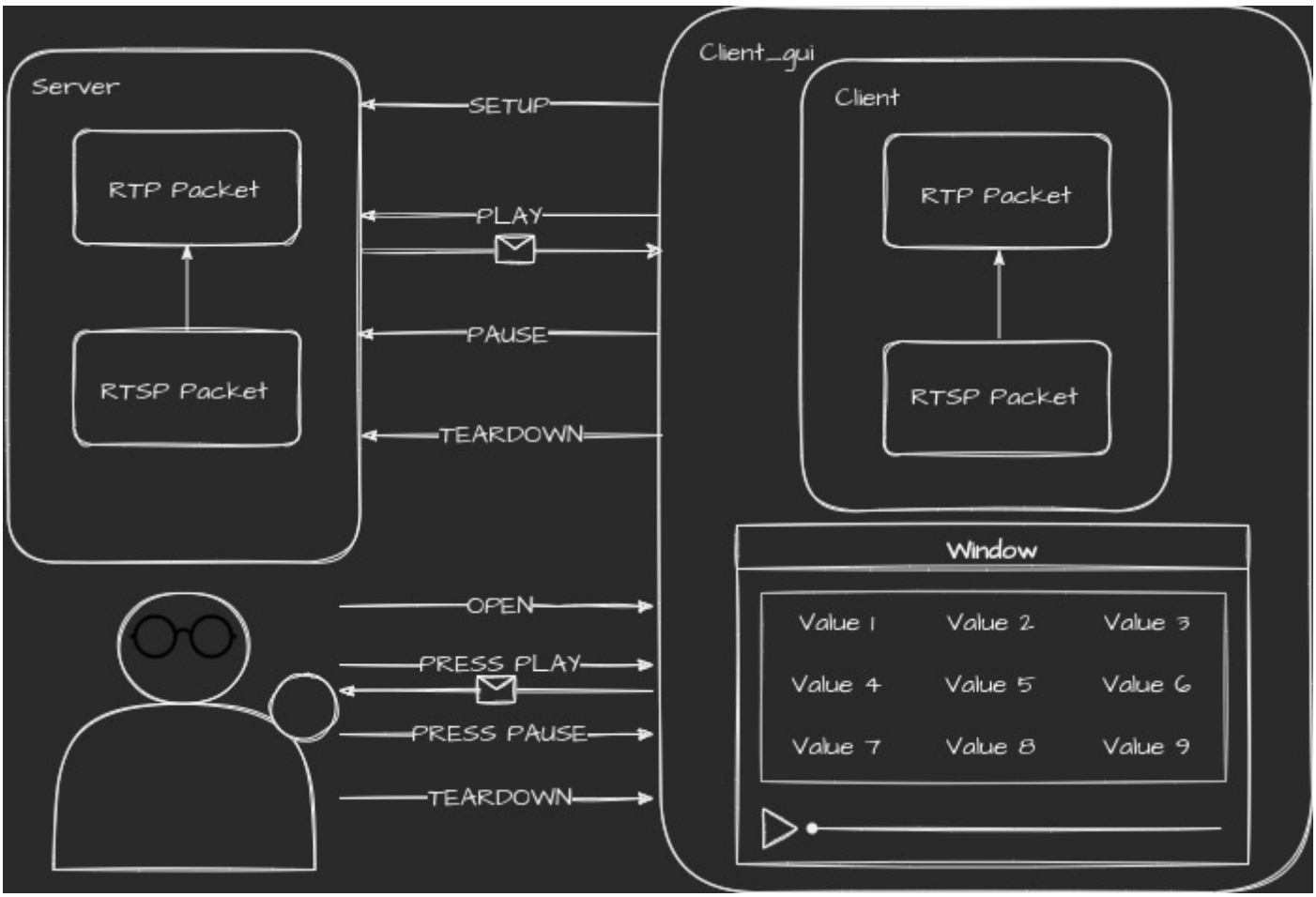
RTSP/RTP Streaming Project

> 熊彥程 B07901044
> 范育瑋 B07901047
> 王昊謙 B07202020

Project Overview



Advanced Features

- Live Webcam Streaming
- Graphic User Interface

Usage

- Server: `python run_server.py <server address> <server port> <source type>`
- Client: `python run_client.py <file name> <host address> <host port> <RTP port> <source type>`

Client Side

Functioning

- Client class
  - send:
    - RTSP request
  - receive:
    - RTSP request
    - RTP packet
- Client GUI class
  - SETUP
    Sent when GUI initialization
  - PLAY
    Sent when user presses play button
  - PAUSE
    Sent when user presses pause button
  - TEARDOWN
    Sent when user presses the teardown button or close GUI

Implementation

The client is implemented as a class named `Client`, with the following methods:

- `client.establish_rtsp_connection()`: Connect the RTSP socket to the server.
- `client._send_request(request_type)`: Send different RTSP request based on the request type to the server.
- `client._get_response()`: Get the RTP response from the server.
- `client._get_frame_from_packet(packet)`: Extract the payload of the RTP packet, which contains the image data sent from the server.
- `client._handle_video_recv()`: Start the RTP socket, receive RTP packet from the server and extract frame from the payload.

The GUI is implmented as a wrapper class `Client_GUI` of the `Client`, the following is the main methods.

- `client_gui.init_ui()`: Setup the size of the window, the range of the slider and the layout.
- `client_gui.slider_moved()`: This method is connected to the event that the slider is dragged. It will change the current frame sequence number.
- `client_gui.handle_setup()`: Enable the play/pause button and teardown button.
- `client_gui.handle_play(packet)`: Let the client send play request to the server. Update play/pause button status and the slider value.
- `client_gui._handle_teardown()`: Let the client send teardown request to the server. Close the RTSP connection.

Server

Functioning

The server has the following jobs:

- send:
  - RTSP response
  - RTP packet
- receive:
  - RTSP request
  - Response to RTSP request
    - SETUP
      Read assigned video file and responese to client
    - PLAY
      Send RTP packets per frame for every frames in the video
    - PAUSE
      Stop sending RTP packets and wait until next PLAY request
    - TEARDOWN
      Close every sockets

Implementation

The server is implemented as a class named `Server`, with the following methods:

- `server.run()` : start the server. It will start listening for RTSP requests.
- `server._teardown()` : clean up the sockets, RTP sending thread and video stream.
- `server._setup(packet)` : Setup RTP sending thread, with information provided by the request packet.
- `server._send_rtp_packet(event)` : The method running in the RTP sending thread, sending RTP packets in a loop. `event` is an `Threading.Event` object used to pause and resume the thread.
- `server._get_rtsp_packet()` : Return the RTSP packet received from the socket.
- `server._better_sleep(sec, expected_inaccuracy=0.3)` : make the program sleep for `sec` time more accurately than `time.sleep()`.

The server has the following states:

- `INIT` : waiting for SETUP request.
- `READY` : the RTP socket is set up and ready to send.
- `PLAYING` : sending RTP packets.
- `FINISHED` : finished playing the video.

The RTP sending function is realized using the `Threading` module. Once the client send `Play` request, the RTP sending thread will start sending packets.

The file streaming function is handled by `VideoStreaming` class. It provides frames from either video files or webcam (mentioned later in bonus). We currently only implement the easiest `mjpeg` format. It search for jpeg starting byte (`\xff\xd8`) and EOF byte (`\xff\xd9`) to identify frames. It support the following methods:

- `videostream.get_next_frame()` : return the next frame from either video file or webcam.
- `videostream._get_next_frame_from_file()` : get the next frame from the mjpeg file.
- `videostream._get_next_frame_from_webcam()` : get the next frame from webcam.
- `videostream._close()` : clean up.

RTSP/RTP

RTSP and RTP Packet are implemented in utils/rtsp_rtp.py.

- RTP Packet Class

- Attribute
  - payload_type
  - seq_num
  - time_stamp
  - header

Offsets	Octet	0								1								2								3							
Octet	Bit [a]	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	Version		P	X	CC				M	PT							Sequence number															
4	32	Timestamp																															
8	64	SSRC identifier																															
12	96	CSRC identifiers ...																															
12+4×CC	96+32×CC	Profile-specific extension header ID																Extension header length															
16+4×CC	128+32×CC	Extension header ...																															

  source:wikipedia
- Method
  - frompacket
    bytes -> RTP packet
  - getpacket
    RTP packet -> bytes

- RTSP Packet Class
  - Attribute
    - request_type
    - video_path
    - seq_num
    - rtp_port
    - session_id
  - Method
    - from_response
      RTSP response bytes -> RTSP response packet
    - build_reponse
      RTSP response packet-> RTSP response bytes
    - from_request
      RTSP request bytes -> RTSP request packet
    - to_request
      RTSP request packet -> RTSP request bytes

Bonus

Live Webcam Streaming

Aside from video file streaming, the server also supports live webcam streaming. By specifying `webcam` in the command line input of both server and client, the server will send live webcam video to the client.
Live webcam streaming is realized using the `cv2` module. The frames obtained from the webcam is put into the RTP packets just like the mjpeg frames, and send to the client.

Client GUI

We also implement a client side graphical user interface. The display window enables the user to press the play/pause button and drag the slider. Note it also supports rewind for file streaming mode.

Reference

- RTSP and RTP streaming