CSC3002F Networks Assignment Report

March 2022

Team Members

Luke Bowles (BWLLUK001)
Zenan Shang (SHNZEN001)
Matthew Scott (SCTMAT003)

GitHub: https://github.com/lbowles/Python-UDP-Chat-Group

ABSTRACT

This report will detail a proprietary network application that uses UDP at its transport layer. The application is using the client-server architecture that allows users to interact over the internet in real time. The application will allow multiple clients to connect to the same server and all clients can view and send messages in real time. User Datagram Protocol (UDP) is a communication protocol used to allow time-sensitive transmission between clients. UDP allows fast data transmission, but due to the unreliable nature of UDP connections, there may be data loss. Transmission Control Protocol (TCP) is a communication standard that enables reliable communication between clients. This is generally used for applications that do not require time-sensitive communication.

For this project we were tasked to use UDP to develop our group chat application. As such measures needed to be taken to help with the reliability and accuracy of message data. We implemented a system in which users would be notified should their messages reach the server in order to help combat loss of data.

FEATURES

Group Communication

The application allows for messages to be sent to a number of clients through a common server. It is specifically designed for group communication as it is impossible to send messages to a specific individual on the server.

Real-Time Communication

Users can send and receive messages in real time through our client-server application. The time it takes for messages to go through is negligible.

Text-Based Messaging

Our application uses text-based messaging (ASCII) as per the project specifications. This provides human readable messages.

CLI

We use a command-line interface for our project. While less attractive looking compared to a GUI it is functional and easy to read and interact with.

Historical Messages

When a client joins the server their entire group chat history can be viewed.

Missed Messages

Once clients log back into the group chat, they can view the messages they have missed while being offline. Messages are stored and are specific to an individual client, so only that client will receive the messages they missed when they return.

Message Confirmation

Server will send a confirmation message back to the client when a message is received. This adds an extra bit of reliability, though it still does not guarantee that the message was received by the other clients without errors.

Message Information

The recipient of a message will be able to see the time when the message was sent next to the username of the sender.

Instructions

Upon connecting to the server the client gets basic instructions on how to use the application (like how to close the connection by typing “Exit”).

Usernames

Upon connecting to the server the users are prompted to enter a username. The username will be seen by recipients of messages sent by the user, thus allowing for easier readability and usability. If a username is left blank the user is assigned a guest username consisting of GuestUser + a random number from 1 to 1000. The user will then be informed of their username.

Displays User List

The IPs of connected clients is shown and updated when a client exits or enters. This allows users to know who is connected.

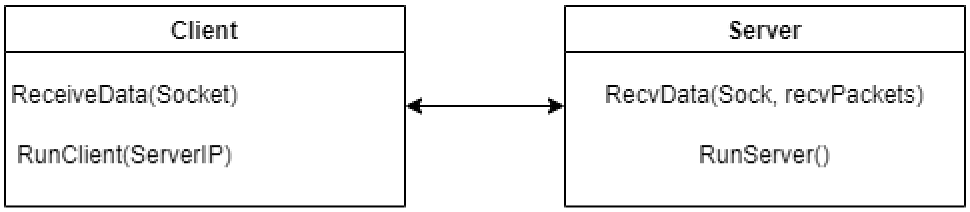
Password and Encryption

All messages sent over the UDP connection will be encrypted. When a server is started a password needs to be created, the client needs to enter the same password to connect to the server. This password is used as a key which is then used to encrypt and decrypt the messages. This password method enables each created chat server to have its own unique encryption security.

When a client sends a message to the server, the message is encrypted, then the server decrypts and reads the message. The server also uses the same encryption method when sending messages back to the clients, the clients receive the message then decrypts and displays it. This adds security to our application, as external viewers on the network that potentially read what data is being sent over our UDP connection won't be able to interpret the actual message due to its encryption.

For example with a generated key, 'Hello' is converted to: '2Duv/A==*vWiLZNJln0+UA3nUN4ZETg==*wUkC0ajWmJrjgGF9opP8vA==*YSdLxschRMuFuq4DGEA5Tg==' before getting encoded and sent over the UDP connection.

Class Diagram



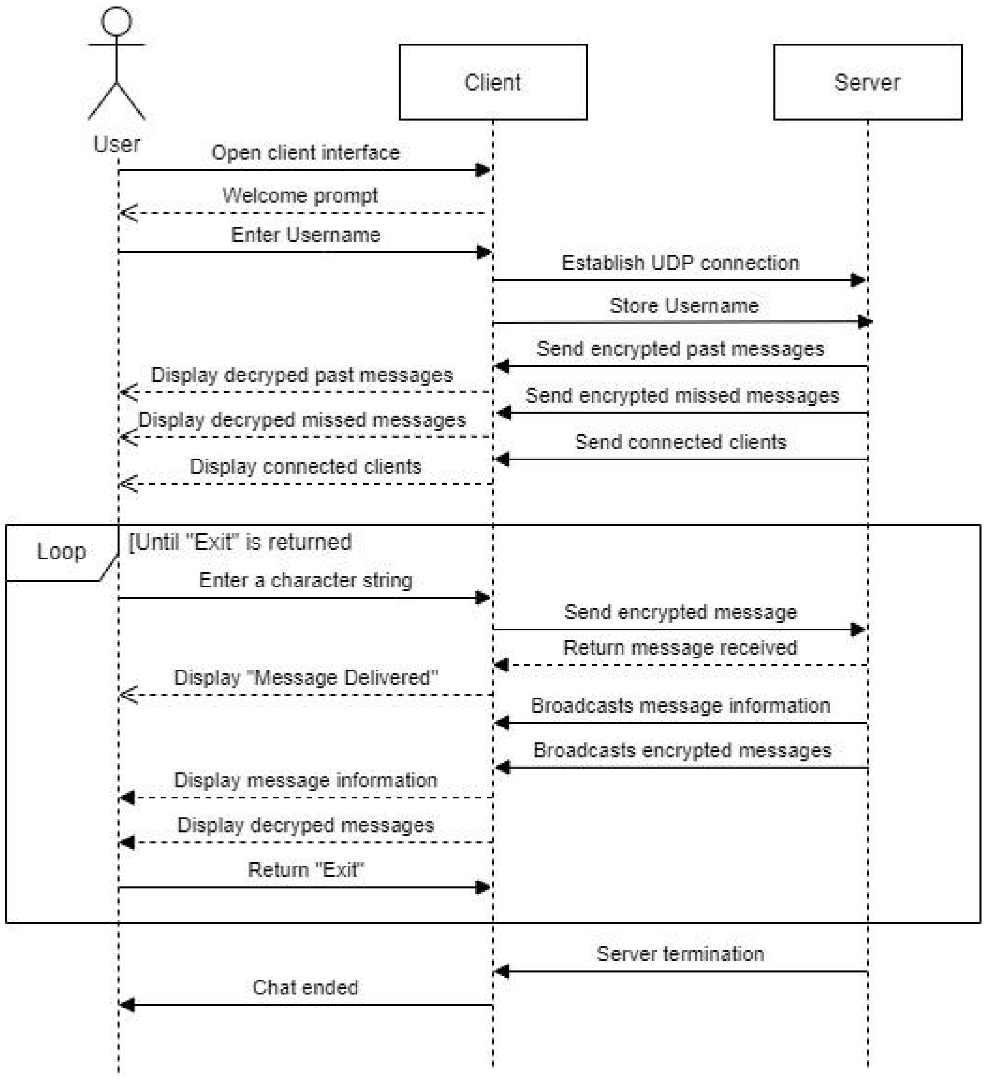
RunClient(ServerIP): This is the method that is used to create the client program. First it will be getting the client connection information, then server connection information, getting the username of the client, reading client messages, encrypting the messages and sending them to the server.

ReceiveData(Socket): This is used to listen to the incoming messages from the server

RecvData(Sock, recvPackets): establish UDP Connection, this will be used to get incoming client data

RunServer(): This is the method that will be used to create a server. First, it will require the user to enter some information needed to create the server, then it will create new threads for every client connection, each new thread will be constantly checking for any incoming client messages. All the incoming messages from the client will be stored in the server and send the past/missed messages to the connected clients.

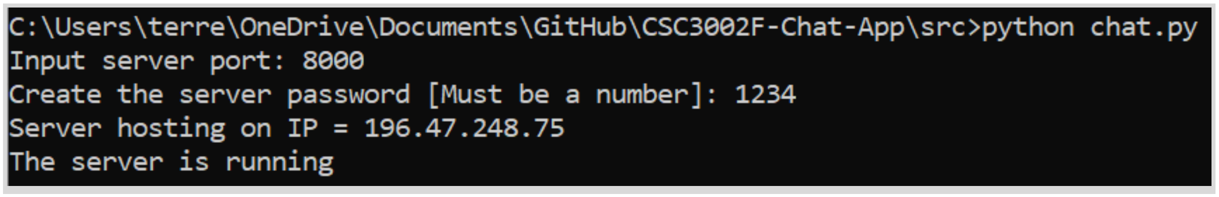
Sequence Diagram


User
Client
Server
Open client interface
Welcome prompt
Enter Username
Establish UDP connection
Store Username
Send encrypted past messages
Display decryped past messages
Send encrypted missed messages
Display decryped missed messages
Send connected clients
Display connected clients
Loop
[Until "Exit" is returned
Enter a character string
Send encrypted message
Return message received
Display "Message Delivered"
Broadcasts message information
Broadcasts encrypted messages
Display message information
Display decryped messages
Return "Exit"
Server termination
Chat ended

This sequence diagram shows us the user-client interaction and client-server interaction. There are 3 objects in this diagram: User, Client and Server. User is the person using the application, client is the application that the user interacts with, server is the program that the client interacts with. The solid line represents synchronous calls and dotted lines represent reply messages. The loop in the diagram indicates that the client will continuously “listen” for user input until the user enters “Exit” as an input. Server terminates when the user closes the command prompt that is running the server. All clients won’t be able to communicate if the server is terminated.

Screenshots

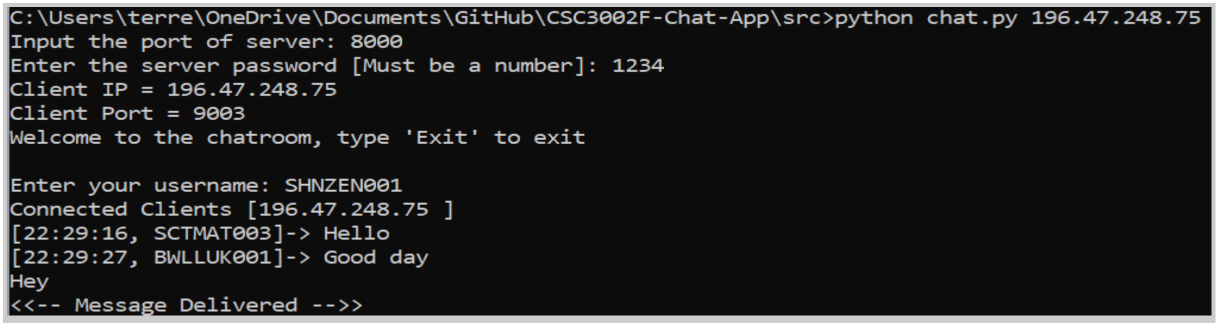
When establishing the server, users will have to follow protocol specification 1 to 3. After those steps, the server will display its IP address so that the user can copy and paste the IP address to create a client application. The server will then “The server is running” to show that the server is ready for clients to establish a connection.



```
C:\Users\terre\OneDrive\Documents\GitHub\CSC3002F-Chat-App\src>python chat.py
Input server port: 8000
Create the server password [Must be a number]: 1234
Server hosting on IP = 196.47.248.75
The server is running
```

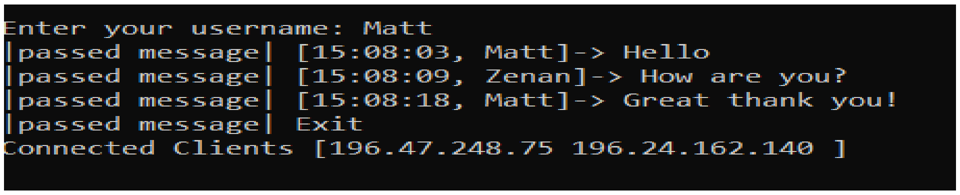
When the user wants to create a client application, they will have to follow protocol specification 4 to 6. After entering the commands, the server will send a welcome message to the client if the connection is successful. It will ask you for an username and then will display all the connected clients. Thereafter, the user can communicate with other users through the application.

This is how the chat room will look like when clients have connected to the server. Display all the connected client IP addresses. When messages are exchanged, the time and username will be displayed once the messages have been received. A message send confirmation will be displayed when the user has sent a message in the app.



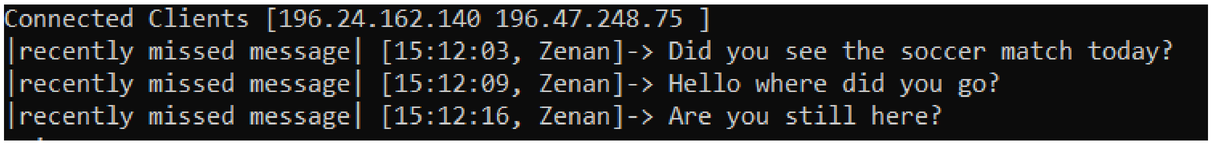
```
C:\Users\terre\OneDrive\Documents\GitHub\CSC3002F-Chat-App\src>python chat.py 196.47.248.75
Input the port of server: 8000
Enter the server password [Must be a number]: 1234
Client IP = 196.47.248.75
Client Port = 9003
Welcome to the chatroom, type 'Exit' to exit

Enter your username: SHNZEN001
Connected Clients [196.47.248.75 ]
[22:29:16, SCTMAT003]-> Hello
[22:29:27, BWLLUK001]-> Good day
Hey
<<-- Message Delivered -->>
```

Here is an example of historical messages when a client joins the chat.



```
Enter your username: Matt
|passed message| [15:08:03, Matt]-> Hello
|passed message| [15:08:09, Zenan]-> How are you?
|passed message| [15:08:18, Matt]-> Great thank you!
|passed message| Exit
Connected Clients [196.47.248.75 196.24.162.140 ]
```

Here is what a user that missed messages while being disconnected to the chat sees when rejoining the chat.



```
Connected Clients [196.24.162.140 196.47.248.75 ]
|recently missed message| [15:12:03, Zenan]-> Did you see the soccer match today?
|recently missed message| [15:12:09, Zenan]-> Hello where did you go?
|recently missed message| [15:12:16, Zenan]-> Are you still here?
```