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

Computer Network

REPORT ASSIGNMENT 1

INSTRUCTOR: NGUYEN LE DUY LAI

STUDENTS: Huỳnh Nhật Quang – 1952112

Trần Kim Tùng – 1953087

Bùi Vĩnh Trung - 1953046

Ho Chi Minh City, April

2022

Table of Contents

1. INTRODUCTION

I. Objective

- We'll build a streaming video server and client that communicate over the Real-Time Streaming Protocol (RTSP) and send data via the Real-Time Transfer Protocol (RTTP) (RTP). The Real Time Streaming Protocol (RTSP) is a network control protocol for controlling streaming media servers in entertainment and communications systems. The protocol is used to create and manage media sessions between two endpoints. Clients of media servers employ VHS-style commands like play, record, and pause to manage media streaming from the server to a client (Video on Demand) or from a client to the server in real time (Voice Recording).

- The Real-time Transport Protocol (RTP) is a network protocol that allows audio and video to be sent over IP networks in real time. RTP is used in telephony, video conferencing applications, including WebRTC, television services, and web-based push-to-talk capabilities, as well as other communication and entertainment systems that utilise streaming media.

- RTP is usually sent via the User Datagram Protocol (UDP). The RTP Control Protocol is used in conjunction with RTP (RTCP). While RTP is responsible for transporting media streams (such as audio and video), RTCP is responsible for monitoring transmission statistics and quality of service (QoS), as well as assisting in the synchronization of multiple streams. RTP is one of the technological underpinnings of Voice over IP, and it is frequently used in this context in conjunction with a signaling protocol such as the Session Initiation Protocol (SIP), which establishes connections across the network.

II. Overview

- Overall, we implement the RTSP protocol in the server, the RTP de-packetization in the client, and takes care of simply displaying the transmitted video. Files need to be implemented are the RTSP protocol in the client (named as Client.py), the RTP

packetization in the server (named as RtpPacket.py), the customized interactive player for displaying the transmitted video.

2. REQUIREMENTS ANALYSIS

I. Functional requirements

a) System requirements

- The system can stream video.
- The system can interact with the Client through the RTSP/RTP.

b) User requirements

- Users can connect to the server through the terminal.
- Users can set up, play, pause or end the video from the server.

II. Non-functional requirements

- The video format can be .Mjpeg or .mp4.
- The response time from server is smaller or equal 0.5 (s).

3. FUNCTION DESCRIPTION

Class name	Function	Description
Server worker	_init_(self,ClientInfo)	Constructor
	run(self)	Run the server
	recvRtspRequest(self)	Receive RTSP request
	processRtpRequest(self,data)	Process the RTP request
	sendRtp(self)	Send RTP packets over UDP
	makeRtp(self,payload,frameNbr)	RPT-packetize the video data
	replyRtsp(self,code,seq)	Send RTSP reply to Client
Server	main(self)	Main function to run the program
Video stream	_init_(self,filename)	Constructor
	nextFrame(self)	Get next frame
	getPosFrame(self)	Get position frame
	getWholeVideo(self)	Get frame of whole video
	calNumFrames(self)	Get total number of frames
	calFps(self)	Get frame per second
	calTotalTime(self)	Get total time of video
	reset_frame(self)	Reset frame
	frameNbr(self)	Get frame number
Client	__init__(self, master, serveraddr, serverport, rtpport, filename)	Constructor
	createWidgets(self)	Build GUI
	setupMovie(self)	Setup button handler
	exitClient(self):	Exit button handler
	teardownMovie(self)	Stop button handler
	pauseMovie(self)	Pause button handler
	describeInfo(self)	Describe butoon handler
	backwardVideo(self)	Backward button handler
	forwardVideo(self)	Forward button handler
	playMovie(self)	Play button handler
	listenRtp(self):	Listen for RTP packets and analysis somethings
	writeFrame(self, data)	Write the received frame to a temp image file
	updateMovie(self, imageFile):	Update the image file as the video frame in the GUI
	connectToServer(self):	Connect to the Server. Start a

		new RTSP/TCP session
	sendRtspRequest(self, requestCode)	Send RTSP request to the server
	recvRtspReply(self):	Receive RTSP reply from the server
	parseRtspReply(self, data)	Parse the RTSP reply from the server
	openRtpPort(self)	Open RTP socket binder to a specified port
	handler(self)	Handler on explicitly closingthe GUI window
RtpPacket	init (self)	Constructor
	encode(self, version, padding, extension, cc,seqnum, marker, pt, ssrc, payload) decode(self, byteStream)	Encode the RTP packet with header fields and payload Decode the RTP packet
	version(self)	Return RTP version
	seqNum(self)	Return sequence (frame) number
	timestamp(self)	Return timestamp
	payloadType(self)	Return payload type
	getPayload(self)	Return payload
	getPacket(self)	Return RTP packet

Figure 1: Function description

4. LIST OF COMPONENTS

- The Server.
- The Client.
- Buttons: PLAY, PAUSE, STOP, DESCRIBE, FORWARD, BACKWARD, Exit
- The video for streaming.
- The GUI of client launcher.

5. MODEL AND DATA FLOW

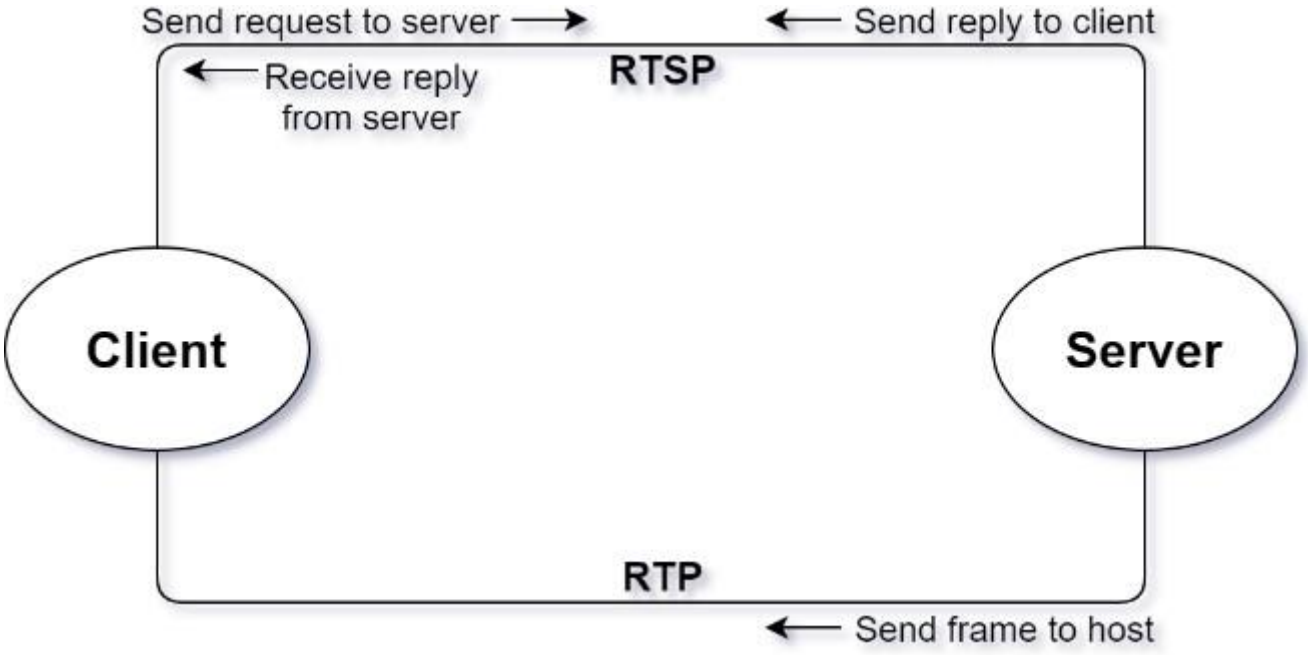

Figure 2: Data flow of video streaming system

6. CLASS DIAGRAM

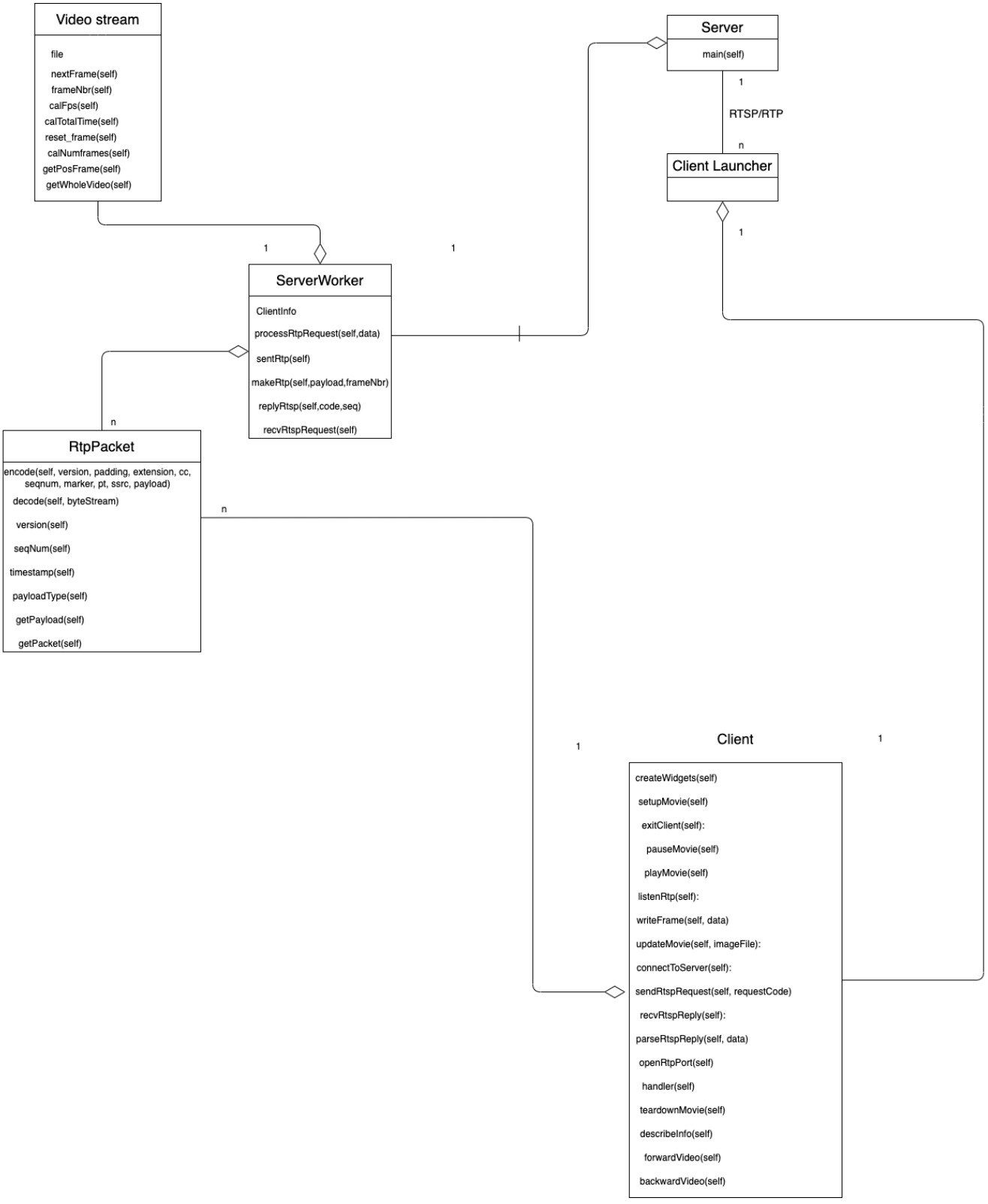

Figure 3: Class diagram

7. IMPLEMENTATION

- We implement and develop more buttons for users in Client.py. We have buttons for STOP, DESCRIBE, FORWARD, and BACKWARD, as well as a button that combines the SET UP and PLAY functions.
- We'll encode and decode certain parameters in RtpPAcket.py, such as sequence number, RTP version, and timestamp.
- We add more functions to VideoStream.py to retrieve the frames from the video, such as calFps(self), calTotalTime(self), calNumFrames(self), getPosFrame(self), and so on.

8. ACHIEVED RESULTS

- Create a streaming video server and client that use the Real-Time Streaming Protocol (RTSP) to communicate and the Real-Time Transfer Protocol to deliver data (RTP).
- The program runs smoothly and generates a user-friendly user interface that is simple to use and leaves a positive impression.
- Extension 1: The SETUP and PLAY buttons were successfully combined into one button.
- Extension 2: The DESCRIBE button allows clients to send requests to the server and get responses.
- Extension 3: The TEARDOWN button is replaced with an Exit button, and a Halt button is added to stop the video, and pressing the PLAY button starts the video from the beginning.
- Extension 4: Two new buttons have been added: FORWARD and BACKWARD.

9. USER MANUAL

- Step 1: Open your terminal in the folder that contains file Server.py. Next, issue this command line: python Server.py <<server_port>> where <<server_port>> is the port your server listens to for incoming RTSP connections. We need to choose a port numbergreater than 1024 for the <<server_port>>.
- Step 2: Create a new terminal in the location where the file ClientLauncher.py is located (we consider this as a Client). Then type the following command line: python ClientLauncher.py <<server_host>><<server_port>> <<RTP_port>> <<video_file>> where <<server_host>> is the name of the machine where the server is running, <<server_port>> is the port where the server is listening on, <<RTP_port>> is the port where the RTP packets are received, and <<video_file>> is the name of the video file you want to request.
- Step 3: The client establishes a connection with the server and displays a window similar to this.:

  By hitting the buttons, you can send RTSP commands to the server: PLAY to play the video, PAUSE to pause during playback, STOP to end the session, and PLAY to restart the film from the beginning. The Ahead and BACKWARD buttons allow users to change the video by one second forward or backward. The DESCRIBE button displays information such as the current frame of video or the total duration of the movie. The Exit button will close the connection and end the session.

Figure 4: GUI for users

10. EXTEND

- **Calculate packet loss rate:**
  To determine the RTP packet loss rate at any particular time, multiply the total number of legal packets received by the current frame number at that time:

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

A valid packet has a sequence number that is greater than our current frame number. Any late packet will be deleted and counted as a lost packet because we are using UDP. For example, if our current frame number is 2 (which means the most recent packet we received was packet number 2) and we get a packet with sequence number 6, we will accept and display the picture frame contained inside this packet while treating packet numbers 3, 4, and 5 as lost (late) packets. When the user presses PAUSE, STOP, or EXIT, we display this rate (close the window).

11. RESOURCE CODE

- Here is the link for the source code of our group:

https://github.com/trungbuivinh/ComputerNetWork-Assignment.git