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

COMPUTER NETWORK

Assignment 1

Streaming Video Using RTSP

Advisors:	Nguyen Le Duy Lai	
	Nguyen Manh Thin	
Candidates:	Tran Hoang Long	– 1852545
	Pham Hoang Hai	– 1852020
	Tran Quoc Anh	– 1852247

Ho Chi Minh City, April 7, 2021

Contents

1 Introduction **2**

2 Requirement Analysis **2**
2.1 Server . . . 2
2.1.1 Functional Requirements . . . 2
2.1.2 Non-Functional Requirements . . . 3
2.2 Client . . . 3
2.2.1 Functional Requirement . . . 3
2.2.2 Non-Runctional Requirement . . . 3
2.3 MJPEG Format . . . 3

3 Implementation Details **5**
3.1 Client . . . 5
3.1.1 GUI . . . 5
3.1.2 RTSPClient . . . 6
3.1.3 RTPReceiver . . . 7
3.2 Server . . . 7
3.2.1 ServerWorker . . . 8
3.2.2 RTPSender . . . 8
3.2.3 VideoStream . . . 9

4 User Manual **9**
4.1 Start Server and Client . . . 9
4.2 GUI Usage . . . 9

5 Result Evaluation **10**

6 Extension **11**
6.1 Statistics . . . 11
6.2 User-Friendly GUI Interface . . . 12
6.3 DESCRIBE Method . . . 13
6.4 Additional Function . . . 13
6.4.1 Display Video Time . . . 13
6.4.2 Forward/Backward . . . 14
6.5 Switch . . . 14

1 Introduction

We will implement 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). This report will guide you through our requirement, work description and result.

2 Requirement Analysis

In this assignment, server and client communicate through RTSP and to send video frame from the server to the client, we will use RTP unicast.

2.1 Server

2.1.1 Functional Requirements

1. Server opens a socket to listen to all client's RTSP requests and responds accordingly:
   - Server processes RTSP request from client to get needed information such as request method, sequence number, ...
   - Server can read data from media source (MJPEG) and packetizes them into RTP packets and send to client after **PLAY** request is received.
   - RTP packet stream is stopped when client request **PAUSE**.
2. RTP packet sent to client follow header format as follows:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|V=2|P|X|  CC   |M|     PT      |       sequence number         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           timestamp                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           synchronization source (SSRC) identifier            |
+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
|            contributing source (CSRC) identifiers             |
|                             ....                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

   More details on the format can be found at RFC 3550.

   In our assignment, our fields are specified as follows:
   - Version field (V) is set to 2.
   - Padding (P), extension (X), CSRC count (CC), marker(M) fields are all set to 0.
   - Payload type (PT) field is 26 for our MJPEG file format (see RFC 1890).
   - The sequence number increments by one for each RTP data packet sent, and may be used by the receiver to detect packet loss and to restore packet sequence.
   - The timestamp reflects the sampling instant of the first octet in the RTP data packet. The sampling instant **MUST** be derived from a clock that increments monotonically and linearly in time to allow synchronization and jitter calculations.
   - The SSRC field identifies the synchronization source. For this assignment, we can pick any integer value we like.
   - The CSRC field does not exist, so our header length is 12 bytes.
3. Server must know if client lost connection.
4. Server logs its status and activities.

2.1.2 Non-Functional Requirements

1. Server must send JPEG frames through RTP packet with delay matching the video's frame rate (we choose 20 FPS).
2. The server creates a new thread for each connected client and terminates it after the client disconnect to keep the number of alive threads low.
3. Communication follows RTCP protocol and data sending follows RTP protocol.
4. Server is always online, listening to opened socket unless being shut down by administrator.

2.2 Client

2.2.1 Functional Requirement

1. Implement RTSP on the client side.
2. Client opens the RTSP socket to the server and uses that for sending all RTSP requests:

   SETUP Sets up the session and transports parameters.
   - Specify the transport mechanism to be used for the streamed media.
   - The Transport header specifies the transport parameters acceptable to the client for data transmission; the response will contain the transport parameters selected by the server.

   PLAY Starts the playback.
   Tell the server to start sending data via the mechanism specified in SETUP. A client must not issue a PLAY request until any outstanding SETUP requests have been acknowledged as successful.

   PAUSE Pause during the playback.
   Cause the stream delivery to be interrupted (pause) temporarily. After resuming playback, the tracks must be maintained and any server resources are kept.

   TEARDOWN Terminates the session and close the connection.
   Stop the stream delivery and freeing the resources associated with it. A SETUP request has to be issued before the session can be played again.
3. Insert the **CSeq** header which starts at 1 and increase the value by 1 for each request sent.

2.2.2 Non-Runctional Requirement

1. Client socket for receiving RTP times out in 0.5 second.
2. Client plays any frame sent to by server without waiting or delaying (no synchronization as of current implementation).
3. Client must ensure socket buffer size large enough to receive RTSP response from server and also RTP data packet from server.

2.3 MJPEG Format

In this assignment, we will encode video in MJPEG format. This format stores the video as concatenated JPEG-encoded images, with each image being preceded by a 5-bytes header storing the size of the image in big-endian binary format.

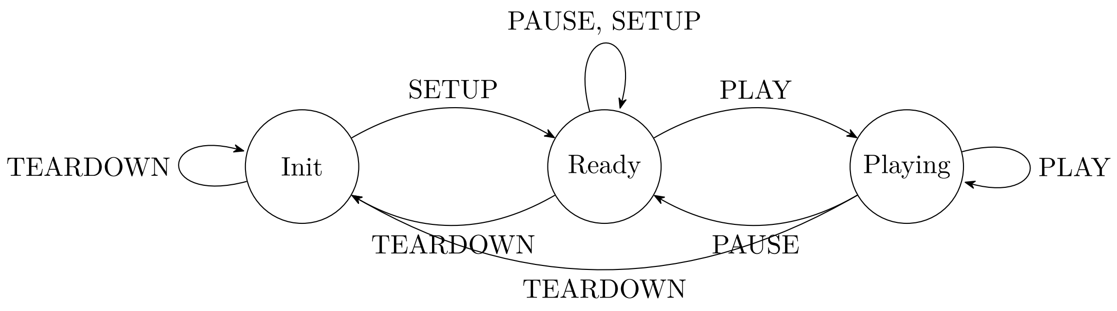


Figure 1: RTSP protocol state machine

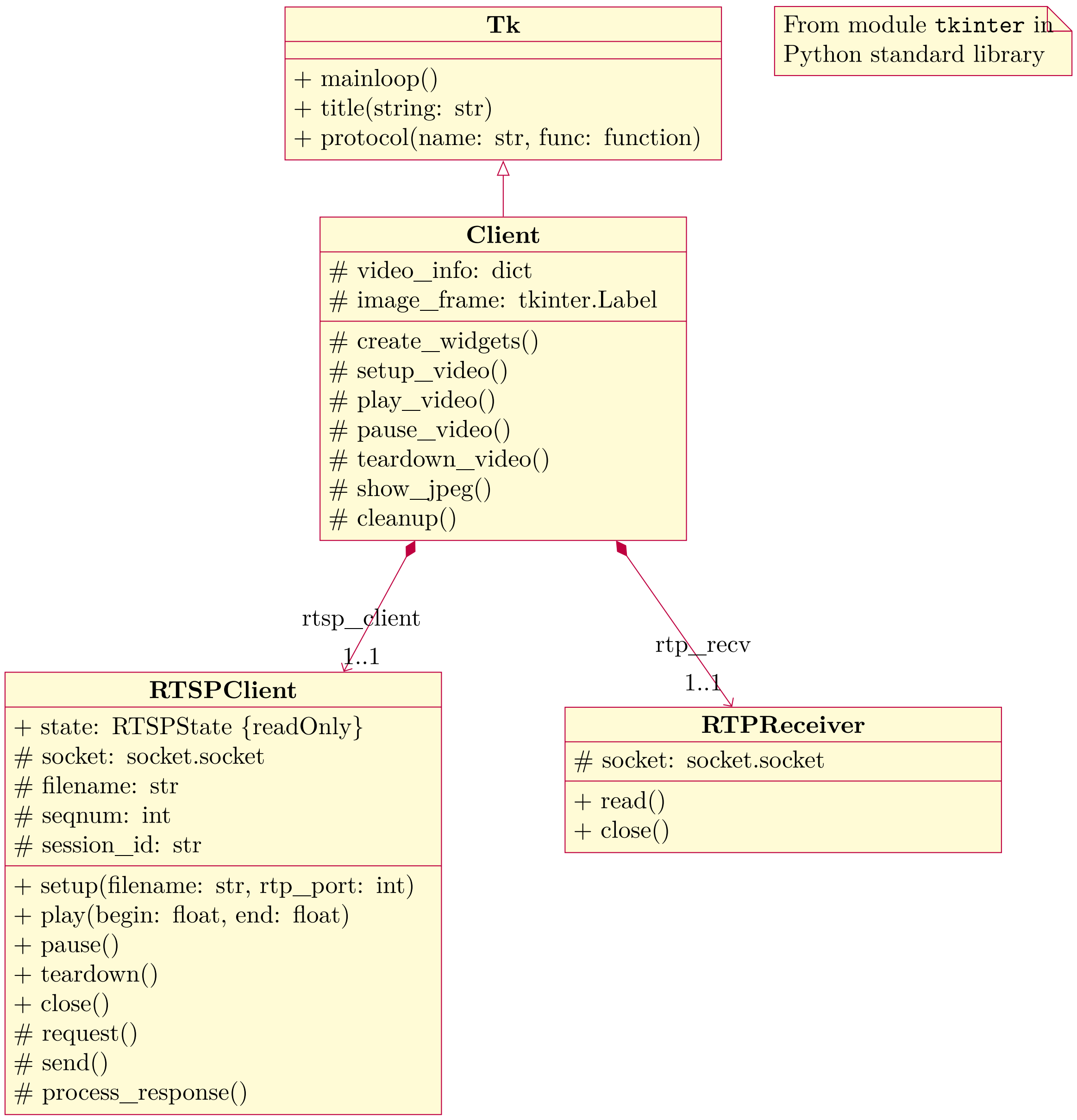


Figure 2: Class diagram for the client

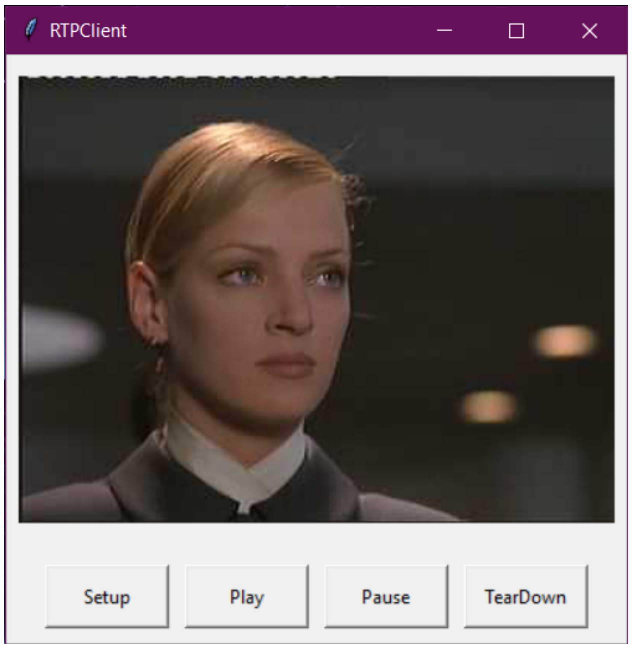


Figure 3: Initial client GUI

3 Implementation Details

3.1 Client

3.1.1 GUI

For the client, we will write GUI interface using `tkinter` module in standard Python library.

The GUI is implemented as **Client** class in `gui.py`, which consists of:

1. A panel above used to hold JPEG frames sent to client by server and display them (this is initially an empty image).
2. Four buttons below linked to each of our client's functions:

   Setup Send setup request to server and initialize RTP port for receiving video data.

   Play Send play request to server and then recevie RTP packets, for each packet, take the packet payload and display the JPEG image to GUI, until server stop sending data (RTP socket timeout).

   Pause Send pause request to server to stop playback.

   TearDown Send tear down request to server, close connection, close RTP socket and shutdown GUI.

3. Our `Client` class uses to helper classes which are:
   - `RTSPClient`: Used by client to send RTSP request and process RTSP response.
   - `RTPReceiver`: Used by client to open RTP socket for receiving RTP data and retrieving payload from RTP packet (which is our raw data of JPEG frame).

GUI's functions

1. `_create_widgets()`: Create GUI as implemented above with a panel and four buttons.
2. `_setup_video()`: Call method `setup()` from RTSPClient with filename and RTP port passed, then receive video data with RTPReceiver.

3. `_play_video(jump=False)`: Call method `play()` from RTSPClient, show warning if user have not set up before playing and display the JPEG image using method `_show_jpeg()`.

4. `_pause_video()`: Call method `pause()` from RTSPClient.

5. `_teardown_video()`: Call method `teardown()` from RTSPClient. Pause the video if it is in Playing state, then create an ok/cancel message box with function `askokcancel()` from module `tkinter.messagebox`. If the answer is ok then clean resources and exit the application, otherwise unpause the video.

6. `_show_jpeg(video_data)`: Receive video data from `_play_video()`, then display each frame continuously to form the video.

3.1.2 RTSPClient

This is our RTSPClient, defined in `rtsp_client.py`. This class define all the functionalities of our client's RTSP requests, including setup, play, pause, teardown, preparing RTSP messages, sending and receiving, decoding RTSP messages.

Functions for processing RTSP message

1. `__init__(server_addr)`: Connect to the server's RTSP socket, set state and sequence number.

2. `_request(method, headers=None)`:
   - This function prepare the proper RTSP request according to the `method` (request type).
   - It then sends the message to server and get the response back.
   - The response is processed using `_process_response()` and returned as a dictionary for easy access.
   - The `headers` parameter is used in case an extra header need to be added (like the transport header in the setup request).

3. `_send(req_message)`: This method send the request message prepared by `_request()` to the server. Then we receive the server response using method `recv()` of class `socket.socket`. Because TCP is a stream-based protocol, so the data returned by `recv()` is not guaranteed to be a complete response message from the server. Therefore, it is important that we have to specify a large buffer size to hold the server response.

4. `_process_response(message)`: Decode the RTSP response, split into lines and create a dictionary containing headers of the message and their respective values.

Client's functions: The client functions make use of the RTSP functions above to make different requests and actions.

1. `setup(filename, rtp_port)`: Send SETUP request to server, including the transport header and retrieve RTSP session ID.

2. `play(begin=None, end=None)`: Send PLAY request with current session ID and read server response.

3. `pause()`: Send PAUSE request with current session ID and read server response.

4. `teardown()`: Send TEARDOWN request with current session ID and read server response.

This class also provide method `close()` to close the RTSP socket after we finish the RTSP session.

3.1.3 RTPReceiver

This is our RTP Receiver for our client, defined in `rtp_receiver.py`.

1. `__init__(listen_port, timeout=0.5)`: Create a datagram socket for receiving RTP data and set the timeout.
2. `read()`: Receive packet from RTP socket, remove header and only return the payload (raw JPEG image).
3. `close()`: Close the RTP socket.

3.2 Server

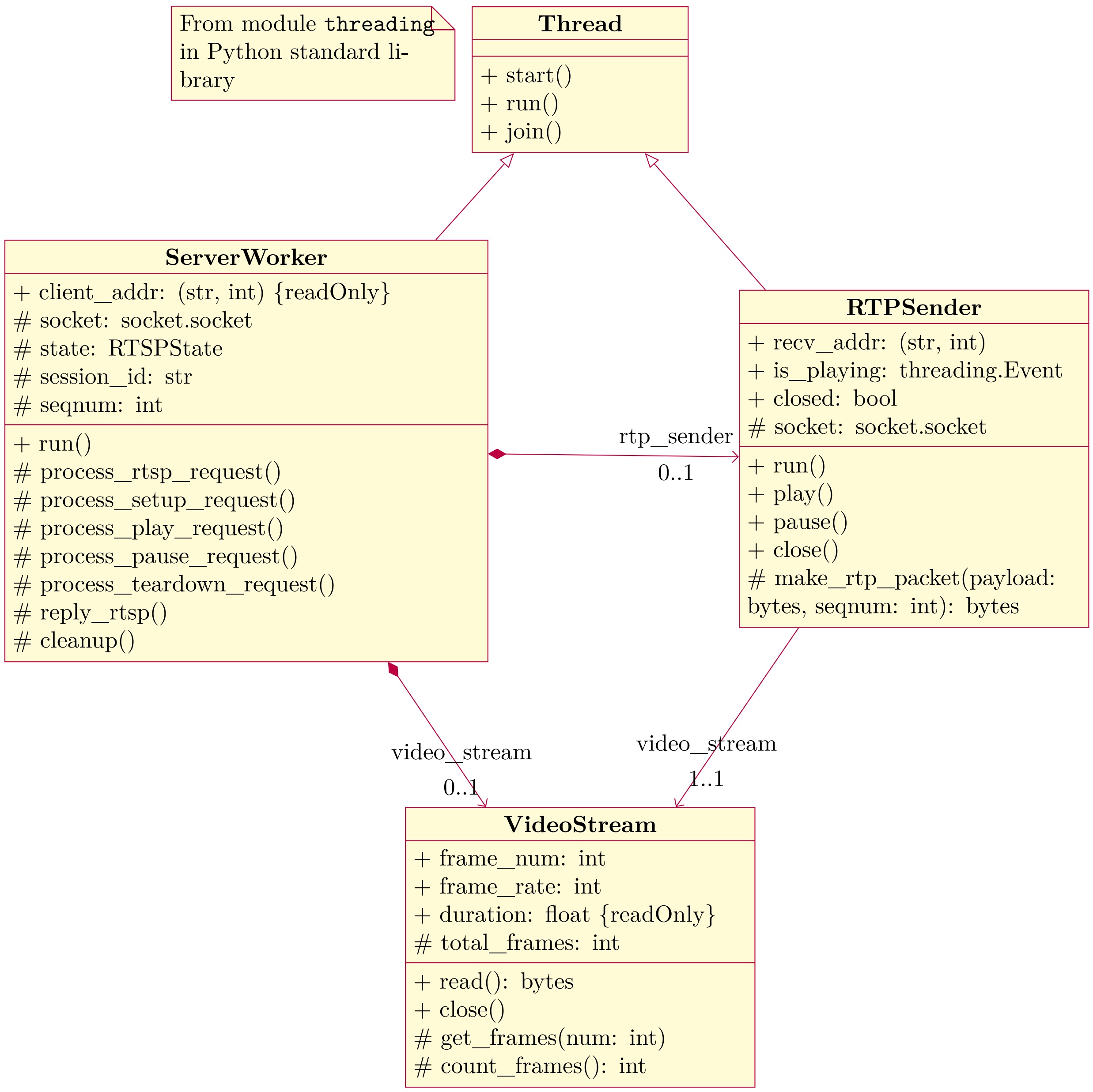


Figure 4: Class diagram for the server

3.2.1 ServerWorker

Each `ServerWorker` object represents a separate thread that listens to client request and reply to those requests.

1. `reply_rtsp(resp)`: Prepare, encode response message and send to client.
2. `run()`: Method representing the server's activity, listens to client request, then it get the request type, and call the appropriate process method for the request.

Process methods

1. `process_setup_request()`: Get media file name and RTP port of client. If the file does not exist, send a 404 Not Found response message to the client. Old video stream is closed if it exits and new stream is set up with new session ID.
2. `process_play_request()`: What this method do is depent on current state of the server:

   Init Send a 455 error message to the client.

   Ready Initialize a `RTPSender` object and start sending RTP packets to client.

   Playing Just send a 200 OK response message to the client and do nothing else.
3. `process_pause_request()`: Signal `RTPSender` object to stop sending RTP packets.
4. `process_teardown_request(request)`: Close RTP connection and clean up resources.

3.2.2 RTPSender

RTPSender is used by ServerWorker to send RTP packet to client.

1. `__init__(client_addr, video_stream)`:
   - Create new socket on server to send RTP packets.
   - Initialize `is_playing` flag as a `threading.Event` object.
   - Set `closed` flag to false.
2. `run()`:
   - First block until the `is_playing` flag is true and check that the `closed` flag is not true, otherwise close the socket and terminate the thread.
   - Use a video stream reader helper class (in our case is `VideoStream`) to read frame by frame the MJPEG video file.
   - For each frame read, make a RTP packet, encode it and send to the client (until the video ends, or paused).
3. `make_rtp_packet()`:
   In this method, video frame will be encoded into a RTP packet. The frame number will be used as sequence number, start from 1 for simplicity. The specification requires the timestamp must be derived from a clock that increments monotonically and linearly in time, so we use function `time.monotonic()`, rounded to millisecond precision. The SSRC (Synchronization Source) field is set to 0 for simplicity. Other fields are set to specified values in the requirement.
4. `play()`: Set `is_playing` flag to true to continue sending RTP packets.
5. `pause()`: Reset `is_playing` flag to false to stop sending RTP packets.
6. `close()`: Set the `closed` flag to true, and also set `is_playing` flag to true in case it is paused.

3.2.3 VideoStream

This class is for reading frame by frame the MJPEG video.

1. `__init__(filename)`: Open the file `filename` and count the total number of frames using method `_count_frames()`.
2. `read()`: This method will read a frame from video stream at position `frame_num`. If the current read frames store in `_read_frames` does not contain frame at that frame number, the method `_get_frames()` will be called to read all neccessary frames into `_read_frames`.
3. `close()`: Close the file object of the video file.

4 User Manual

4.1 Start Server and Client

Server

To start the server, run the following command:

```
python3 -m server <server_port>
```

For example, if you want the server to listen on port 2000, run:

```
python3 -m server 2000
```

Client

To start the client, run the following command:

```
python3 -m client <server_addr> <server_port> <rtp_port> <filename>
```

For example, if you want to stream file movie.Mjpeg using server at localhost and port 2000, receive RTP frames from port 3000, run:

```
python3 -m client localhost 2000 3000 movie.Mjpeg
```

4.2 GUI Usage

After starting the server and client, a window like this will pop up:

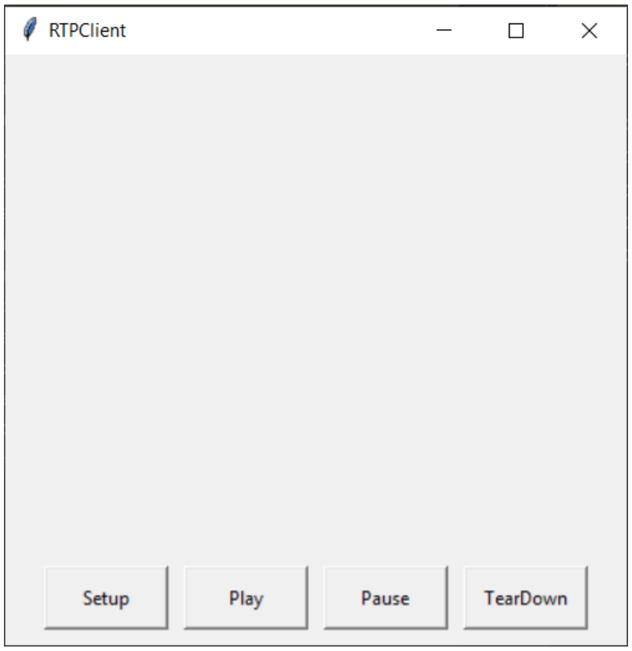


Figure 5: Initial client GUI

There are 4 buttons on the RTPClient window, each one is used to send request to the server and has function as mentioned:

Setup Set up the session and transports parameters

Play Start or resume the playback

Pause Pause during the playback

TearDown Terminate the session and close the connection

Follow the order in figure 1 to make the Client run normally, else you will cause error response from server or nothing will happen.

5 Result Evaluation

This part will evaluate the result of the assignment and what we've learnt during the process. Here are things we achieved:

- How to do socket programming, setup server/client connections.
- RTP and RTSP protocol, including its headers, payload and encoding, decoding process.
- Keeping track of client's state and other in formations.
- Organize the server and client class hierarchy to separate functionalities and jobs.
- Builing simple GUI using `tkinter`.
- Parsing raw JPEG binary data into GUI display.

6 Extension

6.1 Statistics

By adding attribute `data` (for storing data) to our client's RTP receiver, we can record the arrival time and the payload size of each arriving packets. After the transfer session, data can be extracted and stored in a CSV file that we later use for data visualization in the figure below. A CSV file `stats.csv` can be found in root folder when client–server RTP session is running.

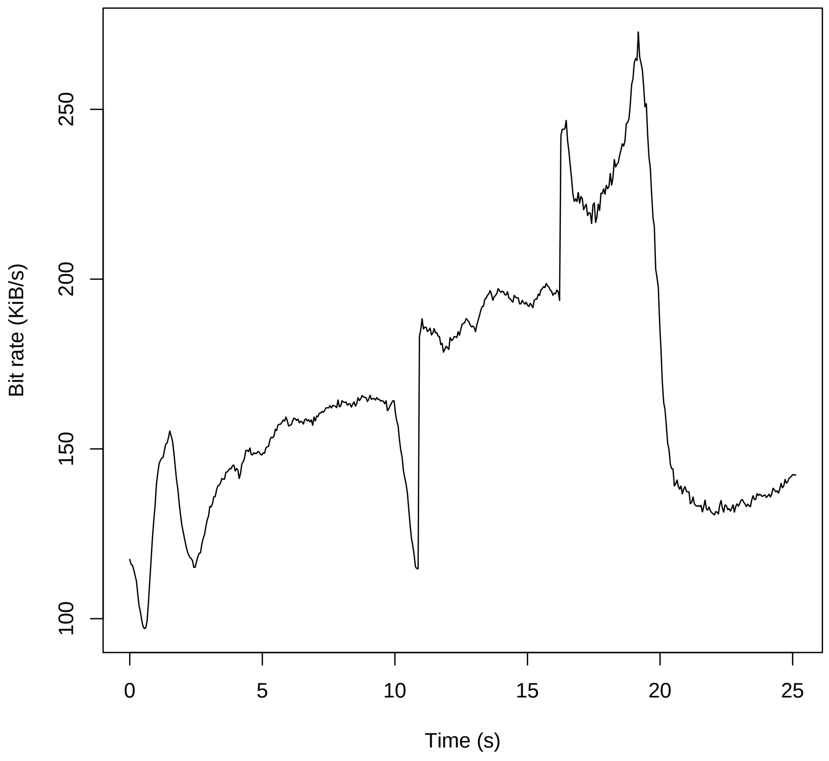


Figure 6: Bit rate over time

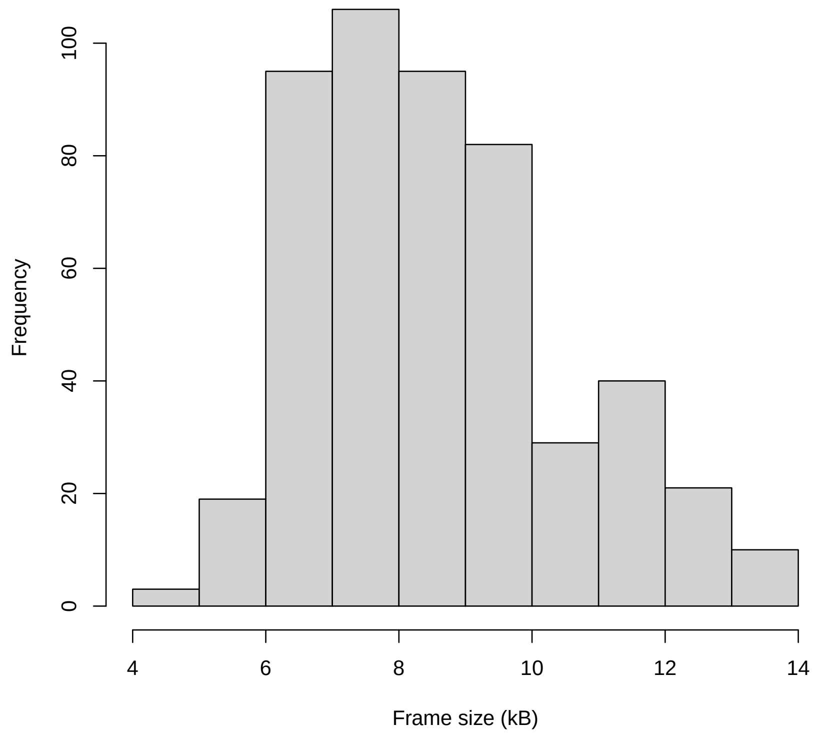


Figure 7: Histogram of video frame size in kilobytes

6.2 User-Friendly GUI Interface

To implement the more user-friendly version of GUI interface using only 3 control buttons (Play, Pause, Stop), we will subclass the class `Client`, the previous version of our GUI interface. To implement button Play, we will call `setup()` and `play()` method of class `Client`. Button Pause and Stop are roughly corresponding to method `pause()` and `teardown()` method of class `Client`. To be more specific:

Play Try setup and start playback

Stop Teardown

Pause Pause

Code usage: To initialize the client with simple GUI instead of the default GUI, just add the flag `--simple` to the command. Example:

```
python3 -m client --simple localhost 2000 3000 movie.Mjpeg
```

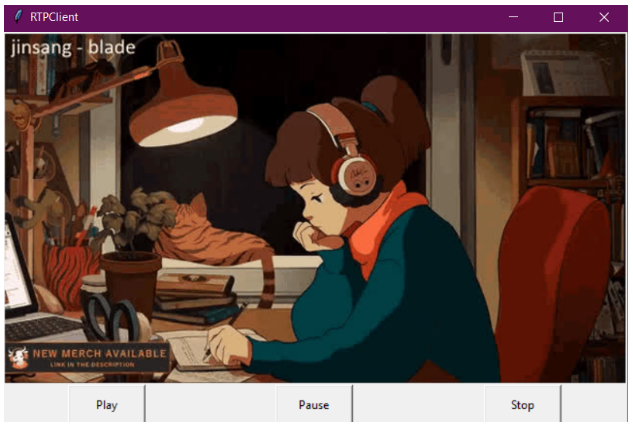


Figure 8: Simple (user-friendly) client GUI

6.3 DESCRIBE Method

We will implement DESCRIBE method using SDP (Session Description Protocol, see RFC 4566). SDP provides a standard representation for media details, transport addresses, and other session description metadata. An SDP session description consists of a session-level section followed by zero or more media-level sections.

Here is an example SDP description of a RTSP session:

```
v=0
o=- 3813623543 3813623543 IN IP4 127.0.0.1
s=RTSP Session
m=video 0 RTP/AVP 26
a=rtpmap:26 mjpeg
a=framerate:20
a=range:npt=0-25.0
```

The meaning of each field is:

v The version of the Session Description Protocol

o The originator of the session plus a session identifier and version number

s The textual session name

m Media description

a Attribute

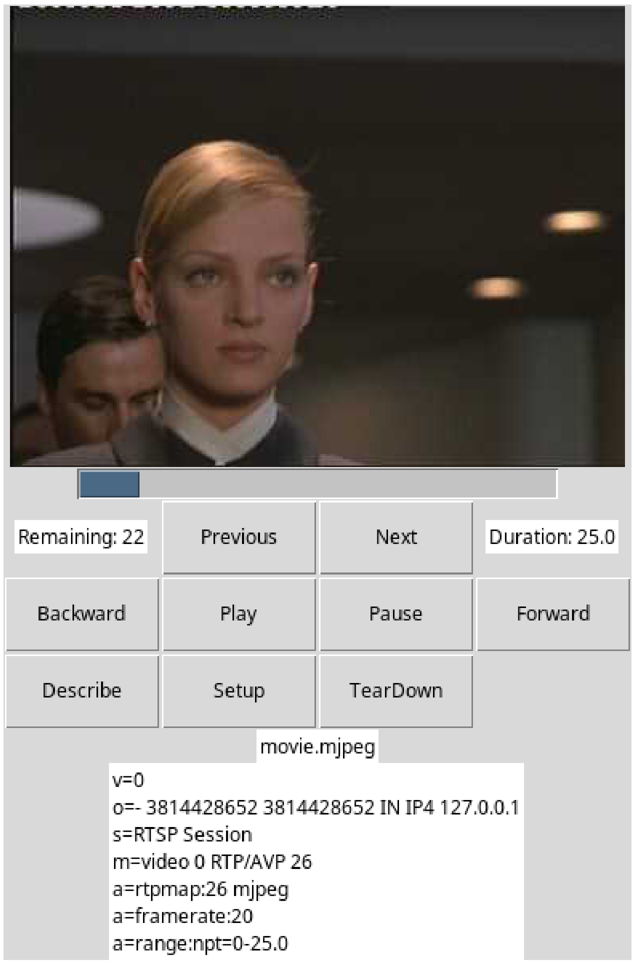
- **rtpmap** The encoding format of the video stream
- **framerate** The frame rate of the video stream
- **range** The total time range of the video stream

6.4 Additional Function

6.4.1 Display Video Time

To get the total time of the video, we parse the range attribute in SDP description from server reply to DESCRIBE request. To get remaining time, we keep track of current progress in the server.

To continuously update remaining time displayed on the GUI, we use class `StringVar` from module `tkinter`.



```
v=0
o=- 3814428652 3814428652 IN IP4 127.0.0.1
s=RTSP Session
m=video 0 RTP/AVP 26
a=rtpmap:26 mjpeg
a=framerate:20
a=range:npt=0-25.0
```

Figure 9: Demonstration of Describe option for GUI interface

6.4.2 Forward/Backward

To add forward/backward functionality to the client, we have to implement `Range` header for PLAY request. The time format is NPT (normal play time). Normal play time (NPT) indicates the stream absolute position relative to the beginning of the presentation. The timestamp consists of a decimal fraction. The part left of the decimal may be expressed in either seconds or hours, minutes, and seconds. The part right of the decimal point measures fractions of a second. For example, the Range header for PLAY request start at 5:00 position till the end of the stream is as follows:

```
Range: npt=300-
```

To allow seeking of video frame, we also implement new method `set_time(time)` for class `VideoStream`. This method set current frame number to the frame number of the frame at specified time. When server receive a PLAY request with `Range` header, it will call the `set_time()` method of `VideoStream` to seek to the frame at the begin time position specified in `Range` header.

6.5 Switch

For the video switching functionality, we implemented a **Next** and **Previous** button in the GUI to allow user to send **Switch** request to the server in order to traverse and change between different MJPEG videos that server offers.

State To switch to the next/prev video, the client's state must be **READY**. Which means the stream is already set up, and if it's in **PLAYING** state, of course a prior `pause` is automatically called by the client. After the video switching, the new stream is in the paused state and the client is now in the **SWITCH** state to indicate video switching.

Client When user press the next or previous function on our GUI, the RTSP client's `switch` function is provoked to request the server to change its video stream. The client also parses server response to get the new video filename being played.

Server On this side, when **SWITCH** request is received, the server get the next video from available name list, replace the video stream with the new video and send response along with the new filename back to client. (The new filename is put in the `New-Filename` header of the RTSP packet.)

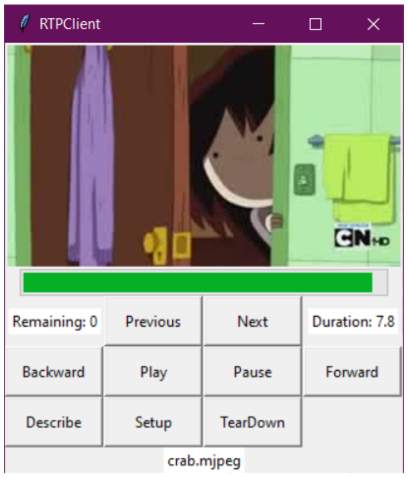


Figure 10: Before switching (old video)

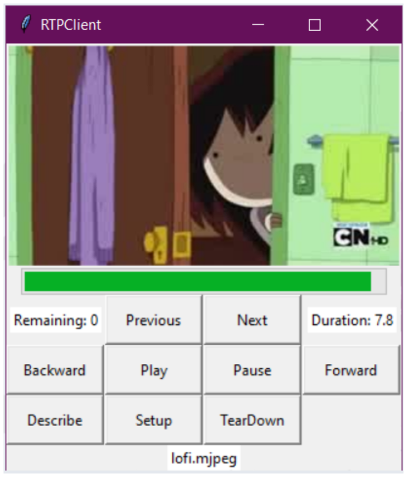


Figure 11: After switching (notice new filename)

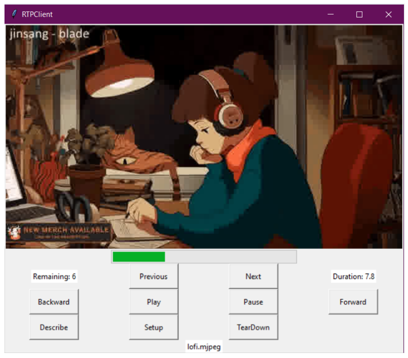


Figure 12: Play is pressed (new video)

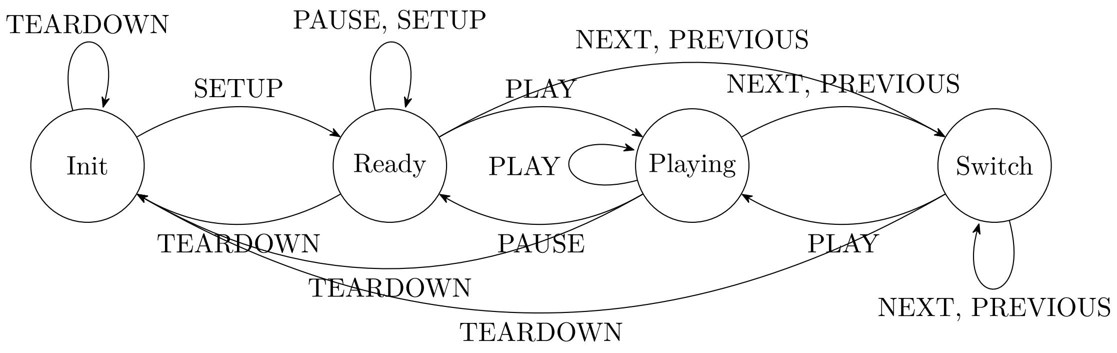


Figure 13: Client state machine after adding Switch state