

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



COMPUTER NETWORK

ASSIGNMENT 1

CC06

Network Application Programming (Socket Programming)

INSTRUCTOR:

Lecturer: Nguyễn Mạnh Thìn (Lab)

GROUP MEMBERS:

Tạ Minh Huy - 1952268

Hoàng Hà Giang - 1952659

Lê Công Trường Thọ - 1852768

APPENDICES

1. OVERVIEW.....	3
2. IMPLEMENTATION.....	4
I. ASSIGNMENT REQUIREMENTS.....	4
A. Functional Requirements:.....	4
i. Administrative functions:.....	4
ii. User Interface:.....	4
B. System Architecture Design:.....	4
II. FUNCTION DESCRIPTION.....	6
A. Client.py.....	6
B. ClientLauncher.py.....	8
C. RtpPacket.py.....	8
III. CLASS DIAGRAM.....	10
3. USER MANUAL.....	10
I. SETUP.....	10
II. APPLICATION'S FUNCTIONS.....	12
3. SUMMATIVE EVALUATION OF ACHIEVED RESULTS.....	14

1. OVERVIEW

Streaming Media may be defined as listening or viewing media in real time. With streaming technology, users can watch and listen to media while it is being sent, instead of waiting for it to completely download and then playing it. Before streaming technology was available, a user might wait an hour (or more!) to completely download a short media file.

In general, media files are huge. For example, five minutes of uncompressed video would require almost one gigabyte of space! So, when the audio and video is prepared for streaming, the media file is compressed to make the file size smaller. When a user requests the file, the compressed file is sent from the video server in a steady stream and is decompressed by a streaming media player on the user's computer to play automatically in real time. A user can jump to any location in the video or audio presentation. Streaming media generally tries keep pace with the user's connection speed in order to reduce interruptions and stalling.

In this assignment, we are asked to create a real video server and client that communicate using the Real-Time Streaming Protocol (RTSP) and send data using the Real-Time Transfer Protocol (RTP).

On the Client side, we will need to implement the RTSP protocol. Initially, when the clients start, we will need to open the RTSP socket to connect to the server. Additionally, we have to define the four main function that are called when the users click on the button on the user interface: SETUP, PLAY, PAUSE, TEARDOWN.

2. IMPLEMENTATION

I. Assignment Requirements

A. Functional Requirements:

i. Administrative functions:

- 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.

ii. User Interface:

- The GUI (Graphical User Interface) provided in the file need to be friendly and easy to use.

B. System Architecture Design:

i. The Client

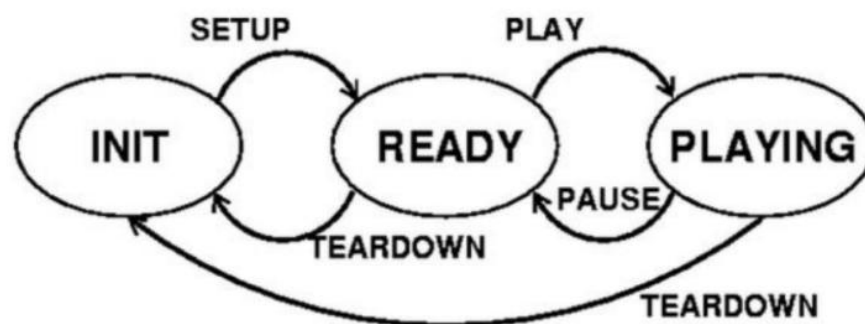


Image 1. The Client States

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 will reply accordingly to The Client's request, as followed:

200: SUCCESSFUL

404: FILE_NOT_FOUND

500: CONNECTION ERROR

ii. 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

Here's the diagram to indicate the RTP-Packet format

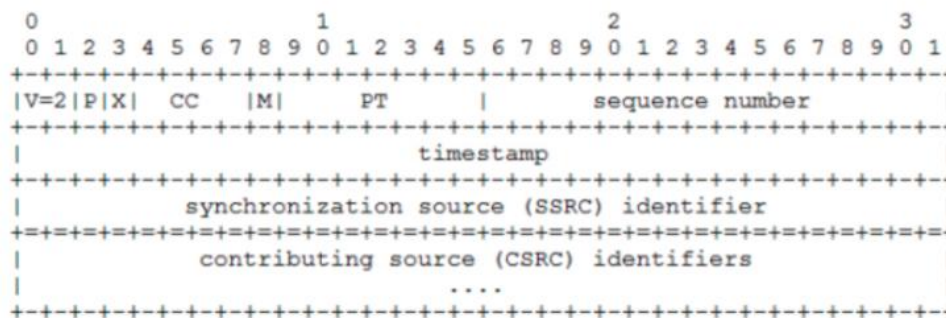


Image 2. RTP-Packet format

II. Function Description

A. Client.py

- This file stores the Client Class to initiate the objects to handles interactions events received from the user (when the buttons are pushed and requests are made by the user) which manage the GUI and handle the given events.

- The constructor of Client will initiate with the following parameters:

- + *master (Tk)*: the root GUI to create the GUI for the application.
- + *serveraddr (str)*: the hostname of the server
- + *serverport (str)*: the unique port for the server
- + *rtpport (str)*: the port for RTP
- + *filename (str)*: the input playback file for the application.

- The following functions (methods) are consisted in the Client class:

CreateWidget(self)

- Create **Setup** button and add 'setup' handler
- Create **Play** button and add 'play' button handler

- Create **Pause** button and add 'pause' handler
- Create **Teardown** button and add 'teardown' handler
- Create Label to display the movie

SetupMovie(self)

- **Setup** button handler where it checks the Client's state is 0 (initial state) or not to send RTSP **Setup** request to server.

ExitClient(self)

- **Teardown** button handler ,when called, will send RTSP **Teardown** request to server, then close GUI window and delete cache of the storing images.

PauseMovie(self)

- **Pause** button handler, can only be called when the video is playing, it will send a RTSP **Pause** request to server.

PlayMovie(self)

- **Play** button handler, can only be called when Client state is set. It'll initiate a new thread that call the *listenRtp()* function then start execution. Then it instantiate *playEvent* variable to store responding *Event object* then returned by threading module. The event flag is set to false then send a RTSP **Play** request to server.

ListenRtp(self)

- This function listens to RTP packets and creates an infinite loop is set to continuously listen for new RTP packets. The variable *data* is created to store bytes returned by RTP socket. If *data* is not null, it will be decoded and update the label in the UI. If Client press **Pause** or **Teardown** buttons, the loop will break and also reaching the end of the image video file will break the loop and close the socket.

WriteFrame(self, data)

- Parameters:
 - + *data*: the frame's data from the image file.
- In this function, we take the frame received by server then store it in a temporary image file (we store it in cache) and return the image file.

UpdateMovie(self, imageFile)

- Parameters:
 - + *imageFile*: the input Mjpeg image file.
- In this function, we need an image as a parameter then we take the image file and write it into the label.

ConnectToServer(self)

- In this function, we instantiate RTSP socket in order to connect to server with server address and server port already provided in the constructor.

SendRtspRequest(self, requestCode)

- Parameters:
 - + *requestCode*: the request data from the user generate after pushing the buttons from the GUI.
- Client send Setup request: create new thread to listen to RTSP response, update RTSP sequence, create new request with type setup, send the request to server.

- Client send Play request: update RTSP sequence, create new request with type play, send the request to server.
- Client send Pause request: update RTSP sequence, create new request with type pause, send the request to server.
- Client send Teardown request: update RTSP sequence, create new request with type teardown, send the request to server.

RecvRtspReply(self)

- This function is responsible for receive RTSP response from server. An infinite loop is created in order to receive the message. If it receives response, it will parse RTSP reply. If request state is change to Teardown, we will stop RTSP socket and break the loop.

ParseRtspReply(self, data)

- This function is responsible for parsing RTSP reply from server. We split the reply according to '\n' and split ' ' in index number 1 to get sequence number. If server response correct sequence number then check the session ID. If session ID is valid and status code is 200 then continue execute. If the request just sent is Setup then update state to READY and open RTSP port. If the request just sent is Play then update state to PLAYING. If the request just sent is Pause then update state to READY and set Event object flag to true. Otherwise, update state to INIT

OpenRtpPort(self)

- This function is responsible for open a RTP socket and port (this socket is different from RTSP socket). RTP socket is responsible for reading RTP packet from server. We set timeout value of the socket to 0.5s. Finally, it binds the socket to the address using the RTP port given by client

Handler(self)

- This function is called when user closing the UI window. Firstly it will pause the movie, than ask the user if they want to close the window, otherwise resume playing the video.

B. ClientLauncher.py

- This is the file where the user instantiate Client object (from the Client class above) and Tk object (this creates a widget of Tkinter to build the window of the application). The application window's will be titled as RTPClient which will present the UI accordingly to the ***CreateWidget()*** function above.

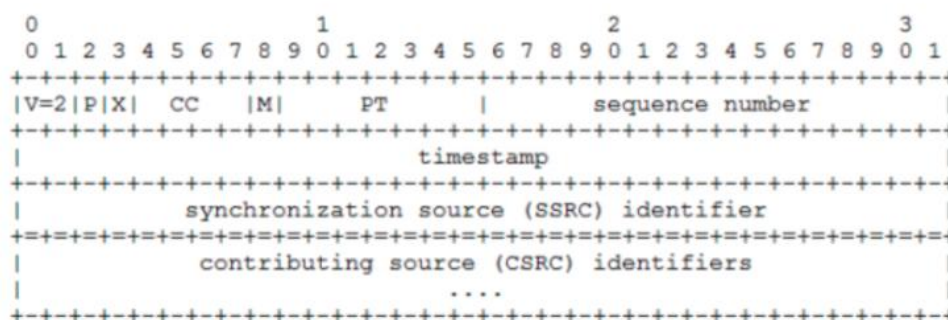
C. RtpPacket.py

- This file stores the RtpPacket class, which create the objects for the Client to de-packet (decode) the data and for the Server to encapsulate the data (encode).
- The RtpPacket class will include these functions (methods):

Encode(self, version, padding, extension, cc, seqnum, marker, pt, ssrc, payload)

- Parameters:

- + *version*: the version code from the header
- + *padding*: the padding bit
- + *extension*: the extension bit
- + *cc*: the 4 bits contribution sources
- + *seqnum*: the 12 bits sequence number
- + *marker*: the marker field bits
- + *pt*: the 7 bits payload
- + *ssrc*: the server identifier
- + *payload*: frame's data
- This function encapsulate the data as a packet from the Server and then send it to the Client. The header of the packer is created using bit-wised operations such as masking and shifting to set, the format is as followed:



Decode(self, byteStream)

- Decoding the RTP packet from the Server by reading the a byte array from the header and a byte stream of the payload byte-by-byte.

Version(self)

- Shift right 6 bit in header[0] to return version

SeqNum(self)

- Return combination of header[2] and header[3]

TimeStamp(self)

- Return combination of header[4], header[5], header[6], header[7]

PayloadType(self)

- Return header[1] & 127 (& 1111111)

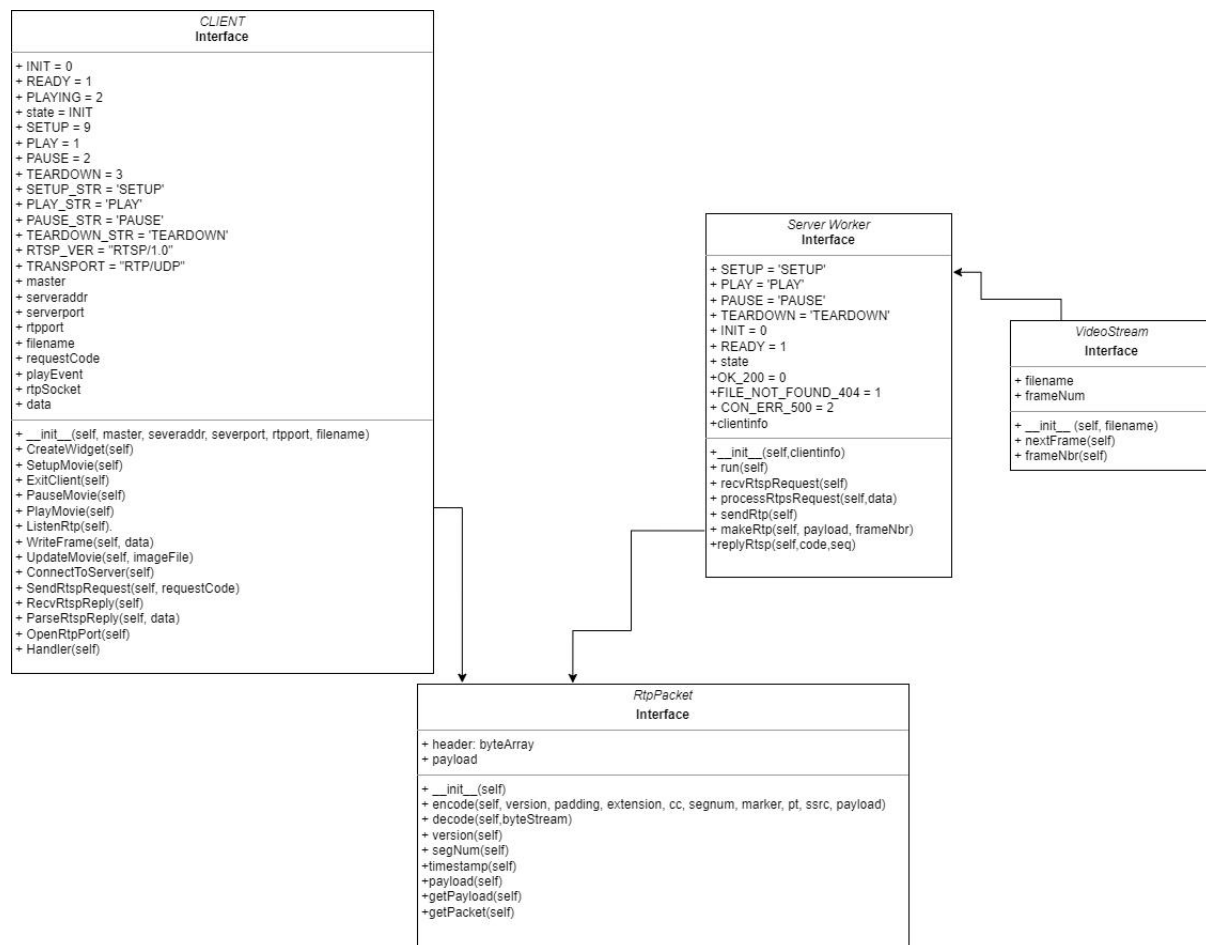
GetPayload(self)

- Return payload from RTP packet

GetPacket(self)

- Return both header and payload

III. Class Diagram



3. USER MANUAL

I. Setup

To play the video streaming application first:

Step 1: Start the Server with the following command in terminal :

`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 terminal 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).

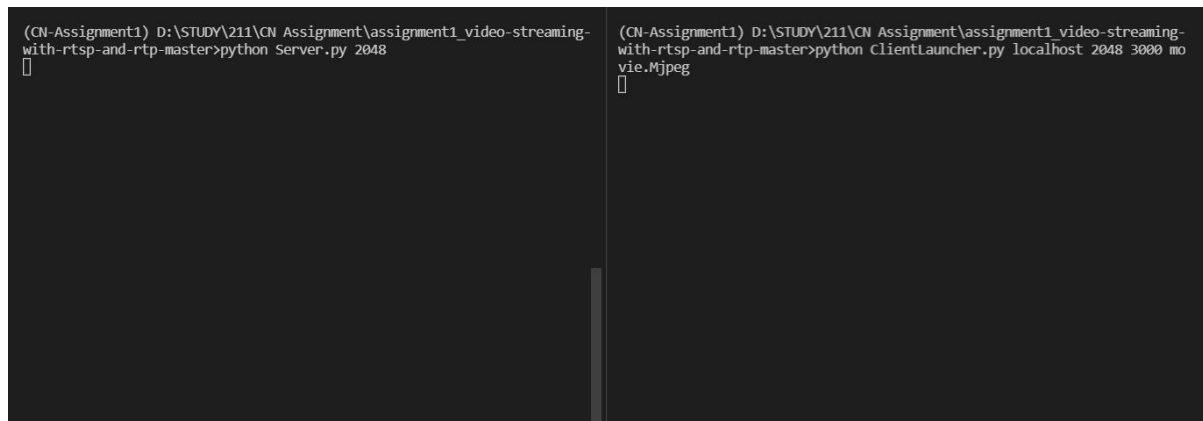


Image 3. Server terminal (left) and Client terminal (right)



Image 4. Application GUI

II. Application's Functions

Step 3: Give the RTSP request to **SETUP** the image file by pressing the SETUP button.

```
Data Sent:
SETUP movie.Mjpeg RTSP/1.0
CSeq: 1
Transport: RTP/UDP; client_port= 3000
```

Image 5. The Client send the request

```
DATA RECEIVED:
SETUP movie.Mjpeg RTSP/1.0
CSeq: 1
Transport: RTP/UDP; client_port= 3000
PROCESSING SETUP
```

Image 6. The server respond to the request

Step 4: Give the RTSP request to **PLAY** the image file by pressing the PLAY button.

```
Data Sent:
PLAY movie.Mjpeg RTSP/1.0
CSeq: 2
Session: 245229
CURRENT SEQUENCE NUM: 1
LISTENING...
CURRENT SEQUENCE NUM: 2
LISTENING...
CURRENT SEQUENCE NUM: 3
LISTENING...
CURRENT SEQUENCE NUM: 4
LISTENING...
CURRENT SEQUENCE NUM: 5
LISTENING...
CURRENT SEQUENCE NUM: 6
LISTENING...
CURRENT SEQUENCE NUM: 7
LISTENING...
CURRENT SEQUENCE NUM: 8
LISTENING...
CURRENT SEQUENCE NUM: 9
LISTENING...
CURRENT SEQUENCE NUM: 10
```

Image 7. The Client will create a loop listening to the RTP Packet send from the Server until PAUSE, TEARDOWN is requested or until the video ends.

```
DATA RECEIVED:
PLAY movie.Mjpeg RTSP/1.0
CSeq: 2
Session: 245229
PROCESSING PLAY
```

Image 8. The Server responds to the PLAY request

Step 5: Give the RTSP request to **PAUSE** the image file by pressing the PAUSE button.

```
Data Sent:
PAUSE movie.Mjpeg RTSP/1.0
CSeq: 3
Session: 245229
```

Image 9. the Client sends the PAUSE request

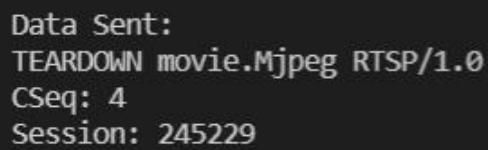
```
DATA RECEIVED:
PAUSE movie.Mjpeg RTSP/1.0
CSeq: 3
Session: 245229
PROCESSING P A U S E
```

Image 10. The Server response to the PAUSE request



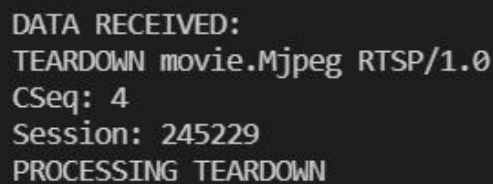
Image 11. The current GUI

Step 5: Give the RTSP request to **TEARDOWN** the image file by pressing the TEARDOWN button and stop the application's session.



```
Data Sent:
TEARDOWN movie.Mjpeg RTSP/1.0
CSeq: 4
Session: 245229
```

Image 12. The Client sends the TEARDOWN request



```
DATA RECEIVED:
TEARDOWN movie.Mjpeg RTSP/1.0
CSeq: 4
Session: 245229
PROCESSING TEARDOWN
```

Image 13. The Server receives the request and close the GUI

3. Summative Evaluation of Achieved Results

- We successfully create a video stream application run smoothly with basic requirement (Set up, Pause, Play, Teardown).
- We are able to **simultaneously 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**.
- The terminal's information is clear and easy to track, as we tested, no bugs or malfunctions found.
- The GUI is basic but fairly easy to use.