

# STREAMING VIDEO SERVER WITH MEDIA PLAYER GUI

# Contents

- **Requirement analysis :**
  - 1.1 Introduction.....
  - 1.2 Specific Requirements.....
    - 1.2.1 Interface requirement.....
    - 1.2.2 Functional requirement.....
    - 1.2.3 Non-functional requirement.....
- **Class diagram:**
- **User manual:**
  - Start the Server and Client.....
  - Use the client GUI to operate the system(original project).....
  - Use the client GUI to operate the system(extended project).....
- **Source code implementation:**
  - Explanation of function implementation:
    - Function sendRtspRequest(self, requestCode) of class Client.....
    - Function parseRtspReply(self, data) of class Client.....
    - Function openRtpPort(self) of class Client.....
    - Function encode(self, version, padding, extension, cc, seqnum, marker, pt, ssrc, payload) of class RtpPacket.....
  - Interaction flow of the whole system:
    - 4.2.1 In the initial starting:.....
    - 4.2.2 In the main flow when client user triggers actions.....
- **Extension:**
  - 5.1 The new PLAY button.....

5.2 The new DESCRIBE button.....	
5.3 The SWITCH button.....	
5.4 The STOP button .....	
5.5 The TOTAL TIME & REMAINING TIME bar.....	

## • Requirement Analysis

### • Introduction

In this assignment, we will implement a video streaming server and client that communicate through the use of ***Real Time Streaming Protocol*** (RTSP). The data sent to client by server will be packetized by ***Real-time Transfer Protocol*** (RTP). In this paper, we will provide detailed specification as well as implementation step of the source code below.

### • Specific requirement

This part consists of three main part: interface requirements, functional requirements and non-functional requirement

#### • Interface requirements

When the connection between the client and the server has been successfully established. We need an interface to display the data received by the server. Thus, the interface that the client sees should satisfy the following requirements:

- ☐ A window interface must appear when the client successfully connects to the server.
- ☐ The user interface shall follow basic style and functionality conventions of the operating system it runs on.
- ☐ The connection interface should work on any devices that run Windows, MacOS or Linux operating systems.
- ☐ The size of the part the displays the media should be at least 19 pixels in height.
- ☐ When the server notifies the client of some information, the interface should show a pop up window with the required message.
- ☐ The interface should display each functionality with each different button. Each button should have some text to indicate its usage.

- **Functional requirements**

This part will give a fundamental presentation of the operations that are available to the server and the client. Please note that the below requirements ending with "(Extension)" are related to the extend part of the assignment. First, the requirements for the **Server**:

- ☐ The server should be the first to initialize the RTSP socket that is responsible for accepting messages from the client and sending back the necessary reply.
- ☐ The server accepts connection when receiving request from the client.
- ☐ Server initializes both the RTSP and RTP connection.
- ☐ Server should always be listening to RTSP messages from the client.
- ☐ Server should send RTSP reply to client every time it receives messages.
- ☐ Upon receiving SETUP request, the server reads data from the video stored on the disk of the server with the name as requested by the client. If no error occurs, the server sends RTSP reply back to client.
- ☐ The server randomizes a session ID after reading the video data from the disk.
- ☐ The server should also calculate the total streaming time of the media when receiving SETUP request.(Extension).
- ☐ Upon receiving PLAY request, server should create a new socket for RTP/UDP and start sending media data to the client.
- ☐ Upon receiving PAUSE request, server should stop sending media data to the client.
- ☐ Upon receiving TEARDOWN request, server should stop sending data as well as close the RTP port.
- ☐ Upon receiving DESCRIBE request, server should send the description of the stream using, the format of the media as well as the encoding style.

- ☐ In the process of sending media data to the client, server must packetize the video data into a RTP packet.
- ☐ All reply from the server should include sequence number and session id.

## **The Client**

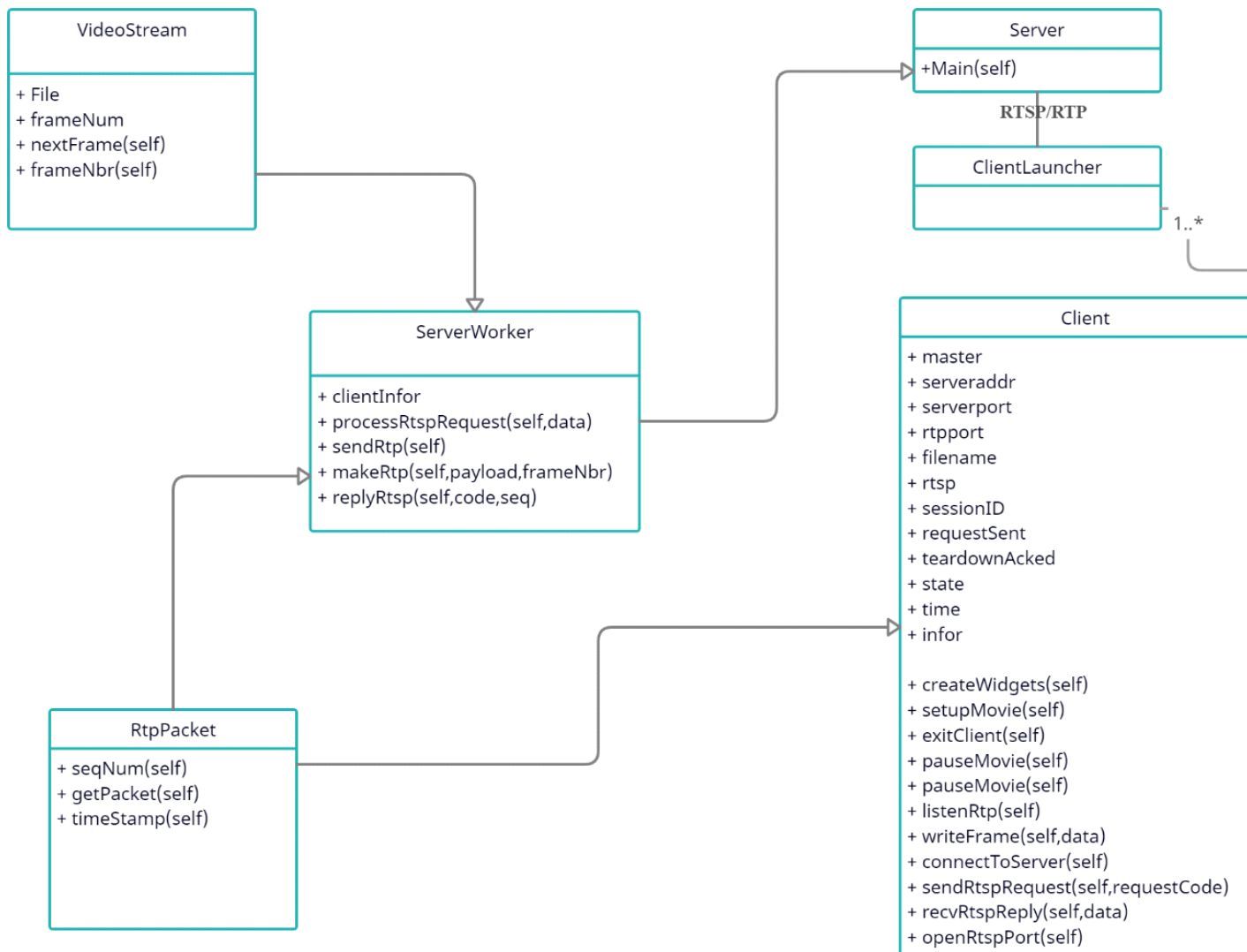
- ☐ When initialized by the user, the client should build the widget that makes up the interface, create a new RTSP socket and send connection request to the server.
- ☐ The client should always listen for RTSP and RTP messages from the server.
- ☐ When the user of the client clicks PLAY button, the client starts listening for RTP packet.
- ☐ Upon receiving SETUP reply from the server, the client creates a new datagram socket to receive RTP packets from the server.
- ☐ Upon receiving PAUSE reply from the server, the client stops listening to the client for RTP packet.
- ☐ Upon receiving DESCRIBE reply from the server, the client shows a pop up window to display the received description. (Extension)
- ☐ Upon receiving TEARDOWN reply from the server, the client closes the RTSP and RTP socket.
- ☐ The client must also display the total time of the video after receiving it from the server. (Extension).
- ☐ The client must also display the total time of the video after receiving it from the server. (Extension).
- ☐ When the user clicks on any button, the client should send the corresponding request to the client.

- **Non-functional requirements**

- ☐ The interface of the service must always be responsive.

- The client must establish the RTP connection within 0.5 seconds. Otherwise the system will raise timeout error.
- There should be no delay in the communication between the client and the server using RTSP protocol and no delay in displaying the frames.
- The performance of the service must be the same across different operating systems.

## • Class diagram:

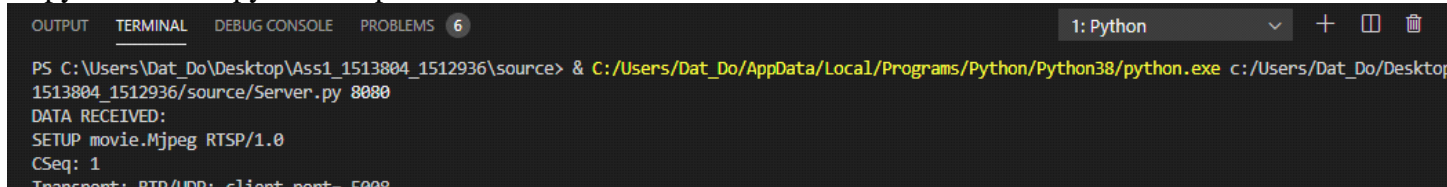


- **User manual:**

- **Start the Server and Client**

Start the server with the command:

```
$ python Server.py server_port
```

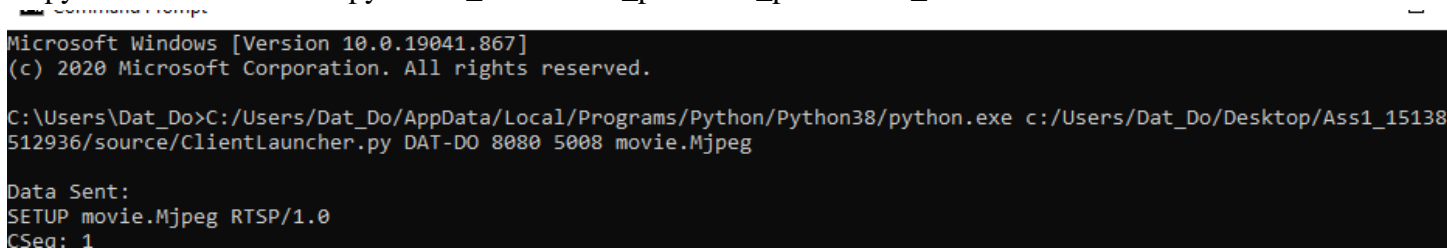
A screenshot of a terminal window with a dark background. The title bar shows '1: Python'. The terminal output shows the command being executed and the server's response: 'DATA RECEIVED: SETUP movie.Mjpeg RTSP/1.0 CSeq: 1 Transport: RTP/UDP; client\_port= 5008'.

```
PS C:\Users\Dat_Do\Desktop\Ass1_1513804_1512936\source> & C:/Users/Dat_Do/AppData/Local/Programs/Python/Python38/python.exe c:/Users/Dat_Do/Desktop/1513804_1512936/source/Server.py 8080
DATA RECEIVED:
SETUP movie.Mjpeg RTSP/1.0
CSeq: 1
Transport: RTP/UDP; client_port= 5008
```

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

Start the client with the command:

```
$ python ClientLauncher.py server_host server_port RTP_port video_file
```

A screenshot of a terminal window showing the client's execution. It displays the command and the output: 'Data Sent: SETUP movie.Mjpeg RTSP/1.0 CSeq: 1'.

```
Microsoft Windows [Version 10.0.19041.867]
(c) 2020 Microsoft Corporation. All rights reserved.

C:\Users\Dat_Do>C:/Users/Dat_Do/AppData/Local/Programs/Python/Python38/python.exe c:/Users/Dat_Do/Desktop/Ass1_1513804_1512936/source/ClientLauncher.py DAT-DO 8080 5008 movie.Mjpeg

Data Sent:
SETUP movie.Mjpeg RTSP/1.0
CSeq: 1
```

Where **server\_host** is 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, and **video\_file** is the name of the video file you want to request (the video file must be in the same directory as the video stream project).

The server is set with **server\_port** 8080. The client is set with **server\_host** 127.0.0.1, **server\_port** 8080, **RTP\_port** 5008 and **video\_file** movie.Mjpeg (the name of the default video file provided by the original source code).

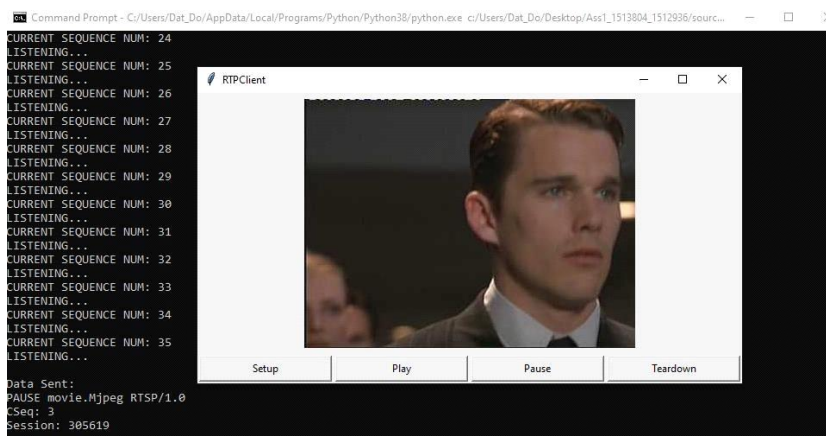
- **Use the client GUI to operate the system(original project):**

After starting the server and the client of the extended project, the client will open a connection to the server and pops up a window as follow:



You can send RTSP commands to the server by pressing the buttons. A normal succession of actions goes as follows:

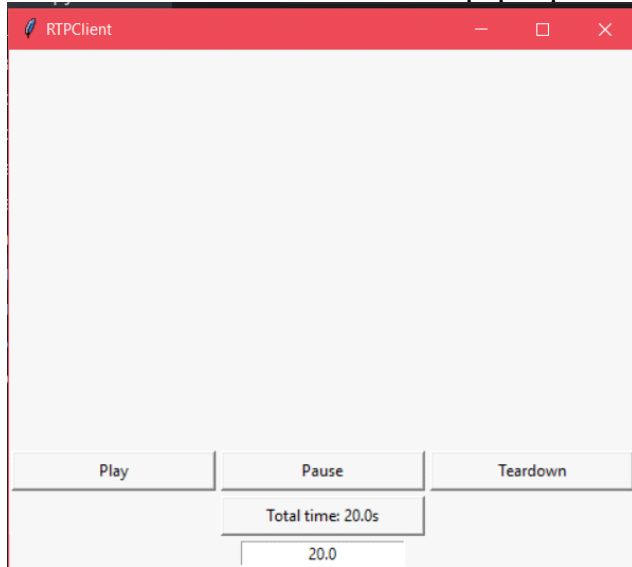
- You must first press **Setup** button to connect the client to the server, set up the session and transport parameters.
- You then press **Play** button to start the playback. After this step, a video frame is displayed above the buttons.
- You may pause the video during playback by pressing **Pause** button.
- Finally, you can press **Teardown** button to end the system operation via terminating the session and closing the connection. The GUI window automatically closes after this step.



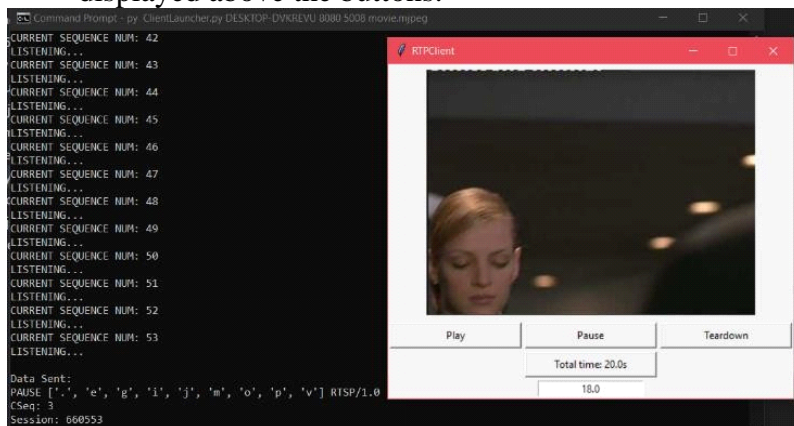
- **Use the client GUI to operate the system (extended project):**



After starting the server and the client of the extended project, the client will open a connection to the server and pops up a window as follow:



- You can send RTSP commands to the server by pressing the buttons. A normal succession of actions goes as follows:
- You first press **Play** button to start the playback. After this step, a video frame is displayed above the buttons:



- The total time, remaining time and name of the video being played are displayed.
- You may pause the video during playback by pressing **Pause** button.
- You may view the information of the media stream by pressing **Describe** button. Then a message box about the session description is displayed.
- **Source implementation:**

- **Explanation of function implementation:**

- **Function `sendRtspRequest(self, requestCode)` of class `Client`**

Function **`sendRtspRequest(self, requestCode)`** is a method of class `Client` that is used to send RTSP request from the client to the server:

**Parameter:** This method takes one additional parameter `requestCode` (`self` is the default parameter for all methods in Python).

**Body operation:** The body of the method is divided into four main branches. Each branch is triggered respectively by each of the only four different actions from the user.

- ☐ On **Setup** request: The **Setup** request is valid if both the **requestCode** is **SETUP** and the **state** of the client is **INIT**.
- ☐ On **Play** request: The **Play** request is valid if both the **requestCode** is **PLAY** and the **state** of the client is **READY**.
- ☐ On **Pause** request: The **Pause** request is valid if both the **requestCode** is **PAUSE** and the **state** of the client is **PLAYING**.
- ☐ On **Teardown** request: The **Teardown** request is valid if both the **requestCode** is **TEARDOWN** and the **state** of the client is **INIT**.

- **Function `parseRtspReply(self, data)` of class `Client`:**

Function **`parseRtspReply(self, data)`** is a method of class `Client` that is used to parse the RTSP reply sent back from the server.

**Parameter:** This method takes one additional parameter **`data`** (**`self`** is the default parameter for all methods in Python).

**Body operation:**

First, the method parses the data received into list of lines. Then it takes the sequence number from the second line of the data.

If the extracted server's reply sequence number is the same as the request's, it then extracts the session ID from the third line of the data.

If this is the first reply received by the client, then its **sessionId**, currently 0, should be assigned to the **sessionId** extracted from the server's reply. If it's not the first reply, checking must be made and the following process only executes if the **sessionId** from the server and the client are equal.

The execution of the method after that is divided into four main branches:

- On **Setup** reply: If the **requestSent** of the client is also **SETUP**, then the RTSP state is set to **READY** and the client opens an RTP port to start receiving data frame.
- On **Play** reply: If the **requestSent** of the client is also **PLAY**, then the RTSP state is set to **PLAYING**.
- On **Pause** reply: If the **requestSent** of the client is also **PAUSE**, then the RTSP state is set to **READY** and the thread which is created to continuously listen for RTP packets sent from the server exits.
- On **Teardown** reply: If the **requestSent** of the client is also **TEARDOWN**, then the RTSP state is set to **INIT** and the attribute **teardownAcked** is set to 1 to close the socket and terminate the client's operation.

- **Function openRtpPort(self) of class Client:**

Function **openRtpPort(self)** is a method of class Client that is used to open RTP socket binded to a specific port so that it can start listening for data from the server.

**Parameter:** This method takes no parameter (**self** is the default parameter for all methods in Python).

**Body operation:**

First, the method initializes a new datagram socket to receive RTP packets from the server.

Then it sets the timeout value of the socket to 0.5sec.

Finally, it tries to bind the socket created to the address using the RTP port input by the client user.

- **Function encode(self, version, padding, extension, cc, seqnum, marker, pt, ssrc, payload) of class RtpPacket**

Function **encode(self, version, padding, extension, cc, seqnum, marker, pt, ssrc, payload)** is a method of class RtpPacket that is used to encode the RTP packet with 12- byte header and payload (data of one frame).

**Parameter:** This method takes up to 8 parameters as the input for the header fields and 1 parameter **payload** as the data of one frame (**self** is the default parameter for all methods in Python).

**Body operation:** The header of an RTP packet has 9 fields of 12 bytes in total. They are 2 bits of **version**, 1 bit of **padding**, 1 bit of **extension**, 4 bits of **cc**, 1 byte of **marker**, 7 bit of **pt**, 16 bit of **sequence number**, 32 bit of **timestamp** and 32 bit of **ssrc**.

- **Interaction flow of the whole system:**

- **In the initial starting:**

- **Server:** When Server start, an object of class **Server** is created, it then initializes an RTSP socket, binds it to a particular **server\_port** defined as an argument, makes it start listening to several RTSP requests. The **Server** also runs a **ServerWorker** object as its backend handler and starts listening for RTSP request from the beginning.
- **Client:** When Client start, an object of class **Client- Launcher** is created, it then takes the **server\_address**, **server\_port**, **rtp\_port** and **fileName** from the command-line arguments and passes them to a **Client** object, which is the backend handler. After that, the **Client** object starts building the GUI and initializes an RTSP socket to communicate with the **Server** through the **connectToServer()** method.

- **In the main flow when client user triggers**

**action: Client main flow:**

When user press the **Setup** button, this action triggers the **setupMovie** method of the Client. Then it would call **sendRtspRequest** on **SETUP** request. As **SETUP** is the first request in the sequence of actions, when this request is sent, the **Client** will send the RTSP request to the **Server** in a predefined format and start a thread that runs the **recvRtspReply** method, which will keep checking and receiving RTSP replies from the server repeatedly, and only terminates when a **TEARDOWN** request is detected. If an RTSP reply is caught, it is parsed by passing into the **parseRtspReply** method and the next operation of the **Client** is taken based on the contents of the reply. If the reply implies the request sent to be a **SETUP** request, **Client** will initialize an RTP port that connect to that of the **Server**.

When client user presses **Play** button, an event thread is created and calls **listenRtp** to listen for RTP packets sent from the **Server** repeatedly. Then a **PLAY** request is passed into **sendRtspRequest**, where the request is encoded to a predefined format and sent to the **Server**. Having received a **PLAY** request, the **Server** sends an RTSP reply that implies a successful RTSP communication and starts sending the data of the video file frame by frame as an RTP packet.

Afterthat, the **Client** starts receiving RTP packets that contain frame data. Finally, after a sequence of actions taken, the RTP packets are decoded and the payloads (each payload is data of one frame) are displayed successively into the window by a particular image-building method.

When client user presses **Pause** button, a **PAUSE** request is passed into **sendRtspRequest**, where the request is encoded to a predefined format and sent to the **Server**. Having received a **PAUSE** request, the **Server** sends an RTSP reply that implies a successful RTSP communication. Immediately after sending and receiving confirmation reply of a **PAUSE** request from the **Server**, the **Client** terminates the thread that is created before to listen for RTP packets sent. Then the window stops updating video.

When client user presses **TEARDOWN** button, a **TEARDOWN** request is passed into **sendRtspRequest**, where the request is encoded to a predefined format and sent to the **Server**. Having received a **TEARDOWN** request, the **Server** sends an RTSP reply that implies a successful RTSP communication. Immediately after sending and receiving confirmation reply of a **TEARDOWN** request from the **Server**, the **Client** raises a **teardownAcked** which flags closing the RTSP and RTP sockets, terminating every executing objects and methods, and destroying the window.

### **Server main flow:**

When the **Server** receives an RTSP request, it encodes the request by the **processRtspRe-quest** method and starts executing based on what the request is:

When **Server** receives **SETUP** request, 1First, it initializes a **VideoStream** object which handles all the processes operated on the video file. Then it extracts the information of the **Client**'s RTP port number and replies with an **OK 200** message.

When **Server** receives **PLAY** request, it creates a new socket for sending RTP packets. Then it replies with an **OK 200** message and immediately creates a new **sendRtp** thread that starts sending RTP packets. In **sendRtp** method, the **VideoStream** will continuously extract the next frame of the video. This data of one video frame is then merged with the RTP header to make a complete RTP packet. Finally, each packet generated from the **Server** is delivered to the **Client** one by one.

When **Server** receives **PAUSE** request, It replies with an **OK 200** message. Then it terminates the **sendRtp** thread to stop the sending of RTP packets.

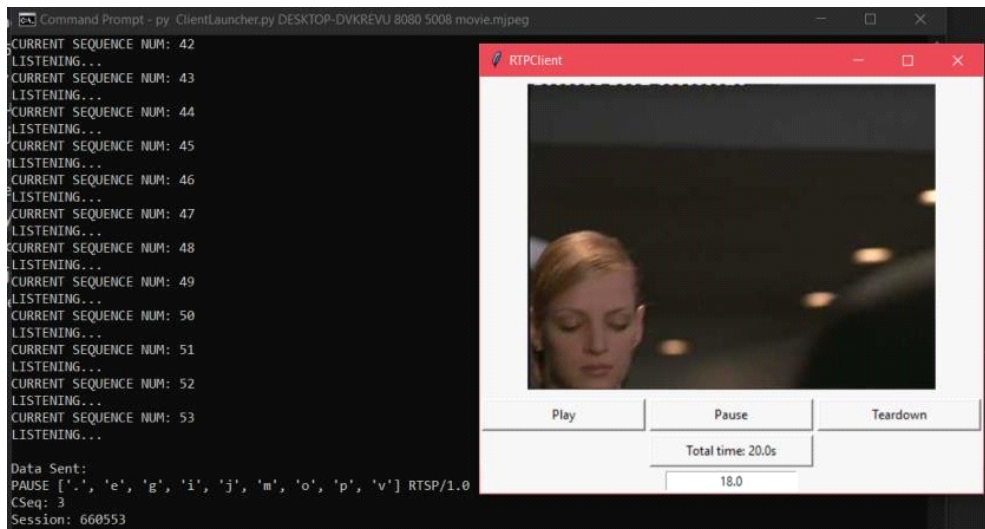
When **Server** receives **TEARDOWN** request, Then it terminates the **sendRtp** thread to stop the sending of RTP packets. Finally, it closes RTP socket to stop the connection to the **Client**.

- **Extension:**

In this question, we are required to implement the streaming service similar to a regular media player such as RealPlayer or Windows Media Player. These player only represents three main buttons, namely **PLAY**, **PAUSE**, **STOP**. Compared to the basic the design of this assignment, we have an excessive **SETUP** , plus the use of the **TEARDOWN** button is also uncommon among media player. In this part, we will address both of these two issues.

- **The new playButton:**

Given that SETUP is mandatory in an RTSP interaction as we need it to establish the connection to the server, the **SETUP** button is no longer necessary. Instead, it would be a more effective approach to set up the connection to the server the moment we create the client. The first step is to remove the SETUP button in the UI design. After that, we put the *setup- Movie* function into the constructor of the client class.

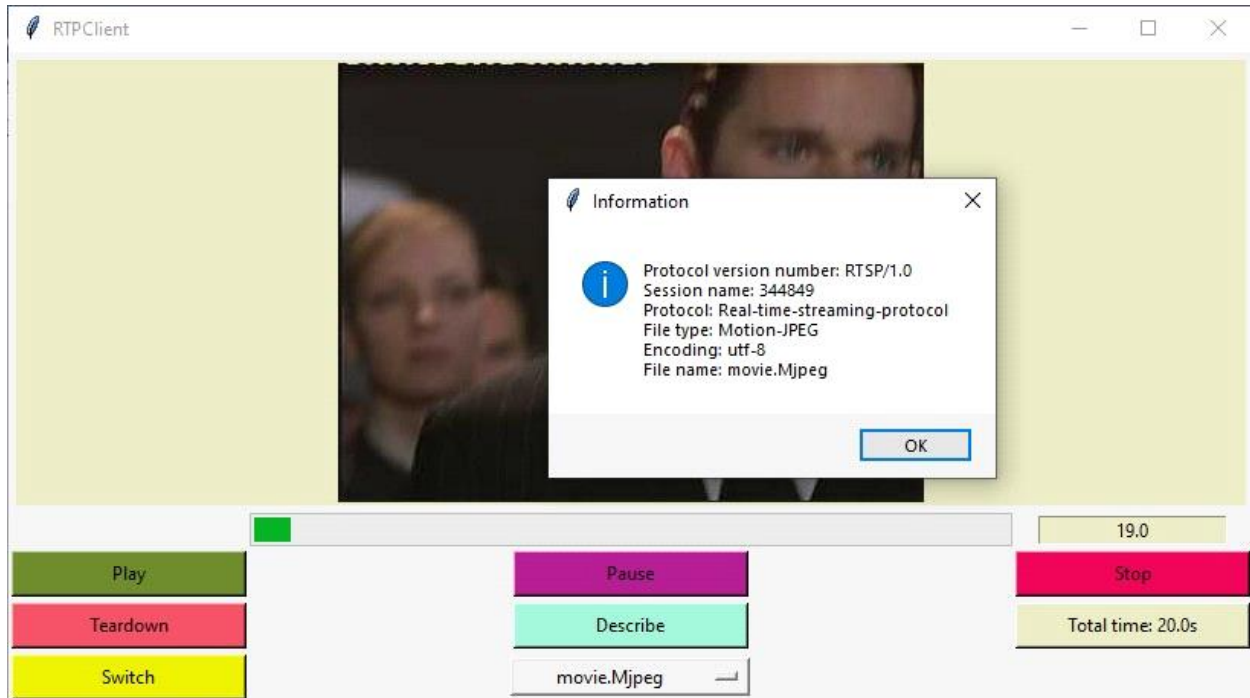


- **The new describeButton:**

For this part, we will implement a new functionality called **DESCRIBE**, which when requested by the client will send back a session description file which tells the client what kinds of streams are in the session and what encodings are used.



The ***showinfo*** function will display a pop up window containing the information received from the server. When the user click on the DESCRIBE button, the result should be as follow:



- **The switchButton:**

First, we add the GUI necessary for this feature. Adding a new button SWITCH, Option- Menu() to the createWidgets() function then a handler for each of them named switchMovie() and updateOptionsMenu(). updateOptionsMenu() is called every time the client receives a list of file names available on the server.

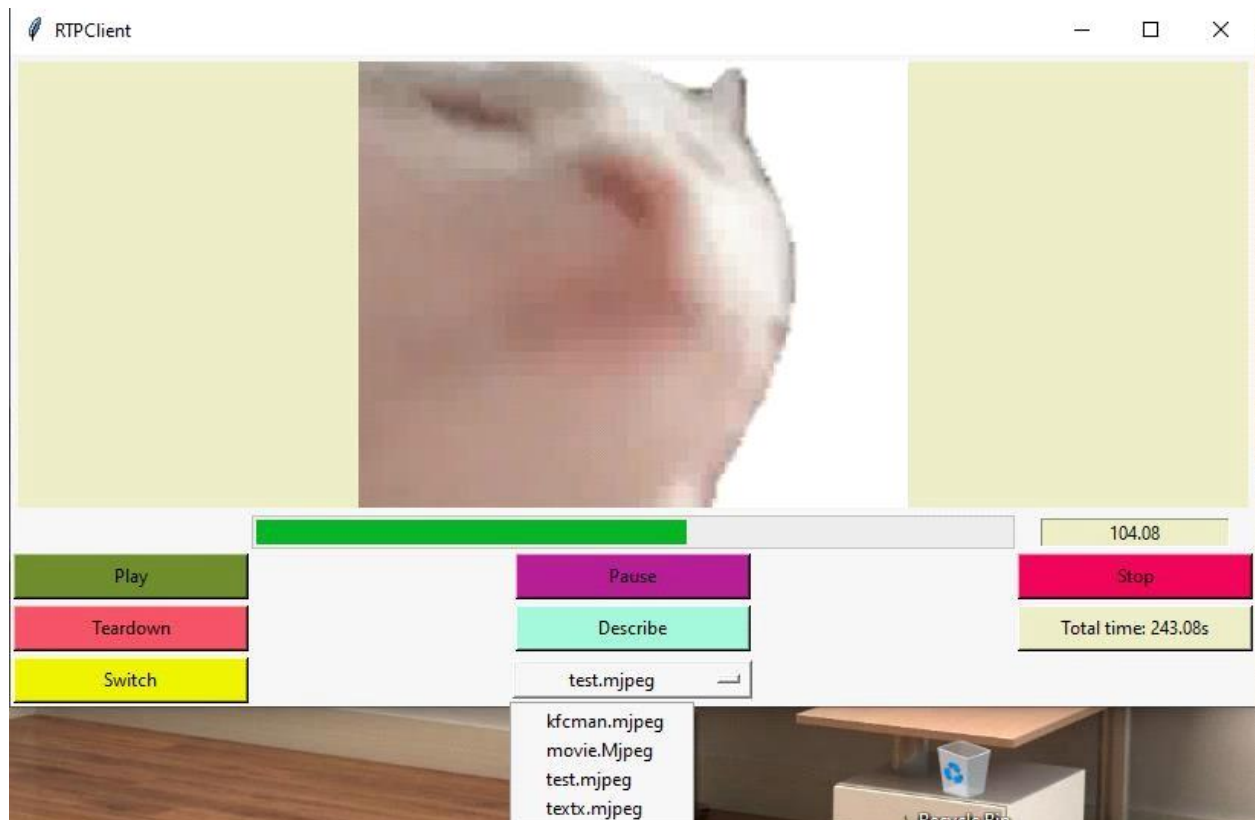
switchMovie() will send the RTSP request to the server with the request code SWITCH and the new file name the user requested. The self.filename as we have known is the current file that the client is asking from the server. In this section, we add self.changedFileName as the new file that the user requests.

When the user selects a new movie from the list received from the server, the StringVar variable named fileNameVar will store the name of the selected file. Therefore each time the user chooses a new file, there should be a call back

function updating the requested filename. `fileNameCallBack()` is created to assigns the current filename the client is sending a request to the server to the new file name the user selected.

You may select another video from a list of videos available in the server through the following ordered steps:

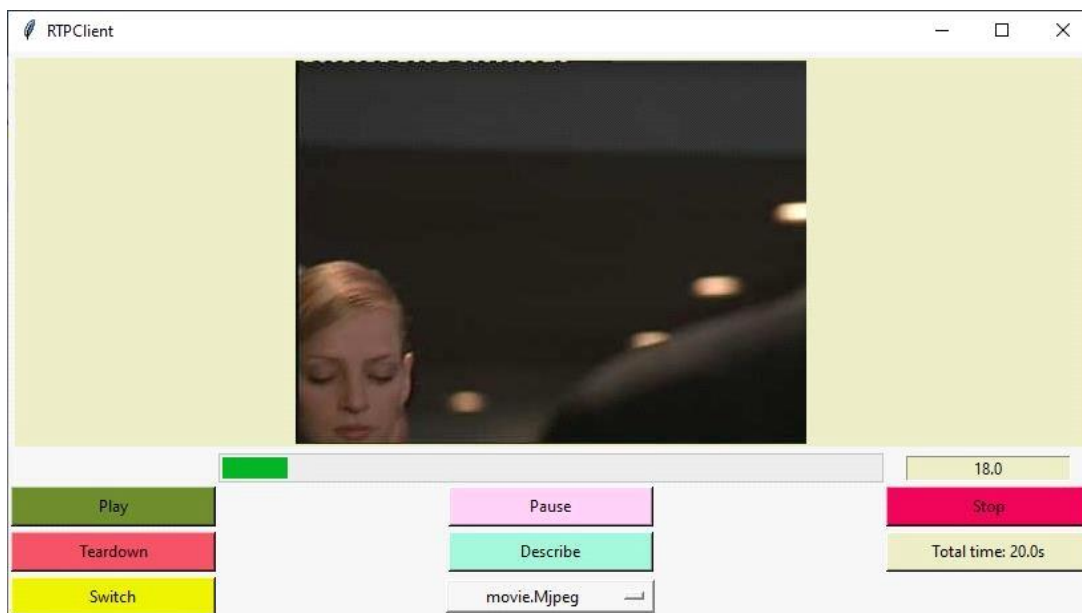
- ☐ If there is a video playing, you have to pause or stop it.
- ☐ Then, choose the names of the video you want to play in the list videos.
- ☐ After that, press the **Switch** button, the video will be set up automatically and ready to play.
- ☐ You can press **Play** button to begin playback.



- **The StopButton:**

The regular media player usually doesn't have a TEARDOWN button, instead the STOP button is used. The functionality of this button is to stop the media completely (not pause), so when the user continues to play it again, it shall start from the beginning. In this part we are going to implement the same procedure.

As for the question in the assignment, when the user click the STOP button, it is not appropriate to send the TEARDOWN request as it will close both the RTP and RTSP connection completely. After the STOP button is clicked, the user should be able to replay the video from the beginning, the TEARDOWN request will close everything.



- **The Total Time and Remaining Time:**

To display the remaining time, each time the server sends back an RTP packet it also includes a current frame number, so we take the total frames of the video minus that

current frame number and divide it by FPS to get the remaining time. The `updateCountDownTimer()` will refresh the GUI every one second.

