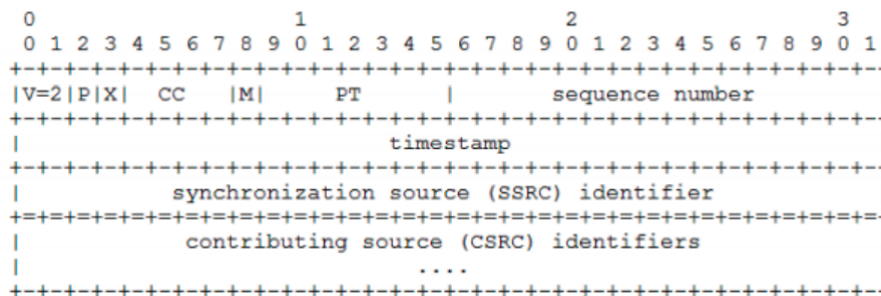


Figure 5: RTP packet header

- Biết về các trường của RTP packet rất cần thiết, sự quan trọng của sequence number khi làm extend 1 để tính packet loss rate và lenght của payload để tính video data rate. Hoặc trong extend 4 để có thể xử lý được việc tua video tới hoặc tua video về.

- Nhóm cũng biết cách dịch bit để lấy thông tin từ data của header, ngoài ra còn biết các trạng thái của 1 video (INIT,READY và PLAYING) tùy với trạng thái mà hiện thực những yêu

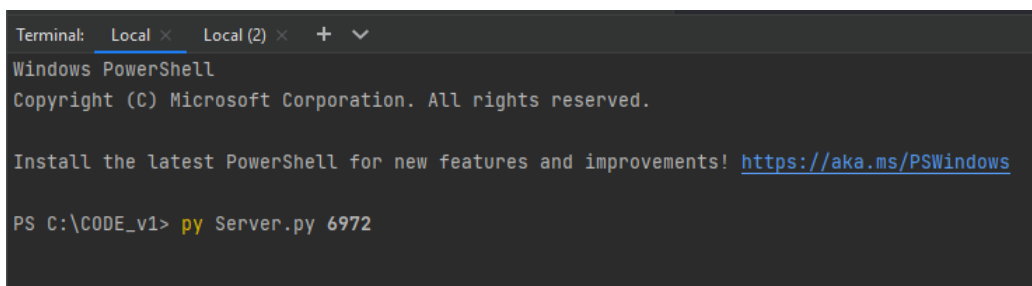
cầu khác nhau.

5 User Manual

5.1 Chạy chương trình

Tại folder chứa source code. Mở 2 terminal

- Terminal thứ nhất chạy lệnh: `py Server.py <server-port>`. Trong đó:
 - + `server-port`: là port mà server của bạn nghe các kết nối RTSP đến.
 - + Port RTSP tiêu chuẩn là 554, nhưng ta phải chọn port > 1024.VD: `py Server.py 6969`

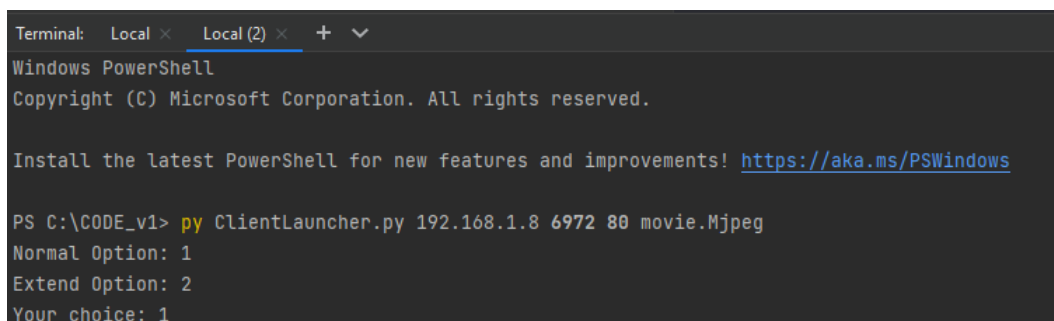


```
Terminal: Local x Local (2) x + v
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\CODE_v1> py Server.py 6972
```

- Terminal thứ hai chạy lệnh: `py ClientLauncher.py <server-host> <server-port> <RTP-port> <video-file>`. Trong đó:
 - + `server-host`: Địa chỉ IP của server (localhost là 127.0.0.1)
 - + `server-port`: port server đang nghe (như ở ví dụ `py Server.py 6969` thì ở đây `server-port` là 6969)
 - + `RTP-port`: port nơi nhận các gói RTP (Có thể chọn một số nguyên dương ngẫu nhiên)
 - + `video-file`: Tên của tệp video bạn muốn yêu cầu stream (ở đây là `movie.Mjpeg`)



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

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\CODE_v1> py ClientLauncher.py 192.168.1.8 6972 80 movie.Mjpeg
Normal Option: 1
Extend Option: 2
Your choice: 1
```

5.2 Cách sử dụng sau khi chạy chương trình

- Sau khi lần lượt thực thi 2 lệnh trên. Cửa sổ giao diện sẽ hiện lên màn hình như dưới:



Figure 6: Giao diện của người dùng

- Sau đó click vào button Setup để gửi yêu cầu SETUP tới máy chủ.
- Bấm vào Play để chạy chương trình

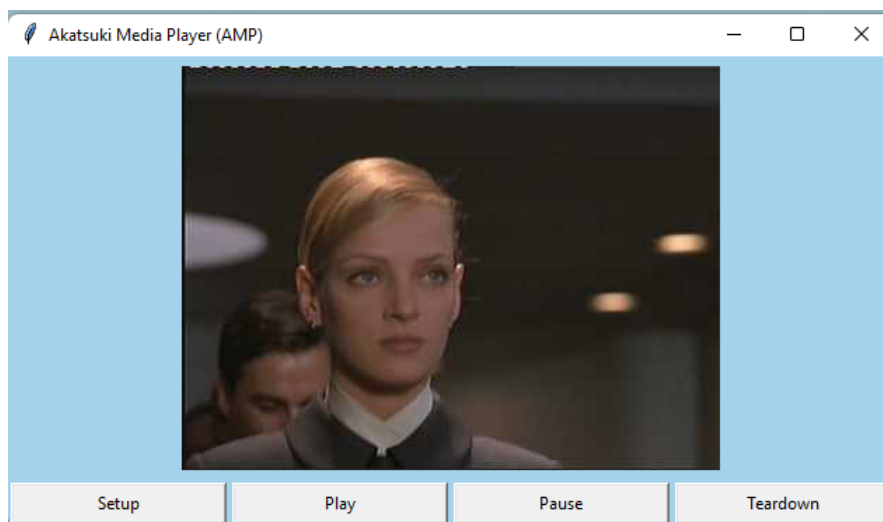


Figure 7: Giao diện của người dùng khi nhấn nút Play

- Khi chương trình đang chạy, nếu muốn tạm dừng thì bấm vào nút Pause

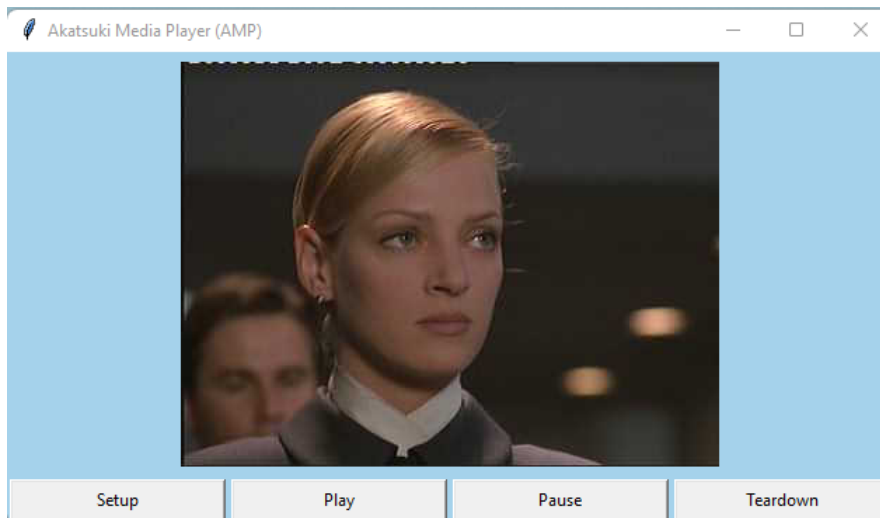


Figure 8: Giao diện của người dùng khi nhấn nút Pause

- Bấm vào Teardown thì chương trình sẽ dừng video và quay về trạng thái ban đầu chưa được SETUP
- Để thoát khỏi chương trình và đóng cửa sổ giao diện người dùng, ta bấm vào dấu X ở góc trên phải màn hình. Bấm OK để thoát, bấm Cancel để hủy bỏ yêu cầu.

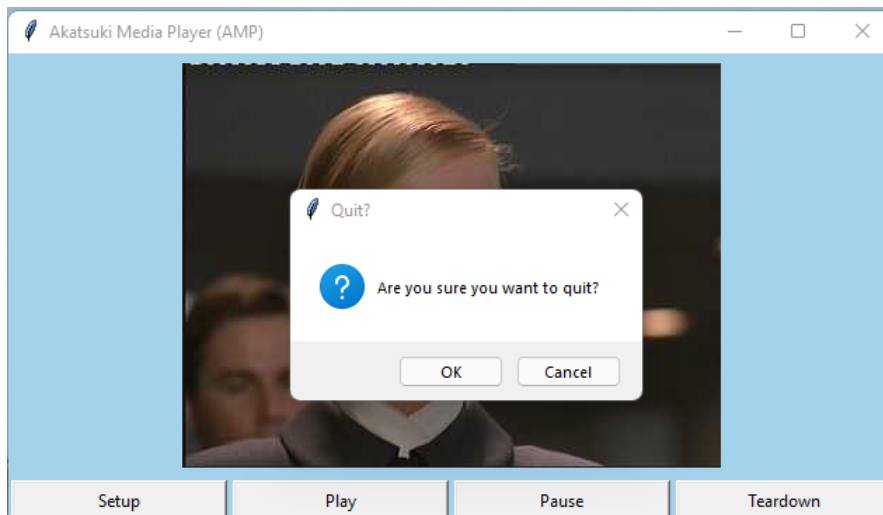


Figure 9: Giao diện của người dùng khi nhấn nút X

6 Extend

6.1 Calculate the statistics about the session

6.1.1 RTP Packet lost rate

Để xác định RTP packet loss rate ta có thể dùng chức năng RTP Stream Analysis của phần mềm Wireshark hoặc xác định dựa trên sequence number ở bên phía Client, nếu như Current Sequence Number không tăng dần theo thứ tự thì chứng tỏ đã có packet bị loss. Trong bài tập lớn này thì nhóm dùng cách thứ hai: Xác định theo Current Sequence Number.

Theo như file ServerWorker.py, cứ mỗi 0.05s server sẽ gửi 1 gói tin, vậy thì tối đa có 20 gói được gửi, nhưng có những lúc mạng không ổn định thì có thể chỉ có 15-16 gói tin được gửi, vậy nên không thể xác định bằng cách đếm có đủ 20 gói tin trong 1s được.

Ta xử lý bằng cách đặt thêm 1 biến counter khởi tạo bằng 0, biến này sẽ tăng mỗi khi listenRtp được gọi và bắt được data.

Khi hàm displayStats() được gọi, rate sẽ được tính bằng công thức: lấy counter chia cho current frame number hiện tại ($\text{maxframeNbr} = 500$).

```
1 def displayStats(self):
2     """Displays observed statistics"""
3     totalPackets = ((self.counter) / (self.countTotalPacket)) * 100
4
5     top1 = Toplevel()
6     top1.title("Statistics")
7     top1.geometry('300x170')
8     Lb2 = Listbox(top1, width=80, height=20)
9     Lb2.insert(1, "Current Packets No.%d " % self.frameNbr)
10    Lb2.insert(2, "Total Streaming Packets: %d packets" % self.
        countTotalPacket)
11    Lb2.insert(3, "Packets Received: %d packets" % self.packets)
12    Lb2.insert(4, "Packets Lost: %d packets" % self.counter)
13    Lb2.insert(5, "Packet Loss Rate: %d%" % totalPackets)
14    Lb2.insert(6, "Play time: %.2f seconds" % self.timer)
15    Lb2.insert(7, "Bytes received: %d bytes" % self.bytes)
16    Lb2.insert(8, "Video Data Rate: %d bytes per second" % (self.
        bytes / self.timer))
17    Lb2.insert(9, "Total Jitter: %.3fms" % (self.totalJitter * 1000))
18    Lb2.insert(10, "Average Jitter: %.3fms" % ((self.totalJitter /
        self.packets) * 1000))
19    Lb2.pack()
```

Code 1: Hàm displayStats() hiện thực trong Client2.py

6.1.2 Video data rate

Về vấn đề video data rate, ta sẽ tính bằng tổng dung lượng Payload (chính là phần data trừ đi số byte của header) chia cho thời gian đã chạy video.

Ta xử lý bằng cách thêm thư viện time của python, khởi tạo 3 biến timerBegin, timerEnd, timer.

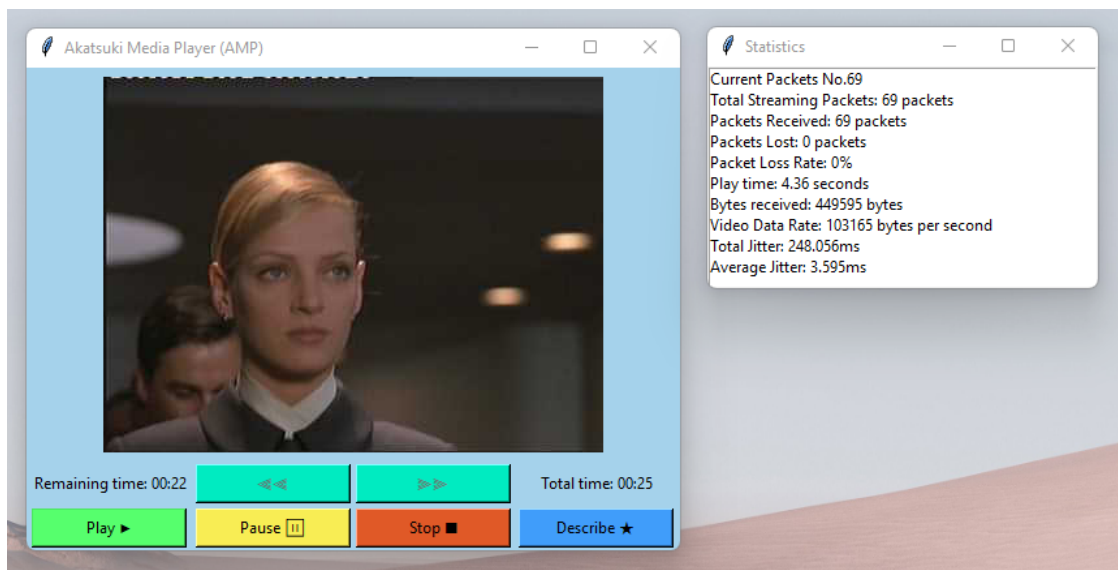
timerBegin được đặt tính từ lúc người dùng bấm Play.

timerEnd được đặt khi Pause được gọi, timer cũng được tính từ lúc này.
 $timer = timerEnd - timerBegin$, vì 1 video có thể bị pause nhiều lần, bởi Pause hoặc nút thoát (X) nhưng lại cancel nên chúng ta sẽ cộng dồn biến này lại.

Tổng số Payload thì được cộng dồn mỗi khi listenRtp được gọi và bắt được data.

Khi hàm displayStats() được gọi, video data rate sẽ được tính bằng công thức tổng số Payload tính được chia cho thời gian từ lúc chạy tới lúc kết thúc việc gửi gói tin (không tính thời gian Pause).

Và đây là kết quả khi hàm displayStats() được gọi:



6.2 Implement PLAY, PAUSE and STOP buttons

- SETUP là bắt buộc trong tương tác RTSP, ở trong bài tập lớn này, chúng ta có 2 cách đơn giản để hiện thực nó:

+ Cách 1: Tự động SETUP ngay sau khi chương trình được chạy (ngay sau khi gõ lệnh ở terminal thứ 2).

+ Cách 2: Chương trình sẽ SETUP ở lần bấm PLAY đầu tiên hoặc bấm PLAY sau khi bấm STOP.

- Ở đây, nhóm chúng em xin chọn cách 2 để hiện thực thao tác SETUP.

- Đối với thao tác STOP thì chương trình sẽ quay về trạng thái ban đầu và reset các biến thống kê. Khi đó, nếu ta tiếp tục nhấn PLAY thì chương trình sẽ được SETUP và khởi chạy video. Ta chỉ có thể thoát bằng cách nhấn vào dấu X ở góc trên bên phải của ứng dụng.

- Dưới đây là các bước thao tác cơ bản:

+ Sau khi gõ 2 lệnh lần lượt trên 2 terminal, chương trình sẽ được khởi chạy:

+ Khi bấm PLAY, chương trình sẽ được SETUP và video sẽ được khởi chạy:

+ Khi bấm PAUSE, video sẽ được tạm dừng, các thông số thống kê từ lần PLAY khởi chạy video sẽ được hiển thị.

- + Khi người dùng bấm vào STOP, video sẽ được tạm dừng trong phút chốc và được đưa về trạng thái ban đầu, chương trình sẽ gọi chức năng TEARDOWN và SETUP lại, thông số thống kê sẽ được hiển thị.
 - + Khi người dùng bấm PLAY, chương trình sẽ được SETUP và phát như lần PLAY đầu tiên.
 - + Và cuối cùng, để thoát ra khỏi chương trình thì người dùng bấm vào dấu X ở góc phải màn hình, chương trình sẽ hỏi lại người dùng về quyết định này, nhấn OK để xác nhận và thoát khỏi chương trình.
 - + Với cách hiện thực trên thì ta nhận thấy: khi bấm vào TEARDOWN ở phần hiện thực trong phần yêu cầu thì video sẽ dừng lại và quay về trạng thái ban đầu chưa được SETUP. Đối với STOP được hiện thực trong phần mở rộng này cũng tương tự, nhưng đồng thời chương trình cũng reset các biến đếm và các biến thống kê.
- Điều gọi chức năng TEARDOWN khi STOP không hẳn là phù hợp, vì người dùng có khả năng sẽ xem lại video sau khi đã nhấn STOP nên việc ngắt kết nối với socket sẽ làm chương trình phải SETUP lại lần nữa nếu người dùng nhấn PLAY. Tuy nhiên, theo thống kê thì tỷ lệ người dùng sẽ xem lại video sau khi đã nhấn STOP là không cao nên cách làm này cũng không mang lại nhiều rủi ro. Ở bài tập lớn này, để đảm bảo khả năng thực thi trơn tru của ứng dụng thì nhóm đã đồng thời đưa việc ngắt kết nối với socket và kết nối lại cho người dùng toàn quyền bằng cách tích hợp các hành động này lần lượt cùng với thao tác STOP và PLAY, có thể xem ở một cách tổng quát, hành động này cũng mang ý nghĩa tương tự việc giữ kết nối với socket. Tuy cách làm này còn khá đơn thuần và chưa tối ưu, nhưng mà nó giúp chương trình vẫn hoạt động hiệu quả

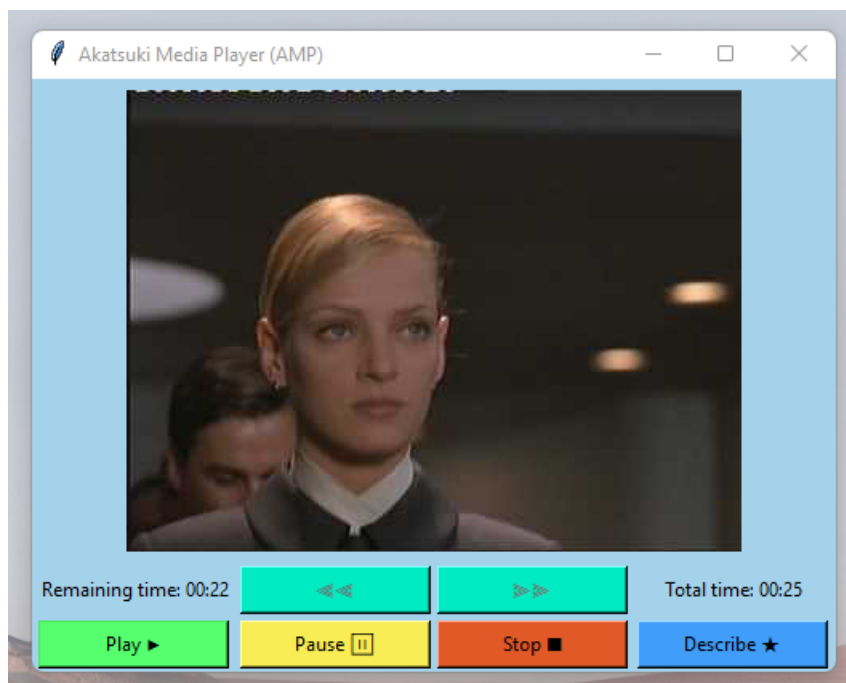


Figure 10: Khi người dùng nhấn vào nút Play để xem video

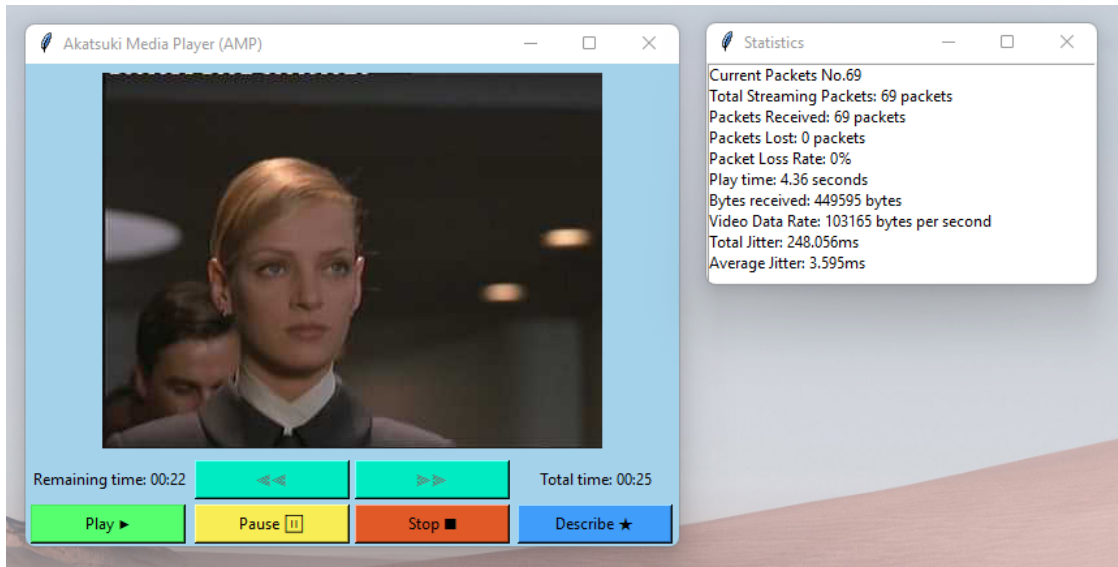


Figure 11: Khi người dùng nhấn vào nút Pause để dừng video

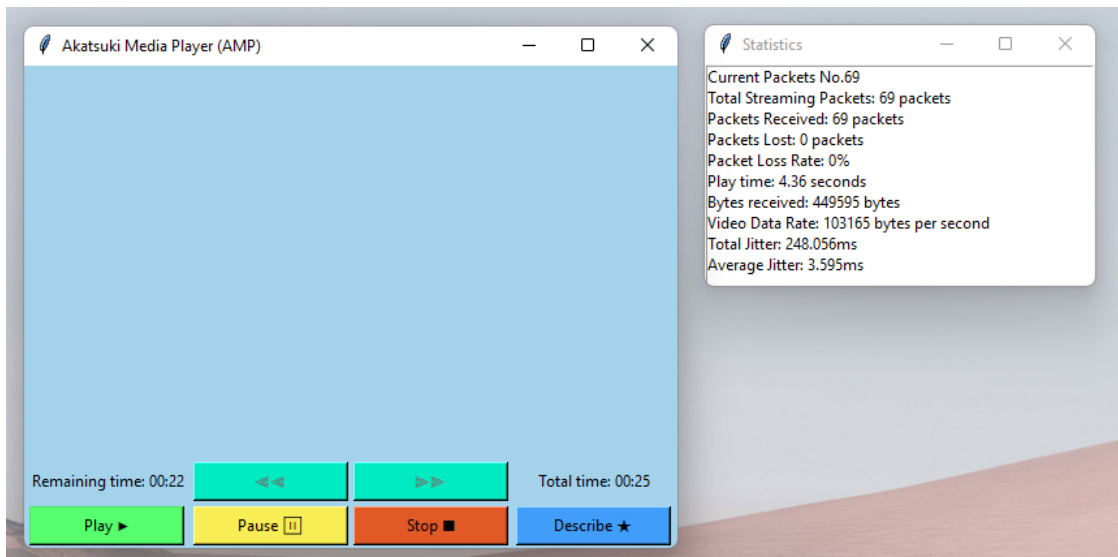


Figure 12: Khi người dùng nhấn vào nút Stop để reset video

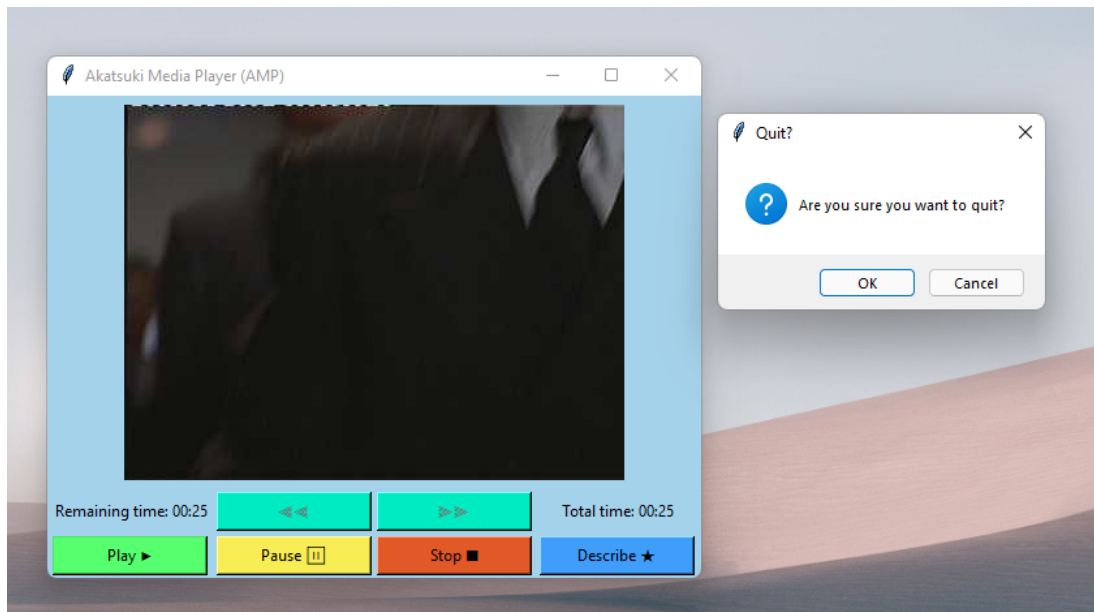


Figure 13: Khi người dùng nhấn vào nút X để thoát

6.3 Implement the DESCRIBE method

Nhóm tạo thêm 1 nút DESCRIBE trong giao diện người dùng và đồng thời viết thêm hàm `describeMovie()` để truyền yêu cầu DESCRIBE đến server. Yêu cầu DESCRIBE chỉ được gửi đi khi người dùng đã SETUP thành công hoặc chương trình đang trong trạng thái READY hoặc PLAYING.

```
1 def describeMovies(self):  
2     """Describe button handler"""  
3     self.sendRtspRequest(self.DESCRIBE)
```

Code 2: Hàm `describeMovie()` dùng để gửi cho server yêu cầu DESCRIBE từ phía Client

Khi Server đã nhận được yêu cầu DESCRIBE qua giao thức RTSP, ta sẽ viết thêm một đoạn code trong hàm `processRtspRequest()` của `ServerWorker.py` để xử lý yêu cầu và sau đó gọi hàm `replyDescribe()` để trả về thông tin của media stream cho phía Client.

```
1 elif requestType == self.DESCRIBE:  
2     if self.state != self.INIT:  
3         print("processing DESCRIBE\n")  
4         self.replyDescribe(self.OK_200, seq[1], filename)
```

Code 3: Đoạn code dùng để xử lý yêu cầu DESCRIBE từ phía Client bên Server



```
1 def replyDescribe(self, code, seq, filename):
2     """Send RTSP Describe reply to the client."""
3     if code == self.oK_200:
4         myreply = 'RTSP/1.0 200 OK\nCseq: ' + seq + '\nSession: '
5             + str(self.clientInfo['session'])
6
7         descriptionBody = "\nv= 0"
8         descriptionBody += "\nm= Video " + self.clientInfo['rtpPort']
9             + "RTP/AVP " + str(MJPEG_TYPE)
10        descriptionBody += "\na= Control: streamid ="
11            + str(self.clientInfo['session'])
12        descriptionBody += "\na= Mime-type: video/MJPEG\""
13
14        myreply += "\nContent-Base: " + filename
15        myreply += "\nContent-Type: " + "application/sdp"
16        myreply += descriptionBody
17        connSocket = self.clientInfo['rtspSocket'][0]
18        connSocket.send(myreply.encode())
```

Code 4: Hàm replyDescribe() dùng để gửi thông tin về Stream từ Server về Client

Cổng mặc định được sử dụng cho giao thức RTSP là 554 và cổng này được sử dụng cho cả giao thức của tầng giao vận UDP và TCP.

Thông điệp trả về theo SDP format (Session Description Protocol) đây là format dùng để mô tả các thông tin phiên giao tiếp của các chương trình multimedia.

Sau khi server gửi nội dung về Stream về cho Client thì bên Client sẽ xử lý thông điệp để có thể hiển thị từng thông tin theo từng dòng một được hiển thị trên form.

```
1 elif self.requestSent == self.DESCRIBE:
2     # self.state =
3     self.displayDescription(Lines)
```

Code 5: Một đoạn code trong hàm parseRtspReply() phía Client2.py để xử lý reply từ Server

```
1 def displayDescription(self, lines):
2     top = Toplevel()
3     top.title("Description")
4     top.geometry('300x180')
5
6     Lb1 = Listbox(top, width=50, height=30)
7     Lb1.insert(1, "Describe: ")
8     Lb1.insert(2, "Name Video: " + str(self.fileName))
9     Lb1.insert(3, lines[1])
10    Lb1.insert(4, lines[2])
11    Lb1.insert(5, lines[3])
12    Lb1.insert(6, lines[4])
13    Lb1.insert(7, lines[5])
14    Lb1.insert(8, lines[6])
15    Lb1.insert(9, lines[7])
16    Lb1.insert(10, lines[8])
17    Lb1.insert(11, "Thoi diem trong video: " + "%02d:%02d" %
18    (self.currentTime // 60, self.currentTime % 60))
19    Lb1.pack()
```

Code 6: Hàm displayDescription() dùng để in ra form gồm những tin về Stream cho người dùng

Nội dung được hiển thị ra cho người dùng như sau:

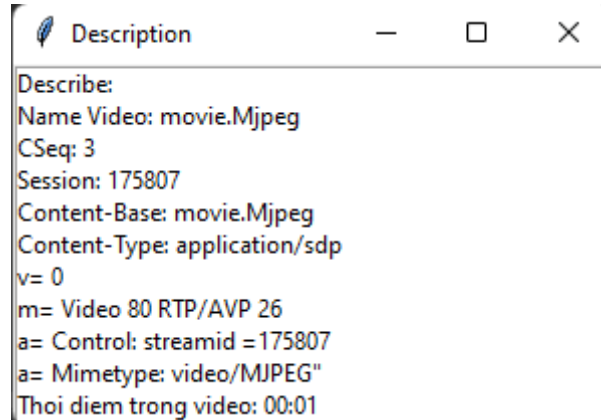


Figure 14: Nội dung về Stream hiển thị ra cho người dùng khi người dùng gửi yêu cầu DESCRIBE

Trong đó, phần nội dung thông điệp chứa những thông tin sau:

- v=0 : phiên bản của giao thức, hiện chỉ là 0.
- m=video 5008 RTP/AVP 26 : kiểu của chương trình media và địa chỉ transport. Theo đặc tả của SDP, 'm' line format theo dạng sau:

$$m=<media> <port> <transport> <fmt>$$

Trong đó, kiểu <media> là video, phần <port> là RTPport, <transport> là RTP/AVP, và <fmt> là MJPEG_TYPE = 26.

- a=control:streamid=997216 : streamid chính là session id của video stream.
- a=mimetype:string;"video/MJPEG" : Kiểu phương tiện (MIME type hay media type) là định danh hai phần cho định dạng file và nội dung định dạng được truyền trên Internet. "video/MJPEG" cho biết kiểu nội dung đang streaming trên session là video và file được mã hóa bằng MJPEG.

6.4 Implement some additional functions for user interface such as: display video total time and remaining time, fast forward or backward video

Ở phần extend này, nhóm đã hiện thực cả label hiển thị total time và remaining time của video, đồng thời thêm chức năng fast forward 10% và backward 10% video.

6.4.1 Label total time and remaining time

Đầu tiên, nhóm khởi tạo 2 label totalTimeBox và remainTimeBox thông qua thư viện tkinter. Khi người dùng bấm vào nút SETUP, RTSP Request sẽ được gửi đến Server. Sau đó, ngoài việc gọi replyRtsp(), ServerWorker còn gọi thêm hàm replySetup() mà trong đó có nhận tổng thời gian của video vào biến totalTime từ hàm get_total_time_video() ở file VideoStream.py

```
1 def replySetup(self, code, seq):
2     """Send RTSP reply to the client."""
3     if code == self.OK_200:
4         totalTime = self.clientInfo['videoStream'].
5             get_total_time_video()
6         reply = 'RTSP/1.0 200 OK\r\nCSeq: ' + seq
7             + '\r\nSession: ' + str(self.clientInfo['session'])
8             + '\r\nTotalTime: ' + str(totalTime)
9         connSocket = self.clientInfo['rtspSocket'][0]
10        connSocket.send(reply.encode())
11        # Error messages
12        elif code == self.FILE_NOT_FOUND_404:
13            print("404 NOT FOUND")
14        elif code == self.CON_ERR_500:
15            print("500 CONNECTION ERROR")
```

Code 7: Hàm replySetup để trả về thông tin sau khi người dùng SETUP

```
1 def get_total_time_video(self):
2     self.totalFrame = 0
3     while True:
4         data = self.file.read(5)
5         if data:
6             framelength = int(data)
7             # Read the current frame
8             data = self.file.read(framelength)
9             self.totalFrame += 1
10        else:
11            self.file.seek(0)
12            break
13        totalTime = self.totalFrame * 0.05
14    return totalTime
```

Code 8: Hàm get_total_time() dùng để lấy tổng thời gian của video

Hàm get_total_time_video() tính tổng thời gian tương đối của video bằng cách lấy tổng số frame của video nhân với khoảng cách thời gian giữa 2 lần server gửi packet cho client. Tổng thời gian tính bằng cách này sẽ có thể bị lệch so với thời gian thực video sẽ chạy, vì trong quá trình gửi packet, có thể đường truyền mạng sẽ gặp sự cố hoặc bị delay cho nên thời gian packet được gửi tới client sẽ bị trễ so với thời gian server gửi packet. Tuy nhiên, tổng thời gian chạy của video có thể được thể hiện tương đối thông qua cách này. Vì đây là Streaming Video, và không có cách nào dự đoán sự delay của đường truyền nên nhóm không thể xác định chính xác tổng thời gian thực mà video sẽ chạy. Label totalTimeBox sẽ được cập nhật ở lần play video đầu tiên:

Label remainTimeBox sẽ được cập nhật ở mỗi khi hàm ListenRtp nhận được data, bằng cách lấy biến totalTime trừ cho biến currentTime (với biến currentTime được tính bằng số thứ tự Frame hiện tại nhân với khoảng thời gian giữa 2 lần server gửi packet (ở bài tập lớn này là 0,05s)). Bởi vì cách tính tương tự nên remaining time này cũng mang tính tương đối giống với total time được tính ở trên.

6.4.2 Fast forward and backward video

Ở bài tập lớn này, nhóm hiện thực chức năng fast forward và backward ở cố định 10% video, nếu người dùng muốn fast forward nhiều hơn thì có thể nhấn nhiều lần. Đầu tiên, nhóm khai báo thêm 2 button "«" và "»", đại diện cho 2 chức năng fast forward và backward video.

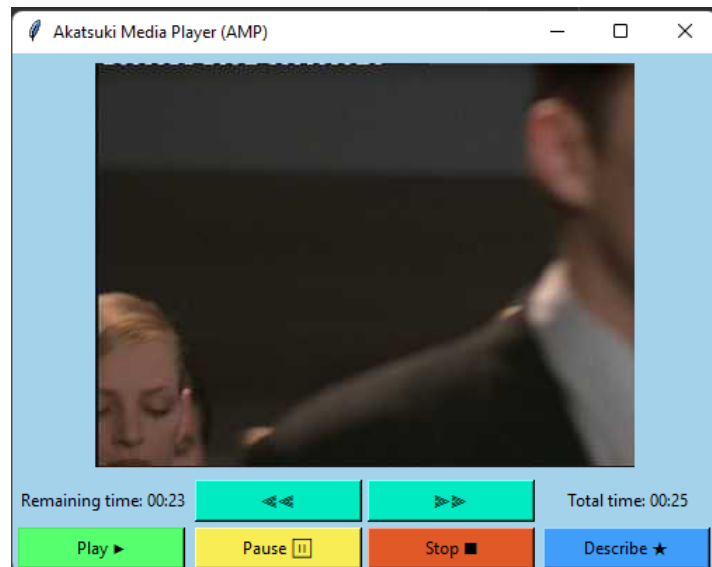


Figure 15: Nút "«" và "»" trong giao diện người dùng

- Khi nhấn nút "»", hàm `forwardMovies` sẽ được gọi và hàm này sẽ tiếp tục gọi hàm `sendRtspRequest` với tham số truyền vào là `"self.FORWARD"`.
 - + Hàm `sendRtspRequest` sẽ tiếp tục gửi request lên cho server.
 - + Sau khi server nhận được request, flag `isNext` ở đối tượng `VideoStream` sẽ được bật lên:
 - + Khi hàm `sendRtp` được server gọi, hàm này sẽ tiếp tục gọi hàm `nextFrame` ở đối tượng `VideoStream`.
 - + Và khi hàm `nextFrame` được gọi với flag `isNext` đã được bật, hàm sẽ trả về data của frame ở thứ tự 10% frame tiếp theo thay vì data ở frame tiếp theo như bình thường, đồng thời flag `isNext` sẽ được đưa về lại giá trị 0.
 - + Tuy nhiên, nếu số lượng frame còn lại của video không còn đến 10%, hàm `nextFrame` sẽ trả về data của frame cuối cùng.

```
1 def forwardMovies(self):  
2     self.sendRtspRequest(self.FORWARD)  
3     self.isForward = 1
```

Code 9: Hàm `forwardMovies()` hiện thực trong `Client2.py`

```
1 def nextFrame(self):
2     """Get next frame."""
3     if self.isNext == 1:
4         forwardFrames = int(self.totalFrame * 0.1)
5         remainFrames = int(self.totalFrame - self.frameNum)
6         if forwardFrames > remainFrames:
7             forwardFrames = remainFrames
8         self.isNext = 0
9
10    else:
11        forwardFrames = 1
12    if forwardFrames:
13        for i in range(forwardFrames):
14            data = self.file.read(5) # Get the framelength from the
15                # first 5 bits
16            if data:
17                framelength = int(data)
18
19                # Read the current frame
20                data = self.file.read(framelength)
21                self.frameNum += 1
22    return data
```

Code 10: Hàm nextFrame() hiện thực trong Client2.py

- Khi nhấn nút "«", hàm prevMovie sẽ được gọi và hàm này sẽ tiếp tục gọi hàm sendRtspRequest với tham số truyền vào là "self.PREV".
 - + Hàm sendRtspRequest sẽ tiếp tục gửi request lên cho server.
 - + Sau khi server nhận được request, flag opt ở đối tượng sẽ được bật lên:
 - + Khi hàm sendRtp được server gọi với flag opt được bật, hàm này sẽ tiếp tục gọi hàm prevFrame ở đối tượng VideoStream.
 - + Và khi hàm prevFrame được gọi, hàm sẽ đưa con trỏ đọc data về vị trí đầu video, đồng thời tiếp tục dịch chuyển vị trí con trỏ đến vị trí chứa data của frame cách frame hiện tại 10% frame video về trước, đồng thời trả về data, flag opt sẽ được đưa về lại giá trị 0.
 - + Tuy nhiên, nếu số lượng frame trước đó của video không còn đến 10%, hàm prevFrame sẽ trả về data của frame đầu tiên.

```
1 if not self.opt:
2     data = self.clientInfo['videoStream'].nextFrame()
3 else:
4     data = self.clientInfo['videoStream'].prevFrame()
5     self.opt = 0
```

Code 11: Đoạn code xử lý yêu cầu BACKWARD từ Client trong Server

```
1 def prevFrame(self):
2     preFrames = int(self.totalFrame * 0.1)
3     if self.frameNum <= preFrames:
4         data = self.file.seek(0)
5         self.frameNum = 0
6         if data:
7             framelength = int(data)
8             # Read the current frame
9             data = self.file.read(framelength)
10            self.frameNum += 1
11    else:
12        data = self.file.seek(0)
13        fFrames = self.frameNum - preFrames
14        self.frameNum = 0
15        for i in range(fFrames):
16            data = self.nextFrame()
17
18    return data
```

Code 12: Hàm prevFrame() hiện thực trong VideoStream.py