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 MÔN HỌC MẠNG MÁY TÍNH

Đồ án lập trình Socket

TRUYỀN PHÁT VIDEO BẰNG GIAO THỨC RTSP VÀ RTP

Giảng viên hướng dẫn: **Lê Hà Minh**

Sinh viên thực hiện: 1. Phan Chí Cao (Nhóm trưởng) - 24120026

2. Nguyễn Hoàng Nhật - 24120110

Thành phố Hồ Chí Minh, 12/2025

Mục lục

1 Lời mở đầu

1.1 Tổng quan

Trong kỷ nguyên số, Video Streaming đã trở thành dịch vụ chiếm tỷ trọng lưu lượng băng thông lớn trên toàn cầu, là nền tảng cốt lõi cho các ứng dụng phổ biến như YouTube, Netflix hay các hệ thống giám sát an ninh (CCTV). Khác biệt với cơ chế tải xuống tập tin truyền thống (File Download), streaming đặt ra những yêu cầu kỹ thuật khắt khe về tính thời gian thực (Real-time), độ trễ thấp và khả năng thích ứng với sự biến động của đường truyền mạng.

Để đáp ứng các yêu cầu này, hệ thống sử dụng bộ giao thức chuyên biệt thay cho HTTP truyền thống: **RTSP (Real Time Streaming Protocol)** để điều khiển phiên làm việc và **RTP (Real-time Transport Protocol)** để vận chuyển dữ liệu đa phương tiện.

Đồ án **"9:53 AM Socket Project"** được xây dựng nhằm nghiên cứu và thực hành Lập trình mạng (Socket Programming), hướng tới việc mô phỏng kiến trúc của một hệ thống Video Streaming cơ bản — từ việc xử lý đóng gói gói tin (packet encapsulation) cho đến xây dựng giao diện tương tác người dùng.

1.2 GitHub Repository Link

Toàn bộ mã nguồn dự án, bao gồm mã nguồn (Source Code), tài nguyên giao diện (Assets), và hướng dẫn cài đặt chi tiết (README) đã được nhóm tải lên và quản lý phiên bản tại GitHub.

Link: https://github.com/cpgod36/9.53-AM-Socket-Project.git

1.3 Mục tiêu đồ án

Đồ án tập trung hoàn thành các yêu cầu cốt lõi của môn học Mạng máy tính, đồng thời mở rộng thêm các tính năng nâng cao để tối ưu hóa trải nghiệm người dùng.

Mục tiêu nền tảng (Core Requirements)

Xây dựng mô hình Client-Server sử dụng thư viện Python Socket, tuân thủ các chuẩn giao thức:

- **Giao thức điều khiển (RTSP over TCP):** Hiện thực hóa máy trạng thái (State Machine) để quản lý phiên làm việc thông qua các phương thức: SETUP, PLAY, PAUSE, TEARDOWN.

- **Giao thức truyền tải (RTP over UDP):** Xây dựng cơ chế đóng gói và vận chuyển dữ liệu video định dạng MJPEG.

Mục tiêu nâng cao (Advanced Optimization)

Tập trung xử lý các vấn đề thực tế của môi trường mạng UDP:

- **Xử lý Video độ phân giải cao (HD Streaming):** Khắc phục giới hạn kích thước truyền tải tối đa (MTU) của mạng.
  - *Phía Server:* Cài đặt thuật toán **Phân mảnh gói tin (Packet Fragmentation)** để chia nhỏ các frame kích thước lớn (720p/1080p).
  - *Phía Client:* Cài đặt thuật toán **Tái lập (Reassembly)** để ghép nối dữ liệu, đảm bảo tái tạo khung hình hoàn chỉnh.
- **Cơ chế ổn định luồng (Jitter Buffer):** Thiết kế bộ đệm phía Client (Client-side Buffer) nhằm giảm thiểu hiện tượng rung pha (Jitter) và tác động của việc mất gói tin (Packet Loss), giúp video phát ổn định hơn.

Hiện đại hóa ứng dụng (Modern UI/UX)

Nâng cấp giao diện người dùng từ thư viện Tkinter sang **PyQt6 Framework**:

- **Thiết kế Responsive:** Giao diện tự động thích ứng với kích thước cửa sổ theo tỉ lệ 16:9.
- **Dashboard giám sát:** Tích hợp hiển thị các thông số kỹ thuật thời gian thực:
  - Tốc độ khung hình (FPS).
  - Tỷ lệ mất gói tin (Packet Loss Rate).
  - Trạng thái bộ đệm (Buffer Health).

1.4 Giới thiệu sản phẩm "9:53 AM Socket Project"

Sản phẩm là ứng dụng Desktop hoàn chỉnh, bao gồm hai thành phần:

- **Server:** Đóng vai trò lưu trữ, xử lý file video, cắt nhỏ gói tin và phân phối luồng dữ liệu (Streaming).
- **Client:** Đóng vai trò nhận dữ liệu, ghép nối gói tin, xử lý đệm (Buffering) và hiển thị.

Các tính năng nổi bật:

- Stream video chuẩn HD mượt mà qua mạng LAN/Localhost.
- Giao diện phong cách Cyberpunk hiện đại.
- Bảng điều khiển (Dashboard) hiển thị trạng thái mạng và Log giao thức chi tiết.
- Quản lý phiên linh hoạt: Tự động kết nối, Phát lại (Replay) và Chuyển đổi video (Switch File).

1.5 Phân chia công việc

Dự án là kết quả của sự phối hợp chặt chẽ giữa tư duy hệ thống (System Thinking) và tư duy sản phẩm (Product Thinking). Nhóm thực hiện gồm 2 thành viên thuộc lớp **24CTT6**, trường **Đại học Khoa học Tự nhiên, ĐHQG-HCM**:

Thành viên	Vai trò chuyên môn	Trách nhiệm chi tiết
Phan Chí Cao	**Trưởng nhóm (Team Leader)** Frontend & UI Architect	• **Kiến trúc hệ thống:** Thiết kế mô hình MVC và luồng tương tác người dùng. • **UI/UX Design:** Thiết kế giao diện trên Figma, hiện thực hóa bằng PyQt6, xử lý các hiệu ứng đồ họa nâng cao (Glow, Responsive). • **Integration:** Tích hợp logic xử lý vào giao diện, đảm bảo trải nghiệm mượt mà. • **Documentation:** Soạn thảo tài liệu kỹ thuật và báo cáo.
Nguyễn Hoàng Nhật	**Đảm nhận Backend (Backend Engineer)** Core Protocol Developer	• **Protocol Implementation:** Hiện thực hóa các class xử lý gói tin RTP và máy trạng thái RTSP. • **Algorithm Optimization:** Phát triển thuật toán phân mảnh gói tin (Fragmentation) cho video HD và tối ưu hóa bộ đệm (Jitter Buffer). • **System Performance:** Xử lý đa luồng (Multi-threading) cho Server và Client để đảm bảo hiệu năng thời gian thực.

2 Cơ sở lý thuyết

Chương này trình bày các nền tảng lý thuyết mạng máy tính được áp dụng trong đồ án, tập trung phân tích sâu vào kiến trúc Socket hỗn hợp (Hybrid Architecture) và cấu trúc chi tiết của bộ giao thức RTSP/RTP.

2.1 Lập trình Socket và Kiến trúc Hỗn hợp

Khái niệm Socket

Socket là điểm đầu cuối (endpoint) trong liên kết truyền thông hai chiều giữa hai chương trình chạy trên mạng. Một socket được xác định duy nhất bởi sự kết hợp của Địa chỉ IP (xác định máy tính) và Số hiệu cổng (Port number) (xác định tiến trình ứng dụng).

Mô hình Client-Server trong đồ án hoạt động dựa trên cơ chế này: Server tạo socket, gắn vào một cổng cụ thể và lắng nghe yêu cầu; Client tạo socket và chủ động kết nối tới Server.

Kiến trúc Hỗn hợp TCP và UDP (Hybrid Architecture)

Thay vì chỉ dựa vào một giao thức truyền tải duy nhất, đồ án áp dụng kiến trúc tách biệt giữa **Mặt phẳng Điều khiển (Control Plane)** và **Mặt phẳng Dữ liệu (Data Plane)**:

1. **Mặt phẳng Điều khiển - Sử dụng TCP (RTSP):**
    - *Lý do:* Các lệnh điều khiển trạng thái (Play, Pause...) yêu cầu độ chính xác tuyệt đối. Một lệnh bị mất hoặc đến sai thứ tự có thể làm hỏng toàn bộ phiên làm việc.
    - *Cơ chế:* TCP đảm bảo truyền tin cậy (Reliable), hướng kết nối (Connection-oriented) và kiểm soát luồng (Flow Control), đảm bảo Client và Server luôn đồng bộ trạng thái.
2. **Mặt phẳng Dữ liệu - Sử dụng UDP (RTP):**
    - *Lý do:* Video streaming ưu tiên tính thời gian thực (Real-time). UDP hoạt động theo cơ chế "nỗ lực tối đa" (Best-effort), không có cơ chế truyền lại (Retransmission) gây trễ, giúp giảm độ trễ (Latency) xuống mức thấp nhất.
    - *Xử lý:* Các vấn đề cố hữu của UDP như mất gói hay sai thứ tự sẽ được xử lý ở tầng Ứng dụng thông qua Sequence Number của RTP.

2.2 Giao thức RTSP (Real Time Streaming Protocol)

Bản chất giao thức

RTSP (RFC 2326) là giao thức tầng ứng dụng dạng văn bản (Text-based), có cú pháp tương tự HTTP. Điểm khác biệt cốt lõi là RTSP có tính **lưu trạng thái (Stateful)**. Server phải duy trì một mã định danh phiên (Session ID) để theo dõi trạng thái hiện tại của Client.

Cấu trúc bản tin (Message Structure)

Mỗi bản tin RTSP bao gồm dòng yêu cầu (Request Line), các dòng tiêu đề (Headers) và dòng trống kết thúc.

- **CSeq (Sequence Number):** Số thứ tự của lệnh. Bắt buộc phải có trong mọi gói tin để khớp cặp yêu cầu-phản hồi (Request-Response Pairing).
- **Session:** Chuỗi định danh duy nhất do Server sinh ra sau lệnh SETUP, dùng để phân biệt các Client khác nhau.
- **Transport:** Quy định phương thức truyền tải (RTP/AVP;unicast) và cặp cổng (client_port) để nhận dữ liệu.

Máy trạng thái (State Machine)

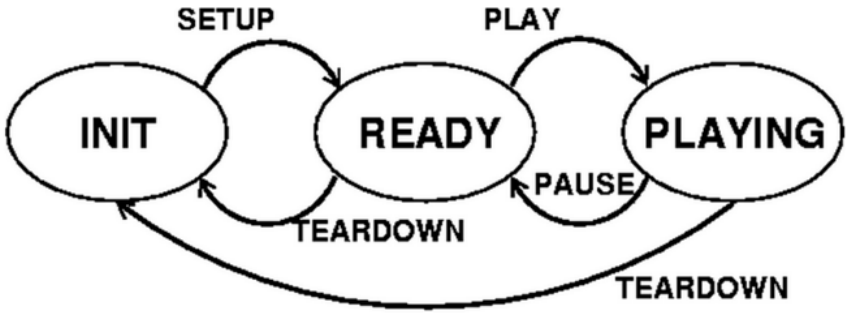
Hệ thống chuyển đổi giữa 3 trạng thái chính:

- **INIT:** Trạng thái khởi tạo. Chưa có tài nguyên nào được cấp phát.
- **READY:** Trạng thái chờ. Server đã khởi tạo socket UDP, phiên làm việc đã được thiết lập (có Session ID), video đang ở vị trí đầu hoặc vị trí tạm dừng.
- **PLAYING:** Trạng thái hoạt động. Server liên tục đọc file và gửi các gói RTP qua socket UDP. Client liên tục nhận và hiển thị.

2.3 Giao thức RTP (Real-time Transport Protocol)

Vai trò của RTP

Trong khi UDP chỉ lo việc chuyển dữ liệu đi, RTP (RFC 1889) cung cấp các thông tin ngữ cảnh cần thiết để tái tạo lại dòng dữ liệu đó ở đích đến.



Hình 1: Sơ đồ chuyển đổi trạng thái RTSP

Cấu trúc Header 12 Bytes

Trong đồ án, Header RTP đóng vai trò cốt lõi giúp Client xử lý các vấn đề của mạng IP không tin cậy.

Cấu trúc Header 12 Bytes

Trong đồ án, Header RTP tuân thủ chuẩn RFC 1889 với tổng kích thước 12 bytes. Dưới đây là ý nghĩa chi tiết của từng trường dữ liệu được đóng gói:

Cơ chế MJPEG qua RTP

MJPEG (Motion JPEG) là định dạng video mà mỗi khung hình là một ảnh JPEG độc lập. Do kích thước một ảnh JPEG (đặc biệt là HD) thường lớn hơn MTU (Maximum Transmission Unit - thường là 1500 bytes) của mạng Ethernet, việc truyền tải nguyên vẹn một frame qua UDP là bất khả thi.

Do đó, RTP Header đóng vai trò như "bản đồ lắp ghép", cho phép chia cắt một frame lớn thành hàng chục gói nhỏ và lắp lại chính xác tại phía nhận thông qua cờ **Marker** và **Sequence Number**.

3 Kiến trúc hệ thống và Giải thuật

Chương này trình bày chi tiết về tổ chức mã nguồn và các giải thuật cốt lõi được áp dụng để giải quyết bài toán truyền tải video thời gian thực.

Trường (Field)	Size	Mô tả và Giá trị thiết lập
Version (V)	2 bits	Phiên bản giao thức RTP. Luôn được thiết lập giá trị = **2**.
Padding (P)	1 bit	Cờ độn. Nếu P=1, gói tin có thêm các byte đệm ở cuối. Trong đồ án, thiết lập P=**0**.
Extension (X)	1 bit	Cờ mở rộng header. Trong đồ án, thiết lập X=**0** (không dùng header phụ).
CSRC Count (CC)	4 bits	Số lượng nguồn góp (Contributing Sources). Do chỉ có 1 nguồn phát video, thiết lập CC=**0**.
Marker (M)	1 bit	**Đây là cờ đánh dấu quan trọng.** - $M = 0$: Gói tin là mảnh dữ liệu nằm giữa frame. - $M = 1$: Gói tin là mảnh cuối cùng của frame. Client dựa vào đây để kết thúc việc ghép mảnh và đẩy ảnh vào Buffer.
Payload Type (PT)	7 bits	Định dạng dữ liệu tải trọng. Sử dụng mã **26** tương ứng với chuẩn Motion JPEG (MJPEG).
Sequence Number	16 bits	Số thứ tự gói tin, tăng dần (ví dụ: 100, 101, 102...). Dùng để phát hiện mất gói (Packet Loss) và sắp xếp lại gói tin.
Timestamp	32 bits	Nhãn thời gian lấy mẫu. Các gói tin thuộc cùng một Frame video sẽ có Timestamp giống hệt nhau.
SSRC Identifier	32 bits	Synchronization Source ID. Một số ngẫu nhiên để định danh Server phát luồng video.

Bảng 1: Chi tiết các trường trong Header RTP

3.1 Mô hình Kiến trúc MVC

Phía Client được thiết kế tuân thủ nghiêm ngặt mô hình **Model-View-Controller**:

- **Model (RtspCore):** Xử lý logic mạng, gửi nhận gói tin.
- **View (GUI):** Hiển thị hình ảnh và thông số.
- **Controller:** Điều phối sự kiện nút bấm.

3.2 Giải thuật Phân mảnh gói tin (Fragmentation)

Để truyền tải video HD qua mạng UDP với giới hạn MTU, Server thực hiện thuật toán cắt nhỏ dữ liệu. Dưới đây là đoạn mã hiện thực tại lớp `ServerWorker`:

```
1    # Maximum payload size (bytes)
2    MAX_RTP_PAYLOAD = 1400
3
4    datalen = len(data)
```

```
    currentTimestamp = int(time.time())

    currPos = 0
    while currPos < datalen:
        # Calculate chunk size
        chunkSize = min(MAX_RTP_PAYLOAD, datalen - currPos)

        # Slice the data
        chunk = data[currPos : currPos + chunkSize]
        currPos += chunkSize

        # Marker Bit: 1 for the last chunk, 0 otherwise
        marker = 1 if currPos >= datalen else 0

        # Send packet via UDP
        self.clientInfo['rtpSocket'].sendto(
            self.makeRtp(chunk, currentSeqNum, marker, currentTimestamp),
            (address, port)
        )
```

Listing 1: Thuật toán phân mảnh gói tin RTP phía Server

3.3 Giải thuật Jitter Buffer và Tái lập (Reassembly)

Phía Client sử dụng một bộ đệm để ghép nối các mảnh dữ liệu và ổn định luồng video.

```
    # Check the Marker bit in RTP Header (Byte 1, Bit 7)
    if rtpPacket.header[1] >> 7 == 1:

        # If Marker = 1, it indicates the end of a frame
        if len(current_frame_buffer) > 0:
            # Package data and push to the Jitter Buffer
            frame_tuple = (current_frame_buffer[:], packet_count, loss)
            self.jitter_buffer.put(frame_tuple)

        # Reset temporary buffer for the next frame
```

```
        current_frame_buffer = bytearray()
```

Listing 2: Logic ghép gói tin và xử lý Buffer phía Client

3.4 Cơ chế Pre-buffering (Nạp trước)

Để đảm bảo video khởi chạy mượt mà và tránh hiện tượng giật cục ngay khi bắt đầu, hệ thống áp dụng cơ chế **Pre-buffering**.

Khi người dùng nhấn Play hoặc chuyển video, Client sẽ không hiển thị hình ảnh ngay lập tức. Thay vào đó, nó sẽ chuyển sang trạng thái "Buffering" và đợi cho đến khi hàng đợi tích lũy đủ một lượng khung hình nhất định (Threshold) trước khi bắt đầu xả dữ liệu ra màn hình.

```
    # Check if the system is in buffering mode
    if self.is_buffering:

        # If buffer size is below threshold (e.g., 60 frames)
        if current_buf_size < self.BUFFER_THRESHOLD:
            return # Keep waiting, do not render video yet
        else:
            # Threshold reached, disable buffering flag
            self.is_buffering = False
            # Start playback
```

Listing 3: Logic Pre-buffering tại giao diện

4 Kết quả thực nghiệm

Sau quá trình hiện thực hóa và kiểm thử, nhóm đã xây dựng thành công ứng dụng Video Streaming đáp ứng đầy đủ các yêu cầu đề ra.

4.1 Môi trường kiểm thử

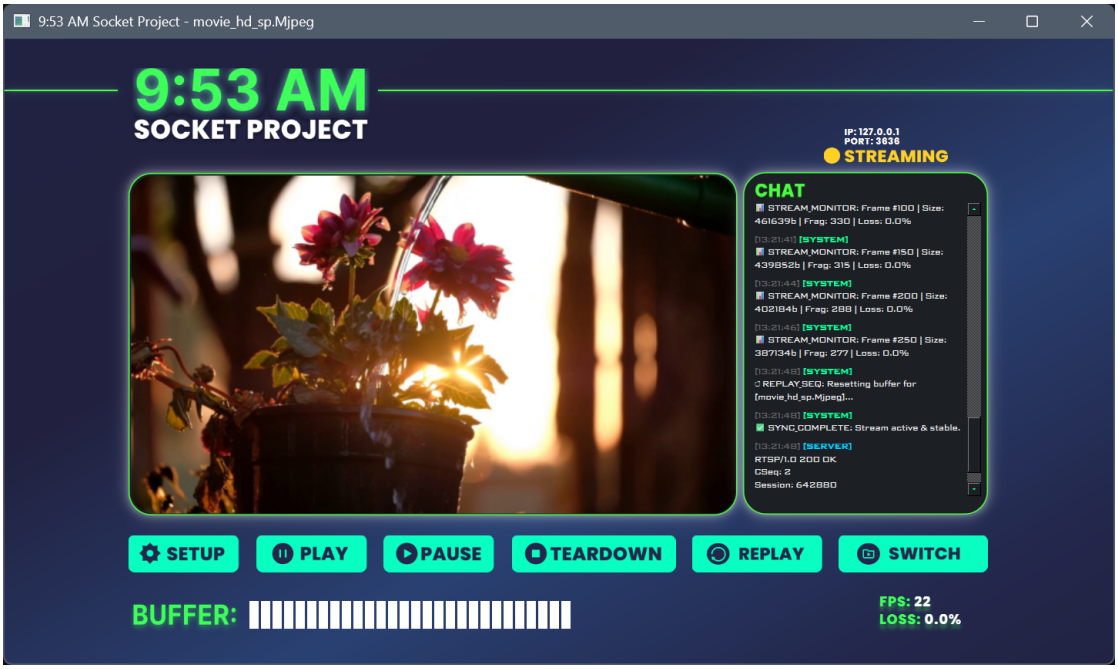
Hệ thống được vận hành và kiểm tra ổn định trên môi trường sau:

- **Hệ điều hành:** Windows 10/11.
- **Ngôn ngữ:** Python 3.x (Thư viện PyQt6, Pillow).

- **Video đầu vào:** Chuẩn MJPEG HD (720p/1080p).

4.2 Giao diện người dùng (Modern UI)

Giao diện được thiết kế theo phong cách Cyberpunk, bố cục Responsive thích ứng với kích thước cửa sổ. Các thông số kỹ thuật (FPS, Packet Loss) được cập nhật theo thời gian thực.

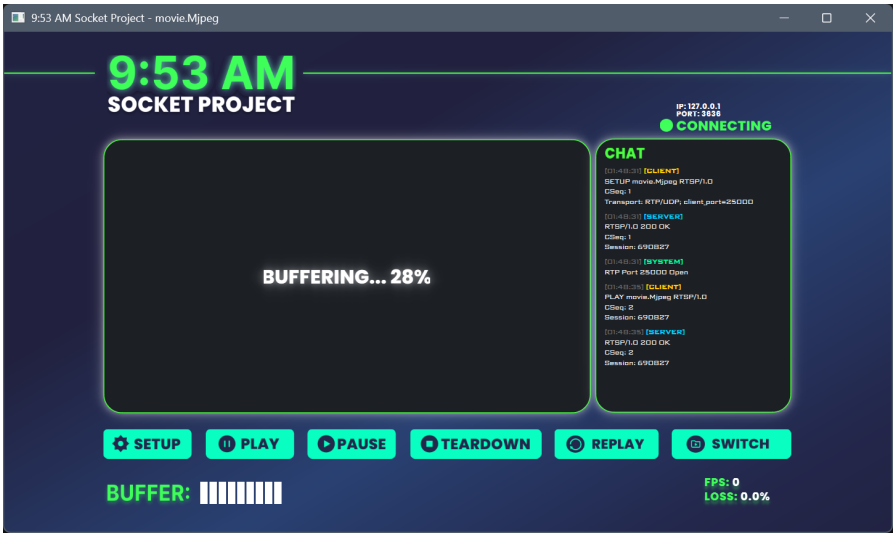


Hình 2: Giao diện chính khi đang Streaming video HD

4.3 Minh chứng các tính năng nâng cao

Cơ chế Đệm (Buffering)

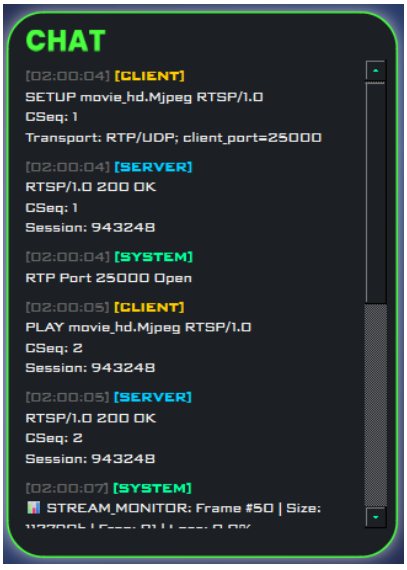
Hình ảnh dưới đây minh họa trạng thái Pre-buffering. Khi bắt đầu phát hoặc chuyển video, hệ thống hiển thị lớp phủ (Overlay) thông báo tiến trình nạp dữ liệu vào Jitter Buffer.



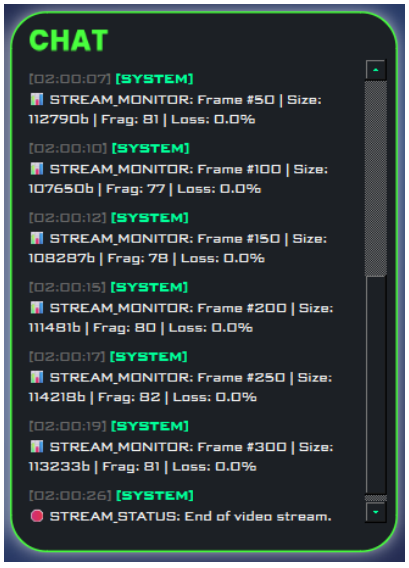
Hình 3: Trạng thái nạp bộ đệm (Pre-buffering) trước khi phát

Nhật ký hoạt động (Activity Log)

Hệ thống Log tích hợp trên giao diện cung cấp thông tin chi tiết về trạng thái hệ thống theo thời gian thực. Hình 4 minh họa quá trình bắt tay giao thức (SETUP, PLAY), trong khi Hình 5 chứng minh kỹ thuật phân mảnh thông qua thông số **Frag** (số lượng gói tin con) và **Size** của từng khung hình.



Hình 4: Giao thức RTSP (TCP)



Hình 5: Thông số Phân mảnh (UDP)

5 Tổng kết

5.1 Kết quả đạt được

Sau quá trình nghiên cứu và hiện thực hóa, nhóm đã xây dựng thành công ứng dụng Video Streaming hoàn chỉnh, đáp ứng 100% các yêu cầu của đồ án môn học và tích hợp thêm nhiều tính năng thực tiễn.

Cụ thể, các mục tiêu đã hoàn thành bao gồm:

- **Về Giao thức mạng:** Hiện thực hóa thành công mô hình Client-Server đa luồng, xử lý chính xác các bản tin RTSP (TCP) và vận chuyển dữ liệu video qua RTP (UDP) tốc độ cao.
- **Về Kỹ thuật nâng cao:** Giải quyết triệt để bài toán truyền tải video HD (720p/1080p) thông qua thuật toán **Phân mảnh (Fragmentation)** và cơ chế **Jitter Buffer** thông minh.
- **Về Trải nghiệm người dùng:** Xây dựng giao diện **PyQt6** hiện đại, tích hợp Dashboard giám sát mạng theo thời gian thực.

5.2 Kiến thức và Kinh nghiệm tích lũy

Qua quá trình thực hiện đồ án, nhóm đã gặt hái được những kiến thức và kinh nghiệm quý báu:

- **Tư duy Lập trình mạng chuyên sâu:** Hiểu rõ bản chất khác biệt giữa TCP và UDP, cách kiểm soát luồng dữ liệu (Flow Control) và cách xử lý các vấn đề thực tế của mạng như mất gói (Packet Loss) hay độ trễ (Latency).
- **Kỹ năng làm việc với Giao thức chuẩn:** Học được cách đọc hiểu các tài liệu đặc tả kỹ thuật (RFC 2326, RFC 1889) và chuyển hóa từ lý thuyết thành mã nguồn thực tế.
- **Kiến trúc phần mềm:** Nắm vững cách tổ chức mã nguồn theo mô hình MVC, tách biệt giữa logic xử lý và giao diện hiển thị, giúp code trong sáng và dễ bảo trì.
- **Kỹ thuật Đa luồng (Multi-threading):** Làm chủ kỹ thuật xử lý song song để đảm bảo giao diện luôn mượt mà trong khi hệ thống mạng vẫn hoạt động ngầm liên tục.

Tài liệu tham khảo

[1] James F. Kurose and Keith W. Ross, *Computer Networking: A Top-Down Approach (Global Edition)*, Pearson Education, 2021.

[2] H. Schulzrinne, A. Rao, and R. Lanphier, "Real Time Streaming Protocol (RTSP)," IETF RFC 2326, April 1998.
https://tools.ietf.org/html/rfc2326

[3] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications," IETF RFC 1889, January 1996.
https://tools.ietf.org/html/rfc1889

[4] Riverbank Computing, "PyQt6 Reference Guide Class Documentation"
https://www.riverbankcomputing.com/static/Docs/PyQt6/

[5] Python Software Foundation, "Socket — Low-level networking interface,"
https://docs.python.org/3/library/socket.html