

# Video streaming with RTSP and RTP

## I. Giới thiệu:

### 1. Mục tiêu:

Xây dựng và triển khai một hệ thống video streaming đơn giản, mô phỏng kiến trúc thực tế trên Internet. Hệ thống bao gồm: server phát video và client nhận. Trong đó, hai giao thức chính được sử dụng bao gồm:

- Kênh điều khiển (Control Channel): thiết lập phiên (session) để stream video và kiểm soát các thao tác của client (SETUP, PLAY, PAUSE, TEARDOWN).  
Giao thức chính là *Real-time Streaming Protocol (RTSP)* sử dụng *TCP*
- Luồng dữ liệu (Media Stream): truyền dữ liệu là video thực tế thông qua việc đóng gói (packetization) bằng phương thức *Real-time Transfer Protocol (RTP)* sử dụng *UDP*

### 2. Nhiệm vụ:

Nhiệm vụ trọng tâm được phân thành 3 phần chính:

- Giao thức cơ bản: Hoàn thiện triển khai giao thức RTSP phía Client và đóng gói RTP ở phía Server thông qua việc hoàn chỉnh mã nguồn còn thiếu.
- HD Video Streaming: Mở rộng hệ thống để hỗ trợ stream video với độ phân giải 720p hoặc 1080p thông qua việc triển khai phân mảnh (fragmentation) cho các khung hình vượt quá MTU (Maximum Transmission Unit - kích thước tối đa của 1 packet) nhằm tối ưu hóa hiệu suất truyền tải.
- Client-side Caching: Thiết lập bộ đệm khung hình ở phía Client để giảm hiện tượng jitter (chậm trễ trong quá trình truyền và nhận dữ liệu) bằng cách tải trước một số khung hình.

## II. Triển khai giao thức cơ bản:

Hệ thống video streaming sử dụng kiến trúc phân tách chức năng. Trong đó, hai giao thức được sử dụng cho hai chức năng riêng biệt:

- RTSP trên TCP: Sử dụng cho kênh điều khiển vì việc thiết lập kết nối cần có mức độ tin cậy cao => sử dụng TCP vì tính chất tương tự.
- RTP trên UDP: Sử dụng cho luồng dữ liệu vì việc phát đi dữ liệu cho client có thể xem không nhất thiết phải tin cậy mà phải đảm bảo tốc độ truyền tối đa.

### 1. Kênh điều khiển (Control Channel)

Kênh điều khiển sẽ thiết lập phiên (session) và quản lý các thao tác mà client yêu cầu.

Giao thức: Real-time Streaming Protocol (RTSP)

Giao vận: Transmission Control Protocol (TCP) để đảm bảo độ tin cậy trong quá trình truyền tải thông điệp quan trọng.

Cổng mặc định: 554

#### 1.1 Các lệnh RTSP:

Client sẽ gửi lệnh văn bản đến server thông qua cổng TCP:

- **SETUP**: C yêu cầu S chuẩn bị một luồng video cụ thể và cổng UDP cụ thể để nhận dữ liệu.
- **PLAY**: C ra lệnh S bắt đầu truyền dữ liệu video qua RTP/UDP.
- **PAUSE**: C ra lệnh S tạm dừng việc gửi dữ liệu video nhưng vẫn duy trì trạng thái **READY**
- **TEARDOWN**: C ra lệnh S kết thúc phiên làm việc và đóng kết nối với S.

## 1.2 Triển khai RTSP bên Client:

Triển khai bằng cách Client gửi request qua socket TCP đã được thiết lập sẵn. Client phải theo dõi trạng thái phiên (session state) và số thứ tự lệnh (CSeq).

- CSeq (Client-sequence): là biến đếm được tăng lên 1 sau mỗi lần 1 request được gửi đi. Header **Cseq**: `{self.rtspSeq}` được chèn vào tất cả các request.
- Session ID: nếu lệnh **SETUP** trả về phản hồi **200 OK**, Client trích xuất Session ID từ header **Session**: `{self.sessionId}` của Server và sau đó được chèn vào tất cả các request tiếp theo.

Server quản lý trạng thái của Client kể từ phản hồi thành công:

- INIT ----(SETUP)----> READY
- READY ----(PLAY)----> PLAYING
- PLAYING ----(PAUSE)----> READY
- READY/ PLAYING ----(TEARDOWN)----> INIT

Lệnh	Trạng thái chuyển đổi	Hành động của Client	Header
SETUP	INIT \$\\to\$ READY	Tạo socket UDP để nhận dữ liệu RTP và thiết lập timeout 0.5s. Gửi Request và chờ Session ID.	Transport: Chỉ định cổng UDP của Client (client_port= <RTP_port>).
PLAY	READY \$\\to\$ PLAYING	Bắt đầu nhận dữ liệu RTP (Video) qua UDP.	Session: Chứa Session ID đã nhận được từ SETUP. Không chèn header Transport.
PAUSE	PLAYING \$\\to\$ READY	Dừng nhận dữ liệu RTP.	Session: Chứa Session ID. Không chèn header Transport.
TEARDOWN	READY/PLAYING \$\\to\$ INIT	Đóng socket TCP RTSP và socket UDP RTP.	Session: Chứa Session ID22. Không chèn header Transport23.

## 1.3 Mã nguồn hoàn chỉnh:

to be added

### 2. Kênh dữ liệu (Data Channel)

Còn về việc server gửi data về cho client thì chỉ cần quan trọng tốc độ truyền data sao cho nhanh nhất là được nên sử dụng UDP.

## Cách hoạt động:

- Server đọc frame: S đọc từng khung hình (trong file .Mjpeg thì mỗi file jpeg và 1 frame)
- Đóng gói RTP (RTP packetization?): S tạo 1 gói tin RTP bằng cách ghép 1 frame (ở đây là 1 frame trong .Mjpeg là 1 jpeg) với 1 file **header**.
- Gửi UDP: S gửi gói RTP này đến cổng UDP của C.
- Client nhận: C nhận datagram UDP (đại khái là gói dữ liệu) tách header và payload (phần data chính cần mở)
- Tái tạo và phát: C sử dụng **seqnum** để sắp xếp gói tin và sử dụng **timestamp** để biết thời điểm phát frame rồi hiện thị frame.

## Cấu trúc gói tin RTP:

Phần dữ liệu cố định 12 byte chứa thông tin điều khiển thời gian thực. Gồm 2 phần: header và payload

- Header:
  - Sequence Number (số thứ tự): giúp client xác định gói nào bị mất và để sắp xếp lại thứ tự.
  - Timestamp: đồng bộ hóa và quản lý biến trễ.
  - Marker bit: đánh dấu ranh giới khung hình, đặc biệt quan trọng trong cơ chế phân mảnh (fragmentation) của HD streaming.
- Payload: chứa dữ liệu thực tế mà người dùng cần.

## Cách chạy

Mở terminal rồi split terminal:

Bên server: `py .\Server.py 12345`

Bên client: `py .\ClientLauncher.py localhost 12345 554 movie.Mjpeg`