CO527

Computer Networks and Distributed Systems

Assessed Coursework

RMI and UDP

Completed by:

Cheng Wei Loon (CID 01416852)

Marcus Neo Jing Quan (CID 01541100)

12 February 2020

RMI vs UDP

After testing our RMI code with two computers located in different parts of London, we concluded that RMI communication mechanism is very reliable. From our observations, we found that not a single message had been lost during transmission despite the computers being in different parts of London. This is because RMI uses Transmission Control Protocol (TCP). This ensures that a connection between the server and client is established before messages are sent over from client to server.

UDP is unreliable as it does not utilize TCP. Packets are sent over the network without first establishing a connection between the client and the server. This results in some loss of packets and the order of receipt being jumbled up.

Causes of lost messages

RMI

Messages can be lost when the network is congested, causing the buffer to become full. Subsequent packets are denied entry into the buffer and are not received properly. However, RMI implements TCP congestion control to deal with this situation. This explains why messages were hardly lost even when sending a large number of messages. However, this also took a toll on the performance as the messages were sent much more slowly due to client-server communication for each packet.

UDP

Messages also can be lost when the network becomes congested. However, there is no in-built congestion control system, so messages are more easily lost. Furthermore, the lack of client-server communication results in faster sending rate of packets, causing the network to become congested more quickly when the number of messages is sufficiently large.

Patterns of lost messages

RMI

There were no patterns found as all messages were received via RMI.

UDP

When testing on the lab computers, two patterns were discovered. The first was that around 70% of the time, all messages above 303 were lost. This could indicate a maximum buffer size or congestion control implemented by the DoC admins. In the other 30% of the time, the message loss rate varied but generally increased as the activity in the lab increased. This could be due to increased traffic over the network which caused the buffer to become full more quickly.

Comparison

We believe that UDP was easier to program. RMI was more difficult because we had to implement protocols like the security manager and the server-client binding to ensure a connection was properly established. On the other hand, UDP only required a common port over which messages are sent and received.

Nevertheless, the simplicity of UDP meant that it was not as reliable in sending information from the client and receiving information from the server as compared to RMI. As the server and the client are not bound, there is a much higher chance of messages being lost from client to server.

APPENDIX

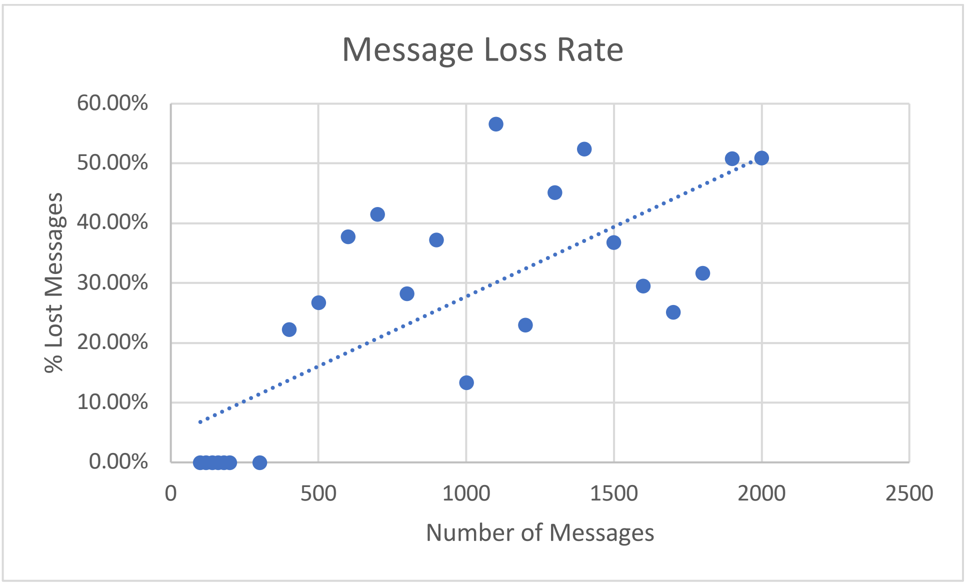


Figure 1: Graph showing the effect of the number of messages on the percentage of lost messages

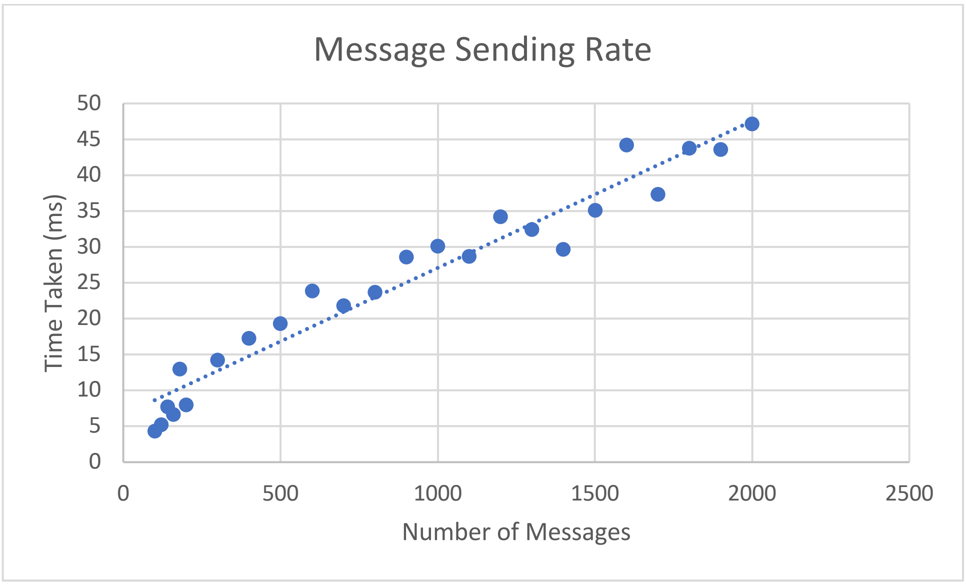
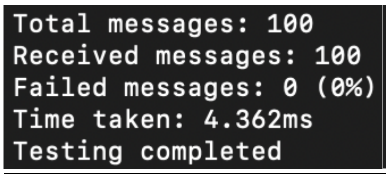


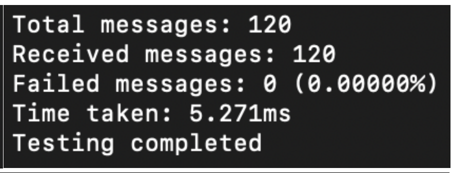
Figure 2: Graph showing the effect of the number of messages on the time taken

Screen Dump

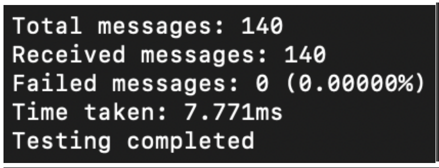
Note: Only the last three lost messages were captured for brevity.



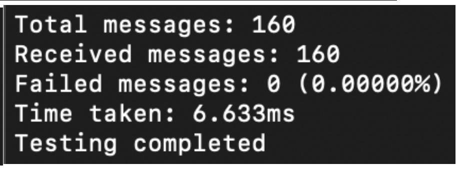
```
Total messages: 100
Received messages: 100
Failed messages: 0 (0%)
Time taken: 4.362ms
Testing completed
```



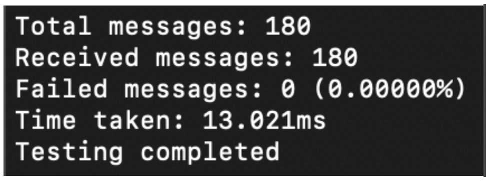
```
Total messages: 120
Received messages: 120
Failed messages: 0 (0.00000%)
Time taken: 5.271ms
Testing completed
```



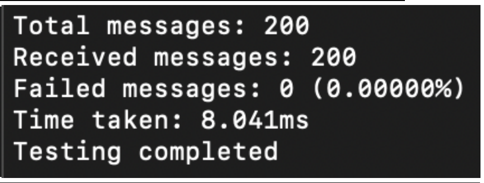
```
Total messages: 140
Received messages: 140
Failed messages: 0 (0.00000%)
Time taken: 7.771ms
Testing completed
```



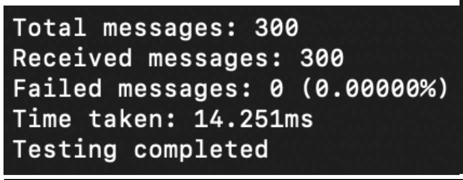
```
Total messages: 160
Received messages: 160
Failed messages: 0 (0.00000%)
Time taken: 6.633ms
Testing completed
```



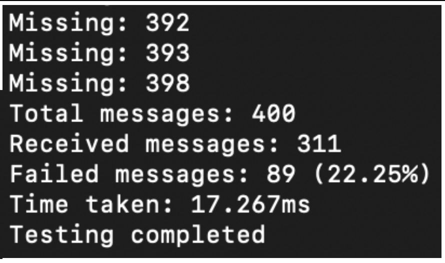
```
Total messages: 180
Received messages: 180
Failed messages: 0 (0.00000%)
Time taken: 13.021ms
Testing completed
```



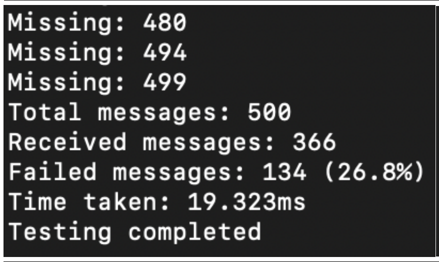
```
Total messages: 200
Received messages: 200
Failed messages: 0 (0.00000%)
Time taken: 8.041ms
Testing completed
```



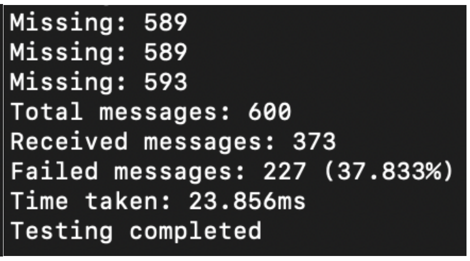
```
Total messages: 300
Received messages: 300
Failed messages: 0 (0.00000%)
Time taken: 14.251ms
Testing completed
```



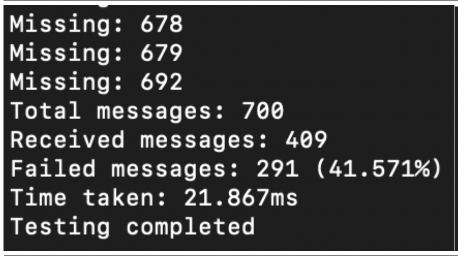
```
Missing: 392
Missing: 393
Missing: 398
Total messages: 400
Received messages: 311
Failed messages: 89 (22.25%)
Time taken: 17.267ms
Testing completed
```



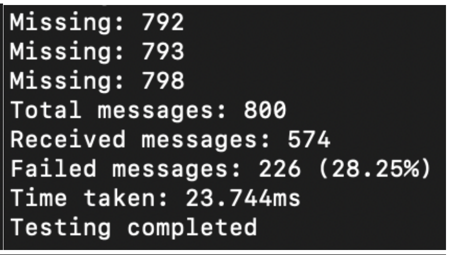
```
Missing: 480
Missing: 494
Missing: 499
Total messages: 500
Received messages: 366
Failed messages: 134 (26.8%)
Time taken: 19.323ms
Testing completed
```



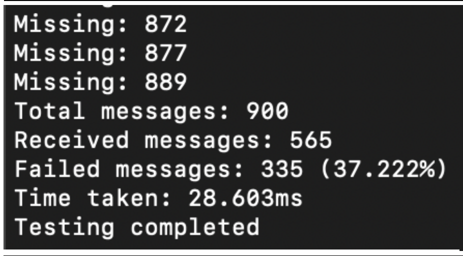
```
Missing: 589
Missing: 589
Missing: 593
Total messages: 600
Received messages: 373
Failed messages: 227 (37.833%)
Time taken: 23.856ms
Testing completed
```



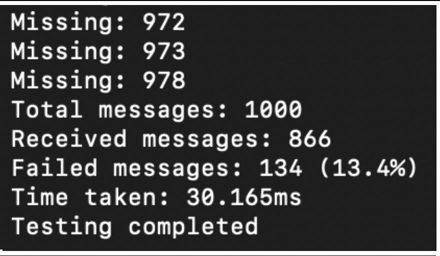
```
Missing: 678
Missing: 679
Missing: 692
Total messages: 700
Received messages: 409
Failed messages: 291 (41.571%)
Time taken: 21.867ms
Testing completed
```



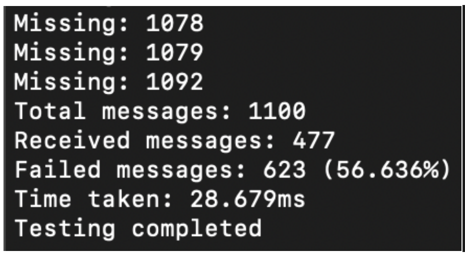
```
Missing: 792
Missing: 793
Missing: 798
Total messages: 800
Received messages: 574
Failed messages: 226 (28.25%)
Time taken: 23.744ms
Testing completed
```



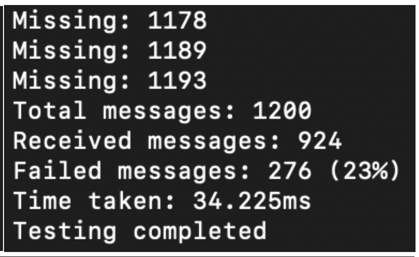
```
Missing: 872
Missing: 877
Missing: 889
Total messages: 900
Received messages: 565
Failed messages: 335 (37.222%)
Time taken: 28.603ms
Testing completed
```



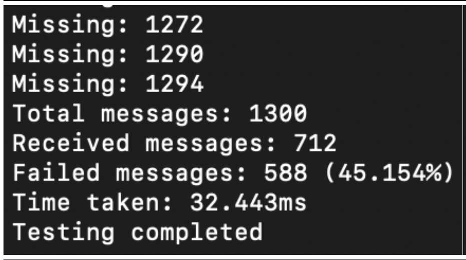
```
Missing: 972
Missing: 973
Missing: 978
Total messages: 1000
Received messages: 866
Failed messages: 134 (13.4%)
Time taken: 30.165ms
Testing completed
```



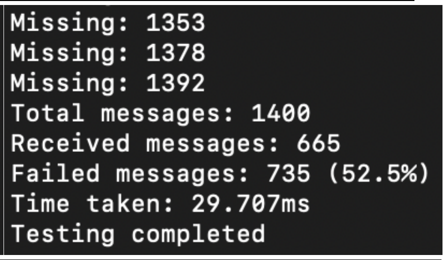
```
Missing: 1078
Missing: 1079
Missing: 1092
Total messages: 1100
Received messages: 477
Failed messages: 623 (56.636%)
Time taken: 28.679ms
Testing completed
```



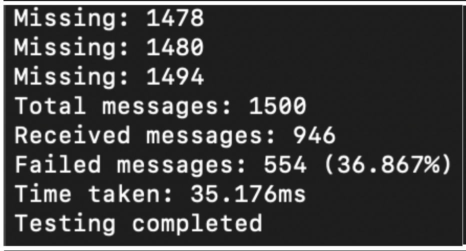
```
Missing: 1178
Missing: 1189
Missing: 1193
Total messages: 1200
Received messages: 924
Failed messages: 276 (23%)
Time taken: 34.225ms
Testing completed
```



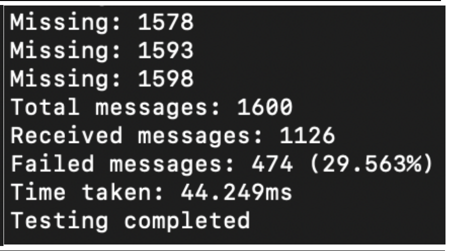
```
Missing: 1272
Missing: 1290
Missing: 1294
Total messages: 1300
Received messages: 712
Failed messages: 588 (45.154%)
Time taken: 32.443ms
Testing completed
```



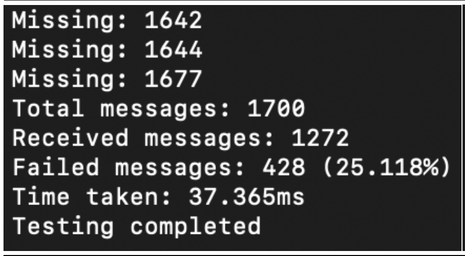
```
Missing: 1353
Missing: 1378
Missing: 1392
Total messages: 1400
Received messages: 665
Failed messages: 735 (52.5%)
Time taken: 29.707ms
Testing completed
```



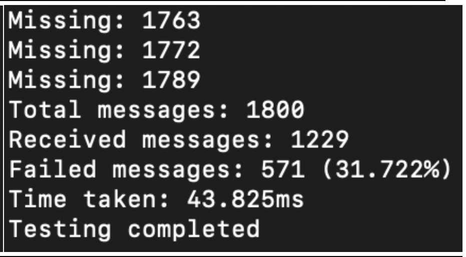
```
Missing: 1478
Missing: 1480
Missing: 1494
Total messages: 1500
Received messages: 946
Failed messages: 554 (36.867%)
Time taken: 35.176ms
Testing completed
```



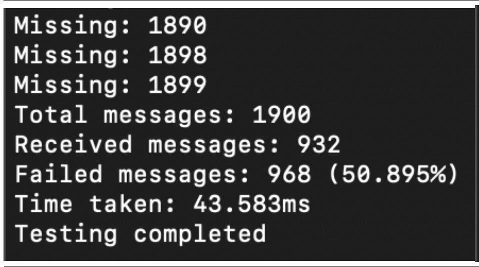
```
Missing: 1578
Missing: 1593
Missing: 1598
Total messages: 1600
Received messages: 1126
Failed messages: 474 (29.563%)
Time taken: 44.249ms
Testing completed
```



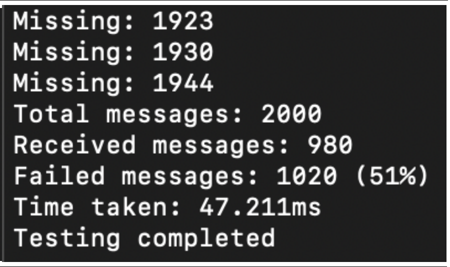
```
Missing: 1642
Missing: 1644
Missing: 1677
Total messages: 1700
Received messages: 1272
Failed messages: 428 (25.118%)
Time taken: 37.365ms
Testing completed
```



```
Missing: 1763
Missing: 1772
Missing: 1789
Total messages: 1800
Received messages: 1229
Failed messages: 571 (31.722%)
Time taken: 43.825ms
Testing completed
```

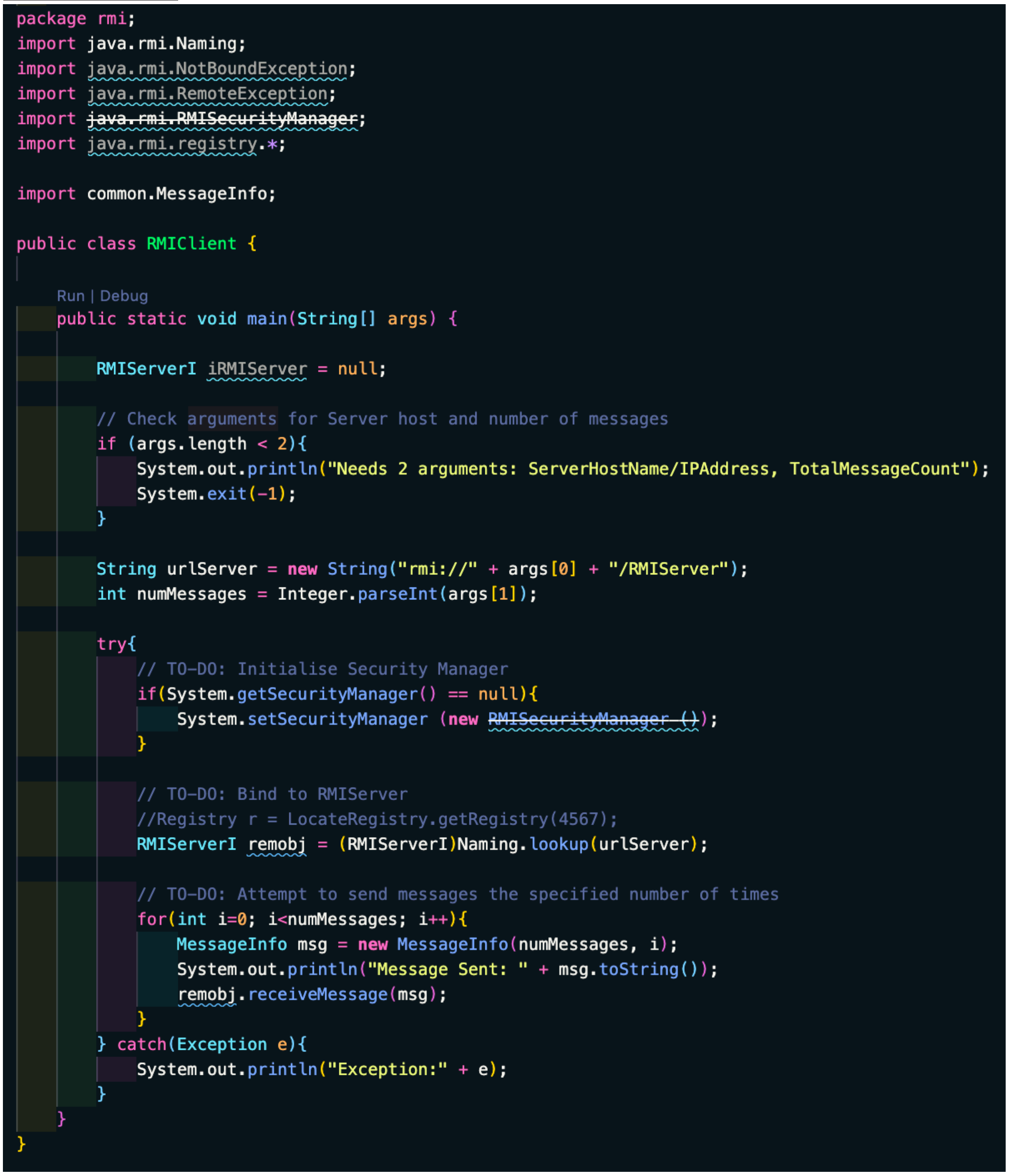


```
Missing: 1890
Missing: 1898
Missing: 1899
Total messages: 1900
Received messages: 932
Failed messages: 968 (50.895%)
Time taken: 43.583ms
Testing completed
```



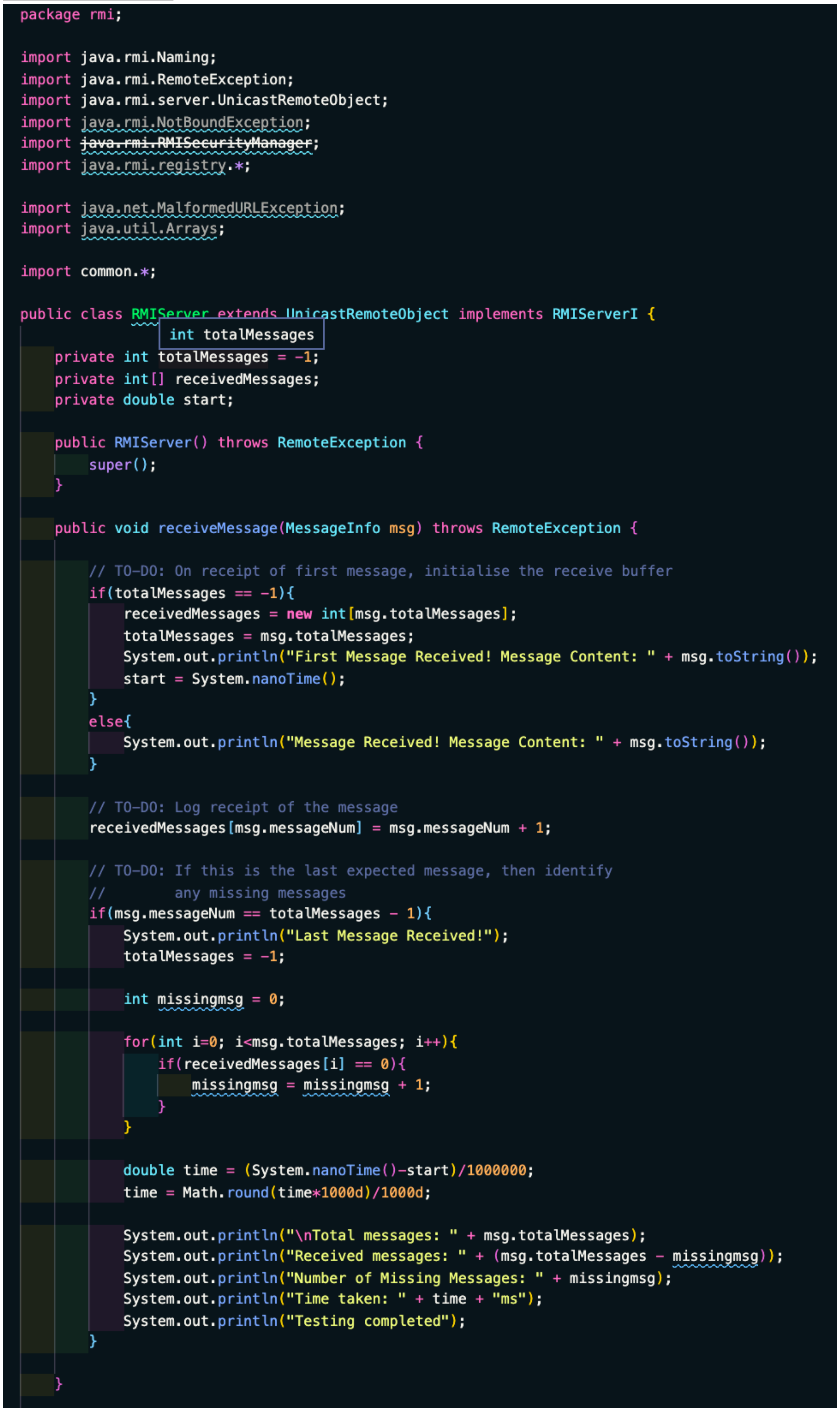
```
Missing: 1923
Missing: 1930
Missing: 1944
Total messages: 2000
Received messages: 980
Failed messages: 1020 (51%)
Time taken: 47.211ms
Testing completed
```

RMI Client

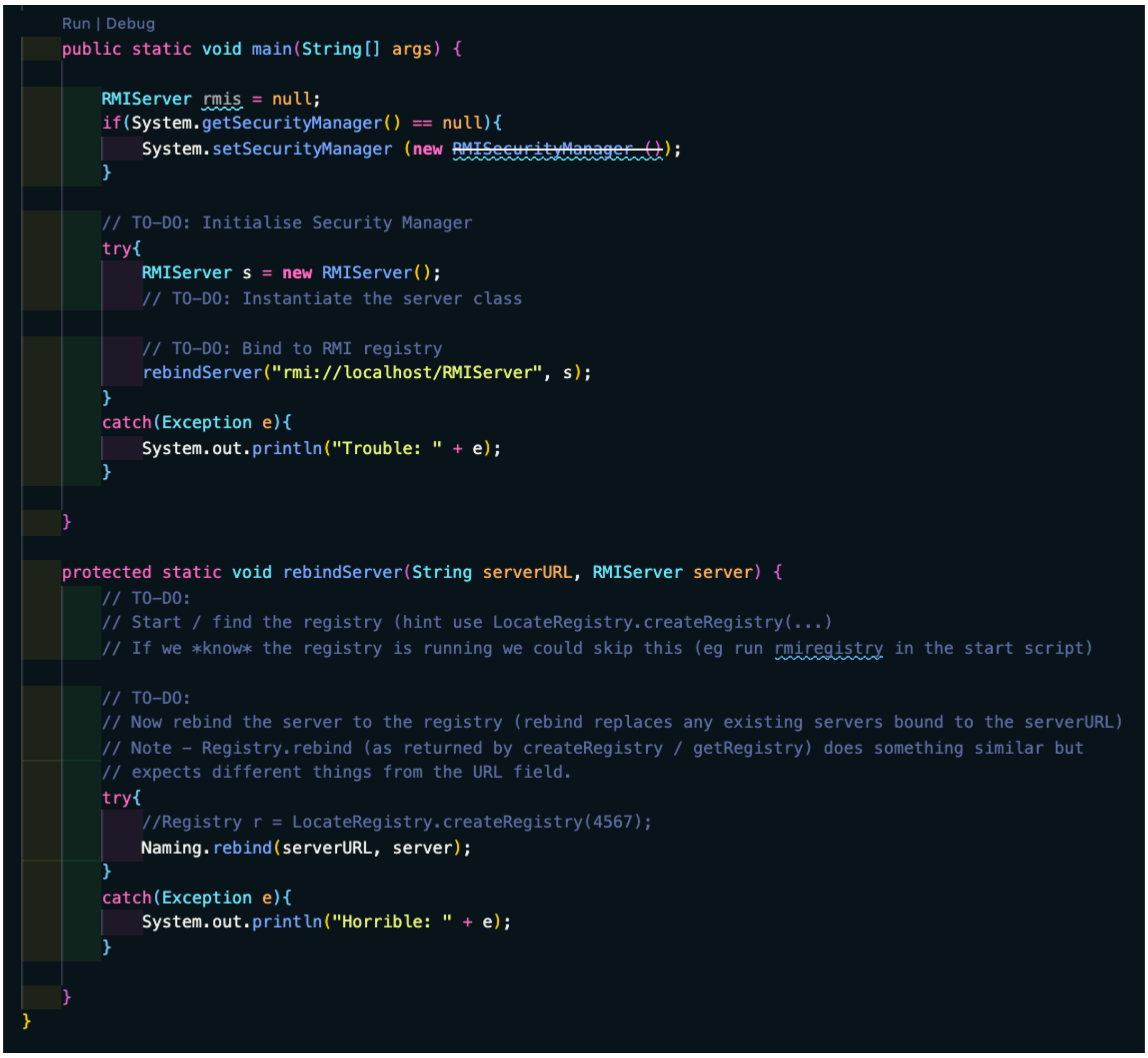


```java
package rmi;
import java.rmi.Naming;
import java.rmi.NotBoundException;
import java.rmi.RemoteException;
import ~~java.rmi.RMISecurityManager~~;
import java.rmi.registry.*;

import common.MessageInfo;

public class RMIClient {

    Run | Debug
    public static void main(String[] args) {

        RMIServerI iRMIServer = null;

        // Check arguments for Server host and number of messages
        if (args.length < 2){
            System.out.println("Needs 2 arguments: ServerHostName/IPAddress, TotalMessageCount");
            System.exit(-1);
        }

        String urlServer = new String("rmi://" + args[0] + "/RMIServer");
        int numMessages = Integer.parseInt(args[1]);

        try{
            // TO-DO: Initialise Security Manager
            if(System.getSecurityManager() == null){
                System.setSecurityManager (new ~~RMISecurityManager ()~~);
            }

            // TO-DO: Bind to RMIServer
            //Registry r = LocateRegistry.getRegistry(4567);
            RMIServerI remobj = (RMIServerI)Naming.lookup(urlServer);

            // TO-DO: Attempt to send messages the specified number of times
            for(int i=0; i<numMessages; i++){
                MessageInfo msg = new MessageInfo(numMessages, i);
                System.out.println("Message Sent: " + msg.toString());
                remobj.receiveMessage(msg);
            }
        } catch(Exception e){
            System.out.println("Exception:" + e);
        }
    }
}
```

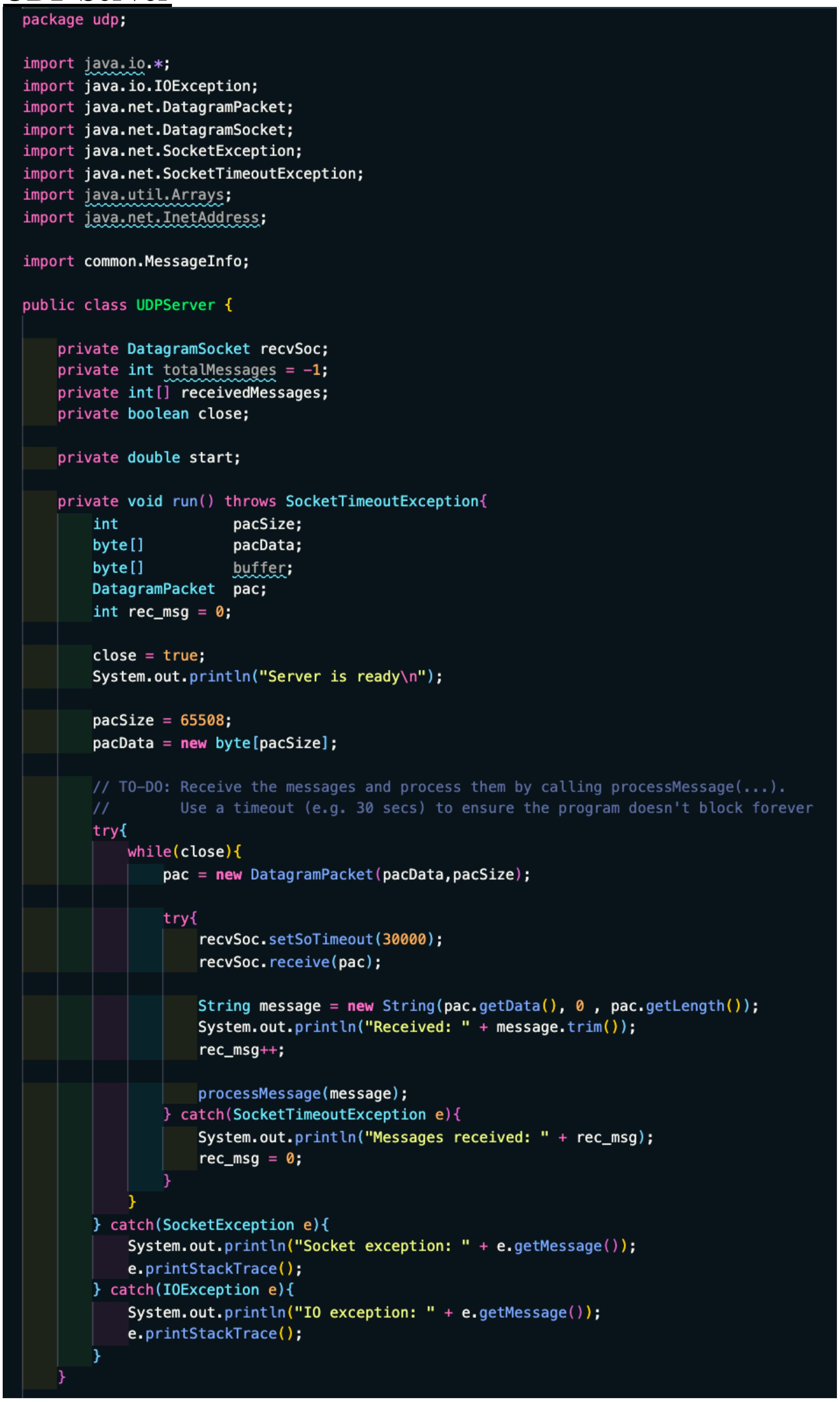
RMI Server



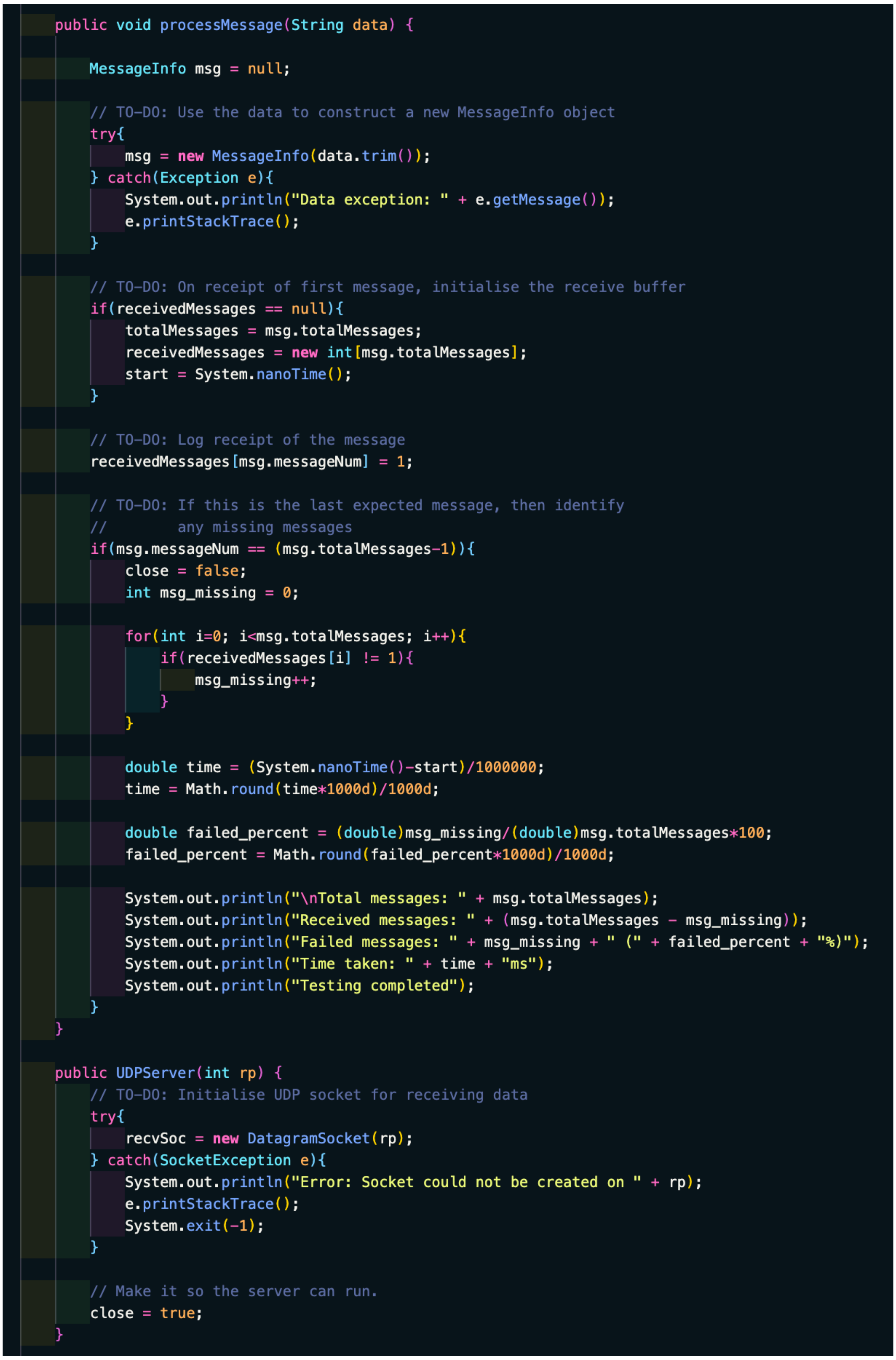
```java
package rmi;

import java.rmi.Naming;
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;
import java.rmi.NotBoundException;
import ~~java.rmi.RMISecurityManager~~;
import java.rmi.registry.*;

import java.net.MalformedURLException;
import java.util.Arrays;

import common.*;

public class RMIServer extends UnicastRemoteObject implements RMIServerI {
                                  int totalMessages
    private int totalMessages = -1;
    private int[] receivedMessages;
    private double start;

    public RMIServer() throws RemoteException {
        super();
    }

    public void receiveMessage(MessageInfo msg) throws RemoteException {

        // TO-DO: On receipt of first message, initialise the receive buffer
        if(totalMessages == -1){
            receivedMessages = new int[msg.totalMessages];
            totalMessages = msg.totalMessages;
            System.out.println("First Message Received! Message Content: " + msg.toString());
            start = System.nanoTime();
        }
        else{
            System.out.println("Message Received! Message Content: " + msg.toString());
        }

        // TO-DO: Log receipt of the message
        receivedMessages[msg.messageNum] = msg.messageNum + 1;

        // TO-DO: If this is the last expected message, then identify
        //        any missing messages
        if(msg.messageNum == totalMessages - 1){
            System.out.println("Last Message Received!");
            totalMessages = -1;

            int missingmsg = 0;

            for(int i=0; i<msg.totalMessages; i++){
                if(receivedMessages[i] == 0){
                    missingmsg = missingmsg + 1;
                }
            }

            double time = (System.nanoTime()-start)/1000000;
            time = Math.round(time*1000d)/1000d;

            System.out.println("\nTotal messages: " + msg.totalMessages);
            System.out.println("Received messages: " + (msg.totalMessages - missingmsg));
            System.out.println("Number of Missing Messages: " + missingmsg);
            System.out.println("Time taken: " + time + "ms");
            System.out.println("Testing completed");
        }

    }
```


```
Run | Debug
public static void main(String[] args) {

    RMIServer rmis = null;
    if(System.getSecurityManager() == null){
        System.setSecurityManager (new ~~RMISecurityManager ()~~);
    }

    // TO-DO: Initialise Security Manager
    try{
        RMIServer s = new RMIServer();
        // TO-DO: Instantiate the server class

        // TO-DO: Bind to RMI registry
        rebindServer("rmi://localhost/RMIServer", s);
    }
    catch(Exception e){
        System.out.println("Trouble: " + e);
    }

}

protected static void rebindServer(String serverURL, RMIServer server) {
    // TO-DO:
    // Start / find the registry (hint use LocateRegistry.createRegistry(...)
    // If we *know* the registry is running we could skip this (eg run rmiregistry in the start script)

    // TO-DO:
    // Now rebind the server to the registry (rebind replaces any existing servers bound to the serverURL)
    // Note - Registry.rebind (as returned by createRegistry / getRegistry) does something similar but
    // expects different things from the URL field.
    try{
        //Registry r = LocateRegistry.createRegistry(4567);
        Naming.rebind(serverURL, server);
    }
    catch(Exception e){
        System.out.println("Horrible: " + e);
    }

}
}
```

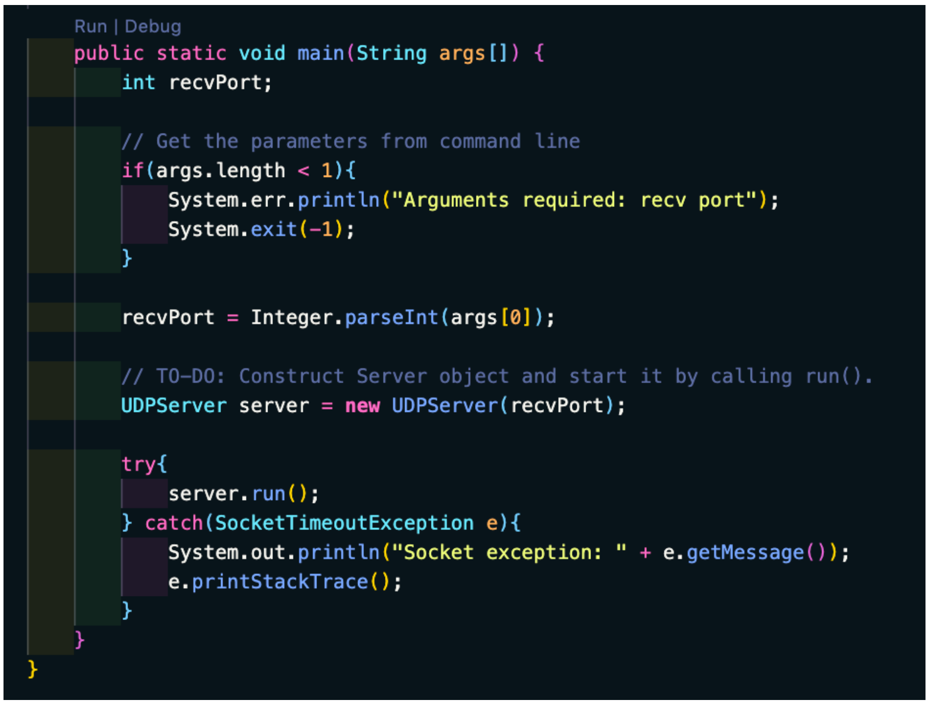
UDP Server



```java
package udp;

import java.io.*;
import java.io.IOException;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.SocketException;
import java.net.SocketTimeoutException;
import java.util.Arrays;
import java.net.InetAddress;

import common.MessageInfo;

public class UDPServer {

    private DatagramSocket recvSoc;
    private int totalMessages = -1;
    private int[] receivedMessages;
    private boolean close;

    private double start;

    private void run() throws SocketTimeoutException{
        int             pacSize;
        byte[]          pacData;
        byte[]          buffer;
        DatagramPacket  pac;
        int rec_msg = 0;

        close = true;
        System.out.println("Server is ready\n");

        pacSize = 65508;
        pacData = new byte[pacSize];

        // TO-DO: Receive the messages and process them by calling processMessage(...).
        //        Use a timeout (e.g. 30 secs) to ensure the program doesn't block forever
        try{
            while(close){
                pac = new DatagramPacket(pacData,pacSize);

                try{
                    recvSoc.setSoTimeout(30000);
                    recvSoc.receive(pac);

                    String message = new String(pac.getData(), 0 , pac.getLength());
                    System.out.println("Received: " + message.trim());
                    rec_msg++;

                    processMessage(message);
                } catch(SocketTimeoutException e){
                    System.out.println("Messages received: " + rec_msg);
                    rec_msg = 0;
                }
            }
        } catch(SocketException e){
            System.out.println("Socket exception: " + e.getMessage());
            e.printStackTrace();
        } catch(IOException e){
            System.out.println("IO exception: " + e.getMessage());
            e.printStackTrace();
        }
    }
```

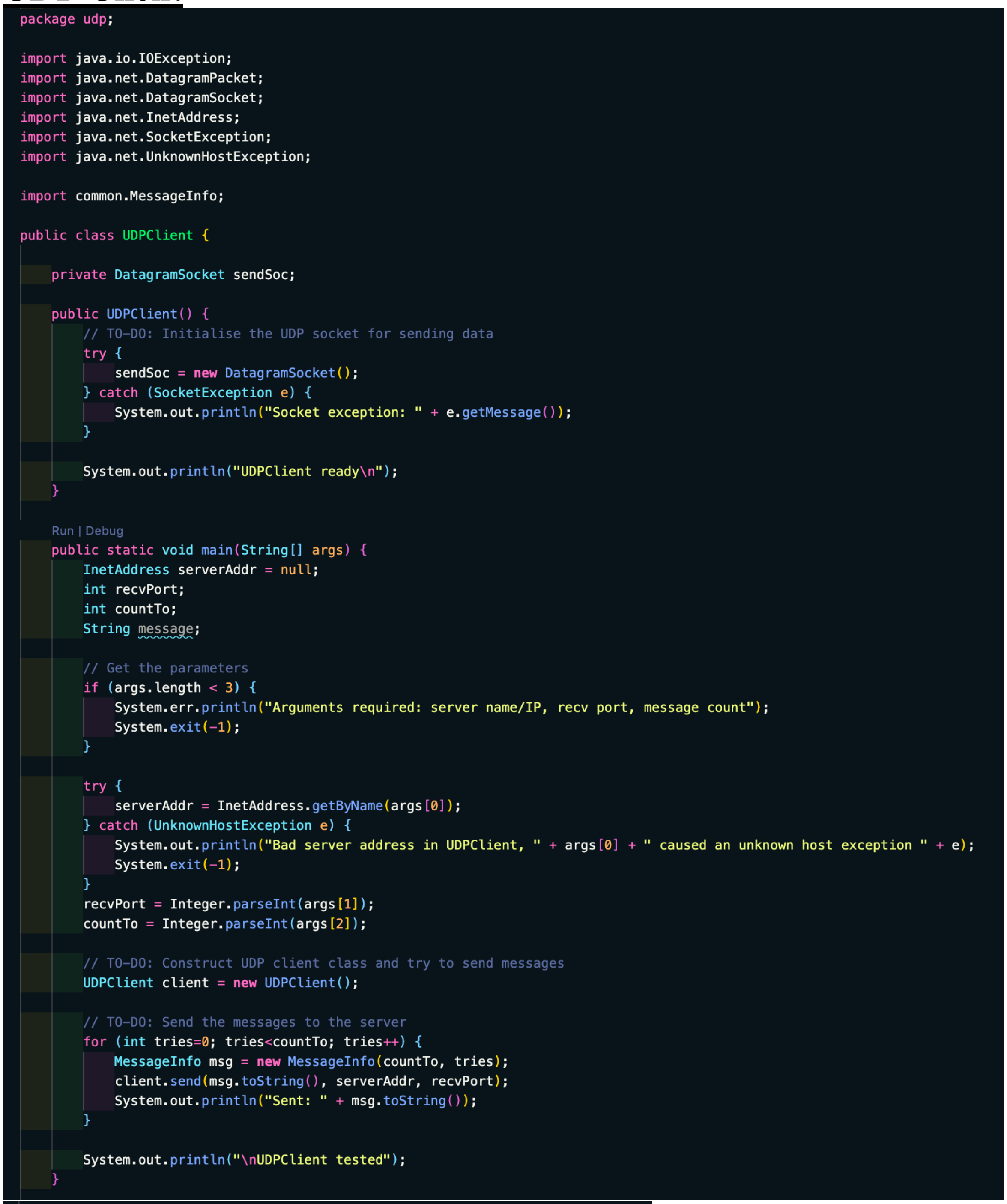


```java
public void processMessage(String data) {

    MessageInfo msg = null;

    // TO-DO: Use the data to construct a new MessageInfo object
    try{
        msg = new MessageInfo(data.trim());
    } catch(Exception e){
        System.out.println("Data exception: " + e.getMessage());
        e.printStackTrace();
    }

    // TO-DO: On receipt of first message, initialise the receive buffer
    if(receivedMessages == null){
        totalMessages = msg.totalMessages;
        receivedMessages = new int[msg.totalMessages];
        start = System.nanoTime();
    }

    // TO-DO: Log receipt of the message
    receivedMessages[msg.messageNum] = 1;

    // TO-DO: If this is the last expected message, then identify
    //        any missing messages
    if(msg.messageNum == (msg.totalMessages-1)){
        close = false;
        int msg_missing = 0;

        for(int i=0; i<msg.totalMessages; i++){
            if(receivedMessages[i] != 1){
                msg_missing++;
            }
        }

        double time = (System.nanoTime()-start)/1000000;
        time = Math.round(time*1000d)/1000d;

        double failed_percent = (double)msg_missing/(double)msg.totalMessages*100;
        failed_percent = Math.round(failed_percent*1000d)/1000d;

        System.out.println("\nTotal messages: " + msg.totalMessages);
        System.out.println("Received messages: " + (msg.totalMessages - msg_missing));
        System.out.println("Failed messages: " + msg_missing + " (" + failed_percent + "%)");
        System.out.println("Time taken: " + time + "ms");
        System.out.println("Testing completed");
    }
}

public UDPServer(int rp) {
    // TO-DO: Initialise UDP socket for receiving data
    try{
        recvSoc = new DatagramSocket(rp);
    } catch(SocketException e){
        System.out.println("Error: Socket could not be created on " + rp);
        e.printStackTrace();
        System.exit(-1);
    }

    // Make it so the server can run.
    close = true;
}
```

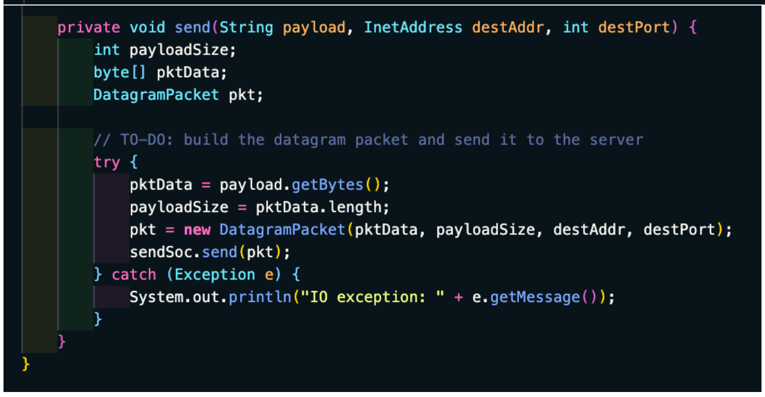


```
public static void main(String args[]) {
    int recvPort;

    // Get the parameters from command line
    if(args.length < 1){
        System.err.println("Arguments required: recv port");
        System.exit(-1);
    }

    recvPort = Integer.parseInt(args[0]);

    // TO-DO: Construct Server object and start it by calling run().
    UDPServer server = new UDPServer(recvPort);

    try{
        server.run();
    } catch(SocketTimeoutException e){
        System.out.println("Socket exception: " + e.getMessage());
        e.printStackTrace();
    }
}
}
```

UDP Client



```java
package udp;

import java.io.IOException;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.net.SocketException;
import java.net.UnknownHostException;

import common.MessageInfo;

public class UDPClient {

    private DatagramSocket sendSoc;

    public UDPClient() {
        // TO-DO: Initialise the UDP socket for sending data
        try {
            sendSoc = new DatagramSocket();
        } catch (SocketException e) {
            System.out.println("Socket exception: " + e.getMessage());
        }

        System.out.println("UDPClient ready\n");
    }

    public static void main(String[] args) {
        InetAddress serverAddr = null;
        int recvPort;
        int countTo;
        String message;

        // Get the parameters
        if (args.length < 3) {
            System.err.println("Arguments required: server name/IP, recv port, message count");
            System.exit(-1);
        }

        try {
            serverAddr = InetAddress.getByName(args[0]);
        } catch (UnknownHostException e) {
            System.out.println("Bad server address in UDPClient, " + args[0] + " caused an unknown host exception " + e);
            System.exit(-1);
        }
        recvPort = Integer.parseInt(args[1]);
        countTo = Integer.parseInt(args[2]);

        // TO-DO: Construct UDP client class and try to send messages
        UDPClient client = new UDPClient();

        // TO-DO: Send the messages to the server
        for (int tries=0; tries<countTo; tries++) {
            MessageInfo msg = new MessageInfo(countTo, tries);
            client.send(msg.toString(), serverAddr, recvPort);
            System.out.println("Sent: " + msg.toString());
        }

        System.out.println("\nUDPClient tested");
    }
```



```java
    private void send(String payload, InetAddress destAddr, int destPort) {
        int payloadSize;
        byte[] pktData;
        DatagramPacket pkt;

        // TO-DO: build the datagram packet and send it to the server
        try {
            pktData = payload.getBytes();
            payloadSize = pktData.length;
            pkt = new DatagramPacket(pktData, payloadSize, destAddr, destPort);
            sendSoc.send(pkt);
        } catch (Exception e) {
            System.out.println("IO exception: " + e.getMessage());
        }
    }
}
```