VIETNAM NATIONAL UNIVERSITY - HCM
Ho Chi Minh City University of Technology
Faculty of Computer Science and Engineering

COMPUTER NETWORKS (CO3004)

Assignment 1

NETWORK APPLICATION PROGRAMMING

Advisors: PhD. Nguyen Le Duy Lai

Candidates: Nguyễn Duy Tịnh - 1852797
Lưu Nguyễn - 1852622

Ho Chi Minh City, April 2021

Contents

1 Requirements Analysis

1.1 Functional requirements

- The program has stable TCP connections.
- UDP socket receives and transfers package in real time.
- When a problem occurs, the program can handle with explicitly closing.
- The program must allow multi-programming and be able to work with multiple threads.
- The program should minimize the loss of RTP packet when transfering.

1.2 Non-functional requirements

- *Performance*: The program should allow at least 100 clients to connect to the server concurently.
- *Usability*: The program should have a friendly GUI, with four buttons (setup, play, pause, tear-down) and a label for movie displaying. Clients can easily understand all functions and be able to use the program at first glance.
- *Maintainability*: The program is written in python standard coding style and organized in OOP approach that helps the developers easy to maintaince and upgrade.
- *Accessibility*: The language and grammar used in GUI is straight forward.

1.3 System requirements

- The Server will run first, open the TCP port and listen for TCP signal.
- When running, the Client will open the TCP port and connect to Server through RTSP socket. At the same time, Client will also generate the GUI.
- When `SETUP` being called, Client will open a new thread to serve. The `requestCode` will be send to Server as an RTP packet through UDP connection.
- After receive and process the RTP Packet, Server will open a new socket to reply back.
- Client then will receive and parse the RTPPacket.

2 Description of different functions for the application

2.1 The Client

2.1.1 `setupMovie`

```
Input parameter: self
```

This function is attached to the Setup button and will be used to initialize the connection between the Client and the Server. We need to check the current state of the Client, only allow this function to work if the state is INIT, avoiding unwanted behaviour during runtime. When the condition is met, we will send a SETUP RTSP request to the Server.

2.1.2 `exitClient`

```
Input parameter: self
```

This function is attached to the Tear down button and will terminate the session and close the connection between the Client and the Server. First, we will send a TEARDOWN RTSP request to the Server, then we will shutdown the GUI window by using the `destroy()` function in Tkinter. Finally we have to remove all of the cache image file, using the `os.remove()` function.

2.1.3 pauseMovie

`Input parameter: self`

This function is attached to the Pause button and will be used when the Client want to pause the streaming. We need to check the current state of the Client, only allow this function to work if the state is PLAYING, avoiding unwanted behaviour during runtime. When the condition is met, we will send a PAUSE RTSP request to the Server.

2.1.4 playMovie

`Input parameter: self`

This function is attached to the Play button and will be used when the Client want to start playing the streaming, or resume the streaming after pausing. We need to check the current state of the Client, only allow this function to work if the state is READY, avoiding unwanted behaviour during runtime. When the condition is met, we will create a new thread to listen to the RTP response from the server. Then

2.1.5 listenRtp

`Input parameter: self`

This function continually tries to listen for data from the Server. If it receives a packet of data, it will encode this packet into an RTP packet and call 2 functions: `writeFrame()` and `updateMovie()` to turn the RTP packet into visible movie frame.

If we trigger the `playEvent` flag, meaning that there is a PAUSE or TEARDOWN request, the function will stop listen for data, and if possible, shut down the socket (TEARDOWN resquest).

2.1.6 writeFrame

`Input parameter: self, data`

This function will try to open the cache file, which is our temporary image file, using the Python built-in `open()` function with 'wb' option, and write the data in the Input parameter to the file using the `write()` function, ultimately, return the cache file after the process.

2.1.7 updateMovie

`Input parameter: self`

This function will take the cache file returned from the `writeFrame()` function and turn the data inside into an image file using `ImageTk.PhotoImage()`, then assign this image to the `label` attribute of the Client class.

2.1.8 connectToServer

`Input parameter: self`

This function will create a TCP socket with the Server using the address and port in `self.serverAddr` and `self.serverPort` to exchange RTSP requests.

2.1.9 sendRtspRequest

`Input parameter: self, requestCode`

This function will send a RTSP request depends on the `requestCode` given in the parameter. Before sending the code, it must check the state of the Client to maintain the state consistency of the structure.

- `requestCode == SETUP and state == INIT`: The function will create a new thread to "listen" for RTSP replies from the server. Then it will increase the value of `rtspSeq` by 1 and keep track of the sent request by updating the `requestSent` to SETUP. Finally it will generate the RTSP request and send it to the Server.
- `requestCode == PLAY and state == READY`: The function will increase the value of `rtspSeq` by 1 and keep track of the sent request by updating the `requestSent` to PLAY. Finally it will generate the RTSP request and send it to the Server.
- `requestCode == PAUSE and state == PLAYING`: The function will increase the value of `rtspSeq` by 1 and keep track of the sent request by updating the `requestSent` to PAUSE. Finally it will generate the RTSP request and send it to the Server.
- `requestCode == STOP and not state == READY`: The function will increase the value of `rtspSeq` by 1 and keep track of the sent request by updating the `requestSent` to STOP. Finally it will generate the RTSP request and send it to the Server.
- `requestCode == TEARDOWN and not state == INIT`: The function will increase the value of `rtspSeq` by 1 and keep track of the sent request by updating the `requestSent` to TEARDOWN. Finally it will generate the RTSP request and send it to the Server.

2.1.10 recvRtspReply

`Input parameter: self`

This function will wait for the RTSP reply from the Server and call `parseRtspReply` to decode the received data. If the Client send a TEARDOWN request, the function will shut down and close the RTP socket.

2.1.11 parseRtspReply

`Input parameter: self, data`

This function will decode the message received from the server. It will only proceed if the sequence number of the RTSP message between the Server and Client are matched. Depending on the type of RTSP request, the function will proceed accordingly.

2.1.12 openRtpPort

`Input parameter: self`

This function will open a RTP port to listen for UDP connection from the server.

2.1.13 handler

`Input parameter: self`

This function will pause the movie and pop up a window to confirm if the user really want to exit. If they do, then we will run the `exitClient` function. If not then we will continue to play the movie.

2.2 RTP Packet

2.2.1 encode

`Input parameter: self, version, padding, extension, cc, seqnum, marker, pt, ssrc, payload`

This function will encode the RTP packet with header fields and payload. The header (12 BYTES) will be the combination of **version, padding, extension, cc, seqnum, marker, pt, ssrc**. The first 2 bytes is shared for `version(2 bits)`, `padding(1 bits)`, `extension(1 bit)`, `contributing source (4 bits)`, `maker(1 bit)`, `payload type(7 bits)`. The next 6 bytes is for `sequence number(2 bytes)` and `timestamp(4 bytes)`. The last 4 bytes id for `synchronization source identifier`.

2.2.2 decode

`Input parameter: self, byteStream`

When receiving the RTP packet from server, this function will take it as a `byteStream` (in array type) and decode into header and payload. The first HEADER_SIZE cell of the array is for the header, the rest is for the payload.

2.2.3 version

`Input parameter: self`

This function return RTP version by shifting the first element in header to right 6 bits.

2.2.4 seqNum

`Input parameter: self`

This function return sequence (frame) number by shifting the header[2] to left 8 bits or returning the header[3].

2.2.5 timestamp

`Input parameter: self`

This function return timestamp by shifting the header[4] to left 24 bits or shifting the header[5] to left 16 bits or shifting the header[6] to left 8 bits or returning the header[7].

2.2.6 payloadType

`Input parameter: self`

This function return payload type by taking the header[1] or number 127.

2.2.7 getPayload

`Input parameter: self`

This function return payload.

2.2.8 getPacket

`Input parameter: self`

This function return RTP packet by merging header and payload.

3 List of components

3.0.1 Client

This component belongs to the client, it is responsible for providing GUI, encoding and sending requests to the server and receiving packets from the server and presenting.

3.0.2 ClientLauncher

This component belongs to the client, it acts as a wrapper of the client class when it receives processing input code and passes parameters to client functions.

3.0.3 Server

This component belongs to the server, it takes the processing input code and passes the number to the Serverwoker's functions as its wrapper.

3.0.4 Serverworker

This component belongs to the server, it receives requests from the client and sends resources by encoding in RTP Packet.

3.0.5 Videostream

This component belongs to the server, it receives requests from the server and processes the video parameters to sync with the system and return data accordingly.

4 Model and data flow

When you run the client, it will open the RTSP socket to the server that will establish a communication link between them. We can change state of client by using the window's button that we implement. By clicking those buttons, it will use RTSP socket for sending all RTSP requests.

In the server side, when link between successfully established, it will be able to listen any requests from the client send to. In real life, when someone have a conversation with you, you need to reply so that they know you're listening to their story, so does the server. After the client send the server a request, it need to reply so that the client know their request is listening. The server always replies to all the messages that the client sends. The code 200 means that the request was successful while the codes 404 and 500 represent $\mathrm{FILE}_N OT_F OU N Derrorandconnectionerror(CON_E RR) respectively$

In the Server.py, we just have a simple function. First it will check whether we enter the command correct or not. If correct, it will create a RTSP Socket and listen to any devices that connect to the server with the same port. When there a connection, the SeverWorker from ServerWorker.py will receive the client info and do it job

Same with Server.py, the ClientLauncher.py will check whether you enter correct it or not by its following form. The program will create a new app window and init a new client through Client.py and continuosly run untill it being shutdown
The purpose of the project is to send data from sever to client, specific is video file.

When the server receives the PLAY-request from the client, the server reads one video frame from the file and creates an RtpPacket-object which is the RTP-encapsulation of the video frame. It then sends the frame to the client over UDP every 50 milliseconds. For the encapsulation, the server calls the encode function of the RtpPacket class

Because we have no other contributing sources (field CC == 0), the CSRC-field does not exist. The length of the packet header is therefore 12 bytes, or the first three lines from the diagram below

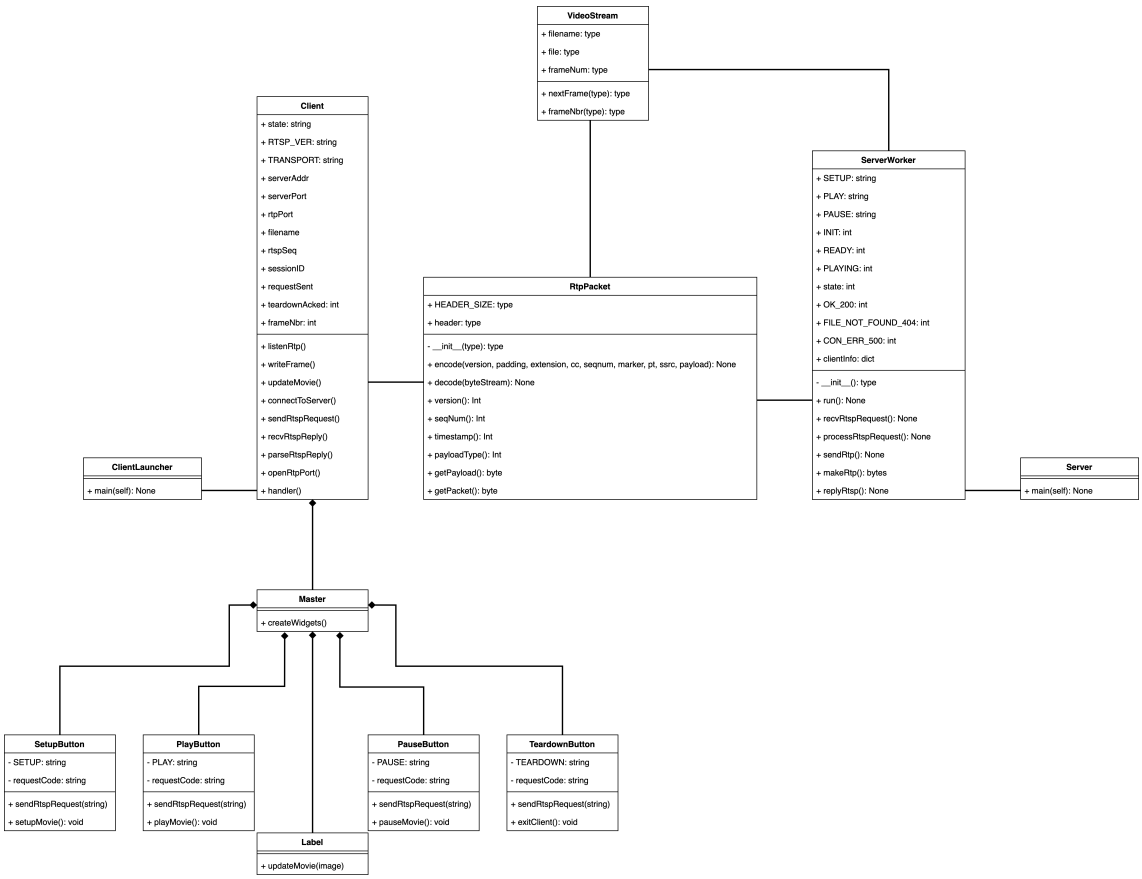
Every header have a maximum capacity at 8 bits and also the data that are pass from the parameter of encode function. In order to store the suitable data for the header, we have to shift bits and override them in the same header

TP-version field (V): 2 bits
Following the diagram, we have padding (P), extension (X), number of contributing sources (CC), and marker (M) fields (1 bit, 1 bit, 4 bit and 1 bit respectively)
Set padding (P), extension (X), number of contributing sources (CC), and marker (M) fields. These are all set to zero in this lab.
The Payload type field (PT) contain 7 bits. This field identifies the format of the RTP payload and determines its interpretation by the application

The timestamp using the Python's time module to reflects the sampling instant of the first octet in the RTP packet. This samping must be derived from a clock that increment motonically and lineary in time to allow synchronization and jitter calculation Set the source identifier (SSRC). This field identifies the server. You can pick any integer value you like.

Video stream is another simple program to help us specific the video file, read next frame and return the frame number.

5 Class diagram



6 Impementation

At first 2 programs will have states (INIT, READY, PLAYING) and actions (SETUP, PLAY, PAUSE, TEARDOWN) while ServerWorker have extend reply status code.

After the client select a command, the server will reponse to that commad by the status code. If it correct, the client will chage the state follow by the server's reply "SETUP" command from the client will include:

- SETUP command
- Video file name to be play Packet Sequence Number starts from 1
- Protocol type: RTSP/1.0 RTP
- Transmission Protocol: UDP
- RTP Port for video stream transmission

When the server receives SETUP command that was send through RTSP socket, it will:

- Check if there is a file match the file name then replying to the client
- Generate a random session number

- Get info from the client's message
- From step 1, if success, the server will open the video specified in the SETUP command and initalize its video frame to number 0
- Reply to the client through RTSP Socket and set STATE to READY
- Reply to the client through RTSP Socket and set STATE to READY

After that, the client will loop to receive the reply from server.

Then the client begins its action in the loop by function parseRtspReply. It will continuosly parse the server's reply to a specific process. After SETUP, the STATE is now READY. We call function openRtpPort() to receive incoming video from server.

Afterward, if PLAY command was sent from client to server. It will repeat the action:

- Update RTSP sequence number
- Send RTSP request through RTSP socket

The server will then create a RTP Socket and start a thread to send video.

The function will continuosly send video's frame that's was chop by VideoStream.py in to many frames. It will send all the frames one by one in order to the client through RTP socket.
In the client side, after the command PLAY was sent, it will open a thread with target listenRtp corresponding to functionRtp from server.

The writeFrame function help us create a temporary image file and override it when ever there a new frame arrives.
The updateMovie function read the image that contain in the file that wrtiteFrame create. With the loop in the listenRtp, we have a video that combined from many pictures.
Now the client's STATE is at PLAYING, we can choose to send PAUSE or TEARDOWN command to server.

When you send PAUSE command, it will repeat smiliar action like PLAY.

At this it will behave a little different, the clientInfo['event'] call set() method, it affect the sendRtp function. If the clientInfo['event'] is Set.

The STATE is now change to READY and we can re-send PLAY command again, at this the curFrameNbr and client's frameNbr still doesn't change, so it can continue run where it stop.
Whether its STATE are READY or PLAYING, we can send to server a TEARDOWN request.
At first, it will send a message to server as usual.
But now it will close the UI window and shut down everything. Remember the temporary image file that we create, now we will have it remove because it's not necessary any more. After that we calculate the loss rate and close all the threads.

7 Summative evaluation of achieved results

- The program can work smoothly, and it also generate a friendly user interface that user can easily use and have the best experience.
- Multiple user can connect to the Server at the same time. Moreover, the Server can handle many request concurently.

- By measuring the elapsed time between current and last frame, we can improve the performance of the stream.
- Extension 1: We can display many information of the RTP Packet on it but still keep the GUI as user friendly as expected.
- Extension 2: We successfully merge the SETUP and PLAY button into 1.
- Extension 3: Client can send DESCRIBE request to server and receive corresponding information.

8 User manual

The program will have 5 files and 1 example file (movie.Mjpeg):

- Client.py
- ClientLauncher.py
- RtpPacket.py
- Server.py
- ServerWorker.py
- VideoStream.py

To run the program, we will follow these steps:

- Run Server.py

  Open the terminal and enter following command:
  `python Server.py <server_port>`
  where <server_port> represents for port that client can listen to

  - we can give it the value 1025
  - Standard RTSP port is 554
  - In this project we should make the value bigger than 1024

- Run ClientLauncher.py
  Open another terminal and enter following command:

  ```
  python CientLauncher.py <server_address> <server_port> <client_port> <movie_file>
  ```

- How RTSP and RTP work together

By now you can see the program window pop-up. So how the client and server communicate and transfer data to each other

`server_address, server_port`:allows client to connect to server.
`client_port`: provide client a port to listen to RTP Packet.
`movie_file`: movie file expected to display.
`SET UP`: connect to server to send request to display movie.
`PLAY`: receive packets (download movie) then display.
`PAUSE`: hold the displayed movie for a while until client presses PLAY again.
`TEARDOWN`: turn off all connections and shut down

9 Extend (All included in source code)

9.1 Calculate statistics

- Packet loss rate

  To calculate the RTP packet loss rate at any given time, we will take the total number of legitimate packets received and divide it by the current frame number at that specific time:

$$packet\ loss\ rate = \frac{\sum legitimate\ received\ packets}{current\ frame\ number} \tag{1}$$

  A legitimate packet is a packet that has the sequence number higher than our current frame number. Because we are using UDP, any late packet will be discarded and count as a lost packet. For example, if our current frame number is 2 (meaning the latest packet we received is packet number 2) and receive a packet which has sequence number 6, we will accept and display the image frame inside this packet and treat packet number 3, 4, 5 as lost (late) packet. We show this rate every time the user presses PAUSE, STOP or TEARDOWN (close the window).

- Video data rate

  To calculate video data rate at any given time, we will take the total of RTP packet bits received from the Server and divide it by the total of time it takes to receive the data.

$$video\ data\ rate\ (bits\ per\ second) = \frac{\sum RTP\ packet\ bits}{time\ taken\ to\ receive\ data} \tag{2}$$

  In our application, we put the `rtpSocket.recvfrom` function between 2 timers to measure the time neeeded to download RTP packet from the Server, as well as keeping track of the total size of received packets in a variable. We will print out the division of these two variable every time the user presses PAUSE, STOP or TEARDOWN (close the window).

9.2 Change to standard interface

Firstly, to solve the mandatory SETUP phase we made it run automatically without needed user's interaction. This means if user launches client interface, a SETUP message will immediately be signaled to server to prepare the requested movie and all essential connections.

We change the TEARDOWN to STOP button in the client interface. However, if we kept the original function of TEARDOWN message, the RTP connection would be closed and in turn, we don't have the SETUP option to reconnect it. In conclusion, we create a STOP message simulating TEARDOWN message but STOP handler will keep socket connections alive and restart movie. After client receives a response for STOP, client will reset framNbr to 0 and listen all RTP packets again as new. The close window button keeps the original TEARDOWN function.

9.3 DESCRIBE interaction

Same as other message, we create a simple DESCRIBE request to be sent to server when user clicks DESCRIBE button on the interface. To prevent a session description file from congesting the RTP packets, a new port number is embedded in the request and server will connect to that port and send description over the socket. Client and server create separate threads to load description file independent of other processes.

```
LISTENING...
CURRENT SEQUENCE NUMBER: 494
LISTENING...
CURRENT SEQUENCE NUMBER: 495
LISTENING...
CURRENT SEQUENCE NUMBER: 496
LISTENING...
CURRENT SEQUENCE NUMBER: 497
LISTENING...
CURRENT SEQUENCE NUMBER: 498
LISTENING...
CURRENT SEQUENCE NUMBER: 499
LISTENING...
CURRENT SEQUENCE NUMBER: 500

Data Sent:
STOP movie.Mjpeg RTSP/1.0
CSeq: 3
Session: 724437
Packet loss: 0
Packet total: 500
Packet loss rate: 0.0
Video data rate: 161469.05646710898 bits per second
```

Figure 1: Example of packet loss rate and video data rate

References

[1] Kurose, James F (2005). *Computer networking: A top-down approach featuring the internet, 3/E*. Pearson Education India.

[2] H. Schulzrinne, Columbia U., A. Rao, R. Lanphier, and M. Stiemerling, Ed. (1998). *Real Time Streaming Protocol (RTSP)*. URL: https://tools.ietf.org/html/rfc2326.

[3] M. Handley, V. Jacobson, C. Perkins (2006). *SDP: Session Description Protocol*. URL: https://tools.ietf.org/html/rfc4566.

[4] Jonathan Virga (2020). *Wiki Python: UDP Communication*. URL: https://wiki.python.org/moin/UdpCommunication.