ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

KHOA: CÔNG NGHỆ THÔNG TIN

BÁO CÁO ĐỒ ÁN THỰC HÀNH

LẬP TRÌNH SOCKET

MÔN MẠNG MÁY TÍNH

NHÓM THỰC HIỆN:

MSSV: 24120139 – Đỗ Phước Thiện

MSSV: 24120079 – Trần Trung Kiên

MSSV: 24120029 – Võ Văn Cường

Giảng viên lý thuyết: Huỳnh Thụy Bảo Trân

Giảng viên thực hành: Chung Thùy Linh

Lớp lý thuyết/Nhóm thực hành: 24CTT3/24CTT3A

Học Kỳ - Niên khóa: HK1 - 2025-2026

MỤC LỤC

1.GIỚI THIỆU ĐỒ ÁN

1.1.Tổng quan đồ án và phân công công việc

Đây là đồ án được đưa ra nhằm củng cố kiến thức của sinh viên trong môn Mạng máy tính bằng việc yêu cầu xây dựng một hệ thống gồm hai thành phần chính: RTSP client và RTP server, hoạt động thông qua hai kênh truyền độc lập. Kênh điều khiển sử dụng giao thức RTSP (Real-Time Streaming Protocol) chạy trên TCP để đảm bảo độ tin cậy của các lệnh điều khiển, trong khi kênh truyền dữ liệu video sử dụng RTP (Real-Time Transport Protocol) chạy trên UDP nhằm đạt tốc độ truyền cao và độ trễ thấp, phù hợp với yêu cầu video thời gian thực.

Bảng phân chia công việc và mức độ hoàn thành:

Phần việc:	Phân công cho bạn:	Mức độ hoàn thành
Cài đặt RTSP protocol ở phía Client	Đỗ Phước Thiện	100%
Cài đặt RTP packetization ở phía Server	Đỗ Phước Thiện	100%
HD Video Streaming	Trần Trung Kiên	100%

Client-Side Caching	Trần Trung Kiên	100%
Thanh thời lượng	Võ Văn Cường	100%
Tua video	Võ Văn Cường	100%

1.2.Phạm vi yêu cầu

1.2.1.Yêu cầu cơ bản

Là các yêu cầu bắt buộc để hệ thống có thể hoạt động đúng chức năng theo chuẩn RTSP/RTP.

+ **RTSP CLIENT**

Sinh viên cần hoàn thiện phần xử lý RTSP trong Client:

- **SETUP**
  - Gửi yêu cầu thiết lập session.
  - Gửi Transport header khai báo cổng RTP.
  - Đọc và lưu Session ID.
  - Tạo socket UDP để nhận RTP.
- **PLAY**
  - Gửi yêu cầu bắt đầu stream.

- Chèn Session header (dùng Session ID).

- **PAUSE**

  - Gửi yêu cầu tạm dừng.

  - Chèn Session header.

- **TEARDOWN**

  - Gửi yêu cầu kết thúc session.

  - Chèn Session header.

- **Các yêu cầu kỹ thuật khác:**

  - Gửi request theo đúng format RTSP/1.0.

  - Quản lý thứ tự request bằng trường CSeq (tăng dần).

  - Quản lý state machine của client: INIT → READY → PLAYING → READY → INIT.

+ **RTP PACKETIZATION(SERVER)**

Sinh viên cần hoàn thiện hàm encode() trong class RtpPacket:

- **Gói tin RTP phải tuân theo chuẩn**

  - Version (V =2).

  - Padding (P = 0), Extension (X = 0), CC = 0.

  - Marker (M = 0).

- Payload type (PT = 26 — MJPEG).
- Sequence number (từ frame number).
- Timestamp (dùng time.time() nhưng thực tế trong đồ án dùng Frame Number để đồng thanh tiến độ và hỗ trợ Seek).
- SSRC (chọn số bất kỳ).
- Sao chép payload (JPEG frame) đúng định dạng.

- **Server phải**
  - Đọc từng frame từ VideoStream.
  - Đóng gói frame → RTP packet.
  - Gửi packet qua UDP mỗi 50 ms.

1.2.2. Yêu cầu nâng cao

Là các yêu cầu mở rộng vào để hệ thống có thể hoạt động trôi chảy, mượt mà và đạt năng suất cao hơn, cụ thể:

+ **HD VIDEO STREAMING**

- Hệ thống phải hỗ trợ streaming video chất lượng cao (720p/1080p):
  - **Xử lý MJPEG** có kích thước lớn (> 97KB).
  - **Fragmentation:** tách 1 frame thành nhiều RTP packet nhỏ (Payload <= 1400 bytes) để không vượt quá MTU của mạng.
  - **Flow Control (Burst Mode):** Server điều tiết tốc độ gửi gói tin theo

chùm để cân bằng giữa tốc độ nạp bộ đệm và khả năng chịu tải của mạng, giảm thiểu mất gói.

+ **CLIENT-SIDE CACHING**

- Hệ thống phải có bộ đệm để chống giật hình và cải thiện mượt mà:
  - **Smart Buffering:** Sử dụng hàng đợi (Queue) với cơ chế ngưỡng (Threshold). Video chỉ bắt đầu phát khi bộ đệm nạp đủ dữ liệu, mô phỏng trải nghiệm mượt mà. Khi người dùng nhấn PLAY, video phát từ buffer chứ không phải từ socket trực tiếp.
  - **Memory Caching:** Lưu trữ tạm thời các khung hình đã hiển thị vào RAM. Cho phép người dùng xem lại (Instant Replay) các đoạn video đã qua mà không tiêu tốn băng thông mạng và không có độ trễ.

+ **SEEK BAR & PLAYBACK NAVIGATION**

Hệ thống cần hỗ trợ thanh thời lượng (progress bar) cho phép người dùng theo dõi và điều hướng video trong lúc phát.

- **Real-time Progress Bar:** Thanh thời lượng đồng bộ chính xác với số thứ tự khung hình (Frame Number) nhận được từ Server, không bị trôi do độ trễ mạng.
- **Smart Seek Algorithm** (Thuật toán tua 3 lớp)**:**
  - **Lớp 1 (Cache Hit)**: Truy xuất tức thì từ bộ nhớ RAM.
  - **Lớp 2 (Buffer Hit)**: Tua nhanh trong hàng đợi hiện có.

- **Lớp 3 (Network Request)**: Gửi lệnh điều khiển vị trí (Seek Request) lên Server.

- **Trải nghiệm người dùng (UX):** Đảm bảo đồng bộ trạng thái (Play/Pause) trước và sau khi tua. Tự động xử lý và làm sạch bộ đệm cũ để tránh xung đột dữ liệu hình ảnh.

2.LÝ THUYẾT NỀN TẢNG

2.1. Giao thức RTSP

2.1.1.Khái niệm

RTSP (RFC 2326) là một giao thức điều khiển tầng ứng dụng (Application Layer), được thiết kế để thiết lập và kiểm soát các phiên truyền thông tin thời gian thực (như âm thanh, video). Khác với HTTP là giao thức phi trạng thái (stateless), RTSP là giao thức có trạng thái (stateful), nghĩa là máy chủ lưu giữ thông tin về phiên làm việc của client từ lúc khởi tạo cho đến khi kết thúc.

2.1.2.Vai trò

+ RTSP không truyền video trực tiếp mà chỉ dùng để gửi lệnh điều khiển, ví dụ:

  - SETUP – thiết lập phiên và thông số truyền tải

  - PLAY – bắt đầu phát

  - PAUSE – tạm dừng

  - TEARDOWN – kết thúc phiên

+ Media thực tế được truyền bằng RTP.

2.1.3. Đặc điểm kỹ thuật

+ Giao thức dạng text, giống HTTP.
+ Thường chạy qua TCP (đảm bảo tin cậy cho lệnh điều khiển).
+ Cổng mặc định: 554 TCP.
+ Hỗ trợ nhiều session đồng thời nhờ Session ID.
+ Có bộ đếm CSeq để đánh số thứ tự các yêu cầu.

2.2. Giao thức RTP

2.2.1. Khái niệm

RTP (Real-Time Transport Protocol) là giao thức tầng ứng dụng cho phép truyền tải dữ liệu media thời gian thực qua mạng IP. Được thiết kế để truyền video, audio với độ trễ thấp.

2.2.2. Vai trò

Trong hệ thống streaming:

- RTSP gửi lệnh điều khiển.
- RTP vận chuyển dữ liệu media.

RTP thường được truyền qua UDP để giảm độ trễ.

2.2.3. Cấu trúc gói RTP

Gói RTP gồm hai phần:

- Header (12 bytes, chứa phiên bản, sequence number, timestamp, SSRC)
- Payload (dữ liệu video/audio)

Các trường quan trọng:

- Version (V): phải là 2.
- Payload Type (PT): xác định loại dữ liệu — MJPEG là 26.
- Sequence Number: đánh số gói → hỗ trợ phát hiện mất gói.
- Timestamp: hỗ trợ đồng bộ khung hình.
- SSRC: định danh nguồn phát.

2.2.4.Đặc tính RTP

Truyền theo kiểu best-effort (UDP) → có thể mất gói.

Không đảm bảo thứ tự → client dựa vào sequence number để sắp xếp.

Là nền tảng của hầu hết các hệ thống live streaming (WebRTC, VoIP...).

2.3. Mối quan hệ giữa RTP và RTSP

RTSP đóng vai trò điều khiển phiên, RTP mang dữ liệu.

Quy trình điển hình:

- Client gửi SETUP → thương lượng cổng RTP.

- Server mở luồng RTP.
- Client gửi PLAY → media bắt đầu truyền qua RTP/UDP.
- Client muốn dừng → gửi PAUSE.
- Kết thúc → TEARDOWN.

Như vậy, RTSP là "remote control", còn RTP là "đường truyền video".

2.4. Định dạng MJPEG

2.4.1. Khái niệm :

MJPEG là định dạng video trong đó:

- Mỗi frame là một ảnh JPEG riêng biệt
- Các ảnh được ghép liên tục tạo thành chuỗi video.
- Không có nén liên frame (no inter-frame compression) như H.264 → dễ xử lý, giải mã nhanh, thích hợp cho lab.

2.4.2. Đặc điểm :

+ Mỗi khung hình độc lập → dễ truyền qua RTP.
+ Giải mã nhẹ → phù hợp lab/streaming độ phân giải thấp.
+ Nhược điểm: file lớn hơn nhiều so với H.264.

2.4.3. Định dạng MJPEG trong bài lab :

+ Video được lưu theo cấu trúc:

[5-byte size][JPEG image]

[5-byte size][JPEG image]

…

+ Trong đó 5 byte đầu chứa kích thước ảnh (số bit).
+ Server đọc từng frame → gói vào RTP → gửi cho client.

3.THIẾT KẾ HỆ THỐNG

3.1. Tổng quan hệ thống

+ Hệ thống được thiết kế theo mô hình Client-Server với kiến trúc xử lý song song (Multi-threading) và điều khiển ngoài băng tần (Out-of-band Control).
+ Hệ thống bao gồm hai kênh giao tiếp hoạt động độc lập:
  - Kênh điều khiển (Control Channel): Sử dụng giao thức RTSP trên nền TCP Port X). Đảm bảo tính tin cậy tuyệt đối cho các lệnh điều hướng (SETUP, PLAY, PAUSE, TEARDOWN).
  - Kênh dữ liệu (Data Channel): Sử dụng giao thức RTP trên nền UDP (Port Y). Tối ưu hóa tốc độ truyền tải và độ trễ thấp (Low Latency) cho luồng video thời gian thực.

3.2. Thiết kế chi tiết phía Server

3.2.1.Các thành phần module

+ Server.py(Connection Listener):Đóng vai trò cổng tiếp đón, sử dụng vòng lặp vô tận để lắng nghe yêu cầu kết nối TCP. Với mỗi Client kết nối, Server khởi tạo một luồng xử lý riêng biệt (ServerWorker), cho phép phục vụ nhiều người dùng đồng thời.

+ ServerWorker.py (Session Controller): Quản lý máy trạng thái (State Machine) của phiên RTSP. Điều tiết tốc độ gửi dữ liệu RTP thông qua cơ chế Burst Mode (gửi theo chùm) để cân bằng tải mạng.
+ VideoStream.py (Data Access Layer): Chịu trách nhiệm đọc và truy xuất dữ liệu từ file vật lý. Khi khởi tạo, module này quét toàn bộ file video để tạo bảng chỉ mục vị trí (Byte Offset Index), hỗ trợ tính năng truy cập ngẫu nhiên (Random Access/Seek) tức thì.
+ RtpPacket.py (Encapsulation): Đóng gói dữ liệu thô vào cấu trúc gói tin RTP chuẩn (RFC 1889). Xử lý logic Phân mảnh (Fragmentation): Cắt các khung hình kích thước lớn (HD Frame) thành các gói tin nhỏ phù hợp với MTU mạng.

3.2.2. Máy trạng thái

+ Server chuyển đổi giữa 3 trạng thái chính:
    - **INIT:** Chưa có phiên làm việc.
    - **READY:** Đã khởi tạo phiên (có Session ID), file video đã mở, sẵn sàng phát.
    - **PLAYING:** Đang trong vòng lặp gửi dữ liệu RTP.

3.3. Thiết kế chi tiết phía Client

Client đóng vai trò là "Người tiêu dùng" (Consumer) dữ liệu, được thiết kế với kiến trúc đa luồng phức tạp để đảm bảo giao diện không bị "đơ" khi xử lý mạng.

3.3.1. Kiến trúc đa luồng

+ Client sử dụng 2 luồng chính hoạt động song song:
    - **Luồng giao diện**: Xử lý sự kiện người dùng (Click chuột, kéo thanh tiến độ). Vẽ hình ảnh (render) lên màn hình. Quản lý logic Buffering và Caching.

- **Luồng mạng**: Chạy ngầm (Background). Liên tục lắng nghe cổng UDP. Thực hiện lắp ráp gói tin (Reassembly) và lọc lỗi (Packet Loss Detection).

3.3.2.ServerWorker.py (Session Controller):

Quản lý máy trạng thái (State Machine) của phiên RTSP. Điều tiết tốc độ gửi dữ liệu RTP thông qua cơ chế Burst Mode (gửi theo chùm) để cân bằng tải mạng.

3.3.3.VideoStream.py (Data Access Layer):

Chịu trách nhiệm đọc và truy xuất dữ liệu từ file vật lý. Khi khởi tạo, module này quét toàn bộ file video để tạo bảng chỉ mục vị trí (Byte Offset Index), hỗ trợ tính năng truy cập ngẫu nhiên (Random Access/Seek) tức thì.

3.3.4.RtpPacket.py (Encapsulation):

Đóng gói dữ liệu thô vào cấu trúc gói tin RTP chuẩn (RFC 1889). Xử lý logic Phân mảnh (Fragmentation): Cắt các khung hình kích thước lớn (HD Frame) thành các gói tin nhỏ phù hợp với MTU mạng.

3.3.5.Cấu trúc dữ liệu quan trọng

+ **Hàng đợi Frame (frameQueue):**
    - Kiểu dữ liệu: Queue
    - Vai trò: Vùng đệm trung gian giữa Luồng mạng và Luồng giao diện. Giúp triệt tiêu độ trễ mạng (Jitter).

+ **Bộ nhớ đệm Frame (frame_cache):**
    - Kiểu dữ liệu: Dictionary (Key: FrameNum, Value: ImageData)

- Vai trò: Lưu trữ cục bộ các khung hình đã giải mã. Hỗ trợ tính năng tua ngược tức thì (Instant Replay).

3.4. Thiết kế chi tiết từ phía Server

+ Quy trình từ lúc dữ liệu nằm trên đĩa cứng Server đến khi hiển thị trên màn hình Client trải qua các bước:
+ **Data Extraction:** VideoStream đọc frame -> ServerWorker.
+ **Fragmentation:** Frame lớn -> Cắt thành nhiều mảnh nhỏ.
+ **Encapsulation:** Chunks + Header -> RTP Packets.
+ **Transmission:** Gửi qua mạng UDP (Burst Mode).
+ **Reassembly (Client):** Nhận Packets -> Kiểm tra Sequence -> Ghép lại thành Frame hoàn chỉnh.
+ **Buffering:** Frame hoàn chỉnh -> Đẩy vào frameQueue.
+ **Caching & Rendering:** Lấy từ Queue -> Lưu vào frame_cache -> Giải mã JPEG -> Hiển thị lên Canvas.

4.CHI TIẾT THỰC HIỆN

4.1. Tiền xử lí dữ liệu

Giai đoạn này nhằm tạo ra định dạng video cần thiết (.Mjpeg) cũng như lập chỉ mục cho tính năng Tua nhanh (Fast seek).

4.1.1 Chuyển đổi định dạng (converter.py)

- Hàm convert_mp4_to_mjpeg sử dụng thư viện OpenCV (cv2) để đọc từng khung hình từ file MP4 gốc.
- Để tối ưu hóa cho mạng UDP, khung hình được Resize về kích thước

nhỏ hơn và được Mã hóa thành JPEG .

- Trước khi ghi dữ liệu ảnh JPEG, Server tính toán kích thước của ảnh, tạo ra một chuỗi 5 ký tự số có đệm . Chuỗi này được ghi làm Header 5 byte trước dữ liệu ảnh thô, tạo nên định dạng .Mjpeg.

4.1.2 Lập chỉ mục (Indexing - VideoStream.py)

- Logic chính được thực hiện trong hàm _build_index. Ngay khi đối tượng VideoStream được khởi tạo, nó quét toàn bộ file .Mjpeg một lần để ghi nhớ vị trí byte (Byte Offset) của mỗi khung hình.
- Các vị trí này được lưu trong danh sách self.frame_offsets. Mục đích của việc lưu trữ này là để cho tính năng tua nhanh đến 1 frame bất kì trong video.

4.2. Khởi tạo và thiết lập kết nối

Giai đoạn này thiết lập kênh điều khiển giữa Server với Client.

4.2.1 Khởi tạo Server (Server.py)

- Tạo một TCP Socket và gắn với một cổng cố định, sau đó chuyển sang chế độ lắng nghe trong 1 vòng lặp vô tận.

- Khi có Client kết nối (rtspSocket.accept()), Server tạo ra một đối tượng ServerWorker và gọi ServerWorker.run() trong 1 luồng riêng biệt, hỗ trợ đa Client.

4.2.2 Khởi tạo Client (ClientLauncher.py)

- Đọc các tham số cấu hình (IP Server, Port RTP, tên file).
- Khởi tạo cửa sổ giao diện Tkinter và gọi class Client để bắt đầu kết nối.

4.2.3 Lệnh Setup

- Client gửi lệnh RTSP SETUP tới Server.
- Hàm processRtspRequest trong ServerWorker.py xử lý lệnh này.
- Server tính toán Tổng số frame của video (totalFrames()) và gửi con số này về cho Client qua Header mở rộng Total-Frames, cho phép Client vẽ thanh thời lượng chính xác.

4.3. Truyền tải dữ liệu (RTP/UDP) – Phía Server

Các hoạt động truyền tải và đóng gói được xử lý trong ServerWorker.py, bao gồm cả logic kiểm soát luồng.

4.3.1 Xử lý Lệnh PLAY và Tua (Random Access):

Khi Server nhận lệnh PLAY trong processRtspRequest, nó kiểm tra xem Client có gửi kèm Header Frame-Num hay không. Nếu có, Server gọi hàm seek(offset) của VideoStream. Hàm này tra cứu chỉ mục (self.frame_offsets) và dùng lệnh file.seek() để nhảy ngay tới vị trí byte đó trong file .Mjpeg.

4.3.2 Phân mảnh (Fragmentation):

- Nằm trong hàm sendRtp, Server giải quyết vấn đề khung hình JPEG quá lớn (50KB–100KB) so với giới hạn gói tin UDP an toàn (khoảng 1.4KB). Dữ liệu ảnh được cắt thành từng mảnh nhỏ, ví dụ 1400 bytes.

4.3.3 Đóng gói RTP (RtpPacket.py):

- Mỗi mảnh dữ liệu được đóng gói với Header 12 byte. Header này bao gồm Sequence Number (để Client phát hiện mất gói) và Timestamp. Lưu ý, Timestamp được sử dụng để truyền số Frame Number tới Client để đồng bộ hóa.

- Bit Marker (Byte 1) được đặt là 1 cho mảnh dữ liệu cuối cùng của một bức ảnh, báo hiệu cho Client biết rằng đã nhận đủ các mảnh để lắp ráp khung hình đó.

4.3.4 Kiểm soát luồng (Burst Mode):

- Trong sendRtp, Server triển khai kỹ thuật truyền tải theo Chùm (Burst Mode) để tối ưu hóa hiệu suất. Server gửi liên tục 20 gói tin, sau đó ngủ 0.002 giây. Điều này giúp Client nạp buffer nhanh hơn tốc độ xem, đồng thời bảo vệ mạng, tránh tình trạng tràn bộ đệm gây mất gói (Packet Loss).

4.4. Giai đoạn xử lí và hiển thị - phía Client

Các hoạt động nhằm tạo ra trải nghiệm người dùng mượt mà được xử lý trong Client.py.

4.4.1 Lắng nghe UDP và chống lỗi

- Hàm listenRtp chạy trong luồng nền, chịu trách nhiệm nhận dữ liệu RTP

thô.

- Nó kiểm tra Sequence Number của từng gói tin. Nếu Sequence Number không liên tiếp, nó đặt cờ discard_current_frame = True và vứt bỏ toàn bộ các mảnh còn lại của bức ảnh đó.Chấp nhận loại bỏ một khung hình thay vì hiển thị ảnh lỗi.
- Khi nhận đủ các mảnh và thấy Marker == 1, Client lấy Timestamp (Frame Number) và đẩy bộ (FrameNum, ImageData) vào hàng đợi frameQueue.

4.4.2 Hiển thị và buffering

- Hàm update_image_loop chạy theo nhịp cố định.
- Cơ chế Re-buffering: Kiểm tra số lượng frame trong frameQueue. Nếu kho hàng (buffer) dưới mức tới hạn (BUFFER_START_THRESHOLD), Client sẽ dừng hình, hiển thị "Buffering..." và chỉ bắt đầu phát lại khi buffer được lấp đầy.
- Client sử dụng số Frame Number được trích xuất từ gói tin để cập nhật thanh tiến độ màu đỏ, đảm bảo đồng bộ chính xác.

4.4.3 Tua thông minh ba lớp

- Đây là tính năng ưu tiên trải nghiệm người dùng. Khi người dùng tua video, Client xử lý theo ba mức độ:
    - Ưu tiên 1 (Cache RAM): Kiểm tra self.frame_cache (bộ nhớ tạm lưu các frame đã xem). Nếu frame có sẵn, hiển thị ngay lập tức (độ trễ bằng 0).
    - Ưu tiên 2 (Buffer Queue): Kiểm tra frameQueue (vùng xám). Nếu frame đã được tải sẵn, Client hiển thị frame đó.
    - Ưu tiên 3 (Server): Chỉ khi frame không có sẵn ở hai nơi trên (vùng

trắng), Client mới gửi lệnh PLAY kèm Frame-Num lên Server để tải lại.

- Pause Giả lập: Khi người dùng bấm Pause, Client chỉ ngừng vẽ hình lên màn hình nhưng không gửi lệnh dừng tới Server. Điều này cho phép Server tiếp tục gửi dữ liệu để Client nạp đầy Buffer, chuẩn bị cho lần phát tiếp theo.

5.KẾT QUẢ THỰC NGHIỆM HỆ THỐNG

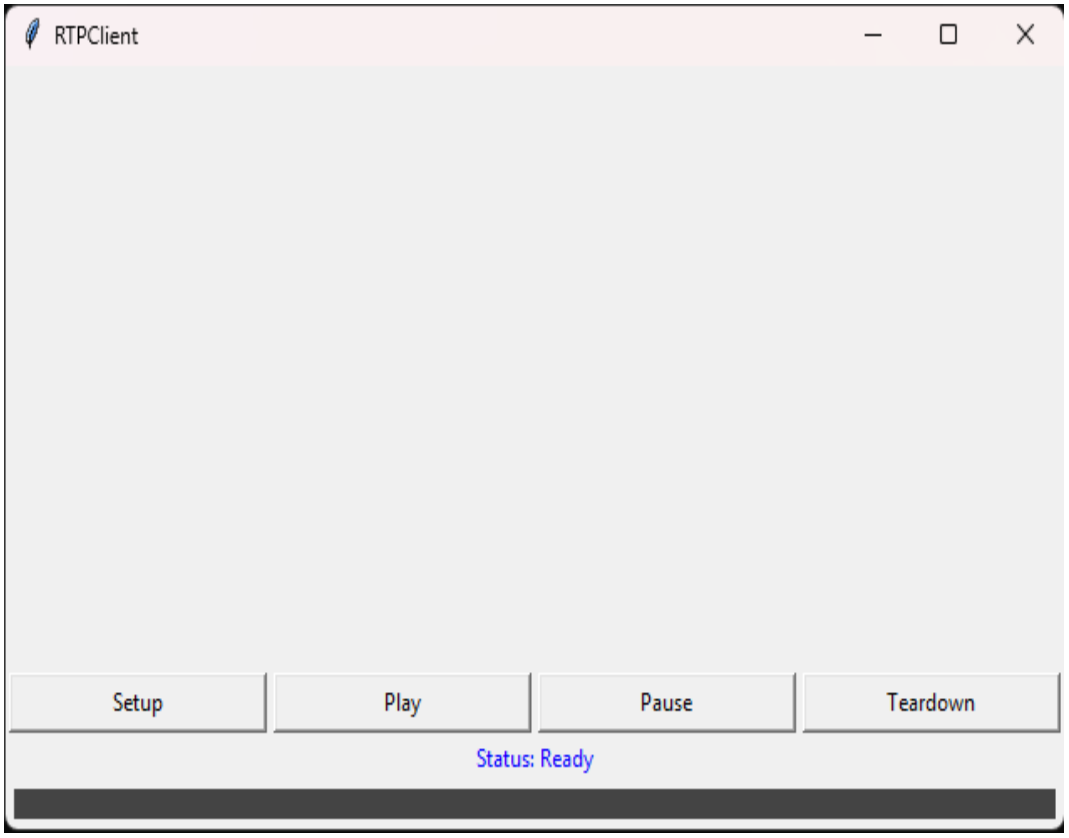
Sau khi hoàn thiện phần cài đặt Server và Client theo mô hình RTSP/RTP, nhóm đã tiến hành chạy thử nghiệm trên máy cá nhân theo cấu hình mạng LAN nội bộ. Các thử nghiệm tập trung vào việc đánh giá mức độ ổn định của phiên RTSP, khả năng truyền frame qua RTP cũng như độ trễ khi phát video.

5.1. Môi trường triển khai (Test Environment)

+ Thông tin phần cứng:
    - Main ASUS B760M - PLUS
    - RAM 32GB
+ Phần mềm sử dụng: Visual studio
+ Thư viện sử dụng: Pillow, OpenCV, Tkinter
+ Dữ liệu thử nghiệm: video đầu vào video.mp4 chuẩn hd 720p/1080p -> file đầu ra movie_hd.Mjpeg đã qua xử lý resize và nén để đảm bảo header < 5 bytes

5.2. Các kịch bản kiểm thử

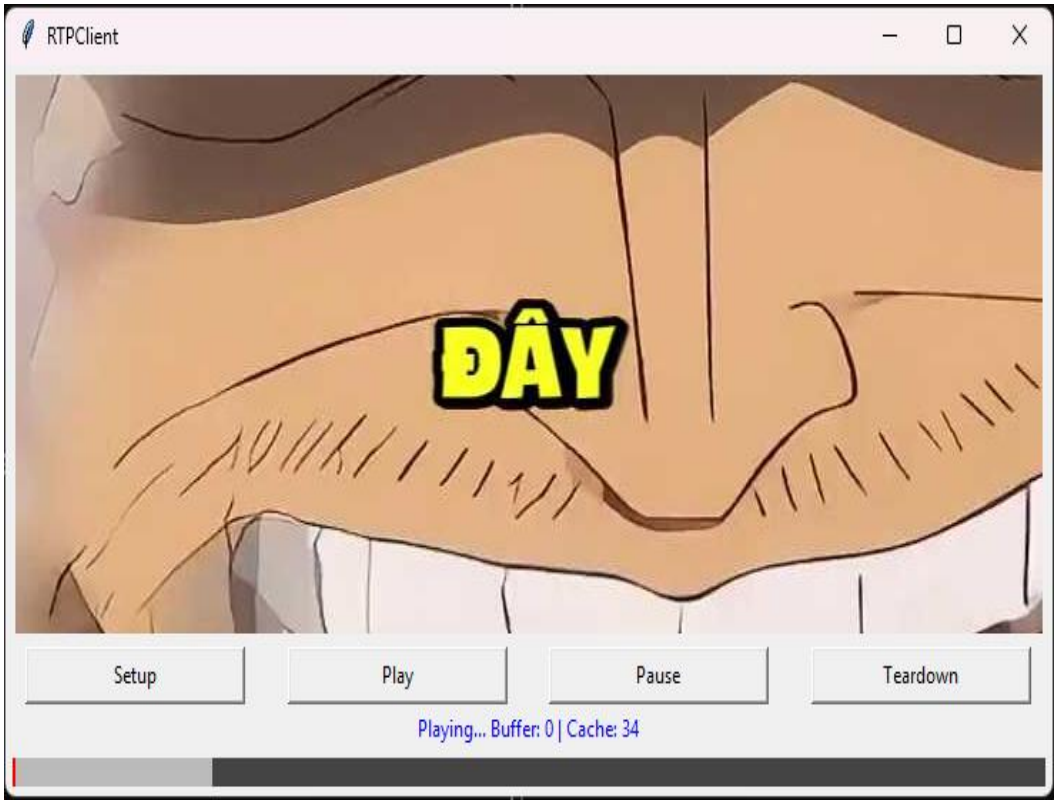
5.2.1.Phát video cơ bản



+ Màn hình giao diện của hệ thống

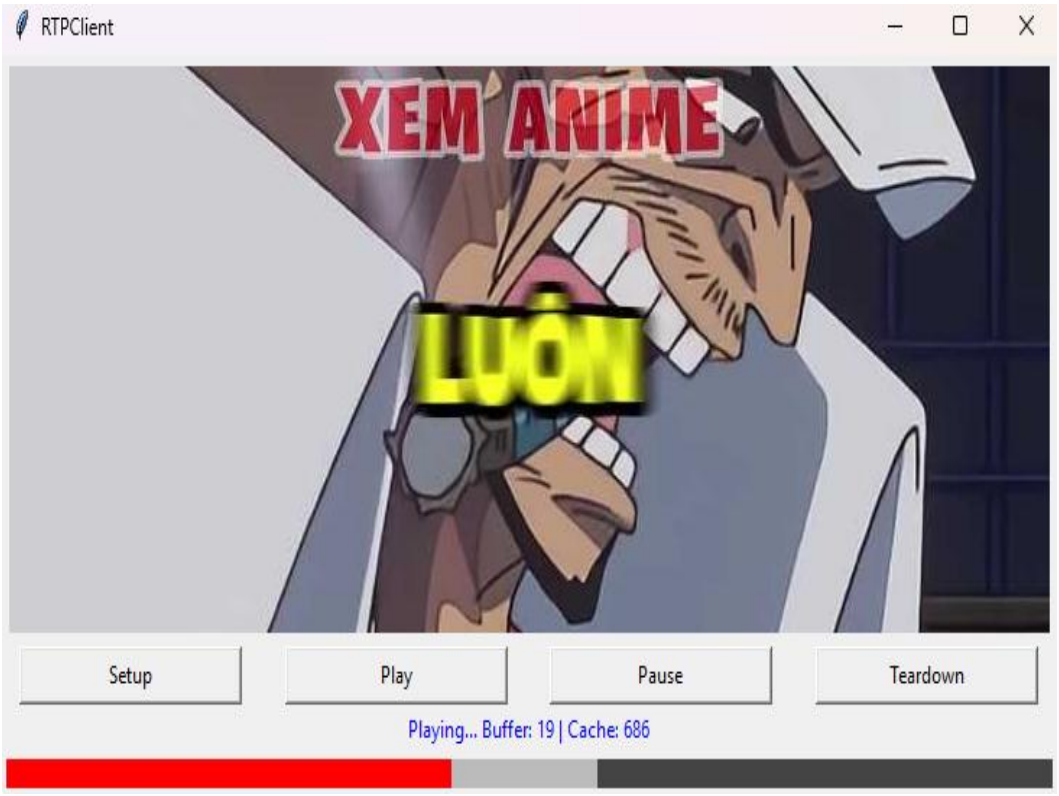
```
Data received:
SETUP movie_hd.Mjpeg RTSP/1.0
CSeq: 1
Transport: RTP/UDP; client_port=5000
Video Info: Total Frames = 1918
```

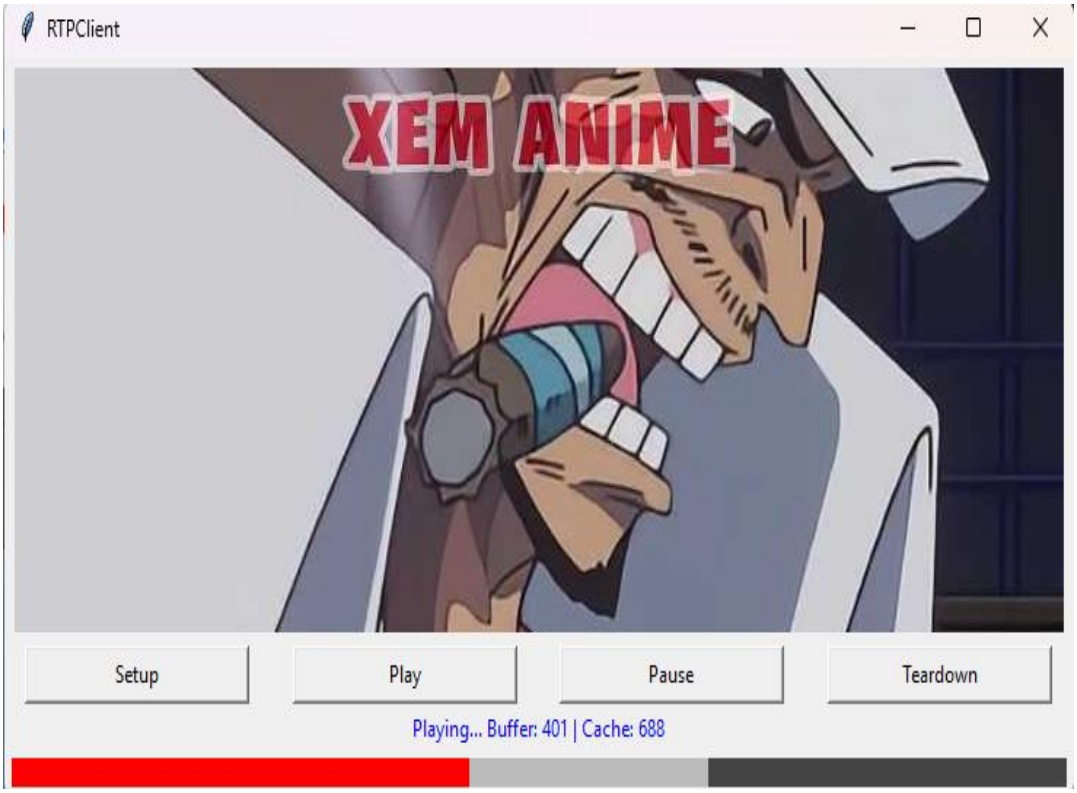
+ Giao diện Client khi bắt đầu phát video. Thông tin 'Total Frames = 1918 được hiển thị trên Terminal chứng tỏ Client đã nhận được Header mở rộng từ Server.



+ Sau khi Nhấn Play thì sẽ đợi Buffer load khoảng 0,5 - 1s để tải đủ 20 buffer rồi video bắt đầu phát

5.2.2.Kiểm soát luồng

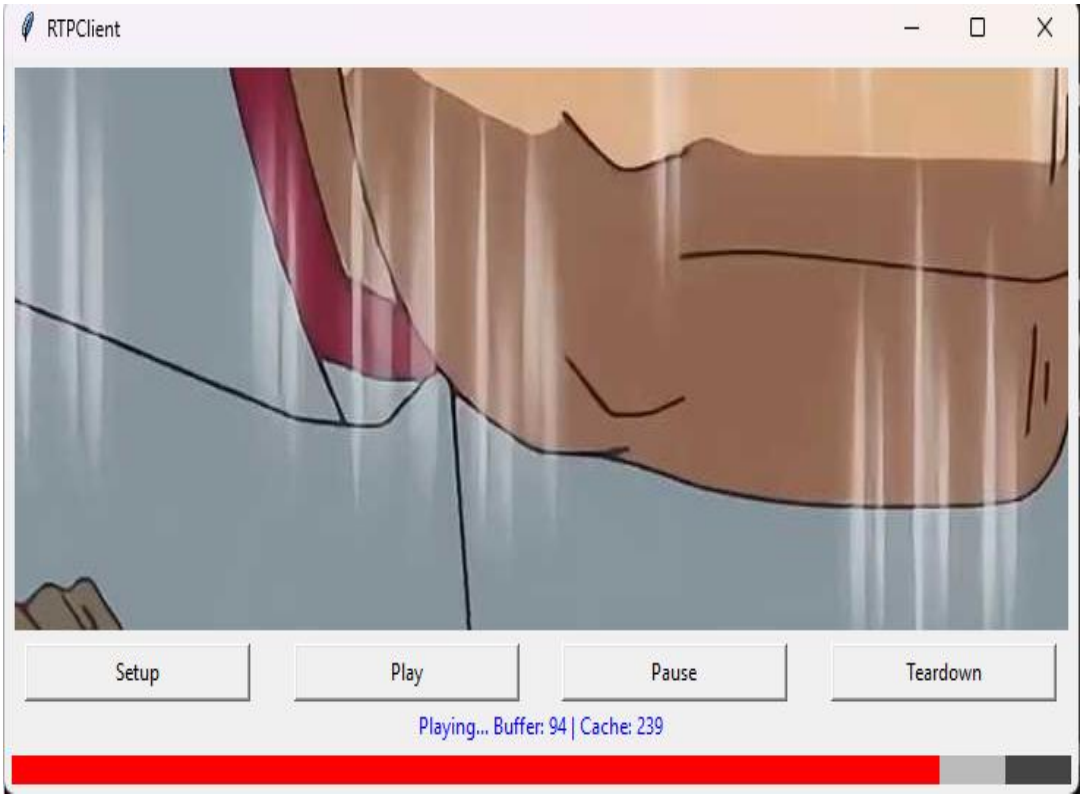


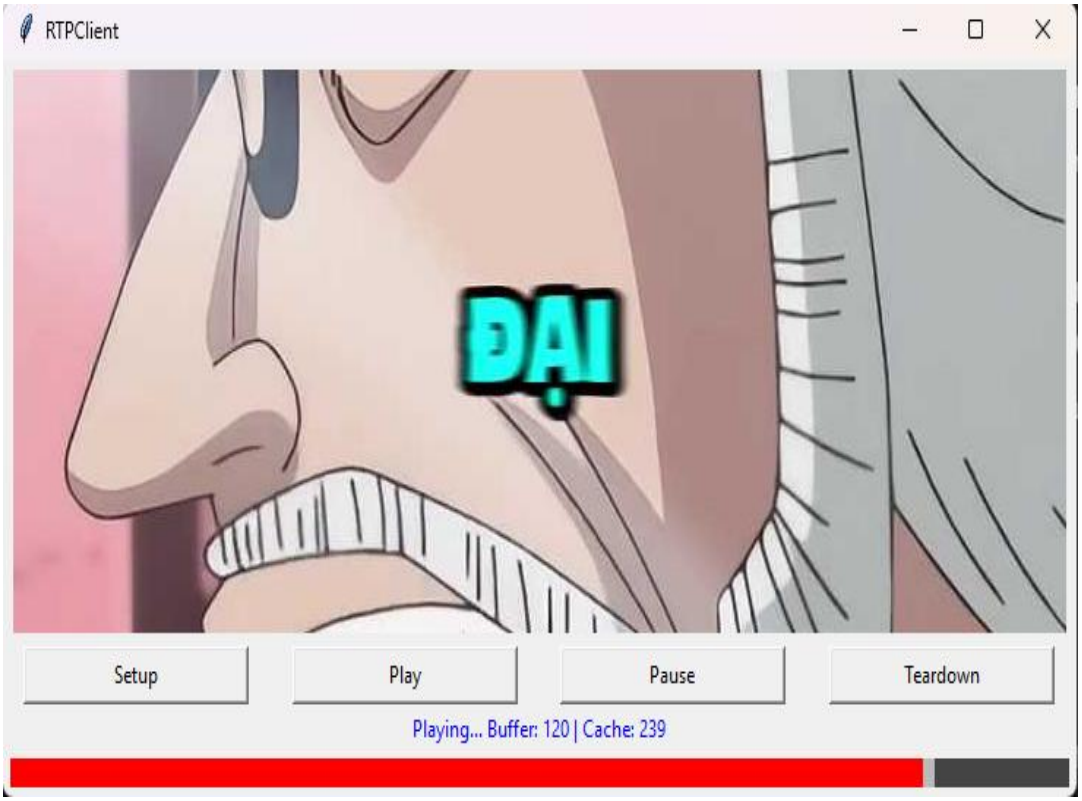



```
--> Paused: Dừng hình, Server vẫn đang nạp Buffer...
```

+ Khi người dùng nhấn Pause, Client ngừng vẽ hình nhưng vẫn duy trì kết nối. Terminal cho thấy bộ đệm (Buffer và bộ nhớ tạm (Cache) vẫn tiếp tục tăng lên (tăng từ 19 lên 401). Điều này chứng minh tính năng Pause giả lập (Simulated Pause) hoạt động hiệu quả, giúp tích lũy dữ liệu cho lần phát tiếp theo

5.2.3.Tua ngược

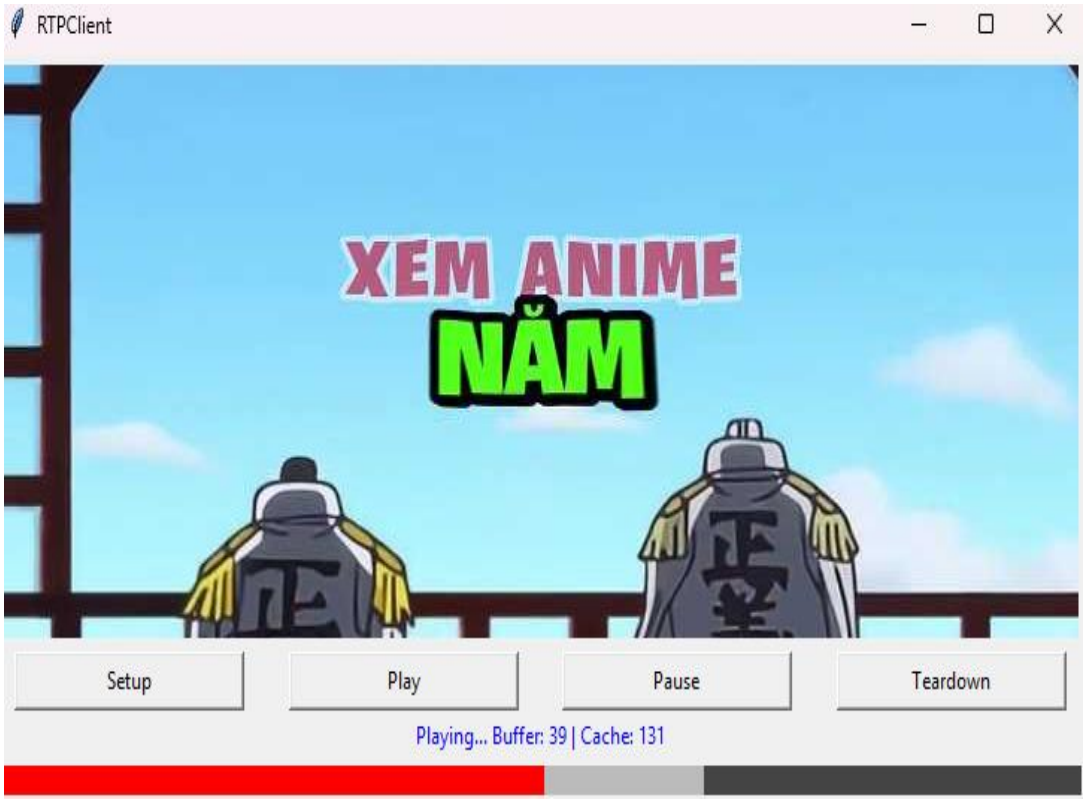




+ Đây là lớp ưu tiên cao nhất trong chức năng tua. Sử dụng để tua ngược về quá khứ (đoạn đã xem)

+ Hệ thống kiểm tra trong từ điển self.frame_cache (nơi lưu trữ các frame đã giải mã). Nếu tìm thấy Target Frame, sẽ lấy dữ liệu hình ảnh từ RAM và vẽ lên màn hình ngay lập từ. Độ trễ gần như bằng 0ms, không tốn băng thông mạng.

5.2.4.Tua nhanh



+ 

+ Đây là lớp ưu tiên thứ 2. Dùng để tua nhanh 1 đoạn ngắn về phía trước nằm trong đoạn xám

+ Hệ thống phát hiện frame đích nằm trong vùng đệm (Buffer Hit). Client thực hiện giải phóng (flush) các frame thừa để nhảy tức thì đến vị trí mới. **Độ trễ ghi nhận xấp xỉ 0ms**.

5.2.5.Tua xa

+ Đây là mức ưu tiên thấp nhất. Dùng để tua đến vùng trắng (chưa tải) hoặc vùng quá khứ đã bị xóa khỏi cache

+ Khi không thấy ở lớp 1 và lớp 2 thì hệ thống sẽ xóa sạch Buffer cũ vì dữ liệu trong đó không còn khớp với vị trí mới. Gửi request PLAY kèm header Frame-Num: <vị trí> lên Server, chuyển sang trạng thái "Buffering" chờ Server nạp đủ dữ liệu mới.

5.2.6. Đồng bộ trạng thái

+ Hệ thống phải đảm bảo hành động tua không làm gián đoạn ý định xem/dừng của người dùng.
+ Mục đích: Khi người dùng PAUSE để soi chi tiết thì sau khi tua video lại tự động PLAY, gây khó chịu.
+ Giải pháp:
+ Trước khi tua, hệ thống lưu trạng thái hiện tại vào biến tạm (was_paused = self.userPaused)
+ Thực hiện tua, chạy thuật toán tua 3 lớp
+ Sau khi tua, nếu was_paused == True -> ép hệ thống giữ nguyên trạng thái PAUSE, chỉ hiển thị khung hình tĩnh tại vị trí mới, nếu was_paused == False -> kích hoạt vòng lặp hiển thị để video chạy tiếp

5.3. Hiệu năng

5.3.1. Độ trễ

+ Start-up Delay: Video bắt đầu sau khoảng 0,5-1s để nạp đủ 20 buffer (BUFFER_START_THRESHOLD). Đảm bảo độ mượt sau khi phát
+ Seek Latency:
    - Vùng Cache/Buffer: 0ms
    - Vùng Server: 0,5 - 1s bằng với Start-up Delay

5.3.2. Tài nguyên hệ thống

+ RAM: tăng dần theo thời gian xem do cơ chế Caching. Tuy nhiên nhờ giới hạn CACHE_LIMIT = 1000, lượng RAM tiêu thụ được kiểm soát ổn định ở mức 50 - 100 MB cho Client, không gây tràn bộ nhớ ngay cả khi xem video dài hàng tiếng đồng hồ.
+ CPU: Tác vụ giải mã JPEG và hình vẽ chiếm lượng CPU vừa phải, giao diện thân thiện với người dùng, phản hồi nhanh, không bị đơ

5.3.3. Độ ổn định

+ Hệ thống chạy ổn định với video dài trên 2000 frames
+ Không xảy ra hiện tượng Crash Server hay Crash Client khi thao tác tua liên tục

5.4. Nhận xét chung

+ Hệ thống đã đáp ứng đầy đủ các yêu cầu khắt khe về xử lý video HD qua giao thức RTP/UDP
+ Các thuật toán tối ưu (Smart Seek, Buffering) đã cải thiện đáng kể trải nghiệm người dùng so với phiên bản cơ bản
+ Vẫn còn nhiều hạn chế với những nền tảng Streaming Video lớn khác như Netflix, Youtube,... như âm thanh, chất lượng hình ảnh, xử lý việc lượng lớn người truy cập 1 thời điểm,....
+ Video test: https://drive.google.com/drive/folders/1iTLFn4orT1-qNFSV51f6UOb_ggL4GA2M?usp=drive_link

6.TÀI LIỆU THAM KHẢO

Lập trình Socket cơ bản: https://realpython.com/python-sockets/

Đa luồng trong Python: https://stackoverflow.com/questions/68425239/how-to-handle-multithreading-with-sockets-in-python

Lập trình GUI bằng Python, sử dụng Tkinter: https://www.youtube.com/watch?v=m9kzSDlzpLE&list=PLBfgLd7V3wpXXBLgiQExUPehRxmAWk2XY

Sử dụng Tkinter để lập trình GUI: https://www.tutorialspoint.com/python/python_gui_programming.htm

Các tài liệu thực hành được đăng tải trên Moodle của môn Mạng máy tính 24CTT3 – Khoa Công nghệ Thông tin – Trường Đại học Khoa học Tự nhiên – ĐHQG TPHCM.