

ĐẠI HỌC QUỐC GIA TP HCM
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA CÔNG NGHỆ THÔNG TIN

Project Socket Programming

Đề tài: Video Streaming with RTSP and RTP

Môn học: Mạng máy tính

Sinh viên thực hiện:

Nguyễn Quốc Nam - 24120098

Võ Hoàng Phúc - 24120123

Chế Nguyễn Thùy Trang - 24120469

Giáo viên hướng dẫn:

ThS. Chung Thùy Linh

Ngày 6 tháng 12 năm 2025



Danh sách thành viên

| Đóng góp | MSSV | Họ và tên | Mã lớp | Nhiệm vụ |
|----------|----------|-----------------------|--------|--|
| 100% | 24120123 | Võ Hoàng Phúc | 24CTT1 | - Xây dựng code thỏa tiêu chí 1. - Tổng hợp tài liệu cho tiêu chí 1. - Đồng bộ code. |
| 100% | 24120098 | Nguyễn Quốc Nam | 24CTT1 | - Xây dựng code thỏa tiêu chí 2. - Tổng hợp tài liệu cho tiêu chí 2. - Viết báo cáo. |
| 100% | 24120469 | Chế Nguyễn Thùy Trang | 24CTT1 | - Xây dựng code thỏa tiêu chí 3. - Tổng hợp tài liệu cho tiêu chí 3. |

Bảng 1: Danh sách các thành viên và nhiệm vụ

Mục lục

| | | |
|----------|---|-----------|
| 1 | Giới thiệu | 1 |
| 1.1 | Mục tiêu bài lab | 1 |
| 1.2 | Tiêu chí đánh giá | 1 |
| 1.3 | Tổng quan thư mục | 1 |
| 2 | Cơ sở lý thuyết | 1 |
| 2.1 | RTP - Real time transfer protocol | 1 |
| 2.2 | RTSP - Real time streaming protocol | 3 |
| 2.3 | Mjpeg - Motion JPEG | 3 |
| 3 | Quy trình thực hiện | 4 |
| 3.1 | Xây dựng RTP/RTSP server và client (4đ) | 4 |
| 3.1.1 | Ý tưởng | 4 |
| 3.1.2 | Hoàn thiện Client.py - Giao thức RTSP của client | 4 |
| 3.1.3 | Hoàn thiện RtpPacket.py - Giao thức RTP của server | 8 |
| 3.1.4 | Thử nghiệm | 10 |
| 3.2 | Bổ sung HD video streaming (3đ) | 11 |
| 3.2.1 | Ý tưởng | 12 |
| 3.2.2 | Hoàn thiện cơ chế chia nhỏ frame thành các fragment | 12 |
| 3.2.3 | Hoàn thiện cơ chế bỏ qua các frame bị trễ để đảm bảo đồ mượt khi phát video | 12 |
| 3.2.4 | Hoàn thiện tính năng hiện số frame mất và lưu lượng mạng trên giao diện | 12 |
| 3.2.5 | Thử nghiệm | 12 |
| 3.3 | Bổ sung client cache (2.5đ) | 13 |
| 3.3.1 | Ý tưởng | 13 |
| 3.3.2 | Hoàn thiện cơ chế bộ đệm phía client | 13 |
| 3.3.3 | Thử nghiệm | 13 |
| 4 | Kết quả đạt được | 13 |
| 5 | Tài liệu tham khảo | 14 |

Danh sách bảng

| | | |
|---|--|----|
| 1 | Danh sách các thành viên và nhiệm vụ | i |
| 2 | Tiêu chí chấm điểm bài lab | 1 |
| 3 | Giá trị có sẵn của bài lab trong RTP header | 3 |
| 4 | Một số MTU cho các phương tiện truyền tải phổ biến | 12 |

Danh sách hình vẽ

| | | |
|---|--|----|
| 1 | Cấu trúc giao thức RTP [1] | 2 |
| 2 | Sơ đồ trạng thái và chuyển đổi của client RTSP | 4 |
| 3 | Cấu trúc mảng header | 10 |
| 4 | Output RTSP client | 11 |

1 Giới thiệu

1.1 Mục tiêu bài lab

Hiểu được và xây dựng được hệ thống streaming video dùng giao thức RTSP để điều khiển và RTP để truyền tải dữ liệu video từ server đến client.

1.2 Tiêu chí đánh giá

| No. | Yêu cầu | Điểm |
|-----|--|------|
| 1 | Triển khai giao thức RTSP của máy khách và đóng gói tin RTP của máy chủ. | 4đ |
| 2 | Truyền phát video HD | 3đ |
| 3 | Bộ nhớ đệm phía máy khách | 2.5đ |
| 3 | Báo cáo | 0.5đ |

Bảng 2: Tiêu chí chấm điểm bài lab

1.3 Tổng quan thư mục

Bài lab được tổ chức theo các thư mục sau:

- **1_RTSP_RTP**: Chứa source code cho tiêu chí 1.
- **2_HD**: Chứa source code cho tiêu chí 2.
- **3_CACHE**: Chứa source code cho tiêu chí 3.
- **4_REPORT**: Chứa báo cáo của bài lab (tiêu chí 4).
- **video**: Chứa các video với các độ phân giải khác nhau phù hợp với từng tiêu chí.

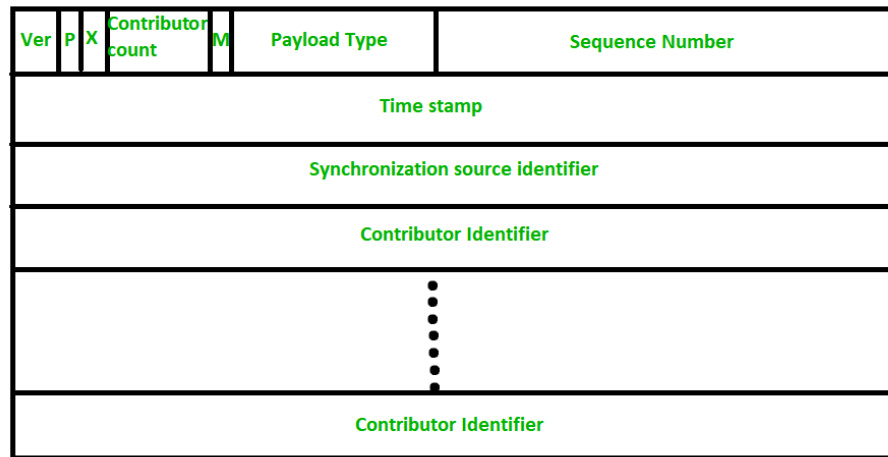
Phần source code cũng đã được up lên [Github](#).

2 Cơ sở lý thuyết

2.1 RTP - Real time transfer protocol

RTP (Real-time Transfer Protocol) là giao thức truyền tải được thiết kế để vận chuyển dữ liệu thời gian thực như video, audio hoặc hình ảnh qua mạng IP. RTP thường hoạt động trên nền tảng UDP

để giảm độ trễ và cho phép truyền tải liên tục, mặc dù đặc tính này có thể dẫn đến mất gói (packet loss). [2]



Hình 1: Cấu trúc giao thức RTP [1]

Gói RTP bao gồm hai phần:

- Header: chứa các thông tin điều khiển quan trọng như:
 - Version: phiên bản của giao thức RTP.
 - Padding: chỉ định xem có thêm byte đệm vào cuối gói hay không.
 - Extension: chỉ định xem có phần mở rộng header hay không.
 - Contributing Source Count: số lượng nguồn đồng thời tham gia vào stream.
 - Marker Bit: đánh dấu các sự kiện quan trọng trong luồng dữ liệu.
 - Payload Type: xác định loại dữ liệu được truyền.
 - Sequence Number: xác định thứ tự gói tin, hỗ trợ phát hiện mất gói.
 - Timestamp: đánh dấu thời gian của gói tin, hỗ trợ đồng bộ hóa.
 - Synchronization Source Identifier: định danh nguồn stream.
 - Contributing Source Identifiers: định danh các nguồn đóng góp vào stream.
- Payload: chứa dữ liệu thực tế của video.

Lưu ý: Một số giá trị đã được quy định sẵn trong giới hạn bài lab

| Ký hiệu | Field | Số bit | Giá trị |
|---------|-----------------------------------|----------------------|----------|
| V | Version | 2 | 2 |
| P | Padding | 1 | 0 |
| X | Extension | 1 | 0 |
| CC | Contributing Source Count | 4 | 0 |
| M | Marker Bit | 1 | 0 |
| PT | Payload Type | 7 | 26 |
| | Sequence Number | 16 | frameNbr |
| | Timestamp | 32 | time() |
| SSRC | Synchronization Source Identifier | 32 | Số tùy ý |
| CSRC | Contributing Source Identifier | 0 | Không có |
| | Tổng số bits | 96 (12 bytes) | |

Bảng 3: Giá trị có sẵn của bài lab trong RTP header

2.2 RTSP - Real time streaming protocol

RTSP (Real-time Streaming Protocol) là giao thức điều khiển cho các phiên truyền tải media. Khác với RTP, RTSP không vận chuyển dữ liệu video mà cung cấp cơ chế "điều khiển từ xa" các hành động như PLAY, PAUSE hay TEARDOWN. RTSP hoạt động trên TCP để đảm bảo tính chính xác của thông điệp điều khiển. [3]

Một phiên RTSP tiêu chuẩn gồm các trạng thái và lệnh sau:

- **SETUP:** yêu cầu server tạo phiên làm việc RTSP và thiết lập port để truyền RTP.
- **PLAY:** yêu cầu server bắt đầu gửi các packet RTP tới client.
- **PAUSE:** tạm dừng việc gửi dữ liệu nhưng giữ phiên RTSP.
- **TEARDOWN:** yêu cầu server kết thúc phiên RTSP và giải phóng tài nguyên.

Client và server giao tiếp bằng mô hình request–response, trong đó mỗi yêu cầu đều kèm theo CSeq (Command Sequence) để đánh số thứ tự và Session ID để xác định phiên làm việc.

2.3 Mjpeg - Motion JPEG

MJPEG (Motion JPEG) là định dạng video được tạo ra bằng cách nối liên tiếp nhiều ảnh JPEG độc lập theo thời gian để tạo thành chuỗi chuyển động. [4]

Cấu trúc của một frame MJPEG bao gồm:

- Mỗi frame được lưu dưới dạng ảnh JPEG đầy đủ.
- Trước mỗi frame có một 5-byte header mô tả độ dài frame.
- JPEG frame bắt đầu bằng 0xFF 0xD8 (SOI) và kết thúc bằng 0xFF 0xD9 (EOI).

3 Quy trình thực hiện

3.1 Xây dựng RTP/RTSP server và client (4đ)

Các file trong phần này nằm trong thư mục **1_RTSP_RTP**

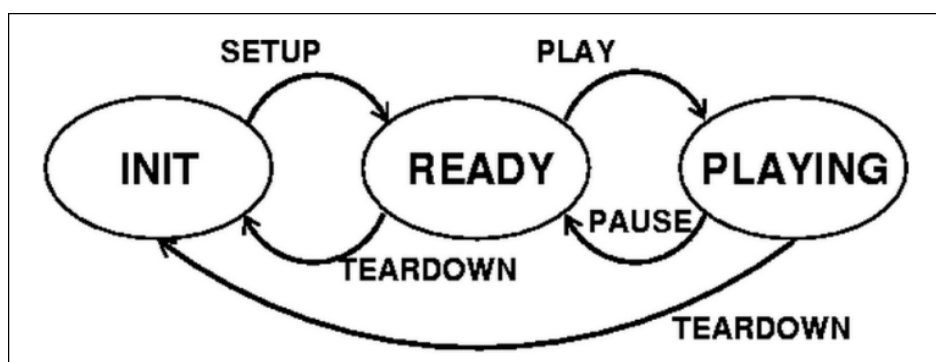
Đây là phiên bản đầu tiên cơ bản nhất của bài lab, yêu cầu xây dựng hệ thống streaming video sử dụng giao thức RTSP để điều khiển và RTP để truyền tải dữ liệu video từ server đến client.

3.1.1 Ý tưởng

Hoàn thiện giao thức RTSP trên client và đóng gói dữ liệu RTP trên server để truyền tải video từ server đến client trên các hàm đã được tạo sẵn.

3.1.2 Hoàn thiện Client.py - Giao thức RTSP của client

Mô hình trạng thái của client:



Hình 2: Sơ đồ trạng thái và chuyển đổi của client RTSP

Hoàn thiện hàm gửi RTSP request - `sendRtspRequest()`:

```
1 import threading
2
3 def sendRtspRequest(self, requestCode):
```

```

4     if requestCode == self.SETUP and self.state == self.INIT:
5         threading.Thread(target=self.recvRtspReply).start()
6         self.rtpSeq += 1
7         request = "SETUP " + str(self.fileName) + " RTSP/1.0\nCSeq: " + str(self
8 .rtpSeq) + "\nTransport: RTP/UDP; client_port= " + str(self.rtpPort)
9         self.requestSent = self.SETUP
10        elif requestCode == self.PLAY and self.state == self.READY:
11            self.rtpSeq += 1
12            request = "PLAY " + str(self.fileName) + " RTSP/1.0\nCSeq: " + str(self
13 .rtpSeq) + "\nSession: " + str(self.sessionId)
14            self.requestSent = self.PLAY
15        elif requestCode == self.PAUSE and self.state == self.PLAYING:
16            self.rtpSeq += 1
17            request = "PAUSE " + str(self.fileName) + " RTSP/1.0\nCSeq: " + str(self
18 .rtpSeq) + "\nSession: " + str(self.sessionId)
19            self.requestSent = self.PAUSE
20        elif requestCode == self.TEARDOWN and not self.state == self.INIT:
21            self.rtpSeq += 1
22            request = "TEARDOWN " + str(self.fileName) + " RTSP/1.0\nCSeq: " + str(
23 self.rtpSeq) + "\nSession: " + str(self.sessionId)
24            self.requestSent = self.TEARDOWN
25        else:
26            return
27
28        self.rtpSocket.send(request.encode())
29        print('\nData sent:\n' + request)

```

- Hàm build chuỗi request. Mỗi lần gửi đều tăng CSeq để server đồng bộ và gán Session để nhận biết sau SETUP. Cụ thể hơn:

- SETUP request: Chương trình sẽ kiểm tra trạng thái hiện tại của client, nếu là INIT thì tạo request SETUP và bắt đầu chạy nền phương thức recvRtspReply (nhận các RTSP reply từ server). Một request SETUP sẽ có dạng:

```

1 SETUP <Video_File> RTSP/1.0
2 CSeq: <CSeq>
3 Transport: RTP/UDP; client_port= <RTP_Port>

```

Ví dụ:

```
1 SETUP video/movie288.Mjpeg RTSP/1.0
2 CSeq: 1
3 Transport: RTP/UDP; client_port= 8090
```

- PLAY request: Chương trình sẽ kiểm tra trạng thái hiện tại của client, nếu là READY thì tạo request PLAY có dạng:

```
1 PLAY <Video_File> RTSP/1.0
2 CSeq: <CSeq>
3 Session: <Session_ID>
```

Ví dụ:

```
1 PLAY video/movie288.Mjpeg RTSP/1.0
2 CSeq: 2
3 Session: 619227
```

- PAUSE request: Chương trình sẽ kiểm tra trạng thái hiện tại của client, nếu là PLAYING thì tạo request PAUSE có dạng:

```
1 PAUSE <Video_File> RTSP/1.0
2 CSeq: <CSeq>
3 Session: <Session_ID>
```

Ví dụ:

```
1 PAUSE video/movie288.Mjpeg RTSP/1.0
2 CSeq: 3
3 Session: 619227
```

- TEARDOWN request: Chương trình sẽ kiểm tra trạng thái hiện tại của client, nếu không là INIT (là READY hoặc PLAYING) thì tạo request TEARDOWN có dạng:

```
1 TEARDOWN <Video_File> RTSP/1.0
2 CSeq: <CSeq>
3 Session: <Session_ID>
```

Ví dụ:

```
1 TEARDOWN video/movie288.Mjpeg RTSP/1.0
2 CSeq: 4
3 Session: 619227
```

- Sau khi dựng xong, request được encode và đẩy qua socket TCP tới server RTSP.

Hoàn thiện hàm phân tích RTSP reply - `parseRtspReply()`:

```
1 def parseRtspReply(self, data):
2     lines = data.split('\n')
3     seqNum = int(lines[1].split(' ')[1])
4
5     if seqNum == self.rtspSeq:
6         session = int(lines[2].split(' ')[1])
7         if self.sessionId == 0:
8             self.sessionId = session
9         if self.sessionId == session:
10            if int(lines[0].split(' ')[1]) == 200:
11                if self.requestSent == self.SETUP:
12                    self.state = self.READY
13                    self.openRtpPort()
14                elif self.requestSent == self.PLAY:
15                    self.state = self.PLAYING
16                elif self.requestSent == self.PAUSE:
17                    self.state = self.READY
18                    self.playEvent.set()
19                elif self.requestSent == self.TEARDOWN:
20                    self.state = self.INIT
21                    self.teardownAked = 1
```

- Phản hồi được tách theo từng dòng để lấy CSeq và Session. Client chỉ xử lý khi số thứ tự trùng với gói vừa gửi để tránh trạng thái cũ. Ví dụ một phản hồi có dạng:

```
1 RTSP/1.0 <HTTP_Status_Code> <Phrase>
2 CSeq: <CSeq>
3 Session: <Session_ID>
```

- Khi mã trạng thái HTTP là 200, client cập nhật trạng thái mới dựa trên request đã gửi:
 - Sau SETUP, trạng thái chuyển từ INIT sang READY. Sau đó mở port RTP bằng hàm `openRtpPort()`.

- Sau PLAY, trạng thái chuyển từ READY sang PLAYING và bắt đầu chạy nền hàm nhận gói RTP.
- Sau PAUSE, trạng thái chuyển từ PLAYING sang READY và dừng hàm nhận gói RTP.
- Sau TEARDOWN, trạng thái chuyển về INIT và đóng socket RTP.

Hoàn thiện hàm kết nối tới port của RTP - openRtpPort():

```

1 import socket, tkinter
2
3 def openRtpPort(self):
4     self.rtpSocket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
5     self.rtpSocket.settimeout(0.5)
6
7     try:
8         self.rtpSocket.bind(('', self.rtpPort))
9     except:
10         tkinter.messagebox.showwarning('Unable to Bind', 'Unable to bind PORT=%d
    ', %self.rtpPort)

```

- Client khởi tạo socket UDP (AF_INET, SOCK_DGRAM) (AF_INET: IPv4, SOCK_DGRAM: UDP) cùng timeout ngắn để việc nhận khung không bị block.
- Hàm bind trực tiếp vào rtpPort; nếu cổng đã được sử dụng, chương trình sẽ cảnh báo không thể bind port đó.

3.1.3 Hoàn thiện RtpPacket.py - Giao thức RTP của server

Hoàn thiện hàm encapsulation - encode():

```

1 from time import time
2 HEADER_SIZE = 12
3
4 def encode(self, version, padding, extension, cc, seqnum, marker, pt, ssrc,
    payload):
5     timestamp = int(time())
6     header = bytearray(HEADER_SIZE)
7

```

```

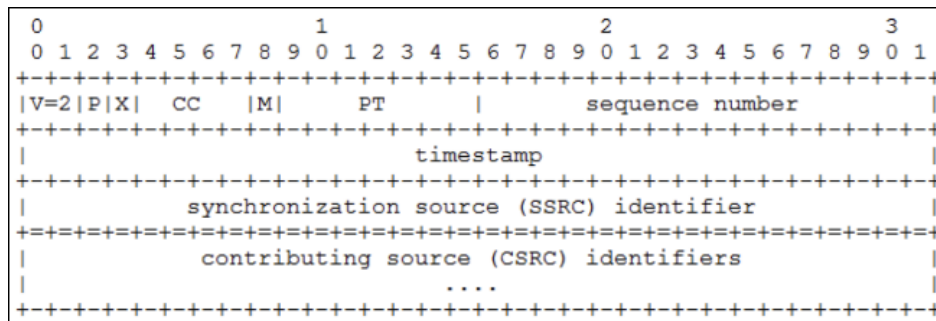
8     header[0] = (version & 0x03) << 6 | (padding & 0x01) << 5 | (extension & 0
11    x01) << 4 | (cc & 0x0F)
9     header[1] = (marker & 0x01) << 7 | (pt & 0x7F)
10    header[2:4] = (seqnum >> 8) & 0xFF, seqnum & 0xFF
11    header[4:8] = (timestamp >> 24) & 0xFF, (timestamp >> 16) & 0xFF, (timestamp
12    >> 8) & 0xFF, timestamp & 0xFF
12    header[8:12] = (ssrc >> 24) & 0xFF, (ssrc >> 16) & 0xFF, (ssrc >> 8) & 0xFF,
13    ssrc & 0xFF
13
14    self.header = header
15    self.payload = payload

```

header: mảng byte có kích thước 12 bytes

- Byte thứ 1 (8 bits): chứa trường Version, Padding, Extension, Contributing Source Count và có dạng V-V-P-X-CC-CC-CC-CC
 - version & 0x03: lấy 2 bit cuối của version (03[16] = 0000 0011[2])
 - « 6: dịch trái 6 bit để đưa về vị trí 2 bit đầu tiên trong byte (eg. V = 2 => 1100 0000[2])
 - Tương tự với các trường padding, extension, lúc này còn 4 bit cuối cùng là CC
 - cc & 0x0F: chỉ lấy 4 bit cuối của cc (0F[16] = 0000 1111[2])
- Byte thứ 2 (8 bits): chứa trường Marker Bit, Payload Type và có dạng M-PT-PT-PT-PT-PT-PT-PT (các logic tương tự byte 1)
- Byte thứ 3-4 (16 bits): chỉ chứa trường Sequence Number
 - (seqnum » 8) & 0xFF: lấy 8 bit đầu của seqnum (0xFF[16] = 1111 1111[2])
 - seqnum & 0xFF: lấy 8 bit cuối của seqnum
- Byte thứ 5-8 (32 bits): chỉ chứa trường Timestamp, logic tương tự byte 3-4
- Byte thứ 9-12 (32 bits): chỉ chứa trường Synchronization Source Identifier (SSRC), logic tương tự byte 3-4

Phần header trong python sẽ có dạng như sau, với mỗi dòng tương ứng với 4 bytes (32 bits):



Hình 3: Cấu trúc mảng header

3.1.4 Thử nghiệm

Thực hiện chạy server RTSP/RTP và client RTSP trên cùng máy tính với video mẫu qua câu lệnh:

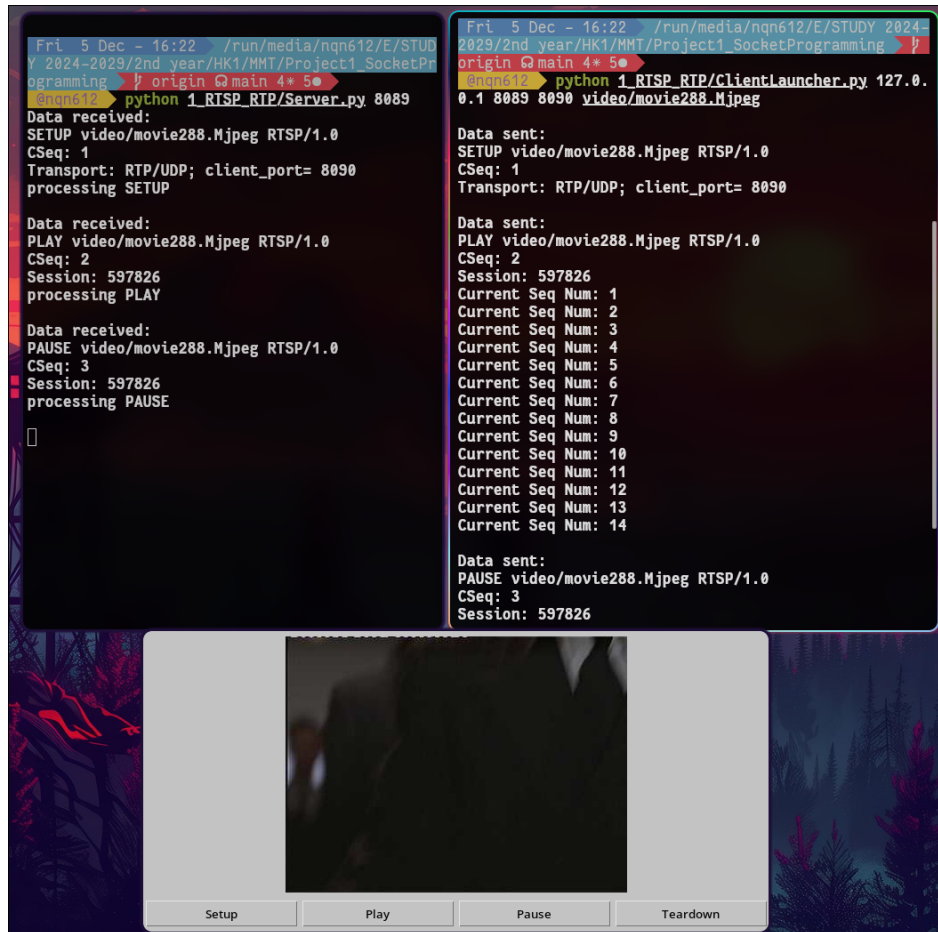
- Terminal 1: chạy server RTSP/RTP (python Server.py <PORT>). Ví Dụ:

```
1 python 1_RTSP_RTP/Server.py 8089
```

- Terminal 2: chạy client RTSP (python ClientLauncher.py <SERVER_IP> <RTSP_PORT> <RTP_PORT> <VIDEO_FILE>). Ví Dụ:

```
1 python 1_RTSP_RTP/ClientLauncher.py 127.0.0.1 8089 8090 video/movie288.
  Mjpeg
```

Output thu được khi chọn các nút (Setup -> Play -> Pause) trên giao diện client:



Hình 4: Output RTSP client

Có thể thấy client đã gửi đúng các request RTSP và chạy được video qua giao thức RTP.

Nhược điểm: Với video có độ phân giải cao (HD trở lên) khi 1 frame vượt quá kích thước gói UDP sẽ bị tràn byte qua các gói khác, dẫn đến việc giải mã sai các frame tiếp theo và báo lỗi.

3.2 Bổ sung HD video streaming (3đ)

Các file trong phần này nằm trong thư mục **2_HD_Video_Streaming**.

Đây là phiên bản nâng cấp của phần 1 nhằm hỗ trợ streaming video có độ phân giải cao (HD - 720p, Full HD - 1080p, 2K - 1440p).

3.2.1 Ý tưởng

Khắc phục nhược điểm của phần 1 bằng cách chia nhỏ frame video thành các mảnh (fragment) có kích thước phù hợp với gói UDP dựa trên MTU - Maximum transmission unit (Hầu hết các hệ thống mạng dùng MTU mặc định 1500 bytes cho IPv4 để đảm bảo tương thích.).

| Các phương tiện truyền tải qua IP | MTU (bytes) | Notes |
|-----------------------------------|--|---|
| Internet IPv4 path MTU | Tối thiểu 68 bytes, tối đa 64 KiB | MTU thực tế phụ thuộc đường truyền; nếu router giảm MTU sẽ xảy ra IP fragmentation. |
| Internet IPv6 path MTU | Tối thiểu 1280 bytes, tối đa 64 KiB | Yêu cầu MTU tối thiểu 1280 bytes; hỗ trợ gói jumbo cho payload lớn. |
| X.25 | 576 bytes khi gửi hoặc 1600 bytes khi nhận | Chuẩn X.25 quy định hai ngưỡng MTU khác nhau cho chiều gửi và nhận. |
| Ethernet v2 | 1500 bytes | Hầu hết mạng LAN sử dụng Ethernet II nên MTU mặc định luôn là 1500 bytes. |
| ... | ... | ... |

Bảng 4: Một số MTU cho các phương tiện truyền tải phổ biến

Mỗi fragment sẽ được đóng gói trong một gói RTP riêng biệt với đầy đủ header và payload. Client sẽ nhận các gói RTP, tái tạo lại các fragment và ghép chúng lại thành frame hoàn chỉnh bằng cách nhận diện các SOI, EOI trong payload (Marker Bit không được sử dụng do đã bị giới hạn giá trị bằng 0).

3.2.2 Hoàn thiện cơ chế chia nhỏ frame thành các fragment

3.2.3 Hoàn thiện cơ chế bỏ qua các frame bị trễ để đảm bảo đồ mượt khi phát video

3.2.4 Hoàn thiện tính năng hiện số frame mất và lưu lượng mạng trên giao diện

3.2.5 Thử nghiệm

Nhược điểm: Dù đã gửi và chạy được nhưng với các video có độ phân giải cao hơn (1440p, 2160p,...) các frame tải chậm hơn tốc độ của video dẫn tới tình trạng mất frame và drop FPS nặng.

3.3 Bổ sung client cache (2.5đ)

Các file trong phần này nằm trong thư mục **3_Client_Cache**.

Đây là phiên bản nâng cấp của phần 2 và cũng là phiên bản cuối cùng nhằm cải thiện trải nghiệm xem video HD bằng cách thêm bộ đệm (buffer) trên client.

3.3.1 Ý tưởng

Khắc phục nhược điểm của phần 2 bằng cách thêm bộ đệm (buffer) trên client để lưu trữ trước một số frame video nhất định trước khi phát. Khi nhấn Play, client sẽ bắt đầu nhận các gói RTP và lưu trữ các frame vào bộ đệm cho đến khi đạt ngưỡng đã định sẵn (ví dụ: 30 frame). Sau đó, client mới bắt đầu phát video từ bộ đệm trong khi tiếp tục nhận và lưu trữ các frame mới vào bộ đệm. Điều này giúp đảm bảo rằng có đủ frame để phát liên tục ngay cả khi tốc độ tải về chậm hơn tốc độ phát video.

3.3.2 Hoàn thiện cơ chế bộ đệm phía client

3.3.3 Thử nghiệm

Thực hiện chạy server RTSP/RTP và client RTSP trên cùng máy tính với video mẫu qua câu lệnh:

4 Kết quả đạt được

Sau quá trình xây dựng, hoàn thiện và thử nghiệm hệ thống theo ba tiêu chí của bài tập lớn, nhóm đã thu được các kết quả như sau:

- Hoàn thiện chức năng RTSP/RTP: Client có khả năng gửi các yêu cầu RTSP và nhận phản hồi đúng định dạng từ server. Phía server có thuật toán đóng gói RTP được hoàn thiện với đầy đủ các trường trong header. Hệ thống đã có thể truyền tải được các video MJPEG có độ phân giải thấp qua UDP với tốc độ ổn định và độ trễ thấp.
- Hoàn thiện chức năng truyền tải video HD: Chia nhỏ mỗi frame MJPEG thành các fragment có kích thước phù hợp với MTU. Đảm bảo tốc độ truyền ổn định và tốc độ phát mượt. Hiện

thì được số lượng frame mất và lưu lượng mạng. Hệ thống đã có thể phát được video HD với tốc độ ổn định.

- Hoàn thiện chức năng bộ đệm phía client: Client luôn duy trì một số lượng frame được lưu trước (pre-buffer), đồng thời bổ sung các frame mới trong quá trình phát nhằm cải thiện khả năng xem lại và hạn chế giật hình khi video có độ phân giải lớn. Hệ thống đã có thể phát video có độ phân giải cao hơn như 1440p, 2160p mà không bị drop frame khi mạng không ổn định.

Hệ thống cuối cùng đáp ứng toàn bộ yêu cầu của bài lab, từ việc thực thi giao thức RTSP và đóng gói RTP, đến việc truyền tải video HD và hỗ trợ client cache. Các chức năng đều hoạt động như mong đợi, thể hiện sự phù hợp của thiết kế hệ thống cũng như cách tiếp cận từng bước của nhóm.

5 Tài liệu tham khảo

Tài liệu

- [1] Geeksforgeeks. [Real-time Transport Protocol\(RTP\)](#), 12-07-2025.
- [2] Wikipedia. [Real-time Transport Protocol](#), 11-12-2022.
- [3] Wikipedia. [Real-Time Streaming Protocol](#), 14-02-2022.
- [4] Wikipedia. [Motion JPEG](#), 21-11-2018.