PROGRAM REPORT GROUP 50

Ndou Arinao
Ramurafhi Thendo (RMRTHE002)
Ratshilumela Phindulo (RTSPHI002)

INTRODUCTION:

The evolution of communication technologies has facilitated the development of various messaging applications, each designed to cater to specific needs and preferences of users. In this era of instant connectivity, the demand for robust, efficient and secure messaging platforms continues to grow. Thus, the design and implementation of messaging applications necessitates careful consideration of protocols, frameworks and features to ensure seamless communication while addressing concerns such as privacy, reliability and authentication. This report outlines the design and implementation of a peer-to-peer chat application that leverages both UDP (User Datagram Protocol) and TCP (Transmission Control Protocol) sockets. A central server is created to coordinate client connections and facilitate queries for available clients. This report would talk more about the protocol design and specification, detailing the structure of messages, communication rules and state transitions. We will also dive into the implementation of client-server interaction model and command specification. There will also be sequence diagrams which will illustrate the flow of messages and interactions between clients in different states.

Program Features

Server features

1. Address Storing and Querying:

- The server maintains a list of connected clients' addresses. This list is stored in the `samplelist` variable, which is a global list that all threads can access.
- When a client connects to the server, its address is appended to the `samplelist`. This allows the server to keep track of all connected clients.
- Clients can send a query to request the list of connected clients. This is achieved by sending the byte string `b'query_list'` to the server.
- Upon receiving the query, the server responds by sending the current list of connected clients to the requesting client.
- This address storing and querying mechanism allows clients to dynamically retrieve information about other connected clients, enabling features such as online user lists or peer discovery in peer-to-peer networks.
- By maintaining this list centrally on the server, clients can obtain up-to-date information about other clients without needing direct communication with each other.

Client features:

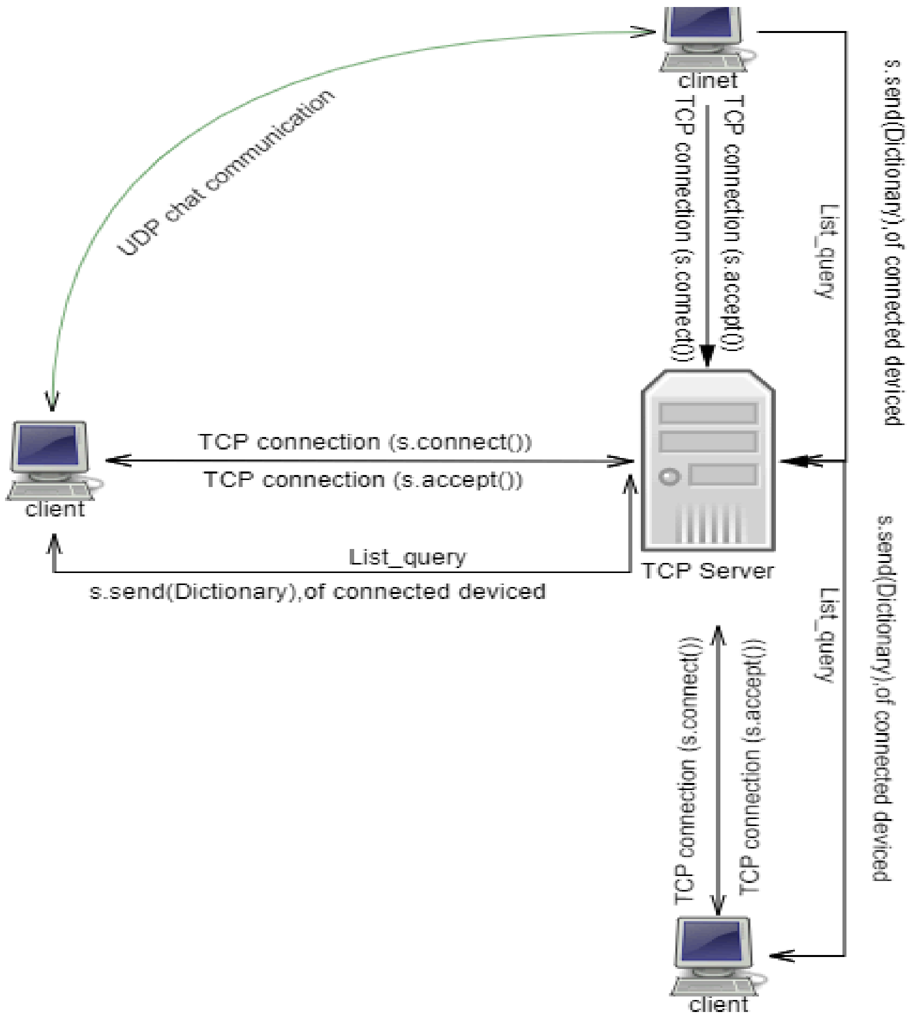
1. User Interaction(phindulo):

- User interaction refers to the process of allowing users to interact with the software application through various input methods such as text input, mouse clicks, or button presses.
- In the code implementation, user interaction is facilitated through the console interface, where the user can input messages to send to the selected device.

- The application continuously prompts the user to input messages, providing a seamless experience for sending and receiving messages.
- Additionally, the application supports a special command (`0`) that allows the user to refresh the list of devices without disconnecting from the server.
- And "$q" to quit talking to other clients and go back to tcp where they can choose a different client they want to talk to
- User interaction enhances the usability of the application by providing an intuitive interface for users to interact with the underlying functionality.

Program Description

- **The program**
  - In simple terms the program is a peer to peer chat application using UDP, The clients would need to connect to a server to see a list of clients that are currently active or connected to the server
  - **Simple program overview diagram :**


clinet
UDP chat communication
TCP connection (s.accept())
TCP connection (s.connect())
List_query
s.send(Dictionary),of connected deviced
client
TCP connection (s.connect())
TCP connection (s.accept())
List_query
s.send(Dictionary),of connected deviced
TCP Server
TCP connection (s.connect())
TCP connection (s.accept())
List_query
s.send(Dictionary),of connected deviced
client

1. Server

- The server is implemented using TCP which will allow clients to be able to connect to it and communicate with it to query some information.

- **Server code implementation(phindulo)**
  - There are 3 main functions implemented in the server code which are :
    1. **send_list_to_client() function**
       - This is the function used to send a list to the client
       - It makes use of a global samplelist
       - The main job of this function is to use pickle inorder to send the list to the client
       - What pickle does is that it serialises the list object then which is to convert it into byte form so that it can transferred over the network.
    2. **handle_client() function**
       - This function is used to handle each client connection, so this function is called after the server has accepted or rejected connection request from a server
       - So it has implementations such as keeping the connection alive until the client disconnected
       - This function also handles the capturing of a device into a list each time a new device is connected to the server
       - It also calls the send_list_to_client function which is used to send the list to the client
    3. **main() function**
       - This is where we accept all the different connections from the different servers, then we implement multi-threading

  - **Multithreading implementation:**
    - Two threads are given two jobs, for example using human analogy, one thread is speaking and the other thread is thinking
      - **In the code implementation we create two threads:**
        1. To listen and accept new connections from new clients, this will be handled by the first three functions which are create_socket(),bind_socket() and accepting_Connections().
        2. The 2nd thread's function is to maintain communication between the server and the connected clients. This means the server will be able to respond to queries such as show all clients and select a client.using the start_turtle() to call all the necessary functions to achieve the desired goal

2. Clients

- **Client code implementation(Phindulo)**
  1. **TCPClient:**

- Manages TCP connection and device selection.Methods: capture_client_info, receive_clients_list, select_device, connect_to_server,connect_to_server`: Establishes TCP connection, selects device, returns info,select_device`: Allows user to choose from connected devices,receive_clients_list`: Receives list of connected devices,capture_client_info`: Captures client's device info.

2. UDPClient:

- Manages UDP connection and message handling.Takes selected device, client device info, and connection flag.Methods: connect_to_device, send_message, receive_messages, check_connection.connect_to_device`: Attempts UDP connection,send_message`: Sends message,receive_messages`: Listens for messages,check_connection`: Prints local and remote endpoints.

3. Main block:

- Creates TCPClient instance, connects to server, selects device. If device selected, creates UDPClient instance.User sends messages.If input is "$q", prompts for new device selection, creates new UDPClient instance, Messages are encoded using message.encode()`.
-

Client code implementation(Arinao)

My code implements simple communication using UDP and TCP protocols, as we can still communicate when the server is off it also utilises a bit of p2p architecture.

TCPClientClass:

the __init__ method initialises the TCPClient object with attributes to store information about connected devices
The receive_clients_list method receives the list of connected device from the server and updates it
The choose_device method allow the user to choose a device they wish to establish a connection with
The connect_to_server method connects to the server, receives a list of all connected devices and allow a user to choose a device to communicate with

UDPClient class:

The __*init*__ method initialises the UDPClient object with attributes to store information about the server, client, and client's socket for communication with the server using UDP.
The *getDevice*_addr method connects to the selected device.
The *send_message* method sends a message to the server.
The *receive_messages* method receives and displays messages from the server.
The *connectionStatus* method checks the connection details, such as local and remote endpoints.

In the **main block**, the TCP client is instantiated, connected to the server, and the user is prompted to enter a username. Then, a UDP client is initialised with the selected device and the user's username. If the user wishes to send the message without their username they can choose an option to be anonymous and the username will be hidden which is a small encryption feature.The program handles refreshing(r) the connection and reinitializing the UDP client if needed. User can choose to stop sending messages and select other devices by pressing q

- **Thendo Client implementation:**
    1. The code defines a Client class responsible for connecting to a server, selecting a device from a list of connected devices, and communicating with the selected device via UDP.
    2. The Client class has methods to capture client information, receive a list of connected devices from the server, select a device from the list, connect to the server, run a UDP client to communicate with the selected device, receive messages from the server, and check the connection details.
    3. The script uses the argparse module to parse command-line arguments for the server's IP address and port number.
    4. When the script is run, it creates an instance of the Client class, connects to the server, and if the connection is successful, runs the UDP client to communicate with the selected device.
    5. The UDP client allows the user to send messages to the selected device and receives messages from the server.

<u>Protocol specification/design</u>

1.Server client protocol

- Tcp connection between the server and the client is established by the client using the connect method to a specific server address that the server is bound to using the bind method. The server uses the accept method to accept incoming connections from the client, the listen method in the server is used to listen for any incoming connection requests from clients.
- **The communication between the server and the client is simple, the process is :**
    1. User connects to the server
    2. The user queries for a list of connected devices(expand)
    3. The server sends the list of connected devices(expand)

(currently the server automatically sends the list of connected devices, so we can just create a prompt from the server side and a decision node on the server side)

Message formats:

- The request from the client is a string and the reply from the server is a string , the message sent from the client has to have a keyword “list ” in-order for the server to be able to send the list of connected devices back to the client. If the data does not include the specified keyword , we have made a loop in the client side to ensure that before the message is sent to the server it contains the required keywords, this is a better way to avoid exception handling on the server side.
- Sending and receiving a string and a list over the network require conversions into bytes that can be sent over the network, for string conversion into bytes, encode method is used by the

sending device to convert the string message into bytes, then the decode method is used by the receiver to convert the byte message into string .

- For a list we use serialisation, so the pickle package is called pickle is a Python module used for serializing and deserializing Python objects. Serialisation is the process of converting a Python object into a byte stream, and deserialization is the reverse process, converting a byte stream back into a Python object

2.Peer to Peer protocol

1.TCPClient Class:

- Utilises the TCP (Transmission Control Protocol) for establishing a reliable, connection-oriented communication channel with the server.
- TCP ensures that data is transmitted in the correct order and that all data is received without errors or loss.
- This involves setting up the TCP connection with the server and fetching the list of connected devices.

2. UDPClient Class:

- Manages communication with other devices using the UDP (User Datagram Protocol) protocol.
- UDP is a connectionless, lightweight protocol suitable for real-time communication where speed and efficiency are prioritised over reliability.
- This relates to sending and receiving messages over UDP, enabling real-time communication with the selected device.

3. Multi-Threading:

- Multi-threading is a programming technique used to achieve concurrency, allowing multiple threads of execution to run concurrently within a process.
- While not directly tied to a specific protocol, multi-threading is crucial for handling asynchronous tasks such as receiving UDP messages while performing other operations.
- It ensures that the client application remains responsive and can handle multiple tasks simultaneously, such as sending messages, receiving messages, and interacting with the user.

4. Dynamic Device Selection:

- This feature involves querying the server for an updated list of connected devices using the TCP protocol.
- It allows the client to dynamically select a device from the updated list obtained from the server, enhancing flexibility and usability.
- So the user had to put in a number that is available from the printed list, if they insert an invalid number or some odd characters this will throw an exception
- TCP is used for sending the query (`b'query_list'`) to the server and receiving an updated list of connected devices in response.

5. User Interaction:

- The client application provides a user-friendly interface for interaction, allowing the user to input messages to send to the selected device.
- While not directly tied to a specific protocol, user interaction is an integral part of the client application's functionality.
- It enhances the usability of the application by providing a seamless experience for sending and receiving messages, along with supporting special commands such as `0` for refreshing the list of devices without disconnecting from the server.
- And other key words are “$q” to quit out of messaging and go back to using tcp so they can choose other clients to connect to.

Message formats:

1. Sending Messages:

- Once the client selects a device to communicate with, it initialises a UDP client for This code implementation consists of two classes: `TCPClient` and `UDPClient`, along with a main block for executing the code. Let's break down what each part does:

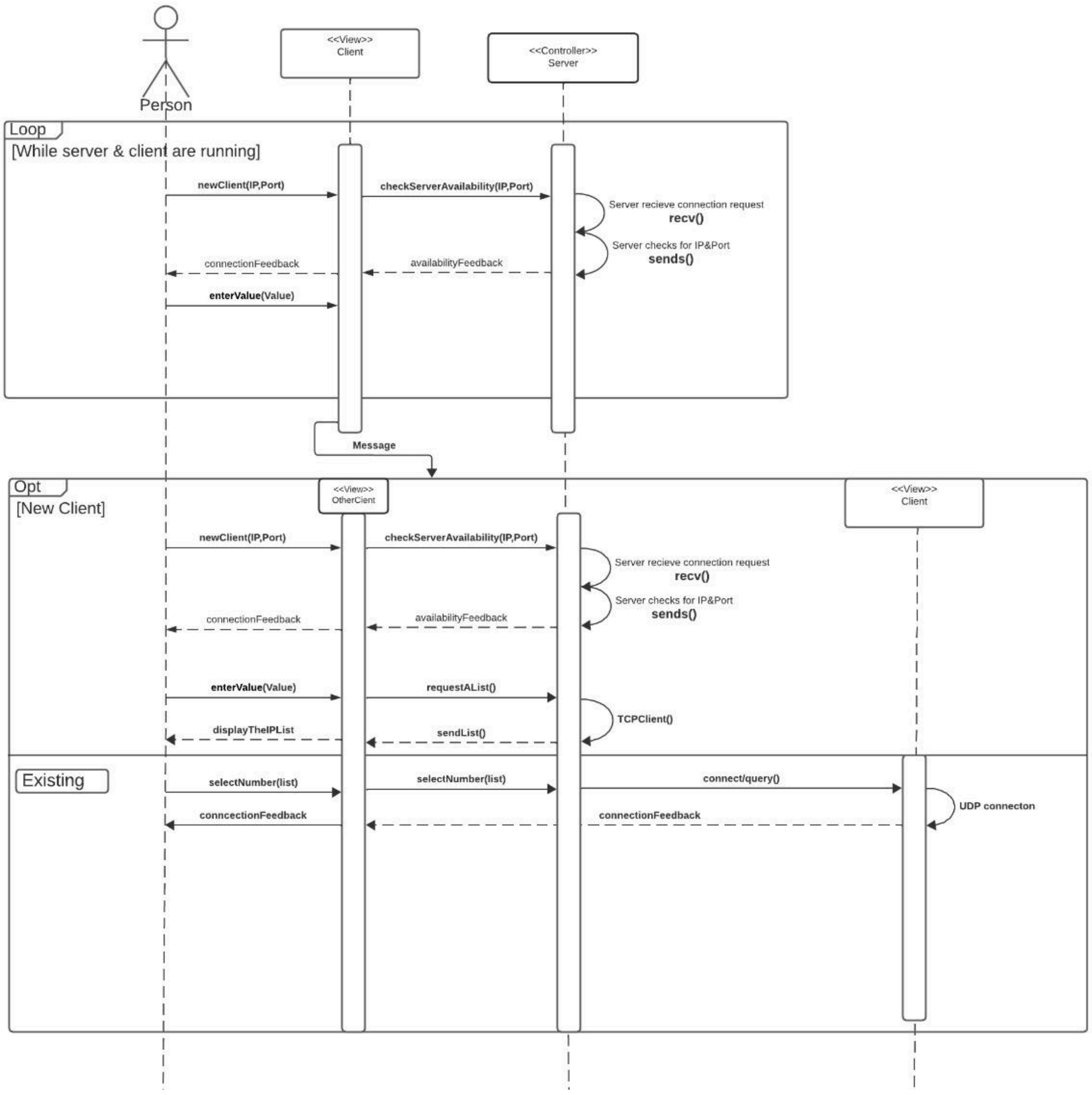
2. Receiving Messages:

- The UDP client continuously listens for incoming messages from the selected device.
- When a message is received, it is decoded from bytes to a string.
- The client prints the received message along with the sender's address.
- Messages received over UDP are likely to be text-based, containing the content sent by the selected device.

Sequence Diagram

Diagram explanation:

The diagram consists of a loop that would be activated when the server and the client code is running. The user would use the client code to initiate the connection with the server which would be based on the IP address and the port number specified in the server code. The server would give feedback to the client which would be visible to the user once the connection have been established. We then have an option part where another client(s) would do the same thing as the previous client(s) in terms of connection and the same would be done by the server, but this new client can request the list of all available clients with will be sent to the server and the server would use TCP connection to connect the clients, thus sending the list to the client that asked of it. The user can now select the number of clients that they want to connect to and the connection would be established via UDP connection as clients have to communicate with UDP.



<<View>> Client
<<Controller>> Server
Person
Loop
[While server & client are running]
newClient(IP,Port)
checkServerAvailability(IP,Port)
Server recieve connection request
recv()
Server checks for IP&Port
sends()
connectionFeedback
availabilityFeedback
enterValue(Value)
Message
Opt
[New Client]
<<View>> OtherCient
<<View>> Client
newClient(IP,Port)
checkServerAvailability(IP,Port)
Server recieve connection request
recv()
Server checks for IP&Port
sends()
connectionFeedback
availabilityFeedback
enterValue(Value)
requestAList()
TCPClient()
displayTheIPList
sendList()
Existing
selectNumber(list)
selectNumber(list)
connect/query()
UDP connecton
conncectionFeedback
connectionFeedback