ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
Khoa Khoa học - Kĩ thuật Máy tính

BÁO CÁO BÀI TẬP LỚN SỐ 1
MẠNG MÁY TÍNH

ĐỀ TÀI: : VIDEO STREAMING SERVER AND CLIENT

GVHD: Bùi Xuân Giang

SV thực hiện: Huỳnh Nhật Long – 1812872
Trương Thanh Lộc – 1812982
Nguyễn Thị Linh – 1812822

TP. HỒ CHÍ MINH, THÁNG 11/2020

Mục lục

Danh sách hình vẽ

1 Requirements Analysis

- Người dùng có thể xem các video được lưu trữ trên server
- Ứng dụng cần có thêm các tính năng khác để đáp ứng những nhu cầu thiết yếu của người dùng khi sử dụng ứng dụng xem video như: pause (tạm dừng video), stop (dừng video và xem lại từ đầu), play (phát video), teardowm (dừng video đóng ứng dụng Client).
- Khi mở ứng dụng Client, một video có thể được phát, nếu người dùng muốn xem video khác cần mở thêm một Client khác.
- Video phải được phát liên tục, hạn chế các gián đoạn không cần thiết gây khó chịu cho người xem.
- Giao diện vừa phải, dễ sử dụng, sử dụng từ ngữ rõ ràng dễ hiểu.

2 Function Description

Ứng dựng cho phép người dùng xem video được truyền tải real time từ Server đến Client tại thời điểm xem

2.1 ClientLauncher

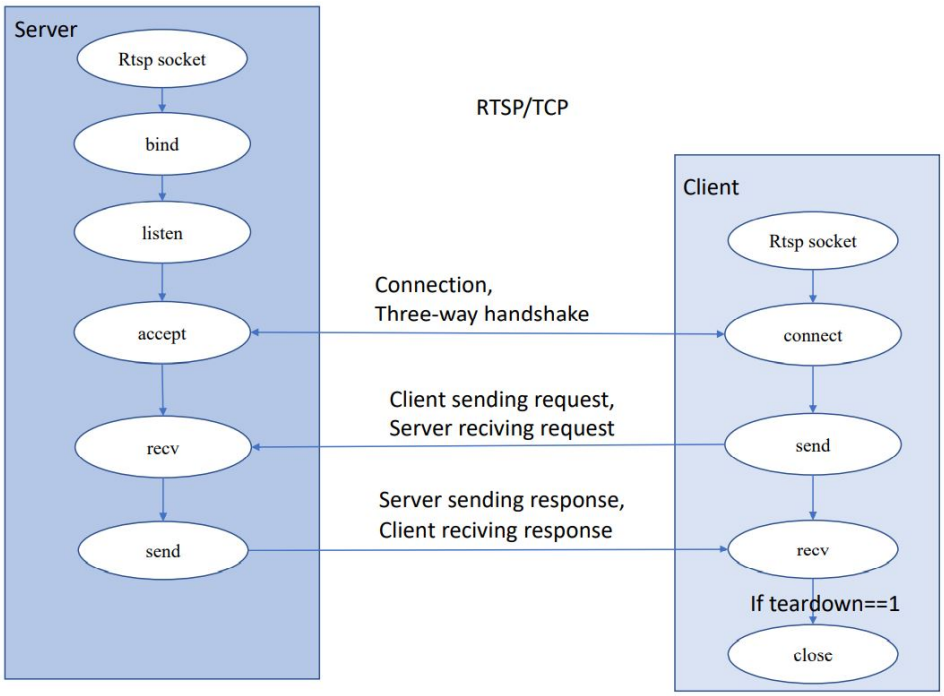
- Tạo một giao diện người dùng sử dụng thư viện tkinter của ngôn ngữ python
- Thông qua giao diện người dùng có thể gửi yêu cầu với giao thức RTSP thông qua việc kích hoạt các nút được tạo trên giao diện và hiện thị video mà người dùng yêu cầu.

2.2 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 tao, 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 nhắm 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 đươc hồi đáp thành công, trang 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 thi 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 click 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.

2.3 Server



Hình 1: *Communication between Client and Server by RTSP/TCP protocol*

Hình 1 minh họa cho quá trình giao tiếp trao đổi dữ liệu giữa Client và Server quá giao thức RTSP/TCP được thiết lập trong hệ thống Video Streaming.

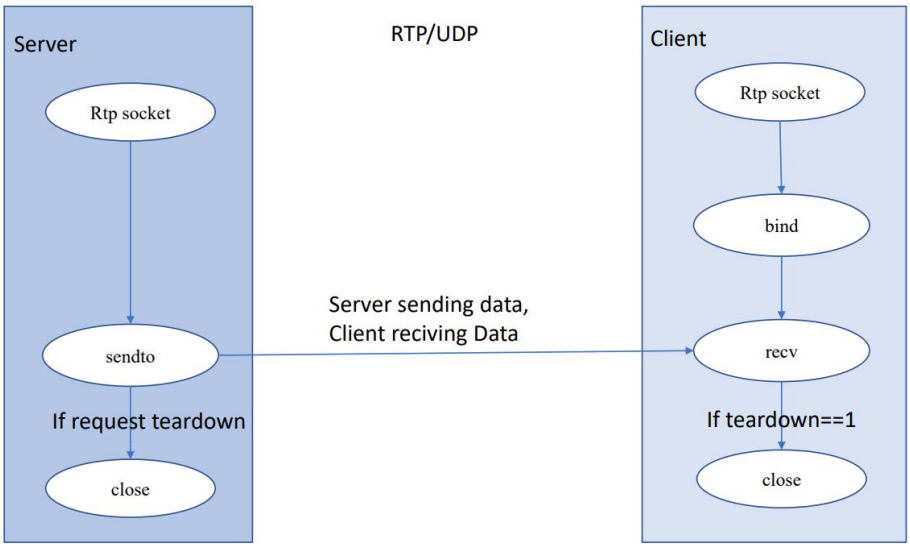
- Tạo RTSP Socket và mở kết nối đến public host và địa chỉ port đã cung cấp, Client và Server sẽ giao tiếp với nhau qua các RTSP socket trên mỗi thành phần - nhận và phản hồi yêu cầu từ Client.
- Biến ClientInfo được tạo ra để chứa dữ liệu mà client gửi cho server và chuyển qua ServerWorker để xử lý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

.

2.4 ServerWorker

- Xử lý Request(Setup, Play, Pause, Teardown) được gửi từ Client thông qua giao thức RTSP.
- Với mỗi client, một luồng xử lý Rquest được tạo ra. Trong mỗi luồn, 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 đươc 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, kiển tra trạng thái hiện tại của Server, nếu trang 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 dọc 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 độc lập và vòng lặp while kết hợp gửi giao thức UDP cho phép dữ liệu được dọc 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 tai 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 dóng RTP socket.

Hình 2 minh họa cho quá trình Server gửi các segment chứa dữ liệu về hình ảnh thông qua giao thức RTP/UDP.



Hình 2: *Communication between Client and Server by RTP/TCP protocol*

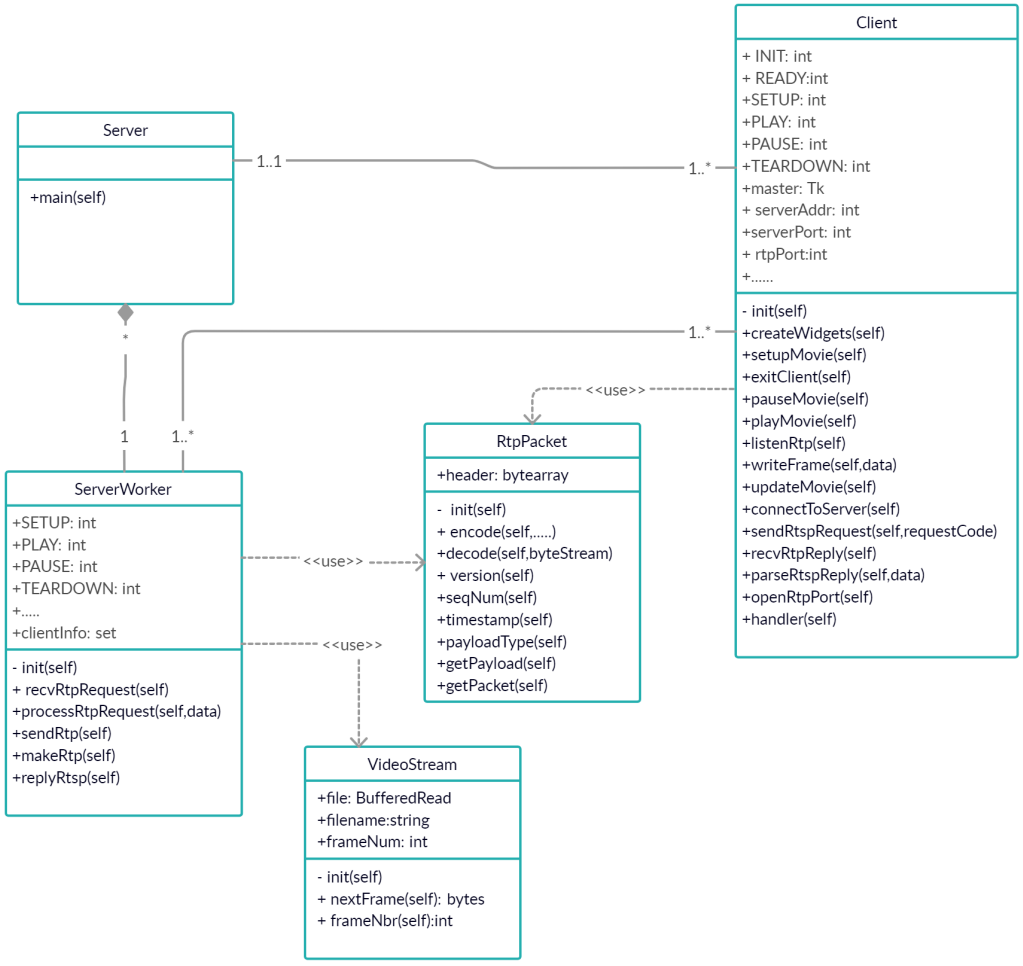
2.5 RtpPacket

- Hàm encode: Thiết lập header cho đối tưởng, 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 các thông số được lưu ở phần header
- Hàm getPayload: trả về payload trong packet
- Hàm getPacket: Đóng gói frame hình ảnh thành packet.

2.6 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 chứ năng đọc một frame dữ liệu trong file video, khi kết hợp vòng lặp while toàn độ frame trong file video sẽ được đọc.

3 Class Diagram



Hình 3: *Class Diagram of system*

4 A Summative Evaluation of Results Achieved

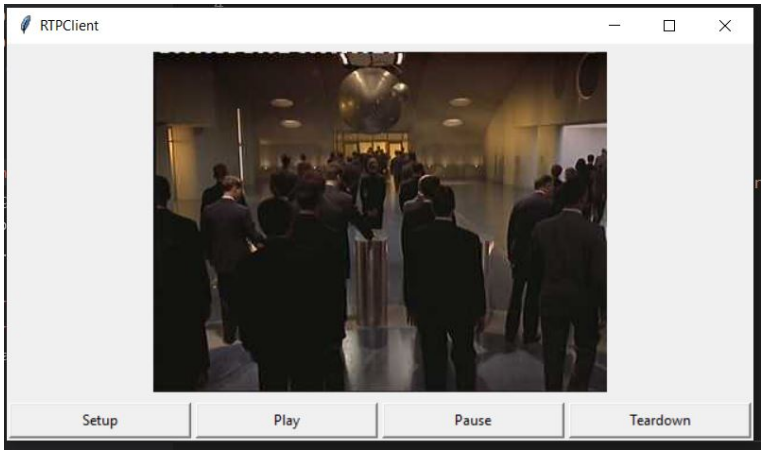
4.1 Về lập trình

- Về rtpPacket: Hoàn thành phần thiết lập header cho rtp packet, hiểu các sử dụng cách phương thức được xây dựng trong class.
- Về Client: Hoàn thành phần viết cấu trúc cho yêu cầu của Client và gửi nó đến Server.
- Flie VideoStream.py được chỉnh sửa thêm hàm getsize() để sử dụng cho phần extend.

4.2 Kết quả

- 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. Hình 4 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

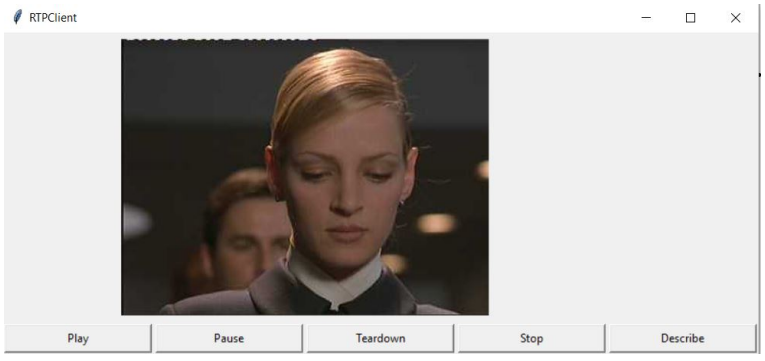
thiết lập. Nhìn chung, chúng tôi đã giải quyết các vấn đề được yêu cầu ở bài tập này và hoàn thành bài tập.



Hình 4: *Application of us*

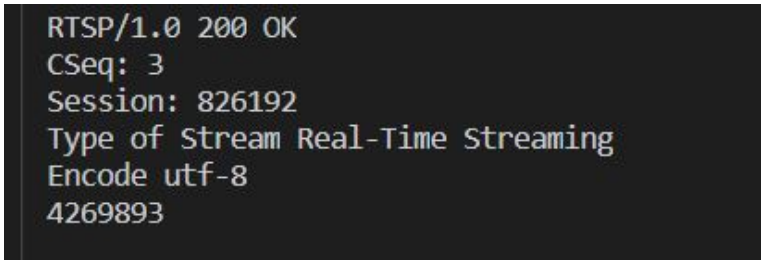
- Với phần mở rộng của đề tài, chúng tôi đã sử dụng WireShark để bắt các gói tin trao đổi giữa Client và Server để tính toán các tỉ lệ được yêu cầu kết quả cụ thể sẽ được trình bày ở phần sau. Về việc hiện thực ứng dụng và không có sử dụng nút "Setup" chúng tôi đã hoàn thành, ngoài ra chúng tôi đã thiết lập thêm nút "Stop" và "Describe" cho ứng dụng, kết quả được thể hiện ở Hình 5,6,7,8. Code hiện thực được lưu ở file Client1.py, ServerWork1.py .

Hình 5: *Interface start*



Hình 6: *Play video*

Hình 7: *Stop video*

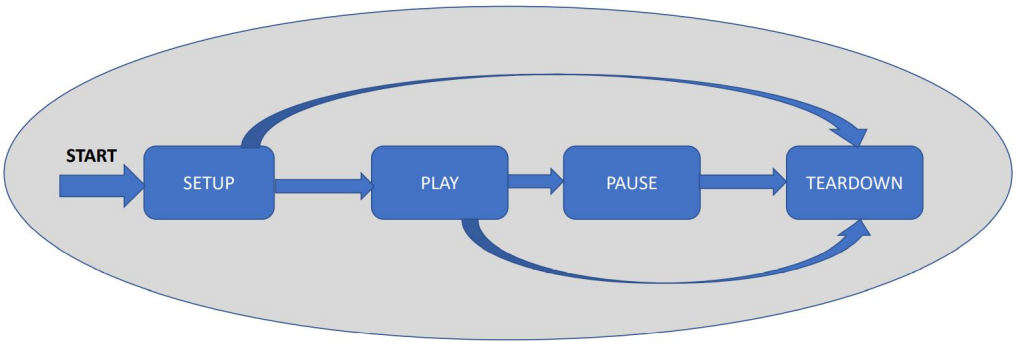


Hình 8: *Session desribe*

- Chú thích: File Client.py, ServerWorker.py là hai file hiện thực có phần bắt buộc của bài tập lớn. File Client1.py và ServerWorker1.py là hai file hiện thực cho phần extend.

5 User manual

Hình 9 mình họa cho quy trình sử dụng các nút trong ứng dụng Client.



Hình 9: *Flow of Client action*

- Để Server chuẩn bị video cho Client, người dùng nhấn "Setup", đây là yêu cầu bắt buộc nếu người dùng muốn xem video
- Sau khi setup, người dùng có thể nhấn "Play" để phát video, sau khi nhấn "Play" video sẽ được hiện thị trên khung giao diện
- Khi video đang phát, người dùng có thể nhấn "Pause" để tạm dừng video. Video dừng, để tiếp tục xem tiếp video người dùng nhấn "Play".
- Để đóng ứng dụng Client, người dùng nhấn "TearDown", khung giao diện đóng.

6 Extend

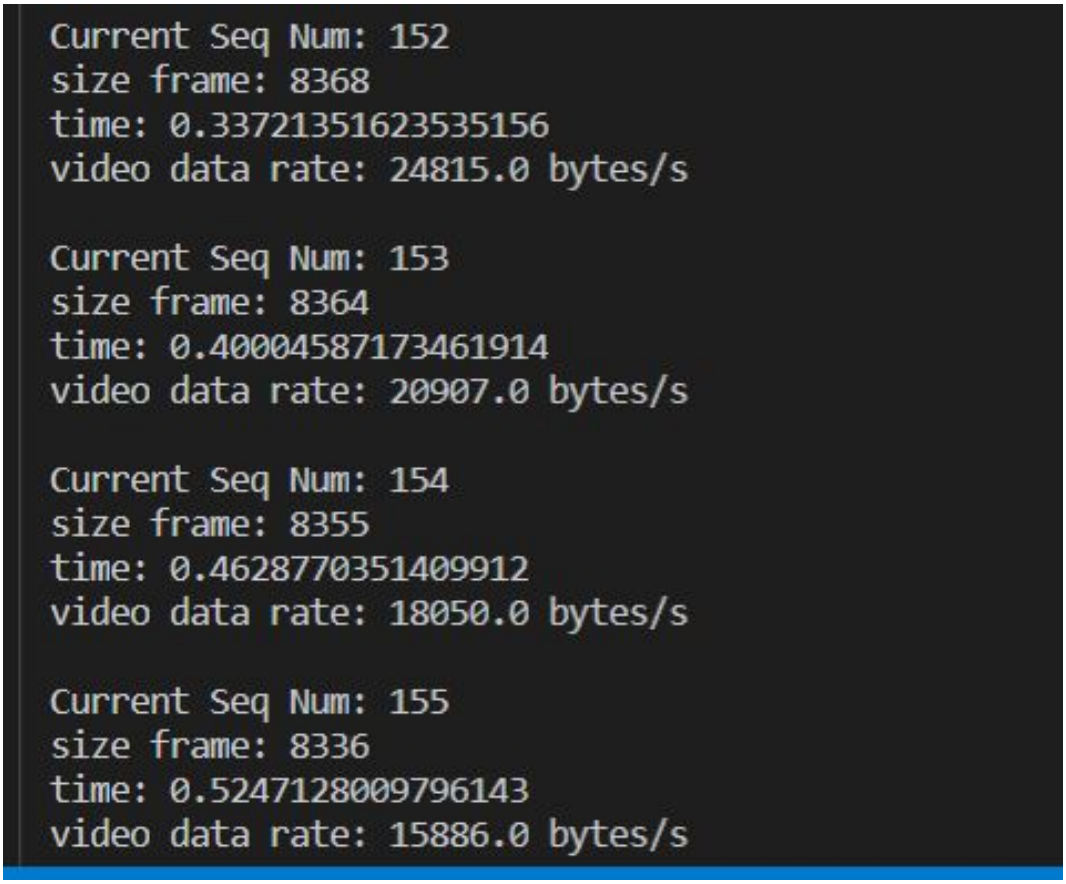
1. Triển khai Server và Client lên localhoat, sử dụng WrieShark để bắt các gói tin trao đổi trên localhost chúng tôi thu được kết quả như Hình 10: Theo quan sát,

1	10:41:08.995516	127.0.0.1	127.0.0.1	TCP	52 50009 → 2500 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM=1
2	10:41:09.079291	127.0.0.1	127.0.0.1	TCP	52 2500 → 50009 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM=1
3	10:41:09.081765	127.0.0.1	127.0.0.1	TCP	40 50009 → 2500 [ACK] Seq=1 Ack=1 Win=2619648 Len=0
4	10:41:12.077131	127.0.0.1	127.0.0.1	TCP	114 50009 → 2500 [PSH, ACK] Seq=1 Ack=1 Win=2619648 Len=74 [TCP segment of a reassembled PDU]
5	10:41:12.077131	127.0.0.1	127.0.0.1	TCP	40 2500 → 50009 [ACK] Seq=1 Ack=75 Win=2619648 Len=0
6	10:41:12.140916	127.0.0.1	127.0.0.1	TCP	79 2500 → 50009 [PSH, ACK] Seq=1 Ack=75 Win=2619648 Len=39 [TCP segment of a reassembled PDU]
7	10:41:12.141914	127.0.0.1	127.0.0.1	TCP	40 50009 → 2500 [ACK] Seq=75 Ack=40 Win=2619648 Len=0
8	10:41:13.959978	127.0.0.1	127.0.0.1	TCP	113 50009 → 2500 [PSH, ACK] Seq=75 Ack=40 Win=2619648 Len=73 [TCP segment of a reassembled PDU]
9	10:41:13.960086	127.0.0.1	127.0.0.1	TCP	40 2500 → 50009 [ACK] Seq=40 Ack=148 Win=2619648 Len=0
10	10:41:14.176960	127.0.0.1	127.0.0.1	TCP	79 2500 → 50009 [PSH, ACK] Seq=40 Ack=148 Win=2619648 Len=39 [TCP segment of a reassembled PDU]
11	10:41:14.177240	127.0.0.1	127.0.0.1	TCP	40 50009 → 2500 [ACK] Seq=148 Ack=79 Win=2619648 Len=0
12	10:41:14.244090	127.0.0.1	127.0.0.1	UDP	6054 52196 → 20000 Len=6026
13	10:41:14.315863	127.0.0.1	127.0.0.1	UDP	5980 52196 → 20000 Len=5952
14	10:41:14.379181	127.0.0.1	127.0.0.1	UDP	5965 52196 → 20000 Len=5937
15	10:41:14.443287	127.0.0.1	127.0.0.1	UDP	5900 52196 → 20000 Len=5872
16	10:41:14.506527	127.0.0.1	127.0.0.1	UDP	5811 52196 → 20000 Len=5783
17	10:41:14.570771	127.0.0.1	127.0.0.1	UDP	5729 52196 → 20000 Len=5701
18	10:41:14.647991	127.0.0.1	127.0.0.1	UDP	5529 52196 → 20000 Len=5501
19	10:41:14.725196	127.0.0.1	127.0.0.1	UDP	5353 52196 → 20000 Len=5325
20	10:41:14.790839	127.0.0.1	127.0.0.1	UDP	5259 52196 → 20000 Len=5231
21	10:41:15.006074	127.0.0.1	127.0.0.1	UDP	5133 52196 → 20000 Len=5105
22	10:41:15.098191	127.0.0.1	127.0.0.1	UDP	5040 52196 → 20000 Len=5012
23	10:41:15.169377	127.0.0.1	127.0.0.1	UDP	5010 52196 → 20000 Len=4982
24	10:41:15.244395	127.0.0.1	127.0.0.1	UDP	5024 52196 → 20000 Len=4996
25	10:41:15.307738	127.0.0.1	127.0.0.1	UDP	5125 52196 → 20000 Len=5097
26	10:41:15.383276	127.0.0.1	127.0.0.1	UDP	5398 52196 → 20000 Len=5370
27	10:41:15.446399	127.0.0.1	127.0.0.1	UDP	5725 52196 → 20000 Len=5697
28	10:41:15.521765	127.0.0.1	127.0.0.1	UDP	6050 52196 → 20000 Len=6022
29	10:41:15.586624	127.0.0.1	127.0.0.1	UDP	6374 52196 → 20000 Len=6346

Hình 10: *Segment are exchanged via localhost*

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 là 500 và số packet nhận được là 500 trong 20 lần chạy liên tục).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. Kết quả thu được như Hình 11:



Hình 11: *size, time and data video rate for each packet*

2. 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 người dùng sau khi mở ứng dụng: xét trạng thái của Client nếu trạng thái hiện tại là INIT thì gửi thông điệp Setup đến Server đến 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 không thích hợp để gửi yêu cầu TEARDOWN vì STOP chỉ đơn giản là ứng dụng sẽ "reset" lại video (đóng RTP spcket) , xóa hình ảnh trong bộ nhớ đệm và người dùng có thể tiếp túc xem video bằng cách Click PLAY tuy nhiên lúc này video sẽ được bắt đầu lai từ đâu, còn với TEARDOWN toàn bộ socket của Client sẽ 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.

3. DESCRIBE: thêm một nút 'Describe' mà chức năng của nó là khi được click Client sẽ gửi yêu cầu DESREIBE 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 xuất ra terminal. Kết quả thu được như Hình 8 (đã được trình bày ở phần 4.1).

7 Phân công công việc

Huỳnh Nhật Long: hiện thực hàm parseRtspReply và describe phần mở rộng
Trương Thanh Lộc: Hiện thực phần sendRtspRequest và phần 1 extend
Nguyễn Thị Linh: Hiện thực phần Encode và Setup,Stop trong phần extend
Báo cáo được hoàn thành bởi tất cả các thành viên.

8 Tài liệu tham khảo

1. James F. Kurose, Keith W. Ross - Computer Networking_ A Top-Down Approach (2013, Addison-Wesley) - libgen.lc.
2.https://en.wikipedia.org/wiki/Real_Time_Streaming_Protocol