

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



COMPUTER NETWORK
CC03

ASSIGNMENT 1

Network Application Programming (Socket Programming)

INSTRUCTOR:

Dr. Nguyễn Lê Duy Lai (Lab)

Assoc. Prof. Phạm Trần Vũ

MEMBER:

Vũ Minh Quang - 1852699

Nguyễn Quang Phúc - 1852668

HCM, November 9th, 2020

TABLE OF CONTENTS

I. Requirements Analysis	3
Functional Requirements:	3
Architecture and Design Requirements:	3
THE CLIENT	3
THE SERVER	4
II. Function Description	4
Client	4
Functions:	5
setup	5
play	6
pause	6
teardown	7
describe	7
jumpTo	8
pauseBeforeJump	8
exit	9
requestList	9
switchChannel	10
changeSpeed	10
receiveRtp	10
RtpPacket	11
Functions:	12
Decode	12
Encode	12
getPacket	13
ServerWorker	13
Functions:	14
run	14
recvRtspRequest	14
processRtspRequest	14
sendRtp	15
makeRtp	15
replyRtsp	15
VideoStream	16
Functions:	16
buildIndex	16
nextFrame	17
frameNbr	17

jumpTo	17
getFileSize	18
getNumberOfFrame	18
Server	18
Functions:	19
main	19
VideoConvert	19
III. Class Diagram	20
IV. Extensions:	21
Extension 1	21
Implementation:	21
Extension 2	22
Extension 3	22
Implementation:	23
Extension 4	23
Implementation:	23
Extension 5	24
Implementation:	25
Bonus Features	25
Implementation:	26
V. Summative Results' Evaluation	26
VI. User Manual	27
Setup	27
Core Functions	29
Extended Functions	32

I. Requirements Analysis

1. Functional Requirements:

When the Client presses the '**Setup**' button the session is being sat up and ready to play the playback.

When the Client presses the '**Play**' button they must be able to see the playback start playing.

When the Client presses the '**Pause**' button the playback must be paused at the current frame but still able to proceed to the next frame play when the Client selects the '**Play**' button.

When the Client presses the '**TearDown**' button the playing session is terminated and the Client has to '**Setup**' to play again.

The Server always replies to all the messages the Client sends.

2. Architecture and Design Requirements:

THE CLIENT

Here are all the replies from the Server:

200: SUCCESSFUL

404: FILE_NOT_FOUND

500: CONNECTION ERROR

When the Client starts, opens the RTSP socket to the server and uses this socket to send all the requests.

Insert the Transport header which contains the port for the RTP data socket, the sequence number, and the playback file's name while sending the '**SETUP**' request.

Get the RTSP session ID from the Session header of the Server's response

Create a datagram socket for receiving RTP data and the time out is 0.5 seconds.

The '**PLAY**' request contains the Session header and the Session ID (returned in the '**SETUP**' response).

The **'TEARDOWN'** request contains the Session header and the Session ID (returned in the **'SETUP'** response).

The **'CSeq'** header must be in every request. The **'CSeq'** header starts at 1 and is incremented in each request sent.

Keep the Client's state up-to-date, the state changes when it receives the reply from the server. There are 3 states needed to keep track of:

INIT: In this state, the Client can only 'Setup' the session

READY: In this state, the Client can 'Teardown' the session and 'Play' the playback.

PLAYING: In this state, the Client can 'Pause' the playback at the current frame and 'Teardown' the session.

THE SERVER

When received the **'PLAY'** request, the server creates the RPT-encapsulation of the video frame and sends the frame to the Client through UDP every 50ms.

The RPT-Packet has the encode function for encapsulation.

The RPT-Packet has the format:

RPT-version (V): 2 bits

Padding (P): 1 bit

Extension (X): 1 bit

Contributing Sources (CC): 4 bits

Marker (M): 1 bit

Payload Type (PT): 7 bits (MJPEG = 26)

Sequence Number: 16 bits

Timestamp: 32 bits

SSRC: 32 bits

II. Function Description

In this Function Description, we will use the Numpy style to write a docstring And we will only write the description for Client and RtpPacket class.

Client

Description:

The Client object handles the actions that are taken when the buttons are pressed and sets up all the UI & event handlers for the application.

Parameters of constructor:

rootTK: Tk

The root GUI to create GUI applications

serverAddr: str

Specifies the hostname of the server

serverPort: str

Specifies the unique port of the server

rtpPort: str

Specifies the port number of RTP

fileName: str

The playback file's name

Attributes:

master: Tk

This is where we store the root GUI

serverAddr: str

This is where we store the hostname of the server

serverPort: int

This is where we store the unique port of the server

rtpPort: int

This is where we store the port number of RTP

fileName: str

This is where we store the playback file's name

state: int

Store the current state of the RTSP in each session

sequence: int

Store the sequence number of the request

session: int

Store the session number

dropdownActivated: bool

Track if the dropdown list of playback is currently active or not

Functions:

setup

Description:

Call when the user presses the play button while the client is in the **INIT** state.

Setups the connection using RTP/UDP

First we setup the rtp socket using UDP, then we bind the rtp-socket to the port number. After that, we encode and send the message to the Server through **rtsp socket** and wait for the message to be response from the

server. If the response is 200 OK so the client is successfully connected to the Server, then we can start the setup process.

Parameters:

None

Raises:

A connection was already setup! (Warning)

The state is not INIT

Out of order sequence : <wrong seq message>. Expecting <expecting seq>

The return CSeq message is not as expected

Error <error msg>

The reply status from the server is **not** 200: Successful

Returns:

None

play

Description:

Call when the user presses the play button while the client is in the **READY** state.

Sends the **PLAY** request message to the server, receives the server's response, and creates a new thread for receiving the RTP packet from the server.

First we check if the state is currently '**READY**'. If so, we try to send the encoded message to the server through **rtsp socket** and read the reply message from the server. Then we try to create the new thread to receive the rtp packet from the server.

Parameters:

None

Raises:

The stream is already playing! (Warning)

Pressing the 'Play' button while the playback is playing

Returns:

None

pause

Description:

Call when the user presses the pause button while the client is in the **PLAYING** state.

Send the **PAUSE** request message to the server to stop the data sending to stop the playback at the current frame

Parameters:
None

Raises:
You need to setup the stream first! (Warning)
Didn't setup the connection
The stream is already paused! (Warning)
Already pressed the 'Pause' button

Returns:
None

teardown

Description:
Call when the user presses the stop button while the client is in the **PLAYING** or **READY** state.

Terminate the session and close the connection and change the playback state

First we also send the encoded message to the Server request to terminate the stream. Then we wait for the response from the Server, after that we can terminate the connection.

Parameters:
None

Raises:
Nothing to teardown (Warning)
Already close the connection and terminate the session

Returns:
None

describe

Description:
Call when the user presses the stop button while the client is in the **PLAYING** or **READY** state.

Pass the information about the media stream contains the session description file which has the kinds of stream, encodings method

We also try to send the message **DESCRIBE** to the server and wait for the full response then we can display the information on to the GUI.

Parameters:
None

Raises:
None

Returns:
None

jumpTo

Description:

Call when the user releases the scroll bar in the video's scrollbar. This function will help to jump to the frame the user wants to continue watching.

Pass the frame the user wants to jump to the Server. The Server will send the reply **200 OK** if it can jump to the requested frame.

Then we may jump to the expected frame, taken by the video's scroll bar and play the stream as soon as we receive the Server's response.

Parameters:
E: Event
Event argument

Raises:
None

Returns:
None

pauseBeforeJump

Description:

Call when the user holds the scroll bar in the video's scrollbar. This function will stop the stream because if we don't do this deadlock will occur.

If the stream is playing then we may call this function but if the stream is not playing then it will call **jumpTo** function.

Parameters:
E: Event
Event argument

Raises:
None

Returns:
None

exit

Description:
Call when the user presses the **X** button on the top right corner of the GUI.

It will teardown the stream by calling the **teardown** function and close **RTSP** socket, which is used to communicate with the Server. Then we terminate the GUI.

Parameters:
E: Event
Event argument

Raises:
None

Returns:
None

requestList

Description:
Call when the user presses the dropdown list of playbacks in the bottom of the GUI.

It will stop the video streaming session if it is playing and then request the **SWITCH** command to the server to retrieve the list of playbacks and add tag '(current)' if it is the same file.

Parameters:
E: Event
Event argument

Raises:
Error <error msg>
The reply status from the server is **not** 200: Successful
Out of order sequence : <wrong seq message>. Expecting <expecting seq>
The return CSeq message is not as expected

Returns:
None

switchChannel

Description:

Call when the user presses the playback in the dropdown list of playbacks.

It will get the name of the playback file that the user wants to play in the list. If the playback file is the same as the playing file then it will start playing the old file. Otherwise, it will teardown the connection and change back to **INIT** state (users have to press play to setup rtp connection for the new file and play again).

Parameters:

arg: many arguments

The arguments passed by the **trace** function

Raises:

None

Returns:

None

changeSpeed

Description:

Call when the user presses the dropdown list of playbacks' speed in the bottom of the GUI.

It will send the **CHANGESPEED** request with the delay to the server.

Parameters:

arg: many arguments

The arguments passed by the **trace** function

Raises:

None

Returns:

None

receiveRtp

Description:

Receive the data frame from the server and configure the image to the player UI. Calculate and display the data rate and statistics (packet loss rate) of the playback.

Parameters:

None

Raises:

Timeout Error!

After 0.5s - time out - but no packet received

Returns:

None

RtpPacket

Description:

The RtpPacket object handles RTP packets, it has the method for Client to de-packetize the data and for Server to encapsulate the data.

Parameters of constructor:

None

Attributes:

header: bytes

This is where we store 12 bytes header

payload: bytes

This is where we store the data of the frame

version: int

This is where we store the version code retrieved from the header

padding: int

This is where we store the padding bit

extension: int

This is where we store the extension bit

cc: int

This is where we store the 4 bits contribution sources

marker: int

This is where we store the marker field bit

pt: int

This is where we store the 7 bits payload

seqnum: int

This is where we store the 12 bits sequence number

timestamps: int

This is where we store the timestamp in the Python's time module
(32 bits)

ssrc: int

This is where we store the identifier for the server

frame: bytes

This is where we store the data of the frame

Functions:

Decode

Description:

Uses to decode the data frame when the Client receives the encapsulated data from the Server

The header of the packet is read byte-by-byte using bit masking and shifting technique to set individual bit

Parameters:

data : bytes

The data needed to decode

Raises:

None

Returns:

None

Encode

Description:

Uses to make the encapsulation of data from the Server before sending the packet to the Client. The header of the packet is create byte-by-byte using bit masking and shifting technique to set individual bit

Parameters:

version: int

This is where we store the version code retrieved from the header

padding: int

This is where we store the padding bit

extension: int

This is where we store the extension bit

cc: int

This is where we store the 4 bits contribution sources

marker: int

This is where we store the marker field bit

pt: int

This is where we store the 7 bits payload

seqnum: int

This is where we store the 12 bits sequence number

timestamps: int

This is where we store the timestamp in the Python's time module (32 bits)

ssrc: int

This is where we store the identifier for the server

payload: bytes

This is where we store the data of the frame

Raises:

None

Returns:

None

getPacket

Description:

Uses to get the whole packet - which contains both the header and the payload

Parameters:

None

Raises:

None

Returns:

bytes

The whole packet (header + payload)

ServerWorker

Description:

The ServerWorker object is used to receive, process and send the data and reply back to the client.

Parameters of constructor:

clientInfo: dict

This is where we store all the information needed to setup the stream.

Attributes:

state: int

This is where we store the state of the stream

clientInfo: dict

This is where we store all the information needed to setup the stream.

Functions:

run

Description:

Uses to receive requests from the client.

A new thread is created to handle the requests from each client.

Parameters:

None

Raises:

None

Returns:

None

recvRtspRequest

Description:

Receive RTSP request from the client.

Create the loop to process the RTSP request from the client.

Parameters:

None

Raises:

None

Returns:

None

processRtspRequest

Description:

Process the RTSP request sent from the client.

Get the request type, media file name, RTSP sequence number. Then process the request and reply to the client.

Parameters:

data: str

String of requests received from the client.

Raises:

None

Returns:
None

sendRtp

Description:
Send RTP packets over UDP.

Sending data with a delay and stopping sending data if request is **PAUSE** of **TEARDOWN**.

Parameters:
None

Raises:
None

Returns:
None

makeRtp

Description:
RTP-packetizer the video data.

Packet the data frame using RTP packet and encode the packet before sending to the client.

Parameters:
payload: bytes
Represents the payload (in this case this is the next frame).
frameNbr: int
Represents the payload's frame number.

Raises:
None

Returns:
data: bytes
Represents the data of both the header and the payload of the frame.

replyRtsp

Description:
Send RTSP reply to the client.

Send the reply code of **200 OK**, **FILE_NOT_FOUND_404**, **CON_ERR_500** and the encoded reply if it is the first 2 message types.

Parameters:

data: str

String of requests received from the client.

Raises:

None

Returns:

None

VideoStream

Description:

The VideoStream object is used to read the frames of the video file, get the frame number and process some helper functions for **JUMP** and other requests.

Parameters of constructor:

fileName: str

This is where we store the path of the current requested video file.

Attributes:

fileName: str

This is where we store the path of the current requested video file.

frameNum: int

This is where we store the number of the current frame.

indexTable: list

This is where we store all the positions of all the frames in the file in order for a better jumping mechanism.

Functions:

buildIndex

Description:

Uses to handle jump requests more efficiently.

Make the indexTable to store all the positions of all the frames in the file.

Parameters:

None

Raises:

None

Returns:

None

nextFrame

Description:

Get the next frame.

Read the first 5-bits to know the length of the frame and retrieve the frame data.

Parameters:

None

Raises:

None

Returns:

Frame data: bytes

The next frame data of the frame sequence.

frameNbr

Description:

Get frame number.

frameNbr is actually the index in indexTable + 1.

Parameters:

None

Raises:

None

Returns:

Frame number: int

The next frame number of the current frame.

jumpTo

Description:

Jump to the frame at the index taken from the indexTable.

Parameters:

frameIndex : int

The index of the frame in the indexTable.

Raises:
None

Returns:
None

getFileSize

Description:
Get the total size of the video file.

Parameters:
None

Raises:
None

Returns:
None

getNumberOfFrame

Description:
Get the total number of frames contained in the file.

This is the length of the indexTable.

Parameters:
None

Raises:
None

Returns:
None

Server

Description:
Bind the server port number to the RTSP/TCP socket and listen for client's requests.
Run the server worker to handle client's requests.

Parameters of constructor:
None

Attributes:
None

Functions:

main

Description:

Uses to get and bind the server port to RTSP/TCP packet protocol.

Start the ServerWorker to handle the client's requests.

Parameters:

None

Raises:

None

Returns:

None

VideoConvert

Description:

Stand alone tool to convert other video files to our special Mjpeg format.

Each jpeg is preceded by **5 byte 'string of digits'**.

Parameters of constructor:

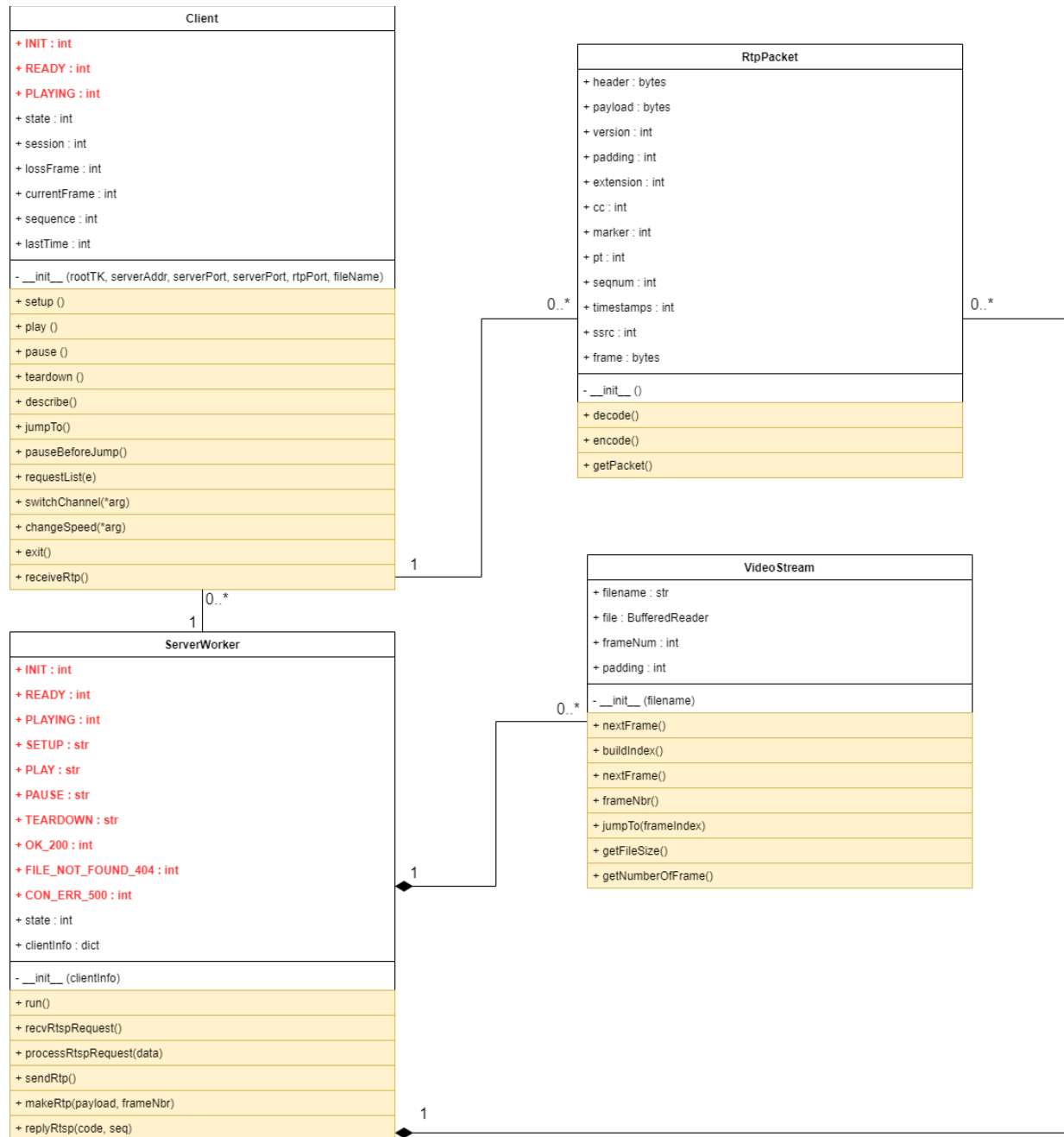
filename: str

Reference to the path of the file needed to be converted.

Attributes:

None

III. Class Diagram



IV. Extensions:

Extension 1

Calculate the statistics about the session. You will need to calculate RTP packet loss rate, video data rate (in bits or bytes per second), and any other interesting statistics that you can think of.

We have successfully added the statistic label on the right hand side of our GUI. The label displays the following information:

- RTP packet number
- RTP loss packet number
- RTP loss rate
- Video data rate (Bytes / s)
- Frame duration - to check if the time before send from the server's worker is as expected
- Frame per second - to check how many frames are delivered per second
- Remaining time - to display the time left for the video streaming
- Total time - to display the total time for the video streaming
- Video SRC - to display the source playback name that are streaming

The three last pieces of information are in **Extension 4** but it will be better looking when we put it all inside the statistic label (Their implementation details will be in **Extension 4** section).

Implementation:

- We setup the display text at the time we receive the rtp packet from the server (only when PLAYING).
- To retrieve the **packet number**, we decode the rtp packet and get from it the seqnum (which is set to the frameNbr in ServerWorker).
- To retrieve the **loss packet number**, we look at the seqnum and check if it is consecutive to the last frame number or not (if not then some frames are missing).
- To calculate the **loss rate**, we take the percentage of the loss frame over the current frame.

- To get the **data rate**, we take the frame's byte over the duration of the current frame and the last frame received.
- To get the **frame duration**, we take the difference in time between current frames and the last frame received from the server in milliseconds.
- To get the **FPS**, we take 1 second over the duration.

Extension 2

Given that SETUP is mandatory in an RTSP interaction, how would you implement that in a media player? When does the client send the SETUP? Come up with a solution and implement it. Also, is it appropriate to send TEARDOWN when the user clicks on the STOP button?

- In this extension we remove the setup button on our GUI so we will proceed the setup process in the **play** function.
- To answer the question of when the client sends the **SETUP** request to the server. First, we only send the setup request to the server when the state of the streaming is **INIT** to setup the rtp connection for retrieving the data frame. Then we may proceed with the stream playing.
- **TEARDOWN** request is needed when the user presses the stop button. Because when the users want to stop the stream, they want to terminate the RTP connection with the server. And when they start the play button it will start setting up and play the playback again.

Extension 3

Implement the method **describe** which is used to pass information about the media stream.

In this extension, we will send the **DESCRIBE** request to the server and we have shown all the information of the stream in another pop up window.

The information including:

- Some information of the session streaming
- **The client's IP (o)**
- **The session number (s)**
- **The file name (m)**

- **File's size (m)**
- **The request status (200 OK)**

Implementation:

To make this extension, we have to modify the following files: Client.py and ServerWorker.py

- In Client.py, we make another function called **describe** to send the request DESCRIBE to the server and display the response details as the pop up window. Noticeably that, we can only have a pop up window when the state of the streaming session is **PLAYING** and **READY**.
- In ServerWorker.py, we track another request type in **processRtspRequest** function named DESCRIBE and then we make the reply with the session's information described above and send back as an encoded reply to the client.

Extension 4

Implement some additional functions for user interface such as: display video total time and remaining time, fast forward or backward video (or make a scroll bar for scrolling video if you can).

As we mentioned above the video's total time and remaining time is displayed in the right label of the GUI.

To make the fast forward and backward (known as the scrolling) we make another UI element which is the scroll bar in the bottom of the video display UI, and we will describe more details about how to use this scrollbar in the **user's manual** section.

Implementation:

- To calculate the **total time**, we multiply the total frame number with 50ms (which is the default delay between two frames packets sent from the server).
- To calculate the **remaining time**, we take the total time minus the current time, which is calculated by using the currentFrame number with the same method of calculating the total time, then we change the format to seconds (s).

- To implement the Scrolling mechanism we make 2 new functions in Client.py (**jumpTo** and **pauseBeforeJump**) and 2 new functions in VideoStream.py (**buildIndex** and **jumpTo**) and add a new request type in ServerWorker.py (**JUMP**).

In **Client.py**

- The **jumpTo** function is used to request the server about the wanted frame to jump to and wait till the server response to start playing at the wanted frame taken from the scrollbar. This function is called when the Client releases the left mouse button the scrolling area.

- The **pauseBeforeJump** function is used to pause the streaming before start jumping. This is used to prevent deadlock from happening in the server. This function is called when the Client starts pressing down the left mouse button in the scroll area.

In **VideoStream.py**

- The **buildIndex** function is used to store all the indexes of all the frames in the file in order to improve the jumping speed. This function is called in the class's constructor to setup the indexTable array at the first time it is initialized.

- The **jumpTo** function is used to jump to the corresponding frame index in the index table, it will setup the frame number and start the streaming at the requested frame index.

In **ServerWorker.py**

- Request **JUMP** is used to call the **jumpTo** function in the VideoStream and start jumping to the wanted frame index requested from the client. Then it sends the reply message to the Client.

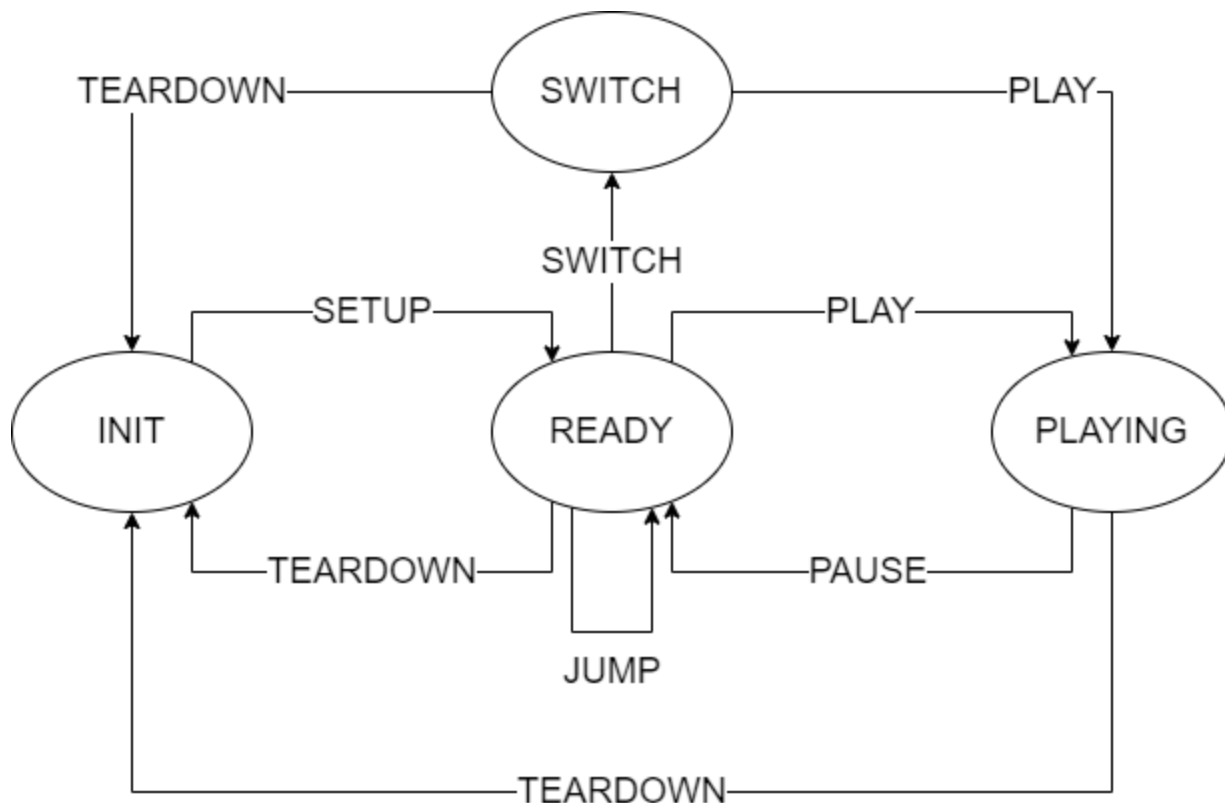
Extension 5

Add one more state to the client (for example **SWITCH** state) so that the user can select another video from a list of videos received from the server.

This is the new state diagram for this extension:

This State Diagram shows the **State** of playback.

The **Changing State** is the message from the Client to the Server.



Implementation:

In **Client.py**

- The **requestList** function is used to request the list of videos from the server, then setups the video list in the 'menu' video list. We need to pause the stream before requesting videos to avoid deadlock. So we only process the request when the state is **READY** and the dropdownList is not currently active.

- The **switchChannel** function is used to setup the new filename. If the file is the same as the old file then we continue playing the video. Else we need to teardown the connection and reset the stream.

In **ServerWorker.py**

- Request **SWITCH** is used to take all the files with the suffix '.Mjpeg' then sends a file's names list back to the client. If the filename is not corrected then the **FILE_NOT_FOUND_404** error is sent.

Bonus Features

This feature is used to change the speed of the playback.

In the GUI, it is implemented the same way as the option menu with the following speeds: **x0.1, x0.25, x0.5, x1, x1.25, x1.5, x1.75, x2, x2.5, x3.**

Implementation:

In **Client.py**:

- The **changeSpeed** function is used to send the requested speed to the server. The speed value is a multiplier with respect to the default playback speed of **20 frames/s** (meaning a 50ms delay between two frames).

In **ServerWorker.py**

- Request **CHANGESPEED** is used to change the frame delay requested by the client. This will change the framerate of the **sendRtp** function.

V. Summative Results' Evaluation

- We are able to make the stream **working smoothly** through all the steps below, and create a formal UI for users to feel the best when using our application.

- We are able to **connect multiple Clients to the Server** and the session code didn't mess up. Moreover, the Server can **handle multiple requests** from different clients **concurrently**.

- We are able to **improve the performance** of the stream by not relying on the **wait()** function but trying to measure the actual elapsed time between current and last time frame, which will contain all the executions of the rtp sending thread's code block.

- With **Extension 1**, we have successfully made all the functions and the GUI as user friendly as expected, so that we can display a lot of information such as packet's rate, video's data rate, video size, ... We also display the information of the video remaining time and total time (**Extension 4**) in the same label.

- With **Extension 2**, we have successfully made the setup and play into one button and we can answer all the questions in this extension. With this feature it is more and more similar to the media player application.

- With **Extension 3**, we have successfully sent the **DESCRIBE** request to the server and received the response with many information about the client's IP address, server's port, filename, etc.

- With **Extension 4**, this extension is quite complicated, we have to modify not only **Client.py**, **ServerWorker.py** but also the **VideoStream.py** to easily jump to the frame the client wants to play. Moreover, we also store all the frame indexes (ie. their starting location on the Mjpeg file) in order for better efficiency in jumping.

In other words, we make the one-directional-sequentially-accessed Mjpeg class capable of performing random access, which helps a lot with jumping forward and is crucial for jumping backward.

- With **Extension 5**, there are a lot of works we have to do in order to accomplish all the requirements. Firstly, we have to add another state and also change the state diagram with the class diagram. We have successfully modified the **SWITCH** state and requested the server. Moreover, the client can choose the playback from the playlist to request to the server to play the playback.

- We also created another class for converting any video format that used jpeg as a frame into our special **5 bytes 'string of digits'** header Mjpeg format used in this assignment. This tool is used to make more videos for the user to choose from in **extend 5**. This file is called **ConvertTo5ByteMjpeg.py**

- Moreover, we created a list of speeds for clients to request the **CHANGESPEED** request to the server with the given delay time so that the Server can change the 'frame delay'.

Summary of results achieved:

We can confidently say that we complete *all the requirements, including a **customized UI** and all of the **extended** sections* in the assignment specification, and additionally implement *some bonus features*.

VI. User Manual

Setup

To play the video streaming application first:

Step 1: Start the Server with the following command in command-line :

```
python Server.py <server-port>
```

server-port : a port number **greater than 1024**. This is to avoid port collisions with already defined ports from other applications.

Step 2: Start the client by using another command-line on the computer or the different computer:

```
python ClientLauncher.py <server-addr> <server-port>  
<rtp-port> <file-name>
```

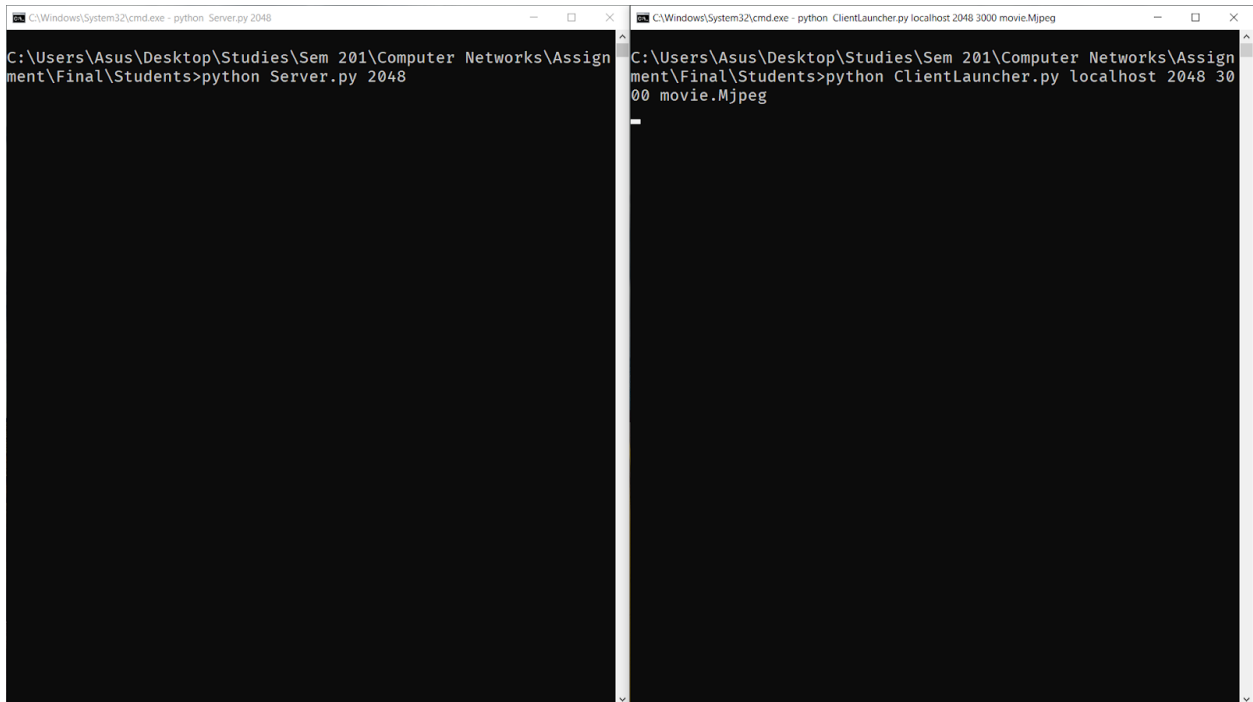
server-addr : localhost if on the same computer ,or IP-addr of the other computer.

server-port : the same as the server-port from the Server.

rtp-port : choose a random client's rtp-port number (that MUST be different from the server port) greater than **1024**.

file-name : the playback file name (and path).

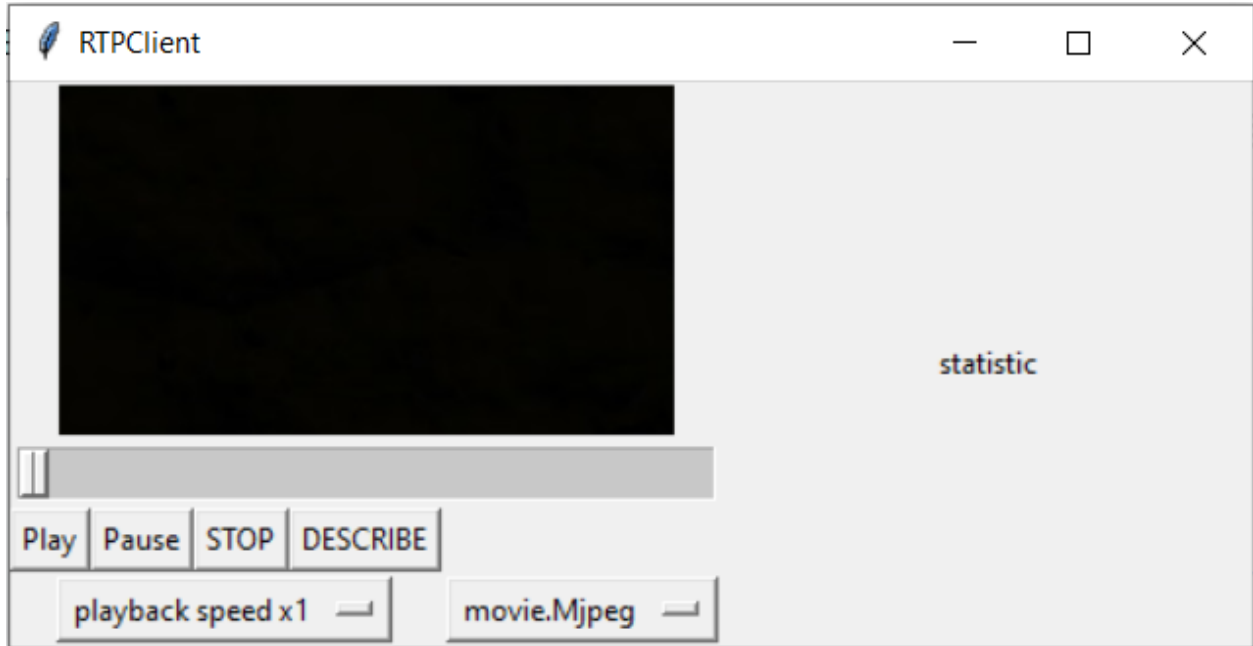
After completing 2 upon steps you should have the following result.



The image shows two terminal windows side-by-side. The left window has a title bar that reads 'C:\Windows\System32\cmd.exe - python Server.py 2048'. The command prompt shows the directory 'C:\Users\Asus\Desktop\Studies\Sem 201\Computer Networks\Assignment\Final\Students' and the command 'python Server.py 2048'. The right window has a title bar that reads 'C:\Windows\System32\cmd.exe - python ClientLauncher.py localhost 2048 3000 movie.Mjpeg'. The command prompt shows the same directory and the command 'python ClientLauncher.py localhost 2048 3000 movie.Mjpeg'.

The terminals from left to right is Server and Client, respectively

The left-hand-side of the application is the viewer and buttons side, the right-hand-side is the statistic of the playback.



GUI of our application

Core Functions

Step 3: To play the playback you should press the **'Play'** button on the bottom left, and the playback will automatically setup and play.

```

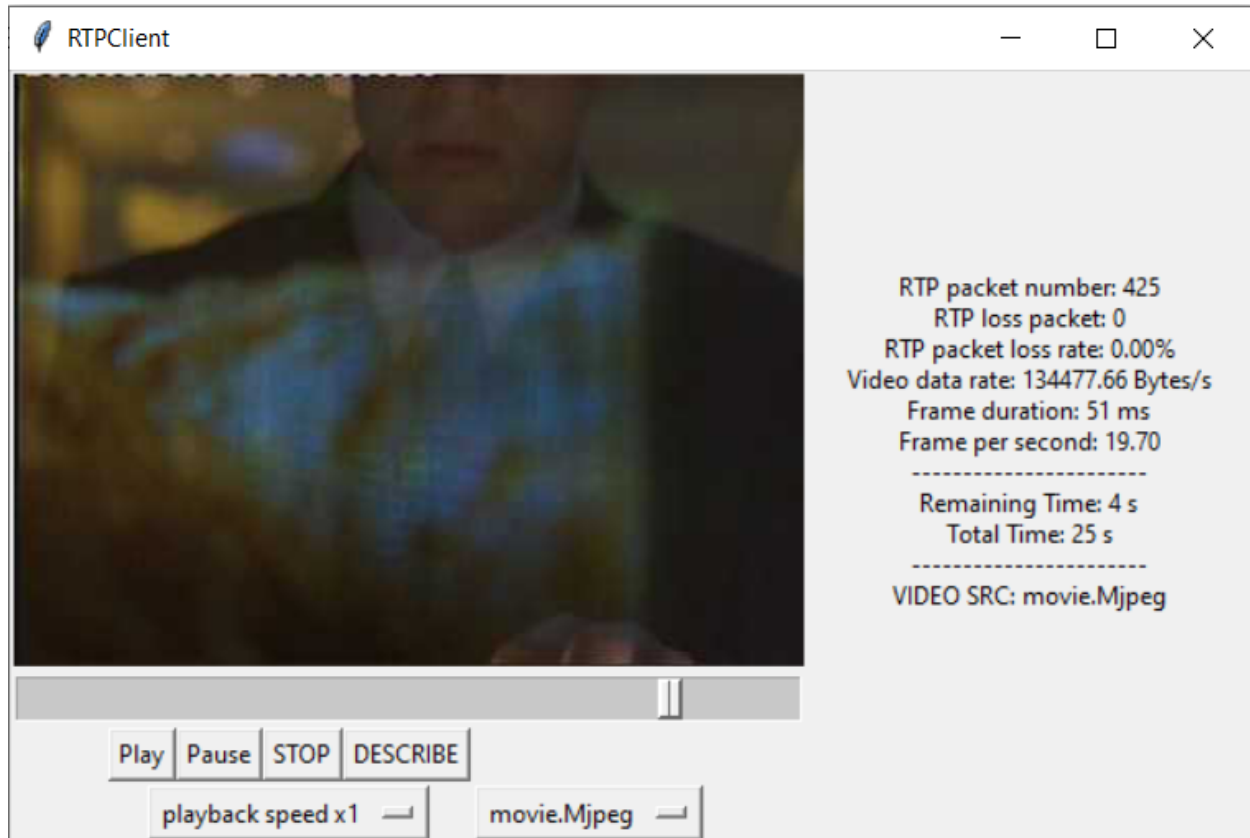
C:\Windows\System32\cmd.exe - python Server.py 2048
C:\Users\Asus\Desktop\Studies\Sem 201\Computer Networks\Assign
ment\Final\Students>python Server.py 2048
Data received:
SETUP movie.Mjpeg RTPS/1.0
CSeq: 1
Transprot: RTP/UDP; client_port= 52669
processing SETUP

Data received:
PLAY movie.Mjpeg RTPS/1.0
CSeq: 2
Session: 736635
processing PLAY

C:\Windows\System32\cmd.exe - python ClientLauncher.py localhost 2048 3000 movie.Mjpeg
C:\Users\Asus\Desktop\Studies\Sem 201\Computer Networks\Assign
ment\Final\Students>python ClientLauncher.py localhost 2048 30
00 movie.Mjpeg
Reply From: localhost:52668
RTSP/1.0 200 OK
CSeq: 1
Session: 736635
TotalFrame 501
-----
Reply From: localhost:52668
RTSP/1.0 200 OK
CSeq: 2
Session: 736635
-----

```

The terminals from left to right is Server and Client, respectively



GUI of our application

After that, you should see the status on the Client command-line showing if the rtsp response is **200 OK** then your command is receivable from the server. Moreover, on the Server side you should see the message showing that it is playing (**PLAY file-name protocol-type**).

The **right hand side** of the GUI is showing all the statistics of the playback and connection with the server.

Step 4: To **pause** the playback you can press the '**Pause**' button next to the '**Play**' button. To continue playing, just press the '**Play**' button again.

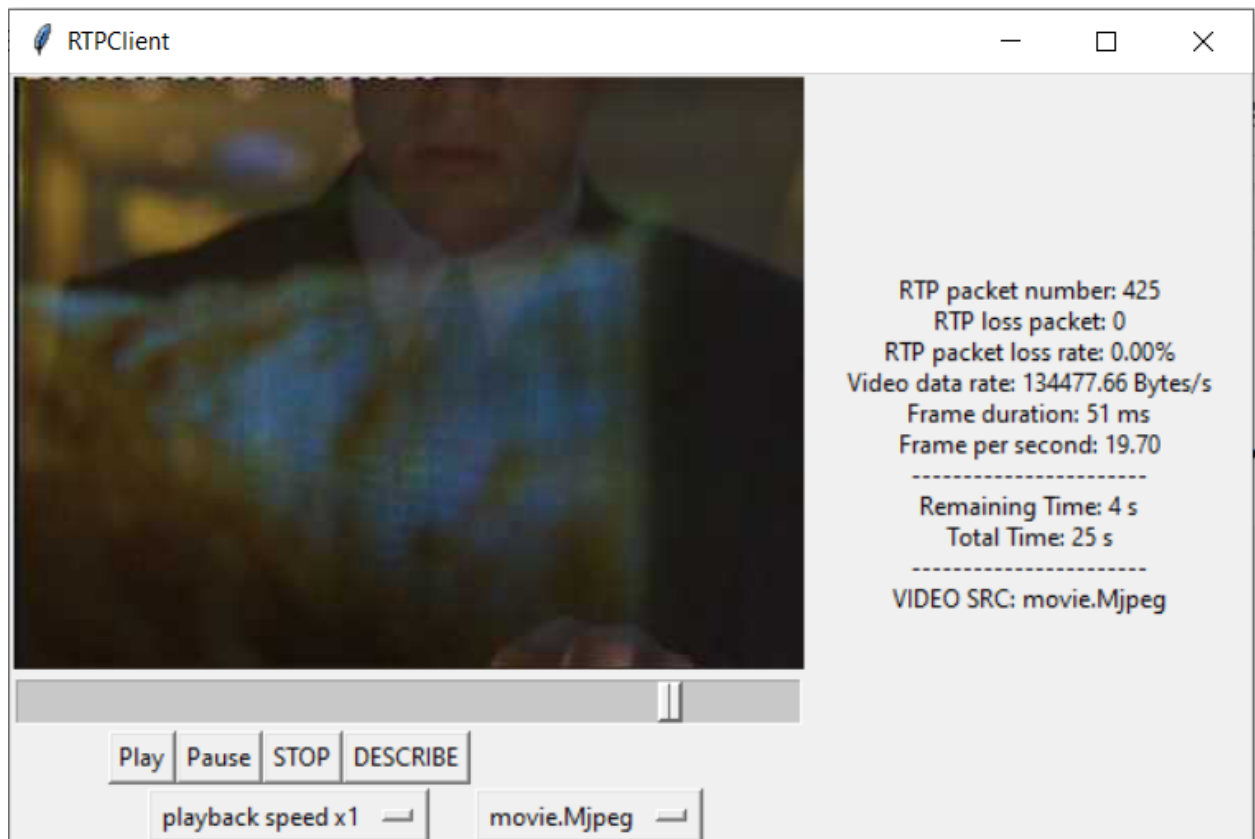
```
C:\Windows\System32\cmd.exe - python Server.py 2048
C:\Users\Asus\Desktop\Studies\Sem 201\Computer Networks\Assign
ment\Final\Students>python Server.py 2048
Data received:
SETUP movie.Mjpeg RTPS/1.0
CSeq: 1
Transprot: RTP/UDP; client_port= 52669
processing SETUP

Data received:
PLAY movie.Mjpeg RTPS/1.0
CSeq: 2
Session: 736635
processing PLAY

Data received:
PAUSE movie.Mjpeg RTPS/1.0
CSeq: 3
Session: 736635
processing PAUSE

C:\Windows\System32\cmd.exe - python ClientLauncher.py localhost 2048 3000 movie.Mjpeg
C:\Users\Asus\Desktop\Studies\Sem 201\Computer Networks\Assign
ment\Final\Students>python ClientLauncher.py localhost 2048 30
00 movie.Mjpeg
Reply From: localhost:52668
RTSP/1.0 200 OK
CSeq: 1
Session: 736635
TotalFrame 501
-----
Reply From: localhost:52668
RTSP/1.0 200 OK
CSeq: 2
Session: 736635
-----
Reply From: localhost:52668
RTSP/1.0 200 OK
CSeq: 3
Session: 736635
-----
```

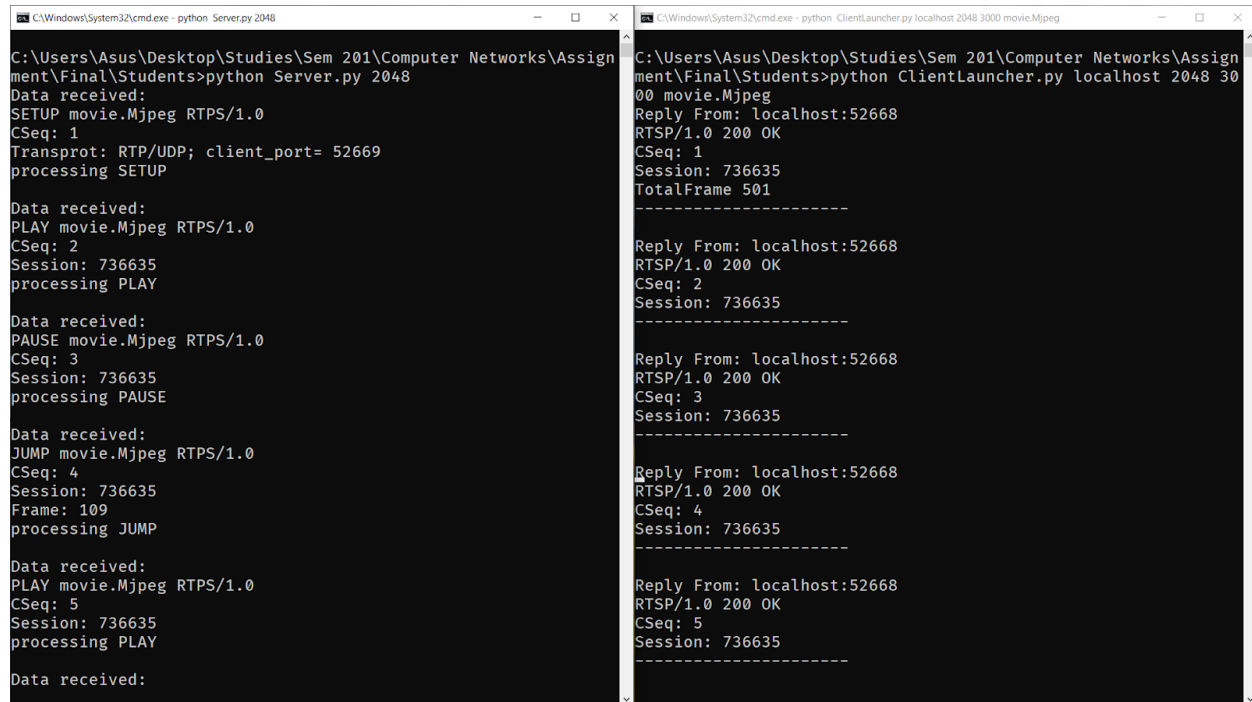
The terminals from left to right is Server and Client, respectively



GUI of our application

Extended Functions

Step 5: To **jump** to the expected frame you must **hold and move the scroll bar** below the video's screen. This will stop the stream and play back at the frame you release the scroll bar.



```
C:\Windows\System32\cmd.exe - python Server.py 2048
C:\Users\Asus\Desktop\Studies\Sem 201\Computer Networks\Assignment\Final\Students>python Server.py 2048
Data received:
SETUP movie.Mjpeg RTPS/1.0
CSeq: 1
Transprot: RTP/UDP; client_port= 52669
processing SETUP

Data received:
PLAY movie.Mjpeg RTPS/1.0
CSeq: 2
Session: 736635
processing PLAY

Data received:
PAUSE movie.Mjpeg RTPS/1.0
CSeq: 3
Session: 736635
processing PAUSE

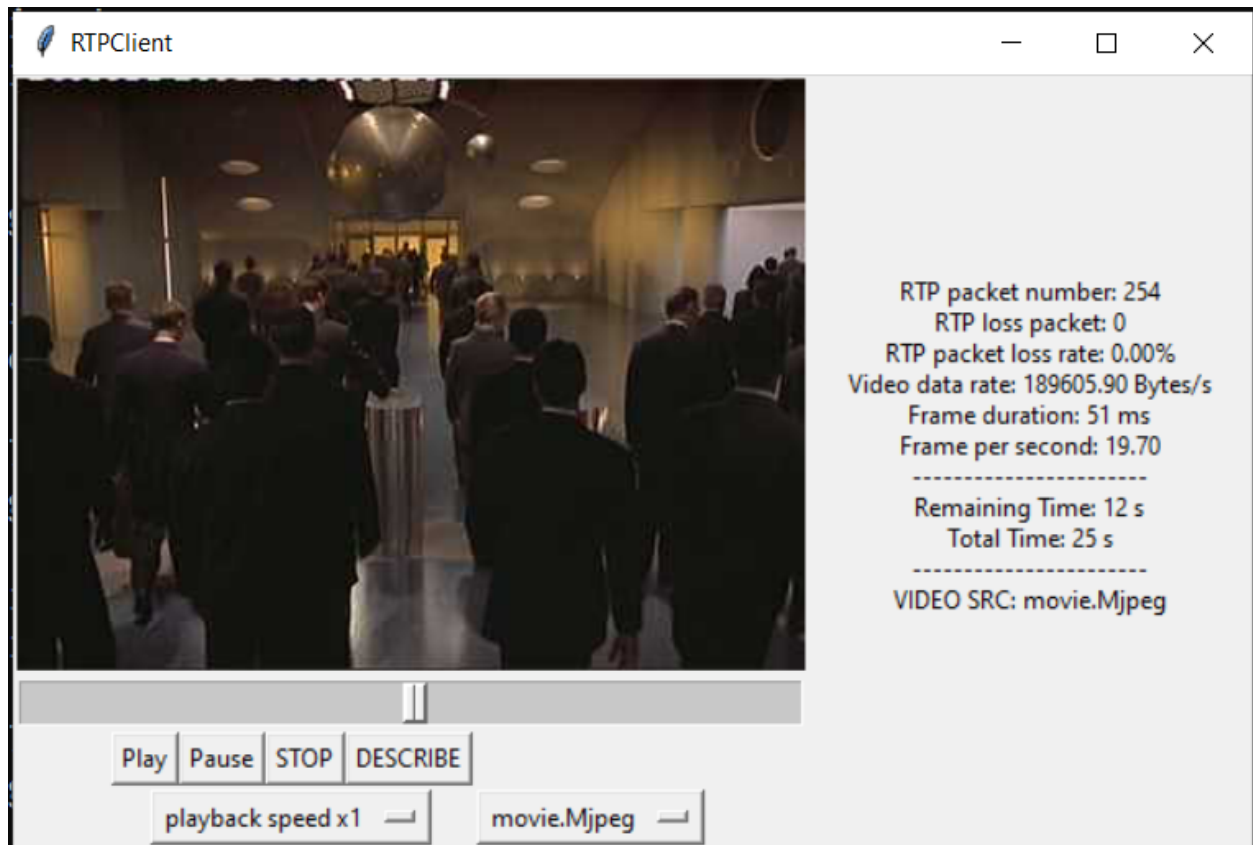
Data received:
JUMP movie.Mjpeg RTPS/1.0
CSeq: 4
Session: 736635
Frame: 109
processing JUMP

Data received:
PLAY movie.Mjpeg RTPS/1.0
CSeq: 5
Session: 736635
processing PLAY

Data received:
```

```
C:\Windows\System32\cmd.exe - python ClientLauncher.py localhost 2048 3000 movie.Mjpeg
C:\Users\Asus\Desktop\Studies\Sem 201\Computer Networks\Assignment\Final\Students>python ClientLauncher.py localhost 2048 3000 movie.Mjpeg
00 movie.Mjpeg
Reply From: localhost:52668
RTSP/1.0 200 OK
CSeq: 1
Session: 736635
TotalFrame 501
-----
Reply From: localhost:52668
RTSP/1.0 200 OK
CSeq: 2
Session: 736635
-----
Reply From: localhost:52668
RTSP/1.0 200 OK
CSeq: 3
Session: 736635
-----
Reply From: localhost:52668
RTSP/1.0 200 OK
CSeq: 4
Session: 736635
-----
Reply From: localhost:52668
RTSP/1.0 200 OK
CSeq: 5
Session: 736635
-----
```

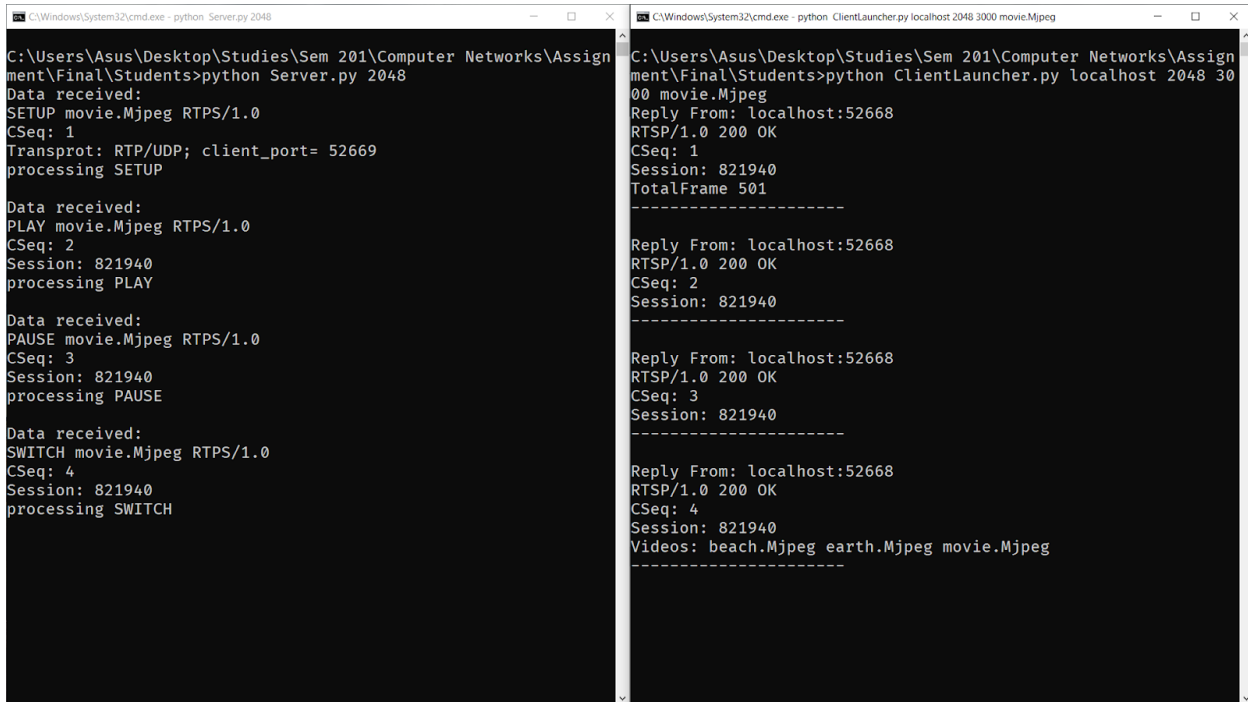
The process of holding and releasing the scroll bar



GUI of our application

Step 6: To **switch** to the other video streaming you can click on the **option menu** and choose the file you want to start streaming.

- When you press the option menu the client will retrieve the list of videos requested from the server.



The image shows two terminal windows side-by-side. The left window is titled 'C:\Windows\System32\cmd.exe - python Server.py 2048' and shows the server's output. It receives three commands: 'SETUP movie.Mjpeg RTPS/1.0', 'PLAY movie.Mjpeg RTPS/1.0', and 'PAUSE movie.Mjpeg RTPS/1.0'. The right window is titled 'C:\Windows\System32\cmd.exe - python ClientLauncher.py localhost 2048 3000 movie.Mjpeg' and shows the client's output. It sends three commands: '00 movie.Mjpeg', 'Reply From: localhost:52668', and 'RTSP/1.0 200 OK'. The client also receives a 'TotalFrame 501' response from the server.

```
C:\Windows\System32\cmd.exe - python Server.py 2048
C:\Users\Asus\Desktop\Studies\Sem 201\Computer Networks\Assign
ment\Final\Students>python Server.py 2048
Data received:
SETUP movie.Mjpeg RTPS/1.0
CSeq: 1
Transprot: RTP/UDP; client_port= 52669
processing SETUP

Data received:
PLAY movie.Mjpeg RTPS/1.0
CSeq: 2
Session: 821940
processing PLAY

Data received:
PAUSE movie.Mjpeg RTPS/1.0
CSeq: 3
Session: 821940
processing PAUSE

Data received:
SWITCH movie.Mjpeg RTPS/1.0
CSeq: 4
Session: 821940
processing SWITCH

C:\Windows\System32\cmd.exe - python ClientLauncher.py localhost 2048 3000 movie.Mjpeg
C:\Users\Asus\Desktop\Studies\Sem 201\Computer Networks\Assign
ment\Final\Students>python ClientLauncher.py localhost 2048 30
00 movie.Mjpeg
Reply From: localhost:52668
RTSP/1.0 200 OK
CSeq: 1
Session: 821940
TotalFrame 501
-----

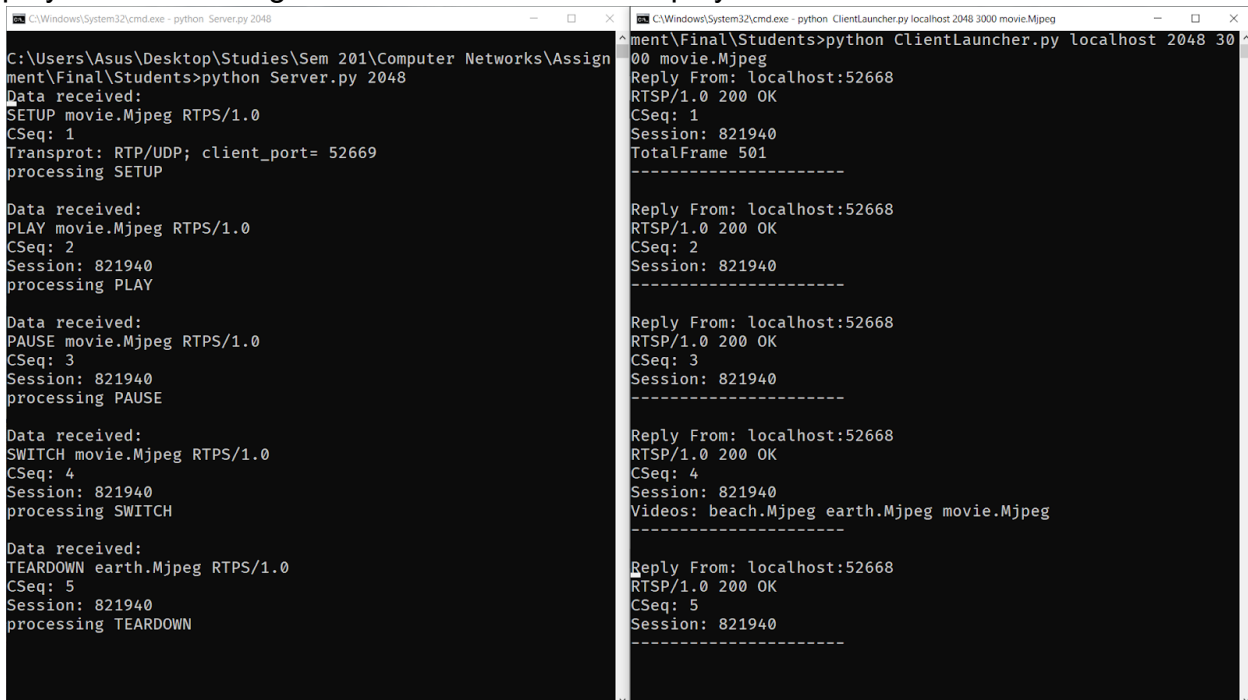
Reply From: localhost:52668
RTSP/1.0 200 OK
CSeq: 2
Session: 821940
-----

Reply From: localhost:52668
RTSP/1.0 200 OK
CSeq: 3
Session: 821940
-----

Reply From: localhost:52668
RTSP/1.0 200 OK
CSeq: 4
Session: 821940
Videos: beach.Mjpeg earth.Mjpeg movie.Mjpeg
-----
```

Retrieving list of videos

- When you choose a video from the list it will teardown the current stream of the current playback and change the filename to the latest playback.



The image shows two terminal windows side-by-side. The left window is titled 'C:\Windows\System32\cmd.exe - python Server.py 2048' and shows the server's output. It receives three commands: 'SETUP movie.Mjpeg RTPS/1.0', 'PLAY movie.Mjpeg RTPS/1.0', and 'PAUSE movie.Mjpeg RTPS/1.0'. The right window is titled 'C:\Windows\System32\cmd.exe - python ClientLauncher.py localhost 2048 3000 movie.Mjpeg' and shows the client's output. It sends three commands: '00 movie.Mjpeg', 'Reply From: localhost:52668', and 'RTSP/1.0 200 OK'. The client also receives a 'TotalFrame 501' response from the server. The left window also shows a 'TEARDOWN earth.Mjpeg RTPS/1.0' command being received by the server.

```
C:\Windows\System32\cmd.exe - python Server.py 2048
C:\Users\Asus\Desktop\Studies\Sem 201\Computer Networks\Assign
ment\Final\Students>python Server.py 2048
Data received:
SETUP movie.Mjpeg RTPS/1.0
CSeq: 1
Transprot: RTP/UDP; client_port= 52669
processing SETUP

Data received:
PLAY movie.Mjpeg RTPS/1.0
CSeq: 2
Session: 821940
processing PLAY

Data received:
PAUSE movie.Mjpeg RTPS/1.0
CSeq: 3
Session: 821940
processing PAUSE

Data received:
SWITCH movie.Mjpeg RTPS/1.0
CSeq: 4
Session: 821940
processing SWITCH

Data received:
TEARDOWN earth.Mjpeg RTPS/1.0
CSeq: 5
Session: 821940
processing TEARDOWN

C:\Windows\System32\cmd.exe - python ClientLauncher.py localhost 2048 3000 movie.Mjpeg
C:\Users\Asus\Desktop\Studies\Sem 201\Computer Networks\Assign
ment\Final\Students>python ClientLauncher.py localhost 2048 30
00 movie.Mjpeg
Reply From: localhost:52668
RTSP/1.0 200 OK
CSeq: 1
Session: 821940
TotalFrame 501
-----

Reply From: localhost:52668
RTSP/1.0 200 OK
CSeq: 2
Session: 821940
-----

Reply From: localhost:52668
RTSP/1.0 200 OK
CSeq: 3
Session: 821940
-----

Reply From: localhost:52668
RTSP/1.0 200 OK
CSeq: 4
Session: 821940
Videos: beach.Mjpeg earth.Mjpeg movie.Mjpeg
-----

Reply From: localhost:52668
RTSP/1.0 200 OK
CSeq: 5
Session: 821940
-----
```

Change the playback file

- When you hit the Play button it will setup the new video playback and start the stream again with the new playback file.

```

C:\Windows\System32\cmd.exe - python Server.py 2048
CSeq: 3
Session: 821940
processing PAUSE

Data received:
SWITCH movie.Mjpeg RTPS/1.0
CSeq: 4
Session: 821940
processing SWITCH

Data received:
TEARDOWN earth.Mjpeg RTPS/1.0
CSeq: 5
Session: 821940
processing TEARDOWN

Data received:
SETUP earth.Mjpeg RTPS/1.0
CSeq: 6
Transprot: RTP/UDP; client_port= 52669
processing SETUP

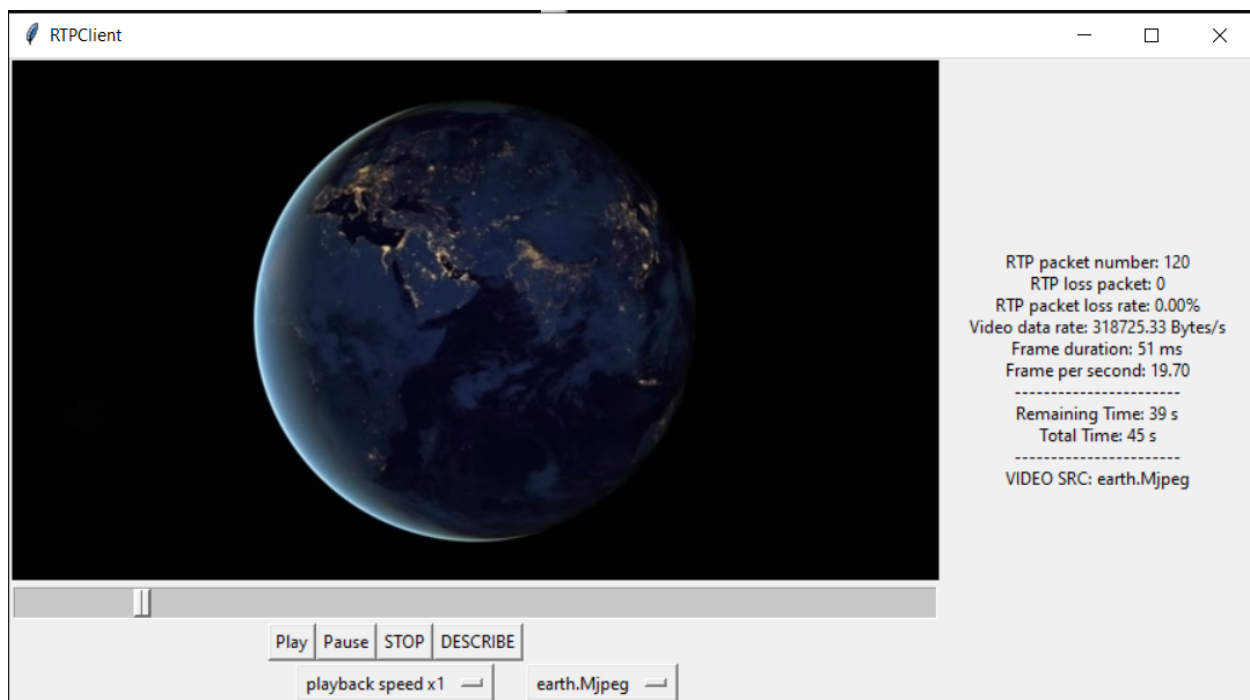
Data received:
PLAY earth.Mjpeg RTPS/1.0
CSeq: 7
Session: 655460
processing PLAY

Data received:
PAUSE earth.Mjpeg RTPS/1.0
CSeq: 8
Session: 655460
processing PAUSE

C:\Windows\System32\cmd.exe - python ClientLauncher.py localhost 2048 3000 movie.Mjpeg
-----
Reply From: localhost:52668
RTSP/1.0 200 OK
CSeq: 4
Session: 821940
Videos: beach.Mjpeg earth.Mjpeg movie.Mjpeg
-----
Reply From: localhost:52668
RTSP/1.0 200 OK
CSeq: 5
Session: 821940
-----
Reply From: localhost:52668
RTSP/1.0 200 OK
CSeq: 6
Session: 655460
TotalFrame 902
-----
Reply From: localhost:52668
RTSP/1.0 200 OK
CSeq: 7
Session: 655460
-----
Reply From: localhost:52668
RTSP/1.0 200 OK
CSeq: 8
Session: 655460
-----

```

Start playing the stream playback



New GUI after switching the stream

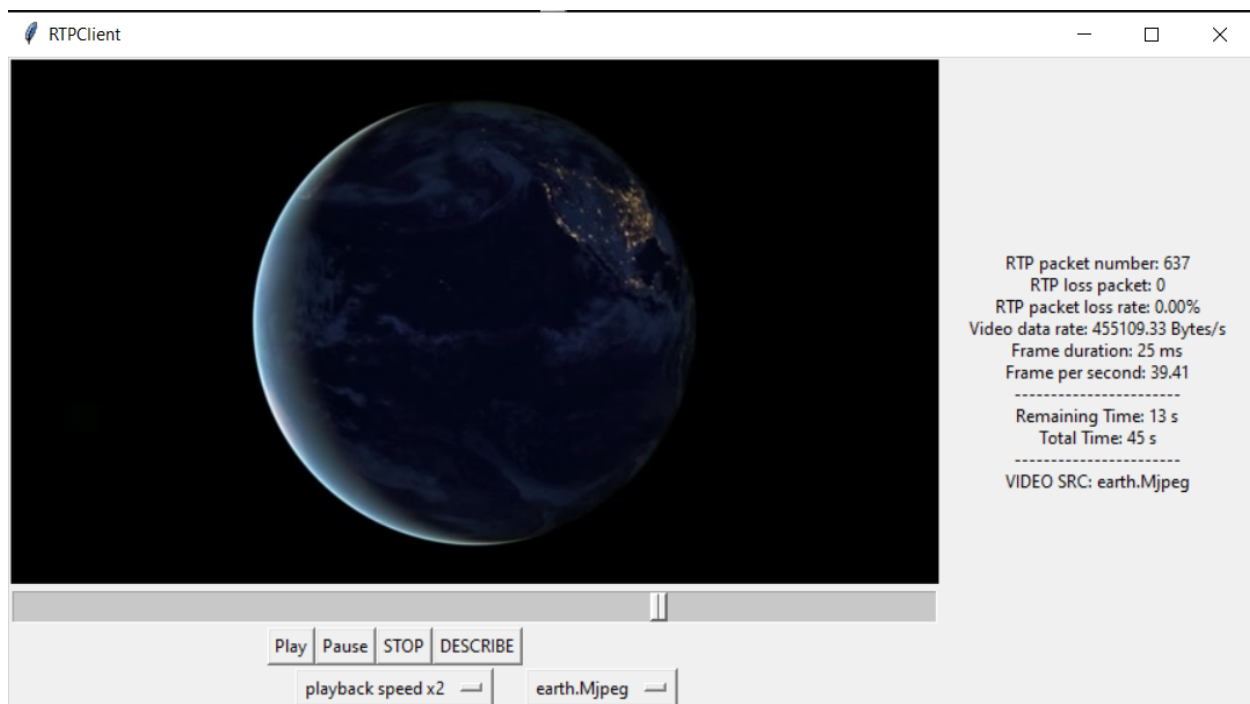
Step 7: To **Change the speed** of the playback you can choose the playback speed option menu and choose the speed you want to change to.

- Now try to change the speed to **x2** option and you will achieve the message below.

```
C:\Windows\System32\cmd.exe - python Server.py 2048
Session: 821940
processing SWITCH
Data received:
TEARDOWN earth.Mjpeg RTPS/1.0
CSeq: 5
Session: 821940
processing TEARDOWN
Data received:
SETUP earth.Mjpeg RTPS/1.0
CSeq: 6
Transprot: RTP/UDP; client_port= 52669
processing SETUP
Data received:
PLAY earth.Mjpeg RTPS/1.0
CSeq: 7
Session: 655460
processing PLAY
Data received:
PAUSE earth.Mjpeg RTPS/1.0
CSeq: 8
Session: 655460
processing PAUSE
Data received:
CHANGESPEED earth.Mjpeg RTPS/1.0
CSeq: 9
Session: 655460
Delay: 0.025
processing CHANGESPEED

C:\Windows\System32\cmd.exe - python ClientLauncher.py localhost 2048 3000 movie.Mjpeg
Videos: beach.Mjpeg earth.Mjpeg movie.Mjpeg
-----
Reply From: localhost:52668
RTSP/1.0 200 OK
CSeq: 5
Session: 821940
-----
Reply From: localhost:52668
RTSP/1.0 200 OK
CSeq: 6
Session: 655460
TotalFrame 902
-----
Reply From: localhost:52668
RTSP/1.0 200 OK
CSeq: 7
Session: 655460
-----
Reply From: localhost:52668
RTSP/1.0 200 OK
CSeq: 8
Session: 655460
-----
Reply From: localhost:52668
RTSP/1.0 200 OK
CSeq: 9
Session: 655460
-----
```

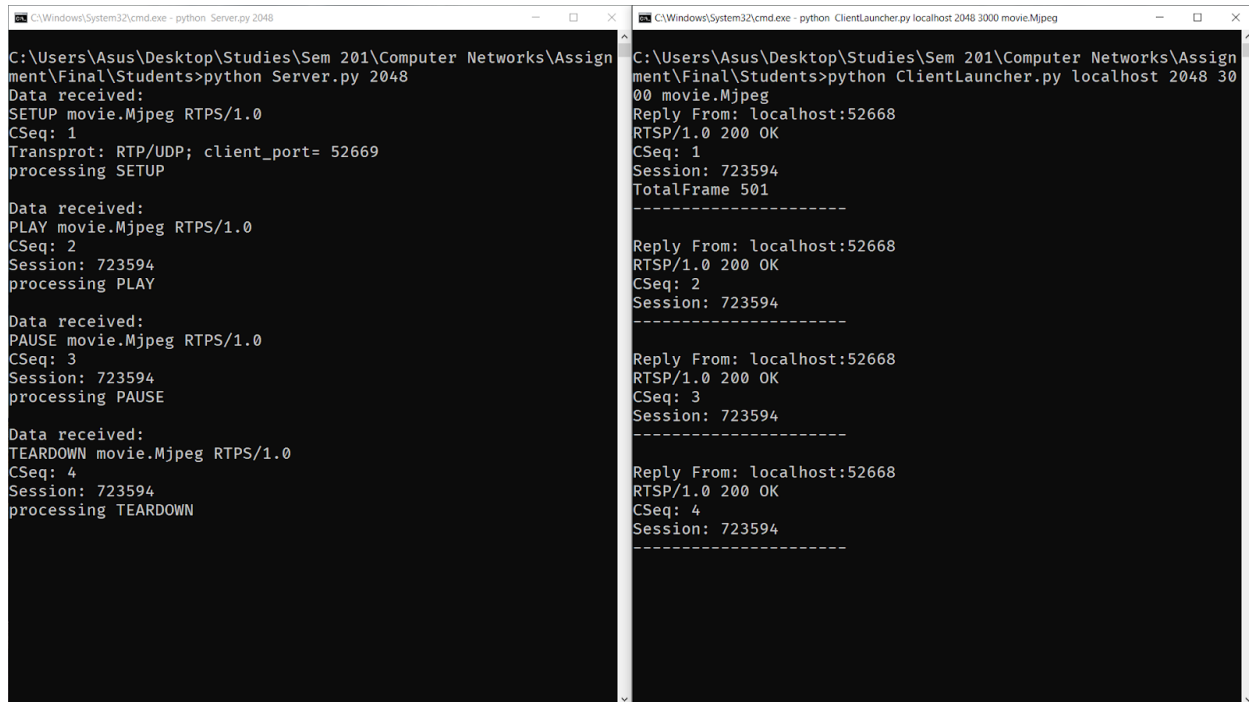
Change the speed to x2



The video stream is doubled the speed

- Notice the difference in the statistic of the playback you can see the differences in **frame duration, FPS**.

Step 8: To terminate the streaming session you can press the ‘**STOP**’ button, if you want to start the session and start the stream over again then go back to **Step 3**.



```

C:\Windows\System32\cmd.exe - python Server.py 2048
C:\Users\Asus\Desktop\Studies\Sem 201\Computer Networks\Assignment\Final\Students>python Server.py 2048
Data received:
SETUP movie.Mjpeg RTPS/1.0
CSeq: 1
Transprot: RTP/UDP; client_port= 52669
processing SETUP

Data received:
PLAY movie.Mjpeg RTPS/1.0
CSeq: 2
Session: 723594
processing PLAY

Data received:
PAUSE movie.Mjpeg RTPS/1.0
CSeq: 3
Session: 723594
processing PAUSE

Data received:
TEARDOWN movie.Mjpeg RTPS/1.0
CSeq: 4
Session: 723594
processing TEARDOWN

C:\Windows\System32\cmd.exe - python ClientLauncher.py localhost 2048 3000 movie.Mjpeg
C:\Users\Asus\Desktop\Studies\Sem 201\Computer Networks\Assignment\Final\Students>python ClientLauncher.py localhost 2048 3000 movie.Mjpeg
00 movie.Mjpeg
Reply From: localhost:52668
RTSP/1.0 200 OK
CSeq: 1
Session: 723594
TotalFrame 501
-----
Reply From: localhost:52668
RTSP/1.0 200 OK
CSeq: 2
Session: 723594
-----
Reply From: localhost:52668
RTSP/1.0 200 OK
CSeq: 3
Session: 723594
-----
Reply From: localhost:52668
RTSP/1.0 200 OK
CSeq: 4
Session: 723594
-----

```

The terminals from left to right is Server and Client, respectively



GUI of our application

Step 9: In case you need to see the information about the media stream, there is the last button named '**DESCRIBE**'. This will give you information about version number, client port, the session and the type of file.

```

C:\Windows\System32\cmd.exe - python Server.py 2048
Data received:
CHANGESPEED earth.Mjpeg RTPS/1.0
CSeq: 15
Session: 244838
Delay: 0.05
processing CHANGESPEED

Data received:
CHANGESPEED earth.Mjpeg RTPS/1.0
CSeq: 16
Session: 244838
Delay: 0.025
processing CHANGESPEED

Data received:
CHANGESPEED earth.Mjpeg RTPS/1.0
CSeq: 17
Session: 244838
Delay: 0.05
processing CHANGESPEED

Data received:
PAUSE earth.Mjpeg RTPS/1.0
CSeq: 18
Session: 244838
processing PAUSE

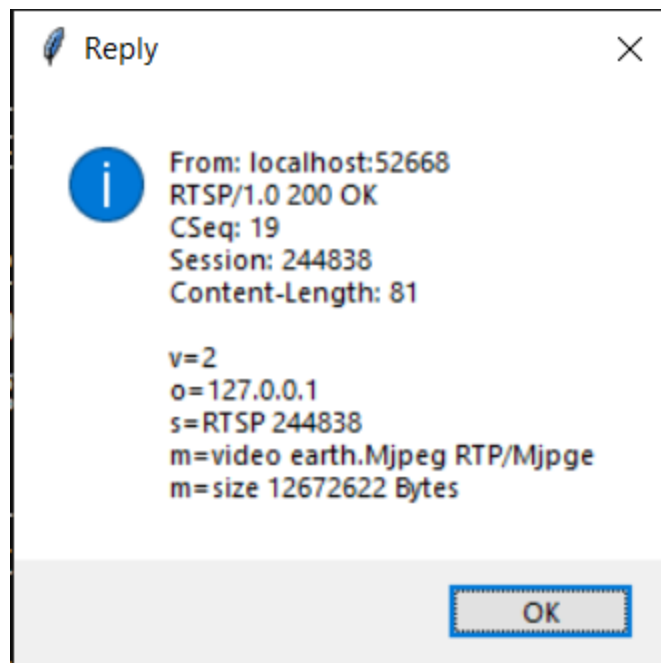
Data received:
DESCRIBE earth.Mjpeg RTPS/1.0
CSeq: 19
Session: 244838
processing DESCRIBE

C:\Windows\System32\cmd.exe - python ClientLauncher.py localhost 2048 3000 movie.Mjpeg
Session: 244838
-----
Reply From: localhost:52668
RTSP/1.0 200 OK
CSeq: 16
Session: 244838
-----
Reply From: localhost:52668
RTSP/1.0 200 OK
CSeq: 17
Session: 244838
-----
Reply From: localhost:52668
RTSP/1.0 200 OK
CSeq: 18
Session: 244838
-----
Reply From: localhost:52668
RTSP/1.0 200 OK
CSeq: 19
Session: 244838
Content-Length: 81

v=2
o=127.0.0.1
s=RTSP 244838
m=video earth.Mjpeg RTP/Mjpge
m=size 12672622 Bytes
-----

```

The message return when you press Describe



Pop up Window to describe the information of the stream