UNIVERSITY OF CAPE TOWN
IYUNIVESITHI YASEKAPA • UNIVERSITEIT VAN KAAPSTAD

Faculty of Science

Department of Computer Science

Socket Programming Project (UDP)
CSC3002F

Declaration

1. We know that plagiarism is wrong. Plagiarism is to use another's work and pretend that it is one's own.
2. We have used the IEEE convention for citation and referencing. Each contribution to, and quotation in this report from the work(s) of other people has been attributed and has been cited and referenced.
3. This report (including diagrams and code) is our own work.
4. We have not allowed and will not allow anyone to copy our work with the intention of passing it off as their own work.
5. We acknowledge that copying someone else's code, schematics or report, or part of it, is wrong, and declare that this is our own work.

Reekoye Gopal (RKYGOP001)
Tshishivhiri Lufuno (TSHLUF009) 14/04/2021

Signature Date

Introduction

This report is on the development of a networked application. It is required to develop a client-server chat application that makes use of UDP at the transport layer. An application layer protocol needs to be implemented to support the client server architecture. Furthermore, a server must be implemented that is able to manage interactions between multiple clients.

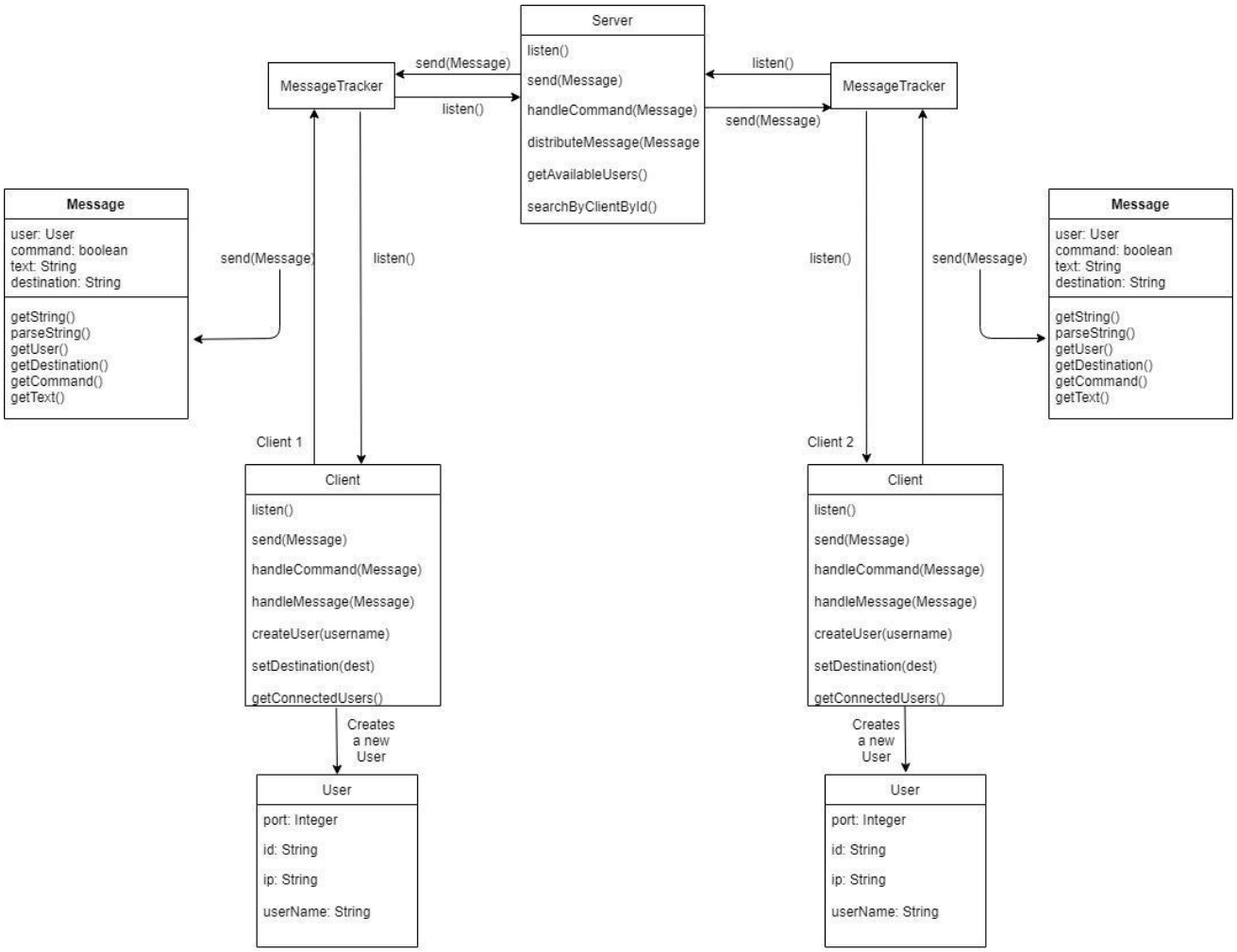
The chat client must allow one or more pairs of users to exchange messages in real time. The server and different clients should be able to run on different hosts. In addition, the application should include the following features:

1. User authentication
2. Group chat
3. Ability to retrieve historical messages.

There are a number of requirements that need to be specified before designing the pattern of communication as well as communication rules. Since the application must make use of UDP which is a connectionless oriented transport layer protocol and provides unreliable service, the application layer protocol must be designed with its own system for attaining reliability.

Protocol Design

There are four classes responsible for the transport layer protocol. These classes are Server, Client, Message, User and MessageTracker. The diagram on the next page shows an overview of the client server architecture.



The **Server** listens and responds to clients' requests. It handles all the messages incoming from different clients. The Server class has the following methods:

- listen() : this method runs on an infinite loop whilst waiting to receive messages from any online clients. This method creates a DatagramPacket to receive message through DatagramSocket of the Server.
- send(Message) : the server makes use of this method when the Server need to send a message to a client. It takes a Message object as argument and checks the meta data of the message to know to which destination to send the message.
- handleCommand(Message) : when the Server receives a command as a Message object through its DatagramSocket, this function reads the command and determine what action to take.
- distributeMessage(Message) : this method handles messages that are not commands. It checks if the received message needs to be passed on to another client or is meant for the Server.

- getAvailableUsers() : this method returns an encoded string of all users who have connected to the Server.
- searchClientById() : this method searches for a specified client among other clients that are online through their unique ID.

The **Client** creates a user and sends messages to the server and to other online clients through the server. The Client class has the following methods:

- run() : this method allocates a DatagramSocket to the client. It also creates a new user associated with this specific client.
- listen(): this method runs on an infinite loop whilst waiting to receive messages(DatagramPacket) from the server through the client's respective DatagramSocket.
- send(Message) : this method gets called when the client wants to send a message to the server or to another online client. It takes a Message object as an argument and checks the metadata of the message to know to which destination to send the message.
- handleCommand(Message) : this method gets called when a command is received instead of a message. It decodes the command to determine what action to take.
- createUser(username) : this method is used to create a new User with the string argument as its username. This method is called once a user opens the client program. It is a representation of the client as a user.
- setDestination(dest) : this method sets the destination to which the client is currently sending messages to.
- getConnectedUsers() : this is used to send a command to the server to provide a list of users that are online.

The **Message** class is used to create Message objects that encode and decode messages being sent by a client or the server. The messages are encoded with the metadata associated with the client. The Message class has the following methods:

- getString() : this is used to create an encoded string using the metadata. The string created will be sent as byte array through the DatagramSocket.
- parseString(): this method decodes the received encoded string.
- getUser() : this method returns the user from whom the message originated from.
- getDestination() : this will return the destination of the message that is being sent.
- getCommand() : this method read through the message object to determine whether it is a command or not. If it is a command, it returns the command received.

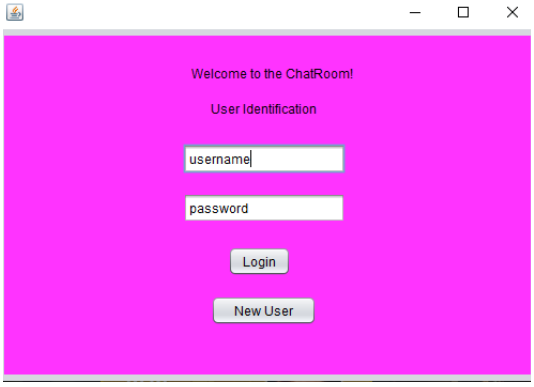
- getText() : this method is used to retrieve the original message from the author before it was encoded.

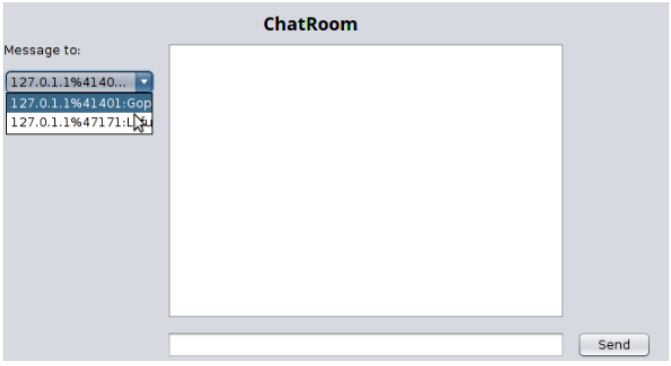
The **User** class is used to create User objects holding the respective information of clients. The IDs of the users were created by combining the user's IP address and the port number with the delimiter '%' between these data.

The **MessageTracker** class keeps track of the Message objects. It establishes which message belongs to which specific client or from the server. It records the ID to be able to identify the owner of the Message object. It also displays the message string on the GUI when the owner is selected.

Functionality

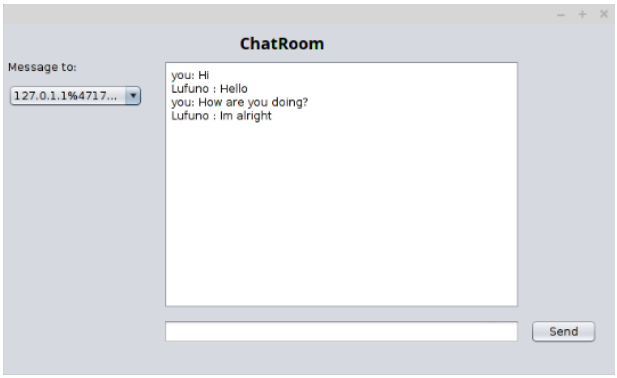
To run the application, the server is first started on a host. On another host the ChatRoomGui program is run, and the user will be prompted to write his/her username. A user will be created in a Client class with the username provided, the ID generated from the IP address and port number. The Client will send a command to the server to register this user and add the latter to the list of online users.





The user can then open a list of online clients that are available to receive messages from the drop down combobox on the top left of the window. When a user is selected, all the text exchanged between the client and the online user will be displayed on the text area found at the centre of the window.

To send a message to the user selected, one must type on the text field found at the bottom of the window and click on send. The Server will receive the message and using the MessageTracker class, it will send the text to the desired client on another host.



Chat Application Design

The diagram below shows the UML of the chat application with all the classes involved.