ĐẠI HỌC QUỐC GIA TPHCM

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. Huỳnh Thụy Bảo Trân

Ngày 5 tháng 12 năm 2025

Mục lục

Danh sách bảng

Danh sách hình vẽ

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 bóc 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 1: Tiêu chí chấm điểm bài lab

1.3 Tổng quan thư mục

Bài lab được tổ chức theo các thư mục sau:

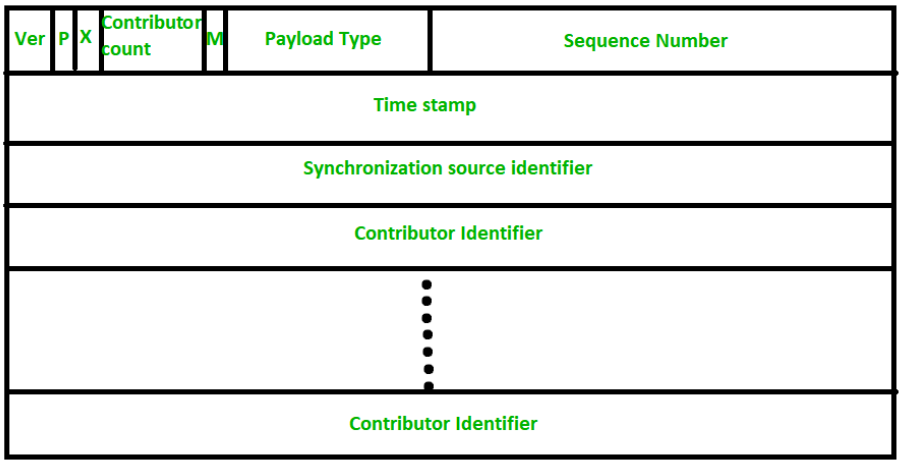
- **1_RTSP_RTP**: Chứa source code cho tiêu chí 1.
- **2_HD**: Chứa source code cho tiêu chí 2.
- **3_CACHE**: Chứa source code cho tiêu chí 3.
- **4_REPORT**: Chứa báo cáo của bài lab (tiêu chí 4).
- **video**: Chứa các video với các độ phân giải khác nhau phù hợp với từng tiêu chí.

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).



Hình 1: Cấu trúc giao thức RTP

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
		96 (12 bytes)	

Bảng 2: 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. 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. 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

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

Hoàn thiện hàm encapsulation - encode():

```
from time import time
HEADER_SIZE = 12

def encode(self, version, padding, extension, cc, seqnum, marker, pt, ssrc,
    payload):
     timestamp = int(time())
     header = bytearray(HEADER_SIZE)

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

     self.header = header
     self.payload = payload
```

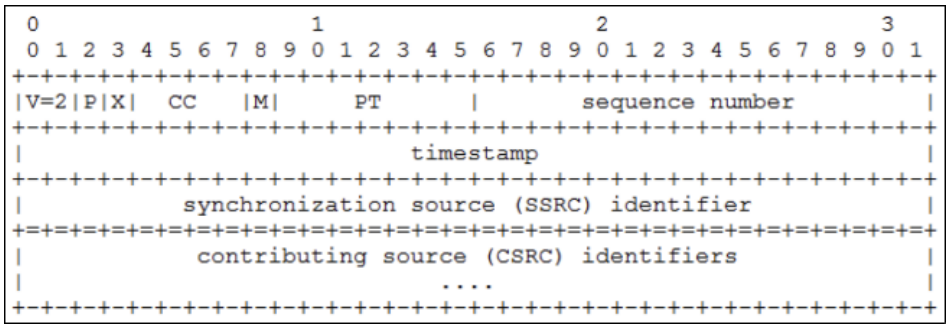
header: mảng byte có kích thước 12 bytes

- 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 2: Cấu trúc mảng header

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

Hoàn thiện hàm gửi RTSP request - sendRtspRequest():

```
import threading

def sendRtspRequest(self, requestCode):
    if requestCode == self.SETUP and self.state == self.INIT:
        threading.Thread(target=self.recvRtspReply).start()
        self.rtspSeq += 1
```

```
        request = "SETUP " + str(self.fileName) + " RTSP/1.0\nCSeq: " + str(self.rtspSeq) + "\nTransport: RTP/UDP; client_port= " + str(self.rtpPort)
        self.requestSent = self.SETUP
    elif requestCode == self.PLAY and self.state == self.READY:
        self.rtspSeq += 1
        request = "PLAY " + str(self.fileName) + " RTSP/1.0\nCSeq: " + str(self.rtspSeq) + "\nSession: " + str(self.sessionId)
        self.requestSent = self.PLAY
    elif requestCode == self.PAUSE and self.state == self.PLAYING:
        self.rtspSeq += 1
        request = "PAUSE " + str(self.fileName) + " RTSP/1.0\nCSeq: " + str(self.rtspSeq) + "\nSession: " + str(self.sessionId)
        self.requestSent = self.PAUSE
    elif requestCode == self.TEARDOWN and not self.state == self.INIT:
        self.rtspSeq += 1
        request = "TEARDOWN " + str(self.fileName) + " RTSP/1.0\nCSeq: " + str(self.rtspSeq) + "\nSession: " + str(self.sessionId)
        self.requestSent = self.TEARDOWN
    else:
        return

    self.rtspSocket.send(request.encode())
    print('\nData sent:\n' + request)
```

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

```
def parseRtspReply(self, data):
    lines = data.split('\n')
    seqNum = int(lines[1].split(' ')[1])

    if seqNum == self.rtspSeq:
        session = int(lines[2].split(' ')[1])
        if self.sessionId == 0:
            self.sessionId = session
        if self.sessionId == session:
            if int(lines[0].split(' ')[1]) == 200: # 200 (from the example)
                if self.requestSent == self.SETUP:
                    self.state = self.READY
```

```
                    self.openRtpPort()
                elif self.requestSent == self.PLAY:
                    self.state = self.PLAYING
                elif self.requestSent == self.PAUSE:
                    self.state = self.READY
                    self.playEvent.set()
                elif self.requestSent == self.TEARDOWN:
                    self.state = self.INIT
                    self.teardownAcked = 1
```

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

```
import socket, tkinter

def openRtpPort(self):
    self.rtpSocket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    self.rtpSocket.settimeout(0.5)

    try:
        self.rtpSocket.bind(('', self.rtpPort))
    except:
        tkinter.messagebox.showwarning('Unable to Bind', 'Unable to bind PORT=%d
    ' %self.rtpPort)
```

3.2 Bổ sung HD video streaming

3.3 Bổ sung client cache

4 Kết quả đạt được

5 Tài liệu tham khảo

Tài liệu