HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY
FACULTY OF COMPUTER SCIENCE & ENGINEERING

COMPUTER NETWORKS (CO3003)

Assignment 1

"Streaming video Server and Client using RTSP and RTP"

Lecturer: Nguyễn Hồng Nam
Group members: Đoàn Sinh Mãn 1813038
Nguyễn Thuý An 1810004
Võ Minh Long 1812951
Lăng Hoàng Long 1812879

Ho Chi Minh City, November 15^{th}, 2020

Contents

1 Introduction

Streaming video is one of the most ubiquitous feature of the Internet. At first, the Internet was designed just for simple text/document based transfers using protocols such as the HTTP, but now, it also can be the platform on which streaming media is passed from one end of the world to another. In this project, we are using Real Time Streaming Protocol (RTSP) and Real-time Transport Protocol (RTP/RTCP) to examine how media can be transmitted over the network. And our project will be implemented in Python 3.

2 Requirements Analysis

2.1 Functional Requirements

Hình 1: *Use-case diagram*

2.1.1 Server

- Server can open a socket to listen clients.
- Server processes four type of request from client: SET-UP request, PLAY request, PAUSE request and TEARDOWN request.
  - SET-UP: Prepare VideoStream object to send.
  - PLAY: Pack video data into RTP packets to send to Client.
  - PAUSE: Stop sending
  - TEARDOWN: Terminate the process and close connection.
- Server send RTSP reply message to Client to confirm a good request.

2.1.2 Client

- Client provides an user interface that has some buttons for user to interact with and to send request to Server.
- Client can request a video from Server each time.
- Client can pause or terminate the process half way.
- Client listens to Server's RTSP reply message.
- Client can send reply back to Server.
- When user clicks the TEARDOWN buttons, the process stops and the connection is closed automatically.

2.2 Non-functional Requirements

- Have simple user interface, easy to use.
- Delay time of PAUSE event is 0.05 second.
- Server allows 5 unaccepted connection before refusing new connections.
- Time out of a RTP socket is 0.5 second.

3 Function Description

3.1 ClientLauncher.py

3.1.1 __main__

This is the only function in `ClientLauncher.py`. It try to receive arguments such as server IP, server port, RTP port and require filename from command line. If everything success, it initialize tk's root windows, initialize client receiver then start tk root main loop, which allow we to interactive with client via GUI.

3.2 Client.py

This file contains `Client` class - the class used to control the GUI and backend of client side. All methods inside `Client` class are:

3.2.1 __init__

This method receive the root window and four arguments from `ClientLancher` and initialize other variable.

3.2.2 createWidgets

This method used to build the GUI of client side: the main window will be used to display video, while four botton Setup, Play, Pause and Teardown used to control.

3.2.3 setupMovie

This method will send RTSP setup request if and only if current state is INIT, which mean that we can only setup once at all.

3.2.4 exitClient

This method send ETSP teardown message, close the GUI and clean tempotary file when called.

3.2.5 pauseMovie

Pause the video when playing.

3.2.6 playMovie

Create a thread to listen data stream then send play command to the server.

3.2.7 listenRtp

Thread routine used to listen data stream from server. It simply receive data, decode data and update buffer with that decoded data. If error orcurs and `playEvent` is set, the connection is kept. If error orcurs and teardown state is set, the connection is terminated.

3.2.8 writeFrame

This method used to update buffer. After receiving data from server, client's subthread write data down a file (with unique name for each session). That file will be read be client main thread and display in screen.

3.2.9 updateMovie

This method read data file image file create by `writeFrame` and display on screen.

3.2.10 connectToServer

Try to establish a socket connection to given server address and por. Display a message box when failed.

3.2.11 sendRtspRequest

This method create corresponding request with request code and current state of program then send it to server.

3.2.12 recvRtspReply

This method receive reply from server and call `parseRtspReply` to parse it. Futhermore, if teardown request is sent, the client close connection to server.

3.2.13 parseRtspReply

This method used to parse the relay from server. First it check whether reply's sequence number is matched with request's sequence number. If they matched, it check whether session id is matched with current session id (or set new current id if it haven't set). Finally, it will change the program state and call other method is needed according to content of the reply.

3.2.14 openRtpPort

Try to open a listening socket on given port. Display a message box if failed.

3.2.15 handler

This method used to handle quit message from user (by clicking [X] botton at top right, pressing Alt+F4, ...)

3.3 Server.py

This file define `Server` class.

3.3.1 main

The only one method of class `Server`. It take one argument from command line - the port number - and try to create a listening socket at given port. Then it start and loop to listen connection. With each connection, it create a worker (in fact, that is a thread) to handle the connection.

3.4 ServerWorker.py

This file define `ServerWorker` class - the class use to handle a connection from client. All of its methods are:

3.4.1 __init__

Simple receive arguments from server's main thread.

3.4.2 run

Start new thread with `processRtspRequest` rountine to handle connection.

3.4.3 recvRtspRequest

This method is simply a loop: receive data then call `processRtspRequest` to process it.

3.4.4 `handler`

This method used to process data receive from client. First of all, it parse data to archive request type, filename and sequence number. Base on the request type, it will run very differently.
If the request type is SETUP: it open the requested media file and assign new session ID the the connection. Finally, it send *200 OK* response to client and setup RTP port number.
If the request type is PLAY: it create a new socket and start new thread with `sendRtp` to send data.
If the request type is PAUSE: it set the event flag and send *200 OK* back to client.
If the request type is TEARDOWN: it set the event flag, send *200 OK* back to client and close the socket.

3.4.5 `sendRtp`

The rountine used to send data to client. It simply read data from video stream and transfer that data via socket. Notice, that this process will be break if event flag is set.

3.4.6 `makeRtp`

Encapsulate RTP packet using `encode` method of `rtpPacket` class with current data and connection state.

3.4.7 `replyRtsp`

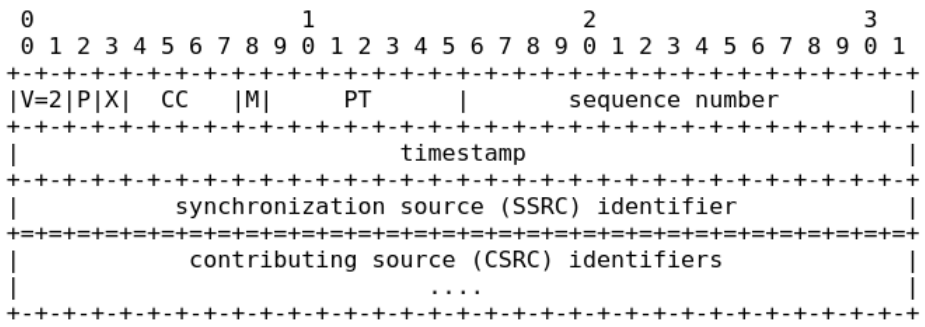
Send reply to client base on the given code. Allowed code are 200 (OK), 404 (Not Found) and 500 (Connection Error).

3.5 `RtpPacket.py`

This file define `RtpPacket` class - which contains some useful method to work with RTP packet.

3.5.1 `encode`

Build RTP header from given arguments and save it for later use.
The structure of TRP header is show as below:



Hình 2: *Structure of RTP header*

Detail of each field are:

- Version (V): 2 bits. Identify the version of RTP. Default is 2 in this assignment.
- Padding (P): 1 bit. If the padding bit is set, the packet contains one or more additional padding octets at the end which are not part of the payload. Not used in this assignment (set to 0).
- Extension (X): 1 bit. If the extension bit is set, the fixed header is followed by exactly one header extension. Not used in this assignment (set to 0).
- CSRC count (CC): 4 bits. The number of CSRC identifiers that follow the fixed header. Not used in this assignment (set to 0).

- Marker (M): 1 bit. The interpretation of the marker is defined by a profile. Not used in this assignment (set to 0).
- Payload type (PT): 7 bits. This field identifies the format of the RTP payload and determines its interpretation by the application. Set to 26 (indicate MJPEG type) in this assignment.
- Sequence number: 16 bits. Used to detect packet loss and to restore packet sequence. Increase by 1 for each RTP packet.
- Timestamp: 32 bits. Reflect the sampling instant of the first octet in the RTP data packet.
- SSRC: 32 bits. Identify the synchronization source. Not used in this assignment (set to 0).

3.5.2 `decode`

Simply split RTP packet into header and payload.

3.5.3 `version`

Return the RTP version.

3.5.4 `seqNum`

Return the sequence number of current packet.

3.5.5 `timestamp`

Return the timestamp of current packet.

3.5.6 `payloadType`

Return the code of payload type.

3.5.7 `getPayload`

Return turn the payload of packet in form of byte stream.

3.5.8 `getPacket`

Return the whole RTP packet.

3.6 `VideoStream.py`

This file define `VideoStream` class - the class used to handle media file.

3.6.1 `__init__`

Try to open given filename. If success, set frame number to 0. Otherwise, throw IOErr exception.

3.6.2 `nextFrame`

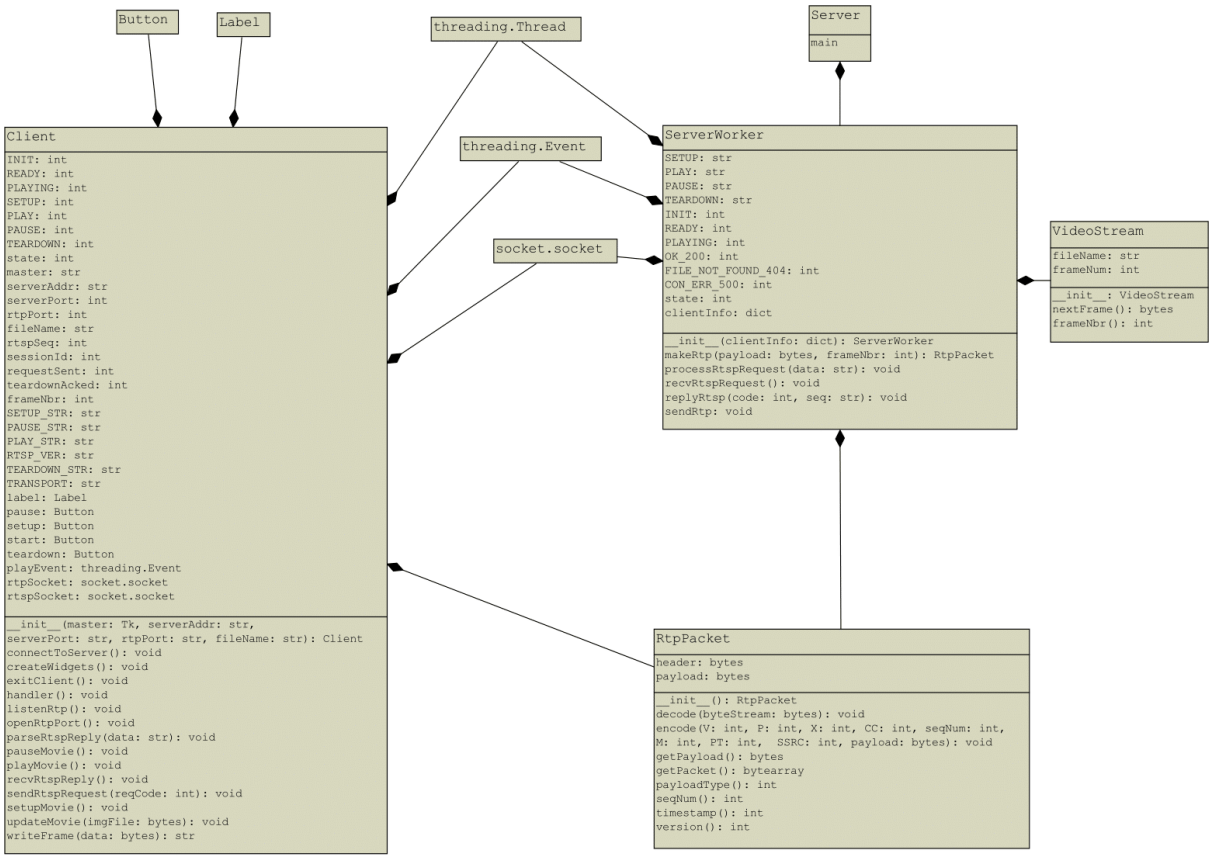
Read next frame in file and return it. Increase frame number by 1.

3.6.3 `frameNbr`

Return the current frame number.

4 Class Diagram

Here's the class diagram in abbreviation:



Hình 3: *Class diagram*

5 A Summative Evaluation of Results Achieved

We have completed all requirements:

1. Implemented the RTSP protocol in the client: 100%
2. Implemented the RTP packetization in the Server: 100%

6 User manual

- First, start the server with the command:

  $ python3 Server.py server_port

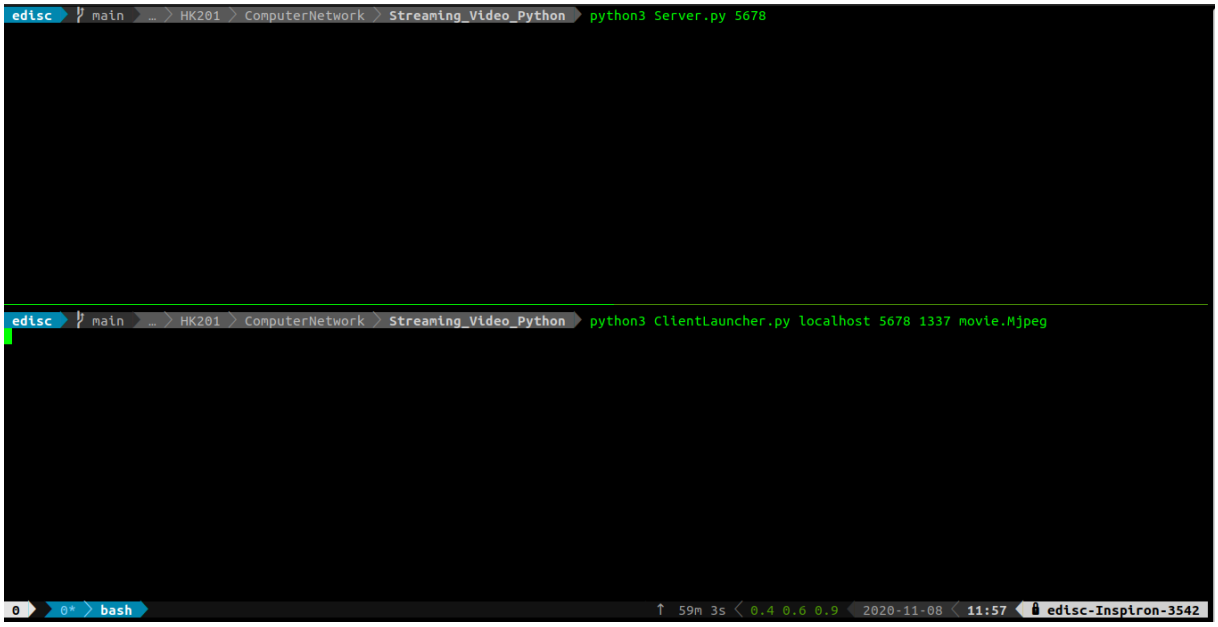
  where **server_port** is the port your server listens to for incoming RTSP connections. The standard RTSP port is 554, but you will need to choose a port number greater than 1024

- Then, start the client with the command:

  $ python3 ClientLauncher.py server_host server_port RTP_port video_file

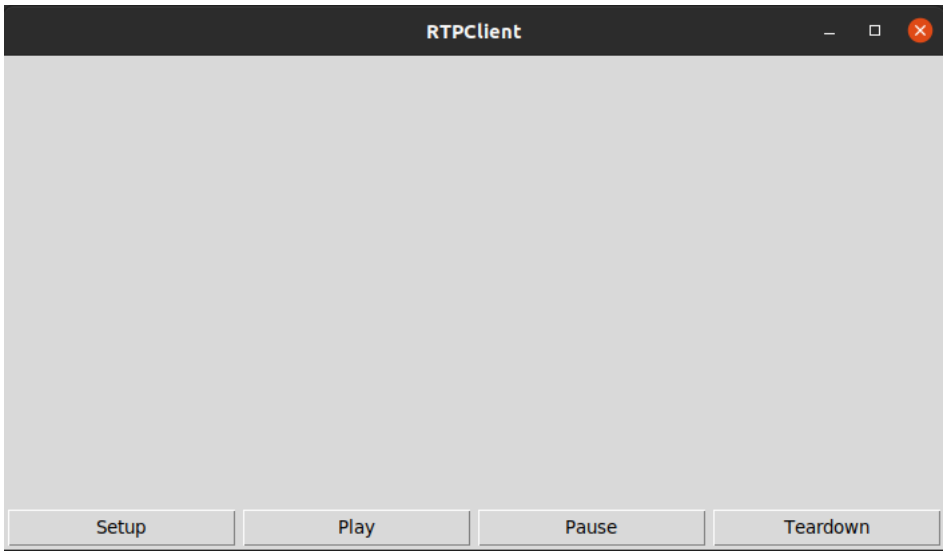
  where:

  - **server_host** is the name of the machine where the server is running;
  - **server_port** is the port where the server is listening on;
  - **RTP_port** is the port where the RTP packets are received;
  - **video_file** is the name of the video file you want to request.



Hình 4: *Start the server and client*

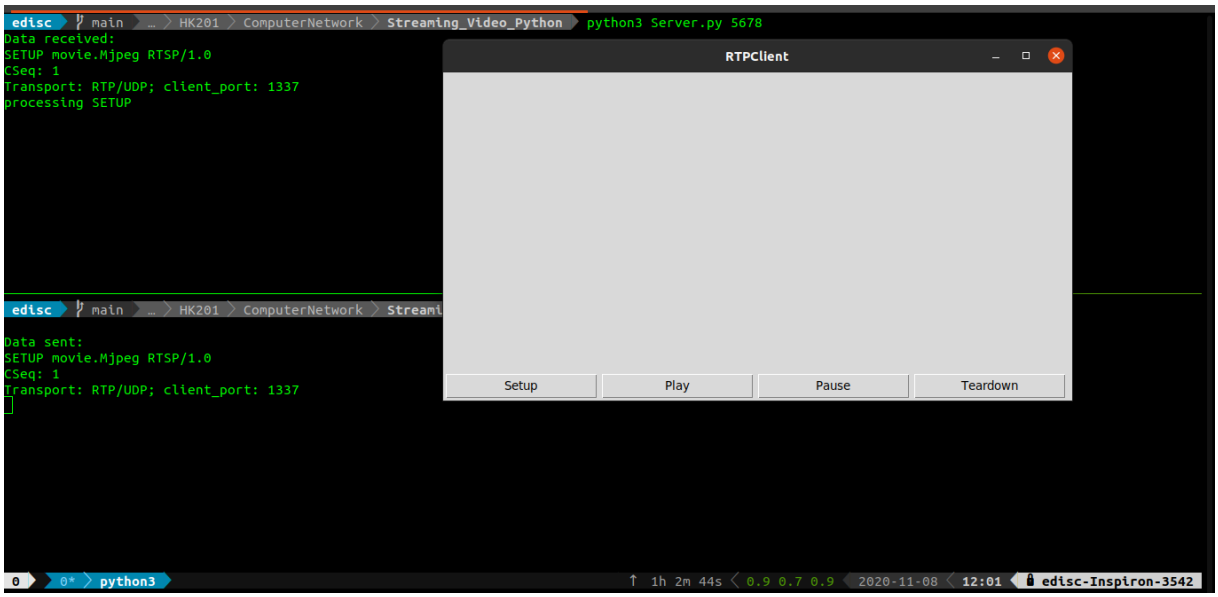
- The client opens a connection to the server and pops up a window like this:



Hình 5: *RTP client window*

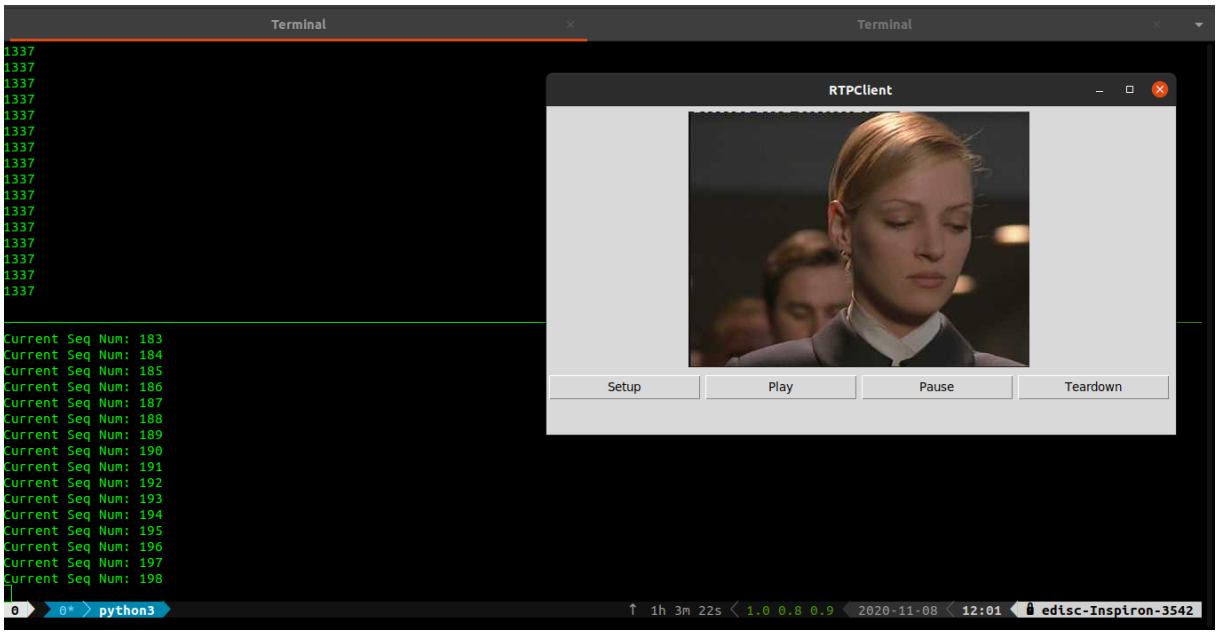
- You can send RTSP commands to the server by pressing the buttons. A normal RTSP interaction goes as follows:

  1. The client sends **SETUP**. This command is used to set up the session and transport parameters.



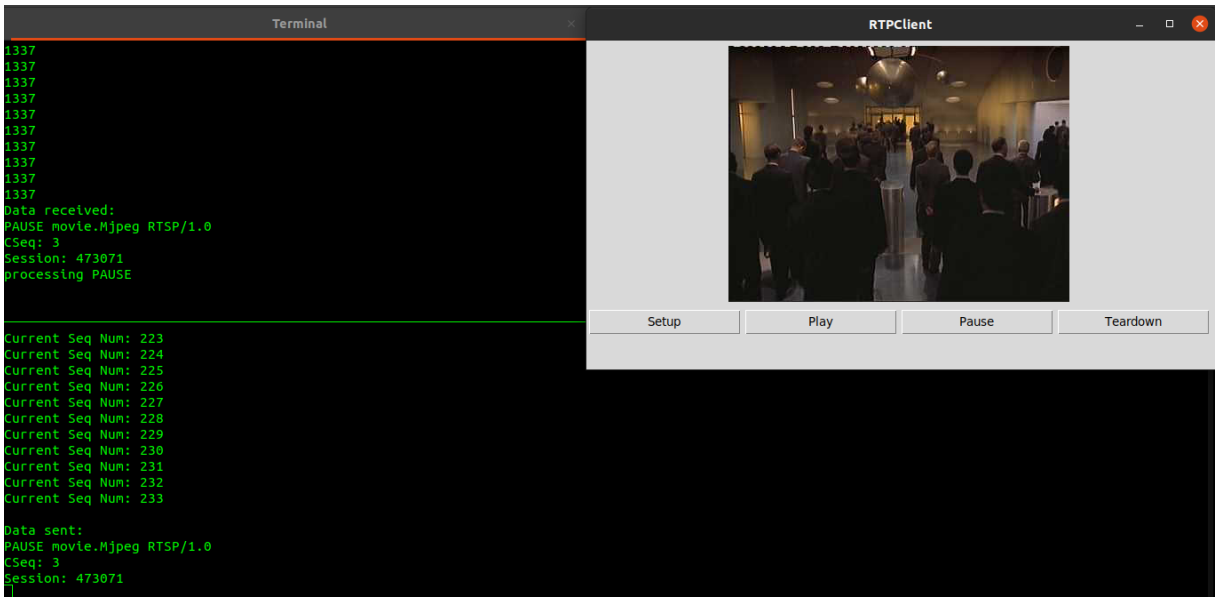
Hình 6: *The client sends SETUP*

2. The client sends **PLAY**. This command starts the playback



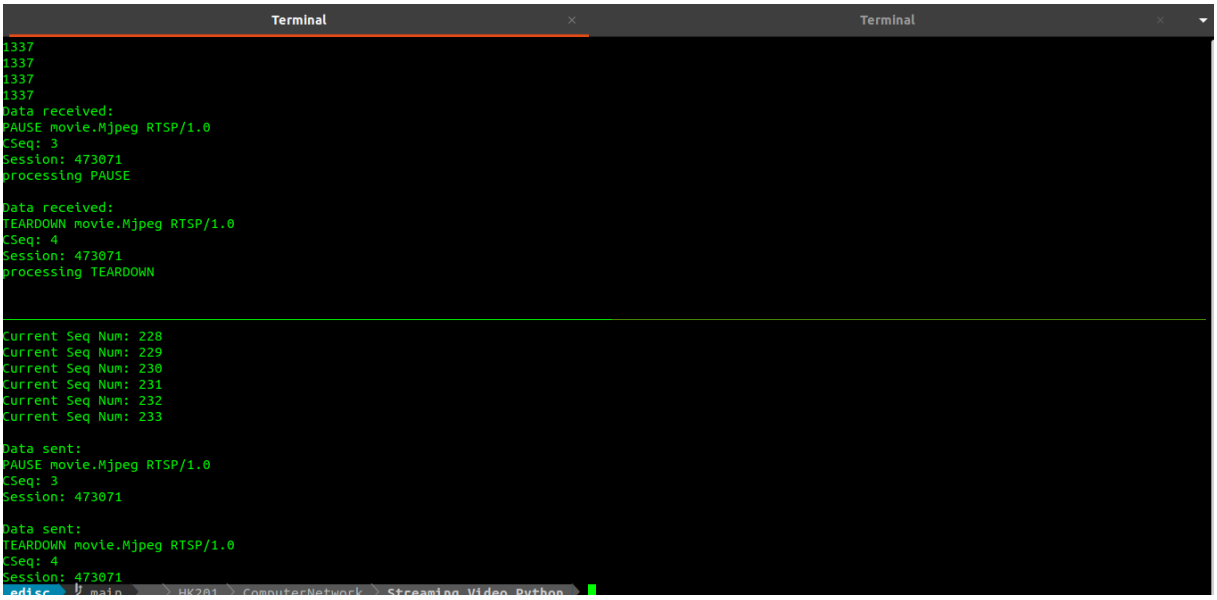
Hình 7: *The client sends PLAY*

3. The client may send **PAUSE** if it wants to pause during playback



Hình 8: *The client sends PAUSE*

4. The client sends **TEARDOWN**. This command terminates the session and closes the connection



Hình 9: *The client sends TEARDOWN*

7 Extend

7.1 Part 1.

We calculated video data rate, RTP packet lost rate and total data received of a session.

7.1.1 RTP packet lost rate

The RTP packet lost rate is a fraction of 2 parameters: the cumulative lost packets and the total number of expected packets. The cumulative lost packets variable is incremented every time the sequence number of a packet is found to be different from the expected sequence number. The expected packets is based on the highest number received (in cases where some packets are lost and sequence number is larger than the counting number).

$$R = \frac{L}{E}$$

In which:

- R is RTP packet lost rate
- L is number of lost packets (packets)
- E is number of expected packets (packets)

7.1.2 Total data received

Total data receive is the cumulative data sent to client

$$T = \sum D_i$$

In which:

- T is total data received (bytes)
- D_i is video data framelength of packet i (bytes)

7.1.3 Video data rate

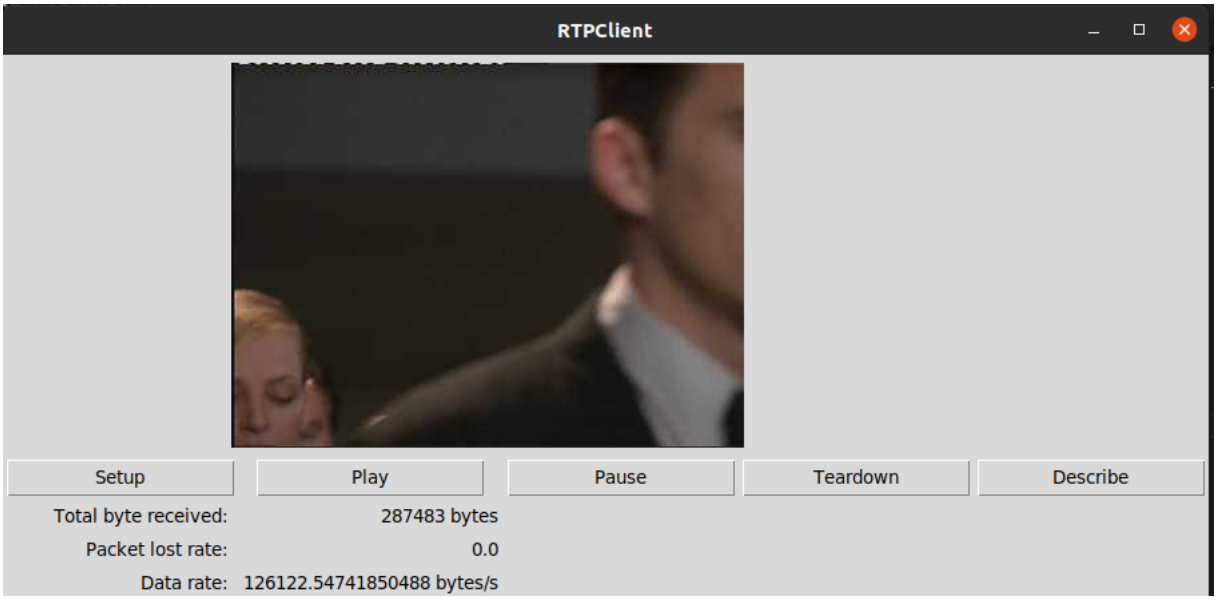
This rate is calculated based on the time where the movie is actually playing and does not count time when the movie is paused. The client measures the interval between each received packet and sums them up to find the total time the movie is playing. The video data rate is the fraction of total data received and total time of playing.

$$V = \frac{T}{TTP}$$

In which:

- V is video data rate (bytes/s)
- T is total data received (bytes)
- TTP is the total data transfering time (seconds)

Here's the result:



Hình 10: *Total bytes received, RTP packet lost rate and Video data rate*

7.2 Part 2.

Because SETUP command need to be called one time, we called setupMovie() method in Client class initially. We also replaced the Setup button with the Describe button.

Here's the result:

Hình 11: *Removed Setup button*

7.3 Part 3.

The DESCRIBE request is sent same as other types of request. However, DESCRIBE reply message is a bit different. It follow Session Description Protocol (SDP). The Session Description Protocol describes a session as a group of fields in a text-based format, one field per line. The form of each field is as follows.

```
<character>=<value><CR><LF>
```

The example of DESCRIBE reply is:

```
RTSP/1.0 200 OK
CSeq: 312
Date: 23 Jan 1997 15:35:06 GMT
Content-Type: application/sdp
Content-Length: 376

v=0
o=mhandley 2890844526 2890842807 IN IP4 126.16.64.4
s=SDP Seminar
i=A Seminar on the session description protocol
u=http://www.cs.ucl.ac.uk/staff/M.Handley/sdp.03.ps
e=mjh@isi.edu (Mark Handley)
c=IN IP4 224.2.17.12/127
t=2873397496 2873404696
a=recvonly
m=audio 3456 RTP/AVP 0
m=video 2232 RTP/AVP 31
m=whiteboard 32416 UDP WB
a=orient:portrait
```

Hình 12: *Example of DESCRIBE reply message*

There are some mandatory fields:

- v field: This is protocol field and must be currently set to 0.
- o field: This is originator identifier field. However, the information show in this field require Session Initiation Protocol (SIP) for creating (according to this RFC. So, we will simply ignore this field in our assignment.
- s field: the session name. We will reuse session ID to fill in this field.
- Time field: No reason to implement it in our assignment!.

To transfer stream type and media encoding using SDP, we use a SDP's optional field: a field - attribute line. Out DESCRIBE reply will look like this:

```
RTSP/1.0 200 OK
CSeq: 2
Session: 964563

v=0
s=964563
a=Real Time Streaming Protocol (RTSP)
a=Motion JPEG (M-JPEG/MJPEG)
```

Hình 13: *DESCRIBE reply message*

References

[1] James F. Kurose and Keith W. Ross. *Computer Networking: A Top-Down Approach*. 7th ed. Boston, MA: Pearson, 2016.

[2] Wikipedia. *Real Time Streaming Protocol*. URL: https://en.wikipedia.org/wiki/Real_Time_Streaming_Protocol (visited on 11/04/2020).