UNIVERSITI TUNKU ABDUL RAHMAN

FACULTY OF INFORMATION & COMMUNICATION TECHNOLOGY

UCCN3034 – NETWORK PROGRAMMING

GROUP PROJECT

GROUP ID: 3002

Student ID	Student Name	Demonstration (10%)	Total (100%)
19ACB03437	**Ng Warren Cin Yen**		
19ACB03076	**Loh Sim Harng**		
20ACB06505	**Miao Wei**		
20ACB01499	**Zhong Ling Hao**		

Report	
Cover Page (1%)	
Content table (1%)	
Introduction (3%)	
Computer Networking (20%)	
Implementation (5%)	
Result and Discussion (10%)	
Conclusion (5%)	
Report Quality (5%)	
Report Total (50%)	

Programs	
Server Program (20%)	
Client - Sender (5%)	
Client - Receiver (5%)	
Code Quality (10%)	
Program Total (40%)	

Table of Content

Chapter 1: Introduction

In this report, we are going to discuss computer networking, which covers Unicast, Multicast, and Broadcast. A brief introduction about the computer networks stated above will be discussed. Besides that, the stated computer networking will be demonstrated by using Java programming language. After that, we are going to develop a system for the Livingway (the Lapporrits on the earth) to collect information and send it to the rest of the Laporrits on the moon. The system will be using Java language which is Java network programming. The systems consist of 2 programs which are the client program and the server program. Within the client program, the user can choose to be a sender or a receiver. If the user chooses to be the sender, the user will be choosing the function to be performed and the Server program will then perform the respective function and send back the output to the sender. The functions included in this system are the function that displays the system time, port number, and reverse characters send information through the multicast address, performs addition through the RMI method, and arranges the number sequence through the RMI method. On the other hand, if the user chooses to be a receiver. The receiver will receive the message through the multicast group. The multicast message will be sent by the Sender and then passed to the Server for multicasting. Additionally, the Caesar Cipher encryption and decryption method is also included to encrypt the message from plaintext to ciphertext and decrypt the message from ciphertext to plaintext. Unicast, multicast, and RMI of the client-server program will be written for this networking system.

Chapter 2: Computer Networking

2.1 Unicast:

Unicast is a one-to-one transmission from one point in the network to another point; that is, one sender and one receiver, each identified by a network address. Transmission Control Protocol (TCP) and User Datagram Protocol (UDP) are two common Internet Protocol unicast delivery techniques. On LANs and the Internet, unicast transmission—in which a packet is sent from a single source to a specific destination—remains the most common type of transmission. Unicast transfer mode is supported by all LANs (such as Ethernet) and IP networks, and most users are familiar with the common unicast applications (such as http, smtp, ftp, and telnet), which use the TCP transport protocol. For example: Browsing a website (http).

Demonstration of Unicast (TCP) using Java code:

TCP Client:

```
// Creates a socket by specifying the host and the port_number
Socket ClientSocket = new Socket(Host, port_number);

// DataInputStream - reads data sent from server
DataInputStream din = new DataInputStream(ClientSocket.getInputStream());

// DataOutputStream - sends data to server
DataOutputStream dout = new DataOutputStream(ClientSocket.getOutputStream());


din.close();
dout.close();
ClientSocket.close();
```

On the client side, a socket is created by specifying the 1st argument: host address (IP address of the server, e.g.: 127.0.0.1) and 2nd argument: port number (The port number where the application will be running on) e.g.: 3000). DataInputStream and DataOutputStream is also initialized for the communications (to send and receive data) between the client and the server. Finally, close() function will be used to close all of the sockets and utilities.

TCP Server:

```
// Init Server Socket with port_number
ServerSocket serverSocket =new ServerSocket(port_number);

// Creates a socket that accepts incoming socket connection
Socket server= serverSocket.accept();

// DataInputStream - reads data sent from client
DataInputStream din=new DataInputStream(server.getInputStream());

// DataOutputStream - sends data to client
DataOutputStream dout=new DataOutputStream(server.getOutputStream());


dout.close();
din.close();
server.close();
serverSocket.close();
```

On the server side, a ServerSocket is created by specifying the port number (E.g.: 3000). A socket will also be created that listens to the particular port number that ServerSocket listens to. If an incoming socket with the same port number that the ServerSocket listens to, the socket will accept the incoming connection. DataInputStream and DataOutputStream is also initialized for the communications (to send and receive data) between the client and the server. Finally, close() function will be used to close all of the sockets and utilities.

2.2 Multicast:

Multicast is group communication where data transmission is addressed to a group of destination computers simultaneously. Multicast refers to communication in which a message is delivered from one or more points to a collection of other points. Primarily, multicast communication uses UDP (User Datagram Protocol). In this instance, there may be one or many senders and a group of receivers receive the information (there may be no receivers or any other number of receivers). There are 2 types of multicast distribution models, which is one-to-many and many-to-many. Examples of one-to-many multicast distribution are audio/video (lectures class, presentation, concerts), push media or announcements (news, weather time). Whereas examples of many-to-many multicast distribution include conferencing or multiplayer games.

Demonstration of Multicast using Java code:

Multicast Sender:

```
InetAddress group = InetAddress.getByName( host: "225.0.0.1");
int port = 1234;
MulticastSocket mcs = new MulticastSocket(port);
mcs.joinGroup(group);

byte[] buffer = new byte[0];
DatagramPacket toMulticastAddr = new DatagramPacket(buffer,buffer.length, group, port);
mcs.send(toMulticastAddr);
```

In the Multicast sender, the InetAddress group specifying the multicast address is stated. Besides that, the port number is also specified. After that, a MulticastSocket will be created with the port number stated. After creating the MulticastSocket, the joinGroup() method will be used so that the MulticastSocket will be joined to the multicast address group. A DatagramPacket will be created with the arguments specified (1st argument: data to be sent, 2nd argument: length of the data, 3rd argument: multicast address, 4th argument: multicast port number). After that, the DatagramPacket created just now will be sent to the multicast receiver by using the send() method through the multicast socket.

Multicast Receiver:

```
InetAddress group = InetAddress.getByName( host: "225.0.0.1");
int port = 1234;
MulticastSocket mcs = new MulticastSocket(port);
mcs.joinGroup(group);

byte[] buffer = new byte[0];
DatagramPacket reply = new DatagramPacket(buffer, buffer.length);
mcs.receive(reply);
```

In the Multicast receiver, the InetAddress group specifying the multicast address is stated. Besides that, the port number is also specified. After that, a MulticastSocket will be created with the port number stated. After creating the MulticastSocket, the joinGroup() method will be used so that the MulticastSocket will be joined to the multicast address group. A DatagramPacket will be created for receiving packets with the length stated. After that, the DatagramPacket created just now will be used to receive the incoming message from the multicast sender by using the receive() method through the multicast socket.

2.3 Broadcast:

Broadcasting is the communication in which the sender sends data to all of the receivers in a network. This is a one-to-all communication model where each sending device transmits data to all other devices in the network domain. The address resolution protocol (ARP) uses this to send an address resolution query to all computers on a LAN and this is used to communicate with an IPv4 DHC server. Broadcast transmission is supported by most LANs (e.g., Ethernet) and may be used to send the same message to all computers on the LAN. The IP network ID and an all-ones host number make up the broadcast packet in IPv4 that can be sent to every machine in a logical network.

Demonstration of Broadcast using Java code:

Broadcast sender:

```
// Creates DatagramSocket
InetAddress addr = InetAddress.getByName( host: "255.255.255.255");
int port = 1234;
DatagramSocket ds = new DatagramSocket(port);

// Sends the message through broadcast
byte[] buffer = new byte[1024];
DatagramPacket sdp = new DatagramPacket(buffer, buffer.length ,addr, port);
ds.send(sdp);
```

In the Broadcast sender, the InetAddress group specifying the broadcast address is stated. Besides that, the port number is also specified. After that, a DatagramSocket will be created with the port number stated. A DatagramPacket that sends the broadcast message will be created with the arguments specified (1st argument: data to be sent, 2nd argument: length of the data, 3rd argument: broadcast address, 4th argument: broadcast port number). After that, the DatagramPacket created just now will be sent to the broadcast receiver by using the send() method through the DatagramSocket.

Broadcast receiver:

```
// Creates DatagramSocket
InetAddress addr = InetAddress.getByName( host: "255.255.255.255");
int port = 1234;
DatagramSocket ds = new DatagramSocket(port);

// Receive the message through broadcast
byte[] buffer = new byte[1024];

DatagramPacket rdp = new DatagramPacket(buffer, buffer.length);
ds.receive(rdp);
```

In the Broadcast receiver, the InetAddress group specifying the broadcast address is stated. Besides that, the port number is also specified. After that, a DatagramSocket will be created with the port number stated. A DatagramPacket that will receive the broadcast message will be created for receiving packets with the length stated. After that, the DatagramPacket created just now will be used to receive the incoming message from the broadcast sender by using the receive() method through the DatagramSocket.

2.4 Summary between unicast, multicast, and broadcast.

	Unicast	Multicast	Broadcast
Distributions	One-to-one	One-to-many Many-to-many	One-to-all
Internet Protocols	Uses TCP or UDP	Uses UDP only	Uses UDP only
Advantages	A unicast server can respond to client requests in a timely manner	Clients who need the same data flow join the same group to share one data flow, reducing server load.	Server traffic load is extremely low because the server does not have to send data to each client individually
Disadvantages	The server might be overwhelmed and congested if the incoming traffic load is massive	The multicast protocol lacks an error correction mechanism in contrast to the unicast protocol	Every host on the network will always receive a broadcast message even if the message is irrelevant to them
Examples	Browsing a website, Downloading a file from an FTP server	Internet Protocol Television (IP TV)	DHCP Discover Message

Summary between unicast, multicast, and broadcast

Chapter 3: Implementation

1. Open a terminal, install the Java libraries by typing the following commands:

```
sudo apt install default-jre
sudo apt install default-jdk
```

2. Unzip the file UCCN_GroupAssignment_3002.
3. In the UCCN_GroupAssignment_3002 folder, open a terminal, and compile the programs by typing the following command:

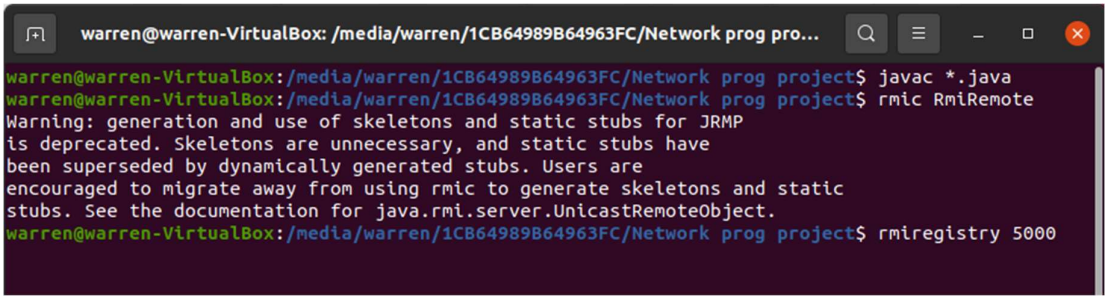
```
javac *.java
```

4. In the same terminal, create stub and skeleton object by typing the following command:

```
rmic RmiRemote
```

5. In the same terminal, start the RMI registry by typing the following command:

```
rmiregistry 5000
```



6. In a new terminal, start the Server program by typing the following command:

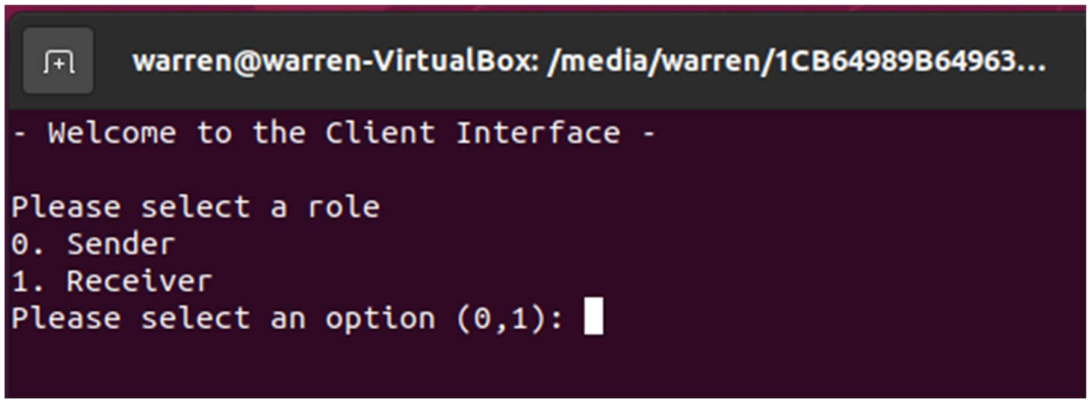
```
java MyServer
```

7. In a new terminal, start the Client program by typing the following command:

```
java MyClient
```

8. You should observe that the Server program will prompt a welcome screen and status. While in the Client program, a welcome message will be prompted in the console and waiting for user to key in an input that select 0 (sender) or 1 (receiver). The outcome is shown in the screenshots below:

Server program



Client program

Chapter 4: Results and discussion

Caesar Cipher:

All of the communication that happens between the client (sender, receiver) and the server will be encrypted and decrypted using Caesar Cipher encryption technique with the shift = 3.
The encrypt() and decrypt() method is declared in the Caesar Cipher class (CaesarCipher.java).
The method will increments/decrements the offset of the value by 3 (E.g. 1 become 4)

```
// Encrypts text using a key(shift=3)
8 usages
public String encrypt(String text)
{
    int key = 3;
    char[] chars = text.toCharArray(); //pass text into chars array

    //Shifting of the value in the array with key=3
    for( int i =0; i < chars.length; i++){
        char c = chars[i];
        c += key; //Encryption, increments offset of the array with key=3
        chars[i] = c;
    }

    return String.valueOf(chars);
}
```

Caesar Cipher – encrypt() method

```
8 usages
public String decrypt(String text)
{
    int key = 3;
    char[] chars = text.toCharArray(); //pass text into chars array

    //Shifting of the value in the array with key=3
    for( int i =0; i < chars.length; i++){
        char c = chars[i];
        c -= key; //Decryption, decrements offset of the array with key=3
        chars[i] = c;
    }

    return String.valueOf(chars);
}
```

Caesar Cipher – decrypt() method

RMI interface and RmiRemote:

The Rmi.java is an interface that will declare the method of the remote object clients may call. The RmiRemote will declare the method that will be called by the client program.

The add() method will return the value of the addition of 2 float values.

```
1 usage
public double add(double x,double y){return x+y;}
```

RMI – add() method

The arg() method will arrange the value of numbers in ascending order and return the arranged string.

```
public String arg(String s) {

    int[] numbers = new int[s.length()];

    for(int i =0; i<numbers.length; i++){
        numbers[i] = Integer.parseInt(String.valueOf(s.charAt(i)));
    }

    int temp;
    for(int i =0; i<numbers.length; i++){
        for(int j=i; j<s.length();j++){
            if(numbers[i]>numbers[j]){
                temp = numbers[i];
                numbers[i] = numbers[j];
                numbers[j] = temp;
            }
        }
    }

    return Arrays.toString(numbers).replaceAll( regex: "\\[|\\]|,|\\s", replacement: "");
}
```

RMI – arg() method

The Welcome menu, Function menu, Clear console related method are declared in the Text class (Text.java). Those methods will be called in the Server and Client program.

```
public static void getWelcomeMenu() {

    System.out.println("- Welcome to the Client Interface -");

    System.out.println("\nPlease select a role");
    System.out.println("0. Sender");
    System.out.println("1. Receiver");
    System.out.print("Please select an option (0,1): ");
}
```

Text – getWelcomeMenu()

```
public static void getFunctionMenu(){
    System.out.println("- Sender Interface -");
    System.out.println("\nPlease select one function to be performed:");
    System.out.println("1. Display the system time");
    System.out.println("2. Display the port number");
    System.out.println("3. Reverse characters");
    System.out.println("4. Send Information through multicast addr");
    System.out.println("5. Perform addition on 2 numbers (RMI)");
    System.out.println("6. Perform arrangement of number sequence (RMI)");
    System.out.println("7. Quit");
    System.out.print("Please key in an option (1-7): ");
```

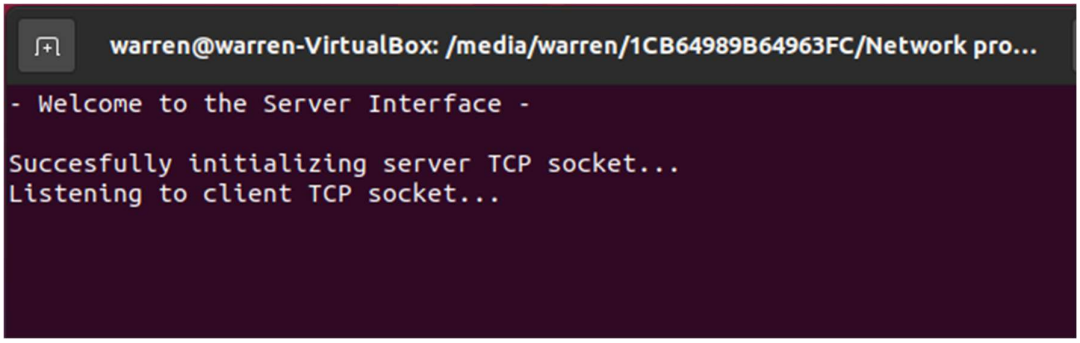
Text – getFunctionMenu()

```java
public final static void clearConsole() {
    System.out.print("\033[H\033[2J");
    System.out.flush();
}

4 usages
public final static void retry(){
    System.out.println("\nPress ENTER to try again...");
    try{System.in.read();}
    catch(Exception ex){}
}

7 usages
public final static void pause(){
    System.out.println("\nPress ENTER to continue...");
    try{System.in.read();}
    catch(Exception ex){}
}
```

Text – clearConsole(), retry(), pause()

Server program:

In the Server Program, the program will first prompt a welcome message and shows the status of the sockets process. A server TCP socket will be created. After setting the socket option and binding, the program will start to listen to incoming connections. As shown in Diagram 1.1.



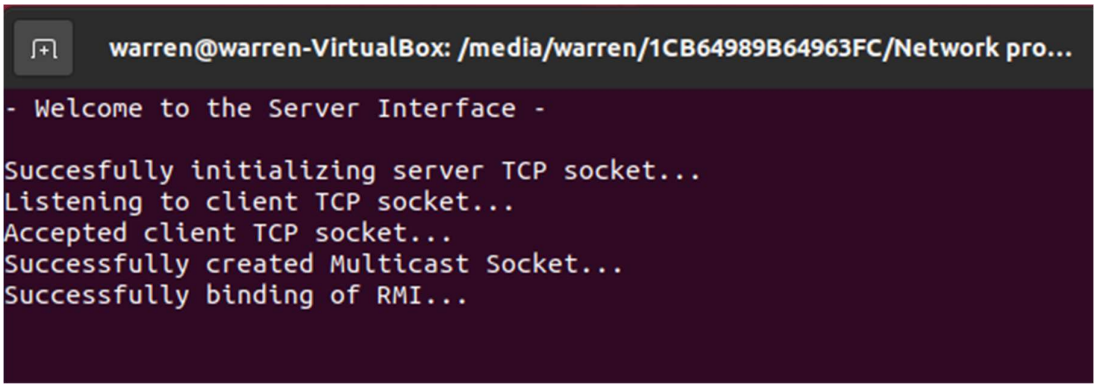
Server - Diagram 1.1

The server TCP socket will be initialized with port number 3002. If the port number is in use, the catch block will handle the error and exception and terminate the program.

```
// init  components needed in TCP server
// ServerTCP socket
ServerSocket serverTCPSock = null;
Socket s = null;

try{
    serverTCPSock =new ServerSocket( port: 3002);
    System.out.println("Succesfully initializing server TCP socket...");
}
catch(Exception e){
    System.out.println("Failed initializing server TCP socket...");
    e.printStackTrace();
    System.exit( status: 0);
}
```

Source code of initializing server TCP socket

If the user selects option 0 (Sender) at the client program, the server accepts the incoming TCP connection from the client. After that, the server will create a Multicast Socket that is used in function 4 (Sends message through multicast), and the binding of the RMI stub. As shown in Diagram 1.2



Server - Diagram 1.2

The processes: accept(), creating Multicast socket, RMI binding; will all be passed through the try-catch block to handle the exceptions. If an error occurred during the process, the system will be terminated.

The multicast socket will be initialized with the multicast address: 255.0.0.1 . If the process is failed, the catch block will handle the error and exception and terminate the program.

```
// init the components needed in Multicast communication
// Specifying the multicast_addr
InetAddress group = InetAddress.getByName( host: "225.0.0.1");
MulticastSocket multicastSock = null;

try{
    multicastSock = new MulticastSocket();
    System.out.println("Successfully created Multicast Socket...");
}catch(Exception e){
    System.out.println("Failed to create Multicast Socket...");
    e.printStackTrace();
    System.exit( status: 0);
}
```

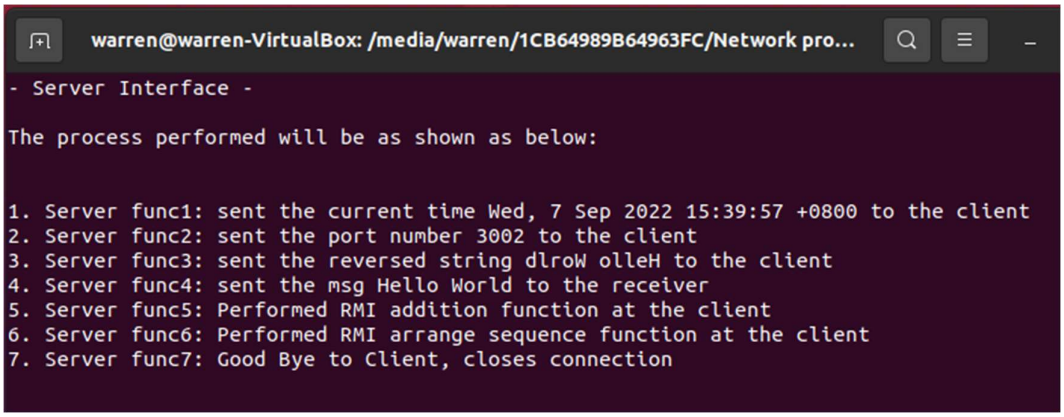
Source code of initializing multicast socket

The RMI will be initialized by constructing a new RMI object and binds that object stub to the name of URL rmi://localhost:5000/sonoo. If the process failed (RMI bound exception), the catch block will handle the errors and terminate the program.

```
// init Rmi registry
try{
    //Constructs a new Rmi object
    Rmi stub =new RmiRemote();

    //Binds that object "stub" to the name of the URI rmi://localhost:5000/sonoo
    Naming.bind( name: "rmi://localhost:5000/sonoo",stub);
    System.out.println("Successfully binding of RMI...\n\n");

}catch(Exception e){
    System.out.println("Failed binding of RMI...\n\n");
    e.printStackTrace();
    System.exit( status: 0);
}
```

Source code of initializing RMI

After initializing all of the required components, the server interface will be displayed. The processes that are performed on the server will be displayed in the console. A counter will be used to keep track of the number of processes performed. As shown in Diagram 1.3



Server - Diagram 1.3

A while loop will be used to loop the process until the user wishes to exit (select 7. Quit at the client side). The server program will read in and decrypts the function option number from the client. The counter value will increment whenever it is passed through the while loop to keep track of the number of processes performed.

```
// Loops until user wish to exit
while (func_opt != 7) {

    //Obtain the function number from client
    String str = din.readUTF();
    func_opt = Integer.parseInt(cipher.decrypt(str));
    counter++;
```

Server – while loop to check user function selection

After getting the function option number from the client. The value will be passed into a switch block to perform the corresponding function requested by the client.

Function 1: System Time

In this function, getCurrentTime() function (defined in Time class) will be called to obtain the latest time. After that, the time will be encrypted and sent back to the client. The function 1 description will also be displayed on the Server Interface. (As shown in diagram 1.3)

```
//1. System time
case 1:

    // Gets the latest time with Time class
    Time time = new Time();
    String cur_time =  time.getCurrentTime();

    // Encrypts and Sends the time to client
    dout.writeUTF(cipher.encrypt(cur_time));
    System.out.println(counter +". Server func1: sent the current time " + cur_time + " to the client");
    break;
```

Server - Function 1 source code

```
public class Time {

    1 usage
    public String getCurrentTime(){

        Date currentDate = new Date();
        SimpleDateFormat timeformat = new SimpleDateFormat( pattern: "EEE, d MMM yyyy HH:mm:ss Z");

        return timeformat.format(currentDate);
    }

}
```

Time class (Time.java)

Function 2: Port number

In this function, getLocalPort() function will be called to obtain the port number used by the client. After that, the port number will be encrypted and sent back to the client. The function 2 description will also be displayed on the Server Interface. (As shown in diagram 1.3)

```
//2. Port number
case 2:

    // Gets the port number
    int port = s.getLocalPort();

    // Encrypts and Sends the port number to client
    dout.writeInt(cipher.encrypt(port));
    System.out.println(counter +". Server func2: sent the port number " + port + " to the client");
    break;
```

Server - Function 2 source code

Function 3: Reverse characters

In this function, the server will receive and decrypts the user input from the client. After that, the user input string will be passed into a reverse() function (defined in ReverseString class). The reverse() function will reverse the character in the string. After that, the reversed string will be encrypted and sent back to the client. The function 3 description will also be displayed on the Server Interface. (As shown in diagram 1.3)

```
//Reverse characters
case 3:

    // Gets user input from client
    String client_str = cipher.decrypt(din.readUTF());

    // Pass the string into reverse()
    ReverseString rev = new ReverseString();
    String reversed_str = rev.reverse(client_str);

    // Encrypts and Sends the reversed_str back to client
    dout.writeUTF(cipher.encrypt(reversed_str));
    System.out.println(counter +". Server func3: sent the reversed string " + reversed_str + " to the client");
    break;
```

Server - Function 3 source code

```
public class ReverseString {

    1 usage
    public String reverse(String s){
        char[] letters = new char[s.length()];

        int index=0;
        for( int i = s.length()-1; i>=0; i--){
            letters[index]=s.charAt(i);
            index++;
        }

        String reverse="";
        for( int i =0; i < s.length();i++){
            reverse= reverse+ letters[i];
        }

        return reverse;
    }
```

ReverseString class (ReverseString.java)

Function 4: Send information through multicast

In this function, the server will receive the message from the sender. After that message will be passed into a Datagram Packet which specifies the message, multicast address: 255.0.0.1, and port number: 3502. After that, a send() function is used to send the Datagram Packet to the multicast group. After sending the message to the multicast group, an encrypted status will be sent back to the sender to state the status of the process. The function 4 descriptions will also be displayed on the Server Interface. (As shown in diagram 1.3)

```
//Send information
case 4:

    // Gets msg from sender
    String msg = cipher.decrypt(din.readUTF());

    // Encrypt the message before sending to multicast_group
    String en = cipher.encrypt(msg);

    // Use UDP packet to send the msg to multicast_group
    DatagramPacket toReceiver = new DatagramPacket(en.getBytes(),en.length(), group, port: 3052);
    multicastSock.send(toReceiver);

    System.out.println(counter +". Server func4: sent the msg " + msg + " to the receiver");
    String status = "Successfully sent the message " + "\"" + msg + "\"" + " to the receiver";

    // Encrypts and Sends the status back to client
    dout.writeUTF(cipher.encrypt(status));
    break;
```

Server - Function 4 source code

Function 5: Perform Addition through RMI

In this function, although the server is not involved in providing the addition function for the client, the status of the process at the client side is sent to the server to indicate that the process is successfully performed. The server will receive the status from the client and decrypts it. After that, the function 5 description will be displayed on the Server Interface. (As shown in diagram 1.3)

```
//Perform addition (RMI)
case 5:

   // Gets the status from client
   int stat1 = cipher.decrypt(din.readInt());
   if (stat1==1) {System.out.println(counter +". Server func5: Performed RMI addition function at the client");}
   else{System.out.println("Server func5: Failed");}
   break;
```

Server - Function 5 source code

Function 6: Arrange sequence through RMI

In this function, although the server is not involved in providing the arrange function for the client, the status of the process at the client side is sent to the server to indicate that the process is successfully performed. The server will receive the status from the client and decrypts it. After that, the function 6 description will be displayed on the Server Interface. (As shown in diagram 1.3)

```
//Arrange Sequence (RMI)
case 6:

    // Gets the status from client
    int stat2 = cipher.decrypt(din.readInt());
    if (stat2==1) {System.out.println(counter +". Server funcó: Performed RMI arrange sequence function at the client");}
    else{System.out.println("Server funcó: Failed");}
    break;
```

Server - Function 6 source code

Function 7: Quit

In this function, the server will send an encrypted goodbye message to the client. After that, the function 7 description will be displayed on the Server Interface. (As shown in diagram 1.3)

```
//Quit
case 7:

    /// Goodbye Msg
    String goodbye = "Goodbye, Have a Nice Day!";

    // Sends the goodbye msg to the client
    dout.writeUTF(cipher.encrypt(goodbye));

    System.out.println(counter +". Server func7: Good Bye to Client, closes connection");
    break;
```

Server - Function 7 source code

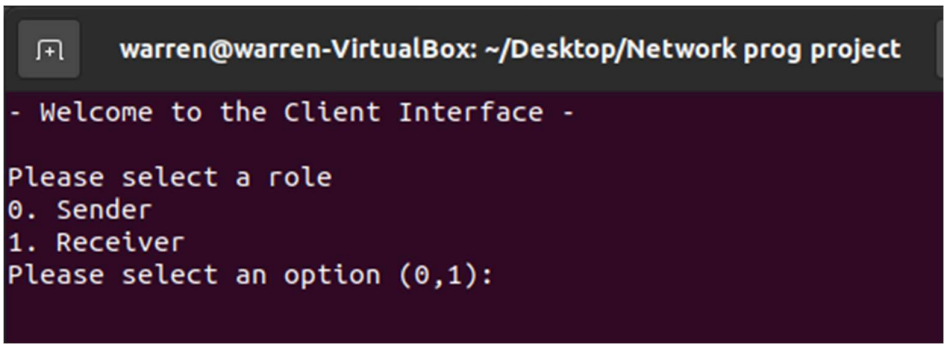
After the server receive the Quit function option (7. Quit) from the sender, the program will not enter the while loop again and all of the socket and utilities will be closed. The Server program will be terminated.

```
// Closes all the utils and exit the program
dout.close();
din.close();
s.close();
serverTCPSock.close();
multicastSock.close();
System.out.println("\n\nTCP Connection Closed");
System.exit( status: 0);
```

Source code of closing all of the server-side sockets

Client program:

In the Client Program, the program will first prompt a welcome message and asks the user to input a selection (0. To be Sender, 1. To be Receiver). The program will validate the user input. As shown in Diagram 2.1



Client - Diagram 2.1

```
// Welcome page, let user select 1.Sender or 2.Receiver
// User input validation for Welcome selection
do{
    Text.clearConsole();
    Text.getWelcomeMenu();
    String str = sc.nextLine();

    try{
        wlc_opt = Integer.parseInt(str);

        // Valid input - 0 or 1
        if(wlc_opt == 0 || wlc_opt ==1){
            wlc_bool=true;
            Text.clearConsole();
        }
        // Invalid input - Wrong int value
        else{
            System.out.println("\nInvalid input entered, please try again.\n");
            wlc_bool=false;
            Text.retry();
            Text.clearConsole();
        }
    }
    // Invalid input - String or char
    catch(NumberFormatException e){
        System.out.println("\nInvalid input entered, Please try again.\n");
        wlc_bool = false;
        Text.retry();
        Text.clearConsole();
    }
}while(!wlc_bool);
```

Client – Welcome page source code

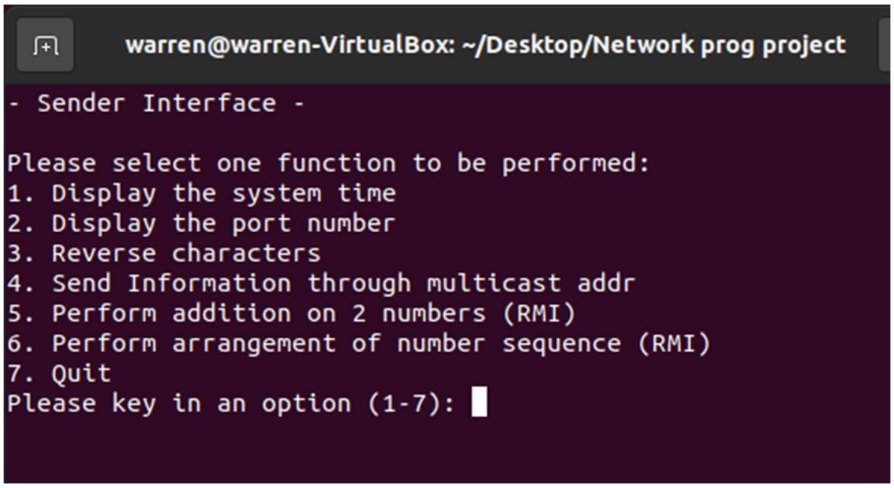
Client – Sender

If the user chooses to be a sender (selects 0. Sender), a TCP connection will be established with the server. A client TCP socket will be initialized by specifying the address: 127.0.0.1, port number: 3002. Besides that, a try-catch block will also use to handle the socket creation process. If the socket creation process failed, the client program will be terminated.

```
// init components needed in TCP client
// Specifying host_addr and port_num
Socket clientTCPSock = null;
int port = 3002;
String host = "127.0.0.1";


try {
    clientTCPSock = new Socket(host, port);
}
catch(Exception ex){
    System.out.println("Failed creating client TCP socket");
    ex.printStackTrace();
    System.exit( status: 0);
}
```

Source code of initialization of client TCP socket

After that, a while loop will be used to display the function menu, prompt user input, and communication with the server to obtain the function's outcome. The while loop will loop the process until the user wishes to exit (select 7. Quit at the client side). The function menu will prompt the user to key in a function option selection input with input validation.



Sender - Diagram 2.2

```
// Loops until user wish to exit
while(func_opt!=7){

    // User input validation for the function selection
    do{
        Text.clearConsole();
        Text.getFunctionMenu();
        String str = sc.nextLine();

        try{
            func_opt = Integer.parseInt(str);

            // Valid function selection input
            if(func_opt>0 && func_opt<8){
                func_bool=true;
            }
            // Invalid function selection input - int value other than 1-7
            else{
                System.out.println("\nInvalid input entered, please try again.\n");
                func_bool=false;
                Text.retry();
                Text.clearConsole();
            }
        }
        //Invalid input - String, char value
        catch(NumberFormatException e){
            System.out.println("\nInvalid input entered, Please try again.\n");
            func_bool = false;
            Text.retry();
            Text.clearConsole();
        }
    }while(!func_bool);
```

Client – Function selection page source code

After validating the user input to be a valid function number selection, the function number selection will be encrypted and sent to the server.

```
// Sends the function selection to the server
dout.writeUTF(cipher.encrypt(String.valueOf(func_opt)));
dout.flush();
```

Source code of client sending the function number to the server

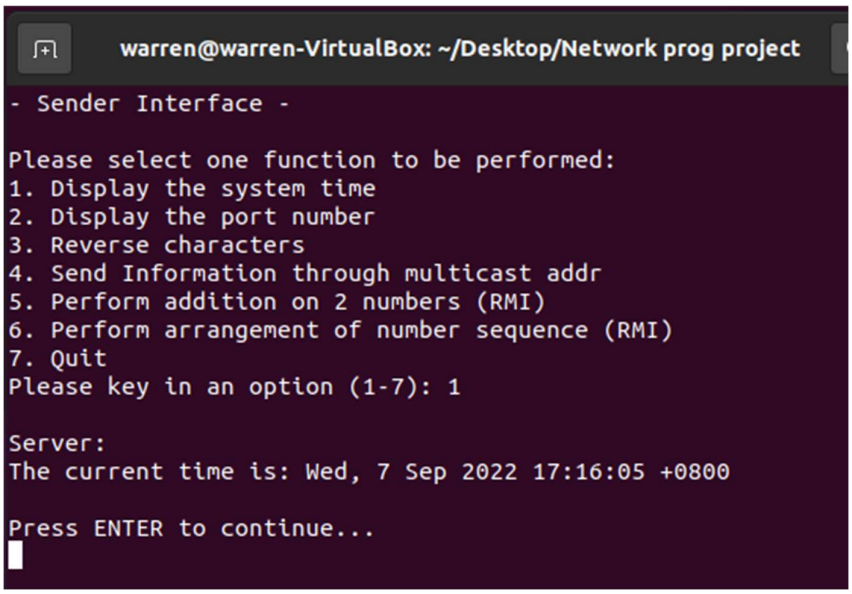
The function selection value will be passed into a switch block to perform the corresponding function that will request the information from the Server or through RMI.

Function 1: System Time

In this function, the client will receive and decrypts the latest time sent by the server. After that, the latest time will be displayed on the client console.

```
// Performs corresponding function
switch(func_opt){

    //1. System time
    case 1:
        String time = cipher.decrypt(din.readUTF());
        System.out.println("\nServer:\nThe current time is: " + time);

        Text.pause();
        break;
```

Client – Function 1 source code



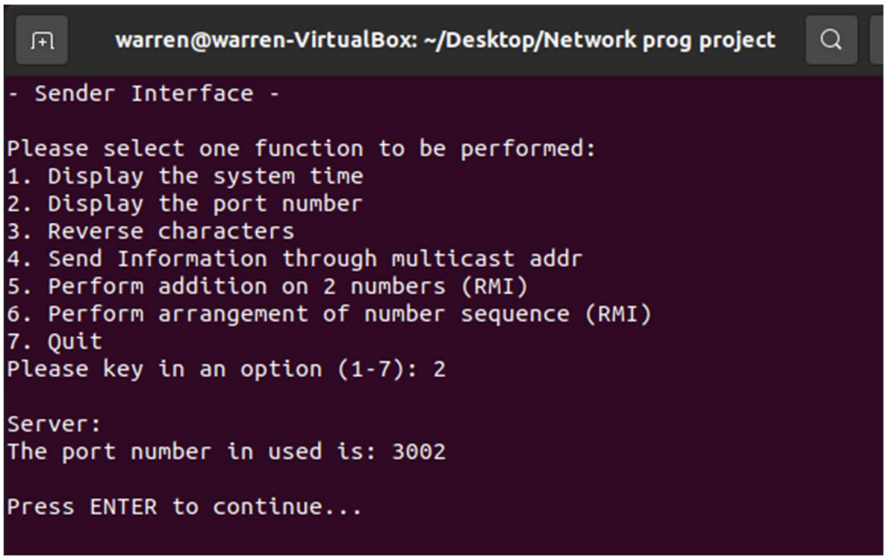
Client - Function 1

Function 2: Port number

In this function, the client will receive and decrypts the port number sent by the server. After that, the port number will be displayed on the client console.

```
//2. Port number
case 2:
    int port_num = cipher.decrypt(din.readInt());
    System.out.println("\nServer:\nThe port number in used is: " + port_num);

    Text.pause();
    break;
```

Client – Function 2 source code



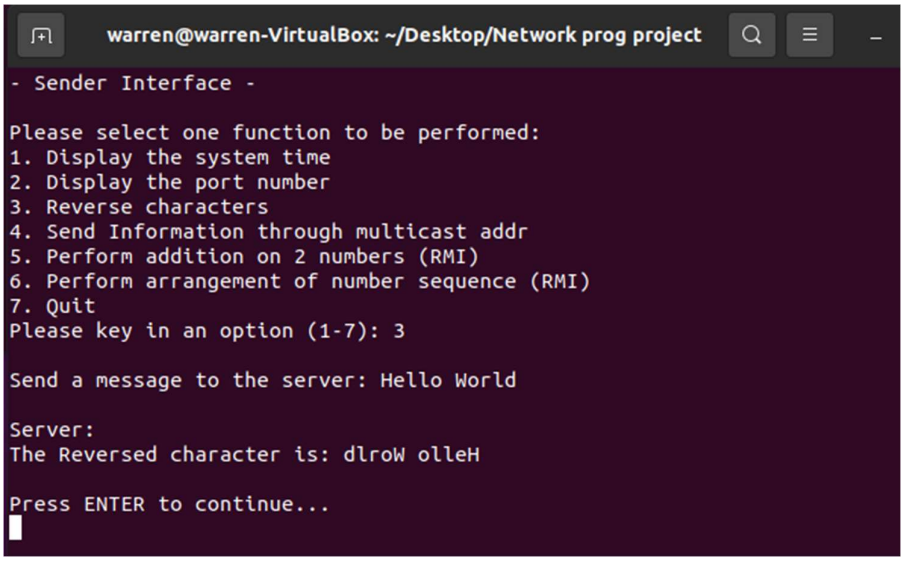
Client – Function 2

Function 3: Reverse characters

In this function, the client will first ask the user to key in a message. The message will then be encrypted and sent to the server. After that, the client will receive and decrypts the reversed message sent from the server. The reversed message will be displayed on the client console.

```
//3. Reverse characters
case 3:
    // User input message
    System.out.print("\nClient:\nSend a message to the server: ");
    String str = br.readLine();

    // Sends the encrypted message to server
    dout.writeUTF(cipher.encrypt(str));
    dout.flush();

    // Decrypts the reversed message from server
    String reverse_char = cipher.decrypt(din.readUTF());
    System.out.println("\nServer:\nThe Reversed character is: " + reverse_char);

    Text.pause();
    break;
```

Client – Function 3 source code



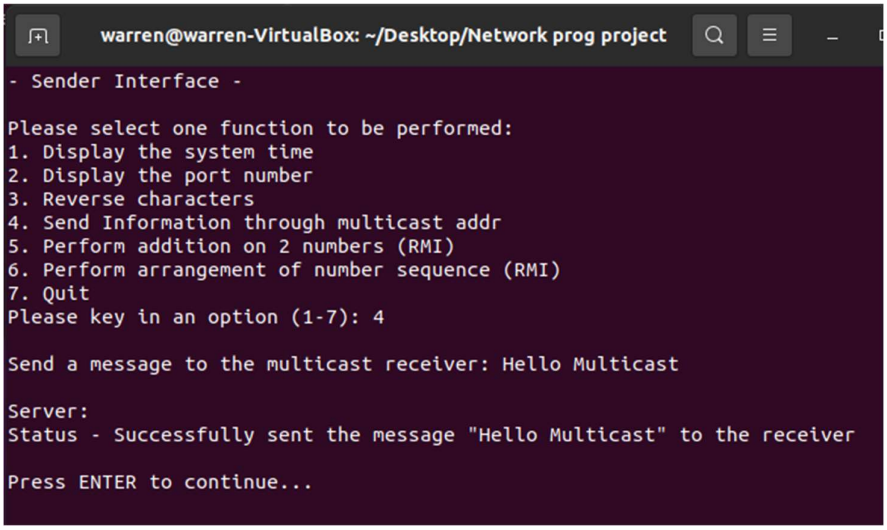
Client- Function 3

Function 4: Send information through multicast

In this function, the client will ask the user to key in a message that will be sent to the multicast group by the server. The message will then be encrypted and sent to the server. After that, the client will receive and decrypts the status sent from the server. The status will be displayed on the client console.

```
//4. Send information
case 4:
    System.out.print("\nSend a message to the multicast receiver: ");
    String multicast_msg = br.readLine();
    dout.writeUTF(cipher.encrypt(multicast_msg));
    dout.flush();

    String status = cipher.decrypt(din.readUTF());
    System.out.println("\nServer: \nStatus - "+ status);

    Text.pause();
    break;
```

Client – Function 4 source code



Client – Function 4

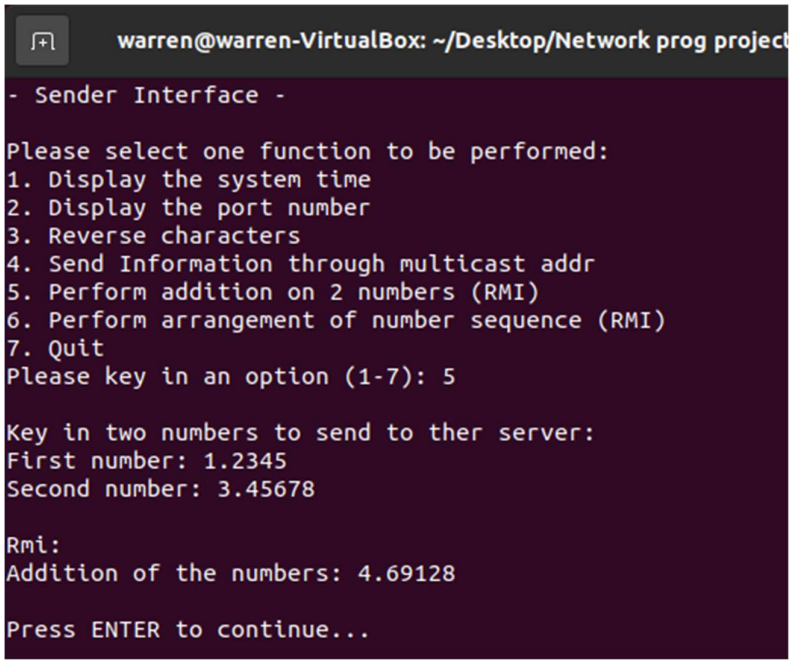
Function 5: Perform Addition through RMI

In this function, the client will ask the user to key in 2 numbers value (float). Both of the value of the numbers will be validated. After validating the user input, an RMI naming lookup function will be used to lookup the function declared in the RmiRemote.java. The 2 numbers value will be passed into the stub's add() function. The value of the addition of the two numbers (5 decimal places) will then be displayed on the client console. Finally, a success status will be sent to the server.

```
//Perform addition (RMI)
case 5:

    String input;
    float num1 = 0, num2= 0;
    boolean isNum1 = true, isNum2 = true;

    System.out.println("\nKey in two numbers to send to ther server:");

    //Num1 user input validation
    do{
        System.out.print("First number: ");
        input = sc.nextLine();

        try{
            num1 = Float.parseFloat(input);
            isNum1=true;

        }
        catch(NumberFormatException e){
            System.out.println("Invalid input entered, Please try again.\n");
            isNum1 = false;
        }
    }while(!isNum1);
```

```java
//Num2 user input validation
do{
    System.out.print("Second number: ");
    input = sc.nextLine();

    try{
        num2 = Float.parseFloat(input);
        isNum2=true;

    }
    catch(NumberFormatException e){
        System.out.println("Invalid input entered, Please try again.\n");
        isNum2 = false;
    }
}while(!isNum2);

//Perform RMI addition function
try{
    Rmi stub=(Rmi)Naming.lookup( name: "rmi://localhost:5000/sonoo");

    System.out.println("\nRmi:\nAddition of the numbers: " + df.format(stub.add(num1,num2)));
}
catch(Exception e){
    e.printStackTrace();
}

//Sends status to server
dout.writeInt(cipher.encrypt( integer: 1));

Text.pause();
break;
```

Client – Function 5 source -code



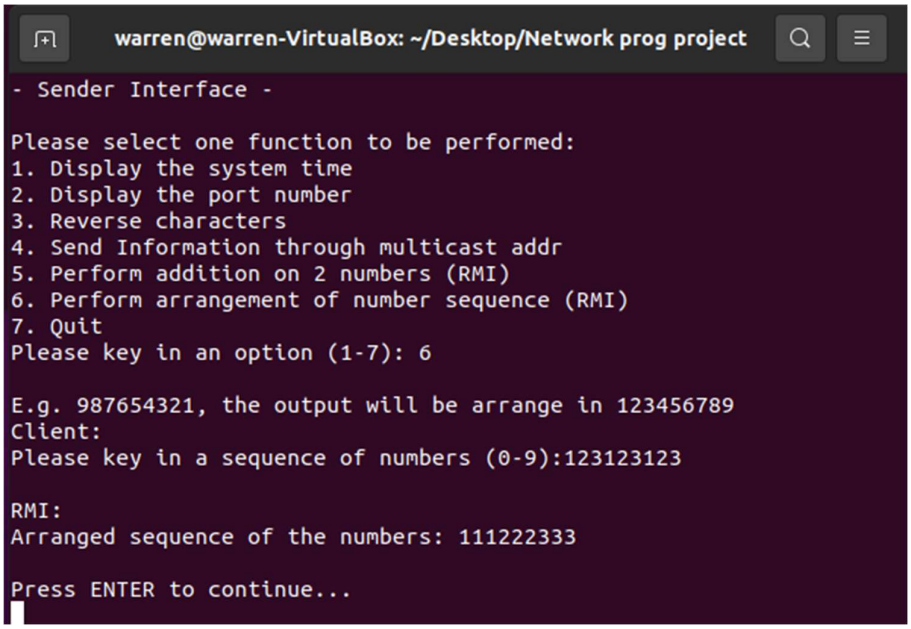
Client – Function 5

Function 6: Arrange sequence through RMI

In this function, the client will ask the user to key in a list of numbers (float). The input value will be validated. After validating the user input, an RMI naming lookup function will be used lookup the function declared in the RmiRemote.java. The 2 numbers value will be passed into the stub's arg() function. The arranged numbers will then be displayed on the client console. Finally, a success status will be sent to the server.

```
//6. Arrange the sequence (RMI)
case 6:
    //Checks
    boolean seq_check = true;
    String seq;

    do{
        System.out.println("\nE.g. 987654321, the output will be arrange in 123456789\n");
        System.out.print("\nPlease key in a sequence of numbers (0-9):");
        seq = sc.nextLine();

        if(seq.chars().allMatch(Character::isDigit)){
            seq_check = true;
        }
        else{
            System.out.println("\nInvalid input entered, Please Key in digits only.");
            seq_check = false;
        }
    }while(!seq_check);

    try{
        Rmi stub=(Rmi)Naming.lookup( name: "rmi://localhost:5000/sonoo");

        System.out.println("\nRmi:\nArranged sequence of the numbers: " + stub.arg(seq));
    }
    catch(Exception e){
        e.printStackTrace();
    }

    //Sends status to server
    dout.writeInt(cipher.encrypt( integer: 1));

    Text.pause();
    break;
```

Client – Function 6 source code



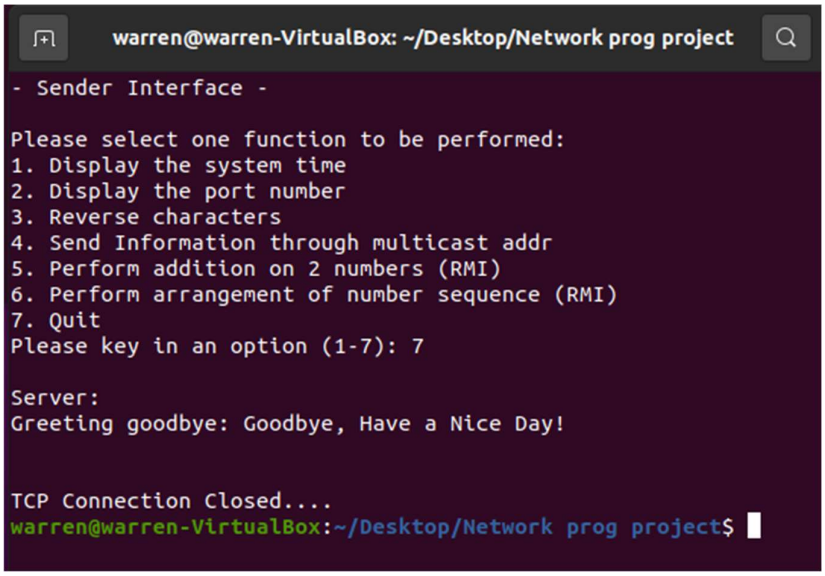
Client – Function 6

Function 7: Quit

In this function, the client will receive and decrypts the goodbye message sent by the server. After that, the goodbye message will be displayed on the client console.

```
//7. Quit
case 7:
    String bye_msg = cipher.decrypt(din.readUTF());
    System.out.println("\nServer:\nGreeting goodbye: " + bye_msg);

    break;
```

Client – Function 7 source code



Client – Function 7

After the user key in the Quit function option (7. Quit), the program will not enter the while loop again and all of the sockets and utilities will be closed. The Server program will be terminated.

```
din.close();
dout.close();
clientTCPSock.close();

System.out.println("\n\nTCP Connection Closed....");
```

Source code of closing all of the sender TCP socket

Client – receiver

If the user chooses to be a sender (selects 0. Sender), a Multicast Socket will be created with the multicast address: 225.0.0.1 and port number: 3052. After that, a joinGroup() function will be called so that the Multicast Socket created just now join the respective multicast group. A try-catch block also is used to handle the errors while creating Multicast Socket.

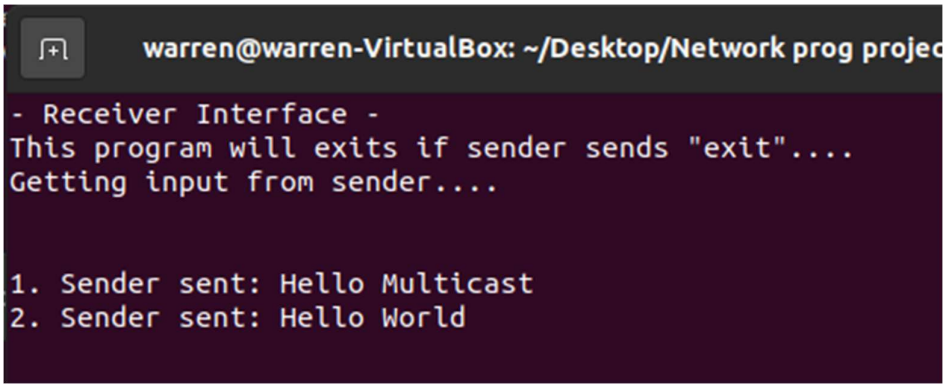
```
// User selects receiver - UDP multicast group
if (wlc_opt==1) {

    try {

        // Initialize the components needed in a Multicast communication
        InetAddress group = InetAddress.getByName( host: "225.0.0.1");
        MulticastSocket multicastSock = new MulticastSocket( port: 3052);
        multicastSock.joinGroup(group);
```

Receiver - creating Multicast socket as a receiver

After that, the console will display the Receiver Interface which will display all of the messages sent from the server through multicast. A counter is also used to keep track of the number count of the message sent from the server. A while loop is used to keep on looping the process until the receiver receives “exit” from the server. The buffer with size[1024] allows the maximum 1024 characters in the communications

```
System.out.println("- Receiver Interface -");
System.out.println("This program will exits if sender sends \"exit\"....");
System.out.println("Getting input from sender....\n\n");


    while (true) {
        byte[] buffer = new byte[1024];
        counter++;
```

Receiver - receiver interface and counter



Receiver – Receiver Interface

In the while loop, a UDP socket is created and listens to incoming messages from the server. After that, the UDP socket is passed into the multicast socket so that it can receive the message sent through the multicast group

```
// Create a UDP socket and receive incoming msg
DatagramPacket reply = new DatagramPacket(buffer, buffer.length);
multicastSock.receive(reply);
```

Receiver – UDP Socket

After the multicast socket received the data, a getData() function is used to retrieve the message from the server into a byte data type variable. Then the message is finally passed into a String data type variable.

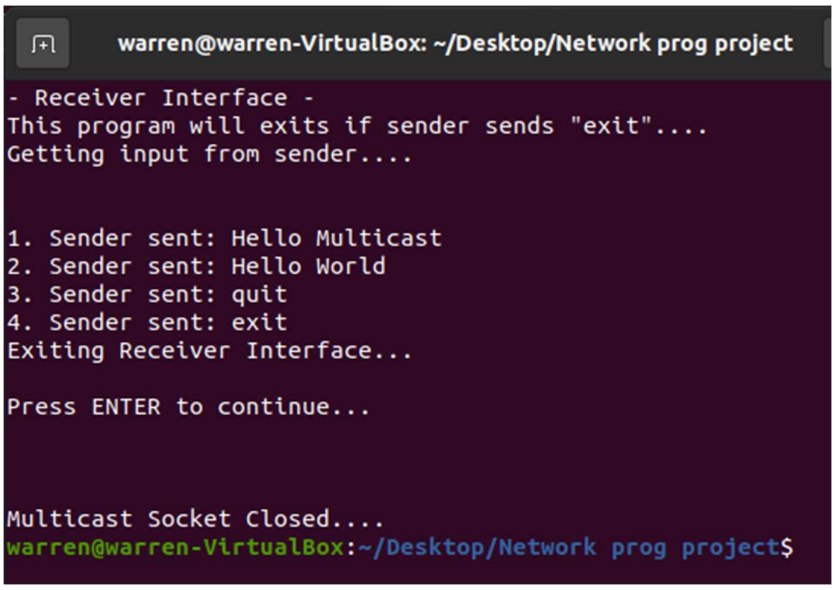
```
// Gets msg from Server
byte[] data = reply.getData();
String val = new String(data,  offset: 0, reply.getLength());
```

Receiver – getData() function

The message will then be decrypted. If the message matches the keyword “exit”, the receiver will display the message and then proceeds to quit the program. On the other hand, if the message does not match “exit”, the receiver will then display the message and keep on looping to receive the message from the server through the multicast group.

```
// Decrypts the msg
String de = cipher.decrypt(val);


// Checks for user input, if input == exit, quit the Receiver Interface
if (de.equals("exit")){
    System.out.println(counter +". Sender sent: " + de);
    System.out.println("Exiting Receiver Interface...");
    Text.pause();
    break;
}
else {
    System.out.println(counter +". Sender sent: " + de);
}
```

Receiver – decryption and checks for input



Receiver – Quits receiver if the received message is “exit”

Finally, the multicast socket will be closed, and the client program will be terminated.

```
            multicastSock.close();
            System.out.println("\n\nMulticast Socket Closed....");

        } catch (Exception e) {
            System.out.println("Failed to create Multicast Socket...");
            e.printStackTrace();
            System.exit( status: 0);
        }
}
br.close();
sc.close();
```

Source code of closing multicast socket and utilities

Chapter 5: Conclusion

In conclusion, we have learned how to distinguish the difference between Unicast, Multicast and Broadcast. The computer networking methods are also demonstrated by using Java code. Additionally, we have also summarized and compared the distribution model, Internet protocols involved, advantages, disadvantages, and implementations between Unicast, Multicast and Broadcast. Other than that, we have also learned Java network programming by using the sockets to create TCP and UDP connections between the client and the server programs. Unicast and Multicast in Java network programming are also being implemented in the client and server program. Besides that, we have also learned the rmic compiler to generate stubs and skeleton class files to host a Java RMI remote object. Additionally, we have also learned the Caesar Cipher technique to encrypt and decrypt the message sent between the client and server for security purposes. In short, this assignment is a great task that not only helped our team to develop skills in Java network programming but also teamwork and coordination between team members to finally complete this assignment.