VIETNAM NATIONAL UNIVERSITY, HO CHI MINH CITY
UNIVERSITY OF TECHNOLOGY
FACULTY OF COMPUTER SCIENCE AND ENGINEERING

Computer Network

Assignment 1

Video Streaming Application

Advisors: Mr. Phạm Trần Vũ
Mr. Nguyễn Mạnh Thìn

HO CHI MINH CITY, OCTOBER 2021

Member list

No.	Full name	Student ID	Percentage of work
1	Đỗ Đức Trung	1952144	100%
2	Đinh Huy Thiện	1952462	100%
3	Nguyễn Duy Thành	1952310	100%

Contents

1 Requirements

1.1 Functional

- Client Application can setup, play, pause the video.
- Client can terminate session by themselves.
- Server can provide error if Client has inputted the wrong file.

1.2 Non-Functional

- Client GUI has 4 buttons: Setup, Play, Pause, Teardown.
- Application can only run Mjpeg files.
- Requests only takes 0.5 seconds before being timed out.
- The video will be played by frames per 0.05 seconds.
- Server only sends data back to the Client via RTP packets.

The goal of this is to make a streaming video server and client that communicate using the Real-Time Streaming Protocol (RTSP) and send data using the Real-time Transfer Protocol (RTP).

2 Application Description

2.1 Communication between Client and Server via RTSP

2.1.1 Client Request

On the Client side, we have 4 main buttons for client to send requests to the Server.

- SETUP: Send SETUP request to the server. Reads server's response and parse Session header for RTSP session ID. Also creates datagram socket for RTP data with timeout of 0.5 seconds

```
SETUP movie.Mjpeg RTSP/1.0
CSeq: 0
Transport: RTP/UDP; client_port = 2345
```

- PLAY: Send PLAY request to the server.

```
PLAY movie.Mjpeg RTSP/1.0
CSeq: 1
Transport: RTP/UDP; client_port = 2345
```

- PAUSE: Send PAUSE request

```
PAUSE movie.Mjpeg RTSP/1.0
CSeq: 2
Transport: RTP/UDP; client_port = 2345
```

- TEARDOWN: Send TEARDOWN request. Read server's response with CSeq header in every request.

```
TEARDOWN movie.Mjpeg RTSP/1.0
CSeq: 3
Transport: RTP/UDP; client_port = 2345
```

2.1.2 Server Respond

Once the Client successfully create a request to the Server, the Server must respond to the request accordingly. Therefore we have 4 server responses to client requests. (in this box, C is denoted as Client and S is denoted as Server):

- SETUP

```
C: SETUP movie.Mjpeg RTSP/1.0
C: CSeq: 0
C: Transport: RTP/UDP; client_port = 2345
```

```
S: RTSP/1.0 200 OK
S: CSeq: 0
S: Session: 123456
```

- PLAY

```
C: PLAY movie.Mjpeg RTSP/1.0
C: CSeq: 1
C: Transport: RTP/UDP; client_port = 2345
```

```
S: RTSP/1.0 200 OK
S: CSeq: 1
S: Session: 123456
```

- PAUSE

```
C: PAUSE movie.Mjpeg RTSP/1.0
C: CSeq: 2
C: Transport: RTP/UDP; client_port = 2345
```

```
S: RTSP/1.0 200 OK
S: CSeq: 2
S: Session: 123456
```

- TEARDOWN

C: TEARDOWN movie.Mjpeg RTSP/1.0
C: CSeq: 3
C: Transport: RTP/UDP; client_port = 2345

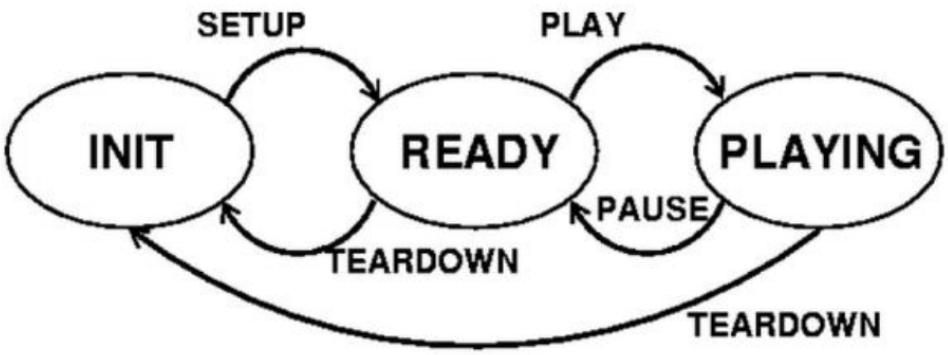
S: RTSP/1.0 200 OK
S: CSeq: 3
S: Session: 123456

2.1.3 Session's State

Another aspect of Server Respond to mention is the session's state. Although the server can respond to all the client's requests, sometimes there are states that server should not respond, which is called session's state. In this assignment, we have 3 states.

- INIT
- READY
- PLAYING

At the beginning, Client should be in INIT state, and Server only responds to Client's request at the correct state according to this diagram.



2.2 Main functionalities of GUI

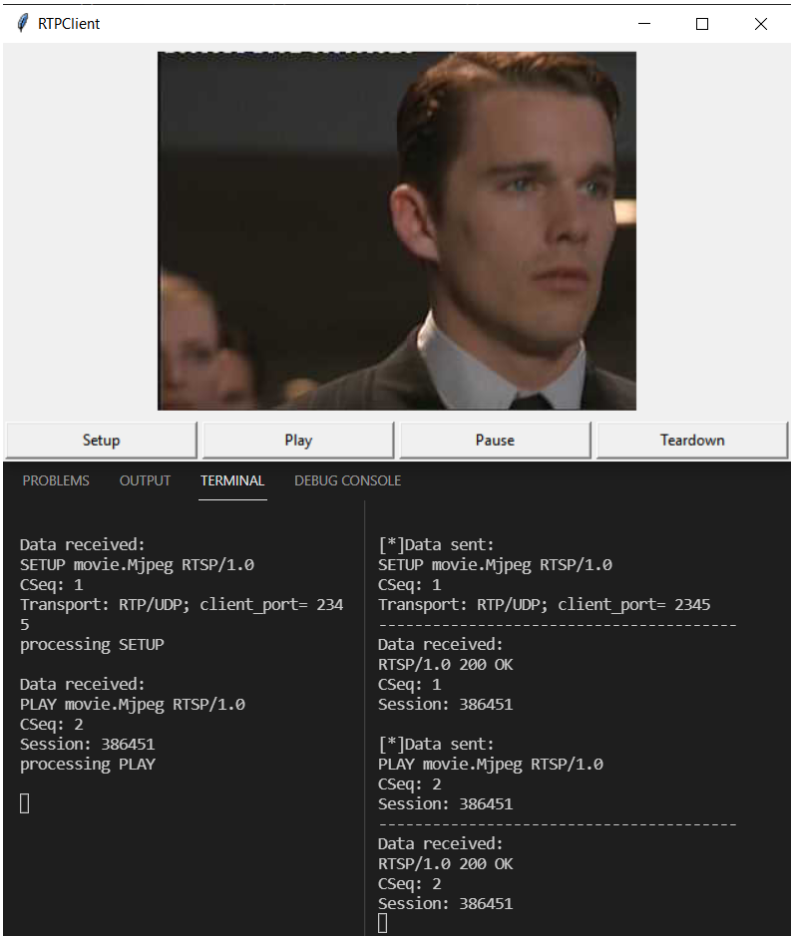
2.2.1 Play video

After the Client makes a PLAY request in READY STATE, server will create a RTP socket to transfer file to VideoStream.py, in which it will be separated into frames and put each frame into RTP packet data per 0.05 seconds.

For each RTP packet, it will be formatted using bit-wise twidling in RTP packet header format:

Bit Offset	0-1	2	3	4-7	8	9-15	16-31
0	Version	Padding	Ext.	CSRC Count	Marker	Payload Type	Sequence Number
32	Timestamp						
64	Synchronization Source (SSRC) Identifier						
96	Contributing Source (CSRC) Identifier						
96+32*CC	Payload						

Once put, packets will be sent to Client side where it will be decoded into frames for GUI. And in the end we'll have this GUI where video is showing 1 consecutive frame per 0.05 second



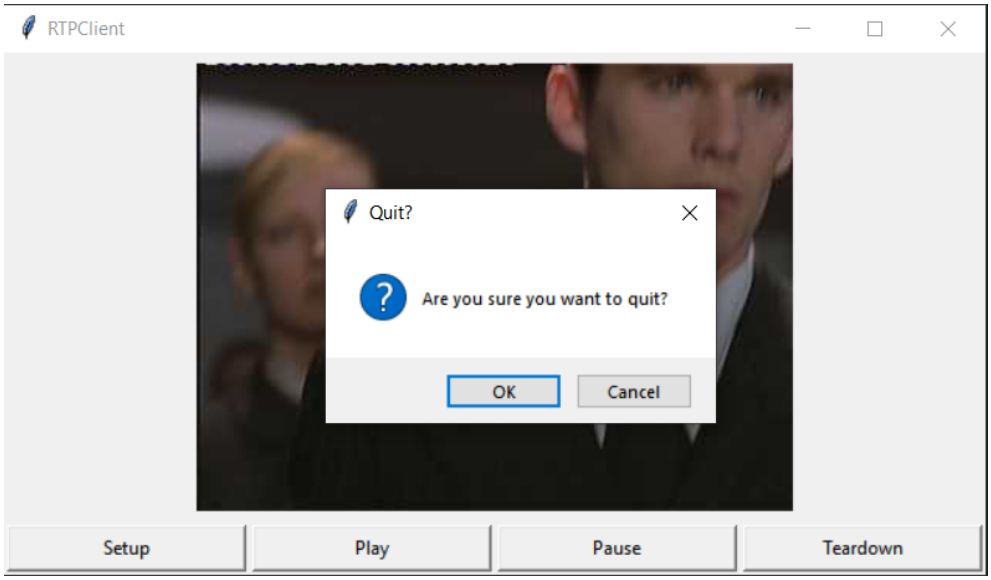
2.2.2 Pause video

While at PLAYING STATE, Client can make a PAUSE request, which basically stops sending frames to Client side. Note that even though frames are not sent to VideoStream.py, the current address of the frame is still saved and continued if Client make a PLAY request.

2.2.3 Terminate program

There are two ways to terminate program:

- TEARDOWN: Server will immediately stop frame conversion, close connection between Client and Server and Client's state will turn into INIT.
- Close the GUI: Another way to call TEARDOWN is via GUI's closing window itself. Slightly different, GUI will create a pop-up for confirmation

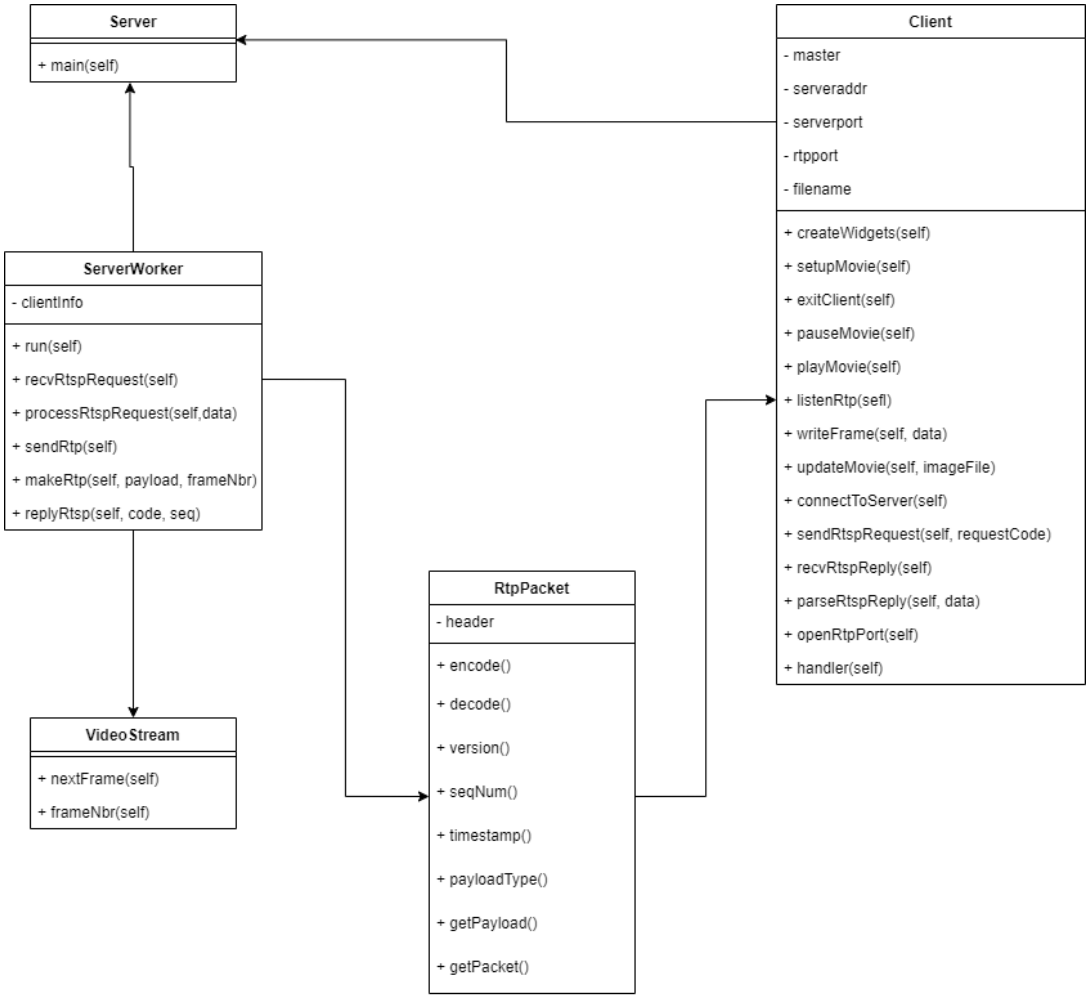


If Client chooses "Cancel", nothing happens and Client can continue making requests to the Server. If Client chooses "Ok", TEARDOWN will be called and Clent's state will turn back to INIT.

3 List Of Components

- Client application:
  - Grapical User Interface (GUI)
  - RTP packet receiver
  - Video Renderer
- Server:
  - RTSP request handler
  - RTP packet sender
- RTP packet handler
- Video stream

4 Class Diagram



5 User Manual

At source code, to start the Server side, type command.

py Server.py <server_port>

With <server_port> is RTSP port. Standard RTSP port is 554, but for this assignment it should be larger than 1024. Example: **py Server.py 1031**

Once compiled successfully, the GUI should look like this.

Then, start the Client side with command

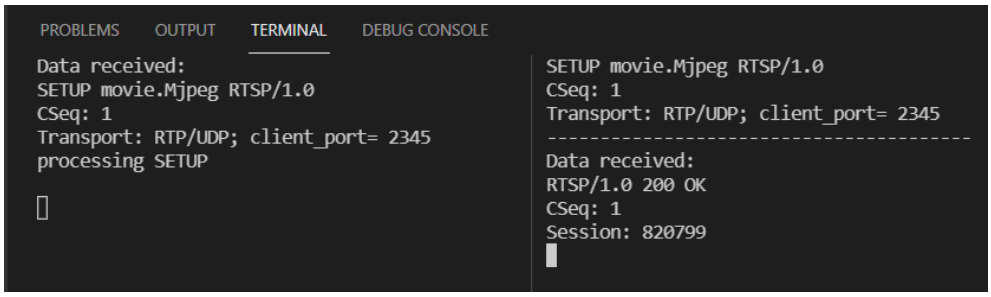
py ClientLauncher.py <server_host> <server_port> <RTP_port> <video_file>

In which:

- <server_host> is Server's IP or your local computer's IP.
- <server_port> is the <server_port> from the Server.
- <RTP_port> to receieve RTP packets. But in this context any number can work.
- <video_file> is path description of your wanted file.

Example: **py ClientLauncher.py 127.0.0.1 1031 2345 movie.Mjpeg**

Then if compiled succesfully, you can send RTSP requests on the GUI, and going back to the Terminal you will see the Terminal is outputting RTSP requests.



A normal session should look like this (Client on the left side and Server on the right side)