Investigating Sending Methods by Remote Method Invocation (RMI) and User Datagram Protocol(UDP)

Calvin Chan(cc6815)
Yuxuan Chen(yc4416)

Introduction:

Sending messages between client and server, by Remote Method Invocation (RMI) and User Datagram Protocol (UDP), investigated by coding in Java, is introduced in this report. Different number of messages are sent between client and server by two different computers, and this report will analyse the performance, efficiency and ease of set-up of each type of system.

UDP:

UDP is based on a connectionless communication model, and there is no guarantee of delivering or ordering, especially in unreliable network.
There are several ways of causing the package loss.

Firstly, congestion could take place easily. UDP lacks overhead, so it can send packets very quickly. Especially in an overloaded router, it receives data in a faster rate than sending. Also， many other users are using the school computers and Internet at the same time, the loading pressure is very high.
Secondly, UDP does not retransmit the lost messages, so this causes the loss rate to increase.

RMI:

Instead of sending messages locally, RMI calls a remote method implementation using stub and skeleton to transmit data between client and server.

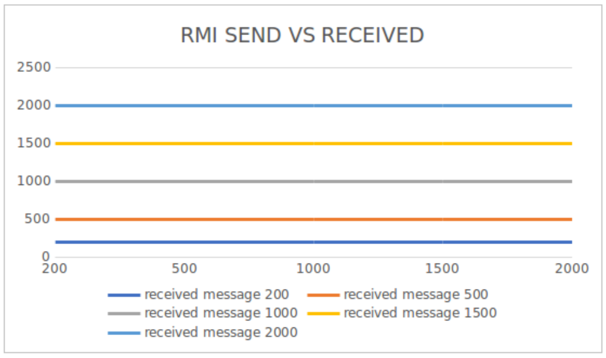
The **stub** acts as a local proxy for a remote object, and it carries out method calls when a method is invoked from a caller(client). The **skeleton** then receives the call from network, and it dispatches the call to the actual remote object.

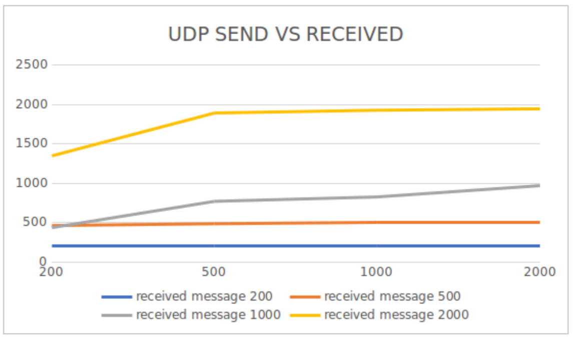
Server binds the service with the RMI Registry, and Security Manager should work for both server and client when skeleton catches the calls sent by the stub.

UDP against RMI (By observation of our testing):

We can easily spot the characteristics in the screen shots below. For UDP, when the distance increases, the loss rate will also increase, but for the RMI, it does not have any lost packages, which means that UDP has a high rate of loss of messages compared to RMI, but it serves quickly.
It is suitable for applications that need fast and efficient transmission such as games.

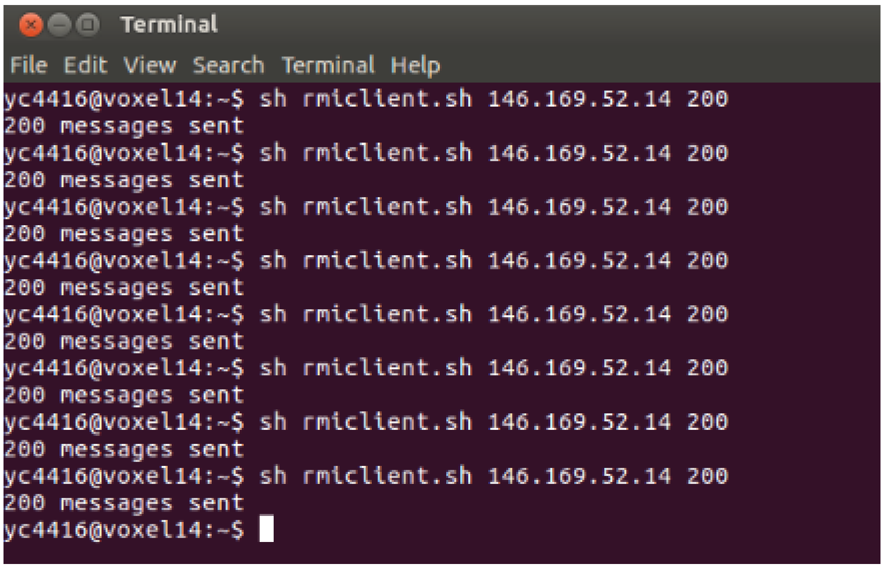
When we are sending packages by RMI, the time waiting is longer.
Although RMI is reliable, it is not suitable for the system requiring lots of exchanging of message such as funds transfer.



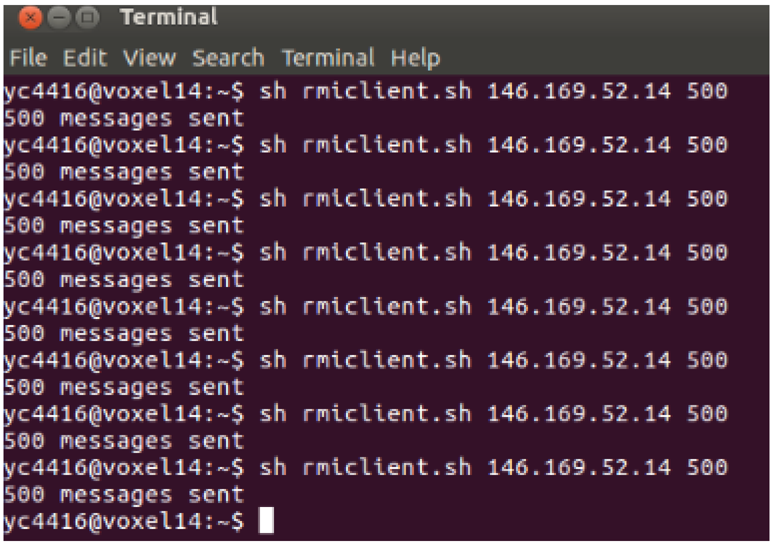


RMI screen shots:

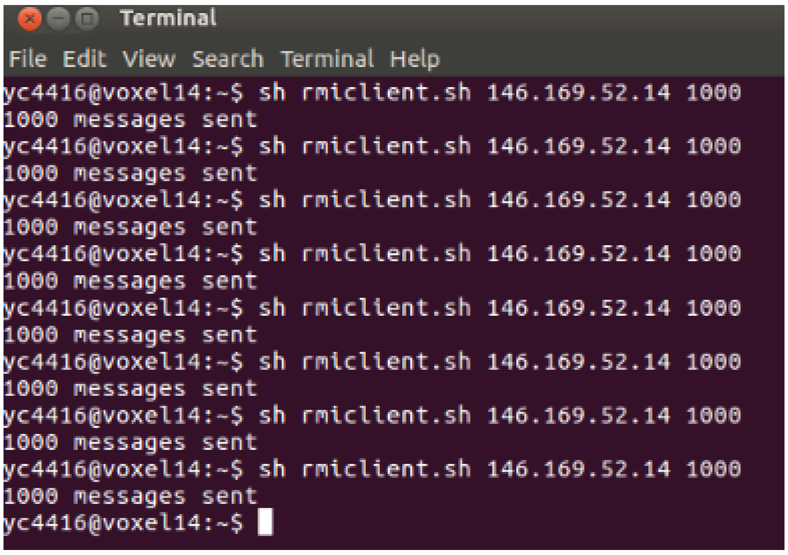
RMI Client:



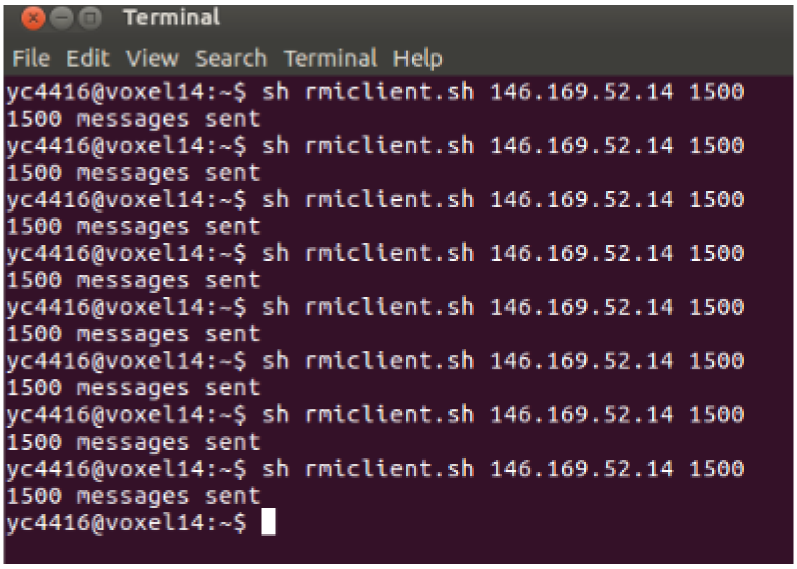
```
yc4416@voxel14:~$ sh rmiclient.sh 146.169.52.14 200
200 messages sent
yc4416@voxel14:~$ sh rmiclient.sh 146.169.52.14 200
200 messages sent
yc4416@voxel14:~$ sh rmiclient.sh 146.169.52.14 200
200 messages sent
yc4416@voxel14:~$ sh rmiclient.sh 146.169.52.14 200
200 messages sent
yc4416@voxel14:~$ sh rmiclient.sh 146.169.52.14 200
200 messages sent
yc4416@voxel14:~$ sh rmiclient.sh 146.169.52.14 200
200 messages sent
yc4416@voxel14:~$ sh rmiclient.sh 146.169.52.14 200
200 messages sent
yc4416@voxel14:~$ sh rmiclient.sh 146.169.52.14 200
200 messages sent
yc4416@voxel14:~$
```



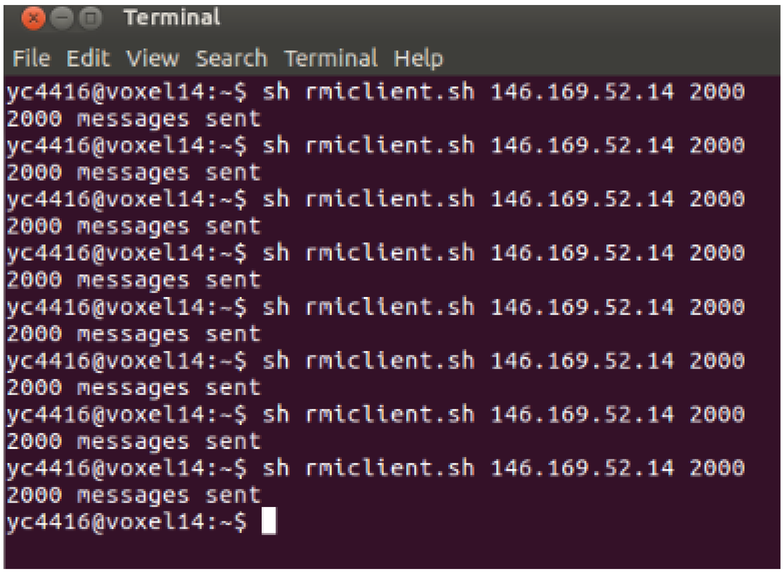
```
yc4416@voxel14:~$ sh rmiclient.sh 146.169.52.14 500
500 messages sent
yc4416@voxel14:~$ sh rmiclient.sh 146.169.52.14 500
500 messages sent
yc4416@voxel14:~$ sh rmiclient.sh 146.169.52.14 500
500 messages sent
yc4416@voxel14:~$ sh rmiclient.sh 146.169.52.14 500
500 messages sent
yc4416@voxel14:~$ sh rmiclient.sh 146.169.52.14 500
500 messages sent
yc4416@voxel14:~$ sh rmiclient.sh 146.169.52.14 500
500 messages sent
yc4416@voxel14:~$ sh rmiclient.sh 146.169.52.14 500
500 messages sent
yc4416@voxel14:~$ sh rmiclient.sh 146.169.52.14 500
500 messages sent
yc4416@voxel14:~$
```



```
yc4416@voxel14:~$ sh rmiclient.sh 146.169.52.14 1000
1000 messages sent
yc4416@voxel14:~$ sh rmiclient.sh 146.169.52.14 1000
1000 messages sent
yc4416@voxel14:~$ sh rmiclient.sh 146.169.52.14 1000
1000 messages sent
yc4416@voxel14:~$ sh rmiclient.sh 146.169.52.14 1000
1000 messages sent
yc4416@voxel14:~$ sh rmiclient.sh 146.169.52.14 1000
1000 messages sent
yc4416@voxel14:~$ sh rmiclient.sh 146.169.52.14 1000
1000 messages sent
yc4416@voxel14:~$ sh rmiclient.sh 146.169.52.14 1000
1000 messages sent
yc4416@voxel14:~$ sh rmiclient.sh 146.169.52.14 1000
1000 messages sent
yc4416@voxel14:~$
```

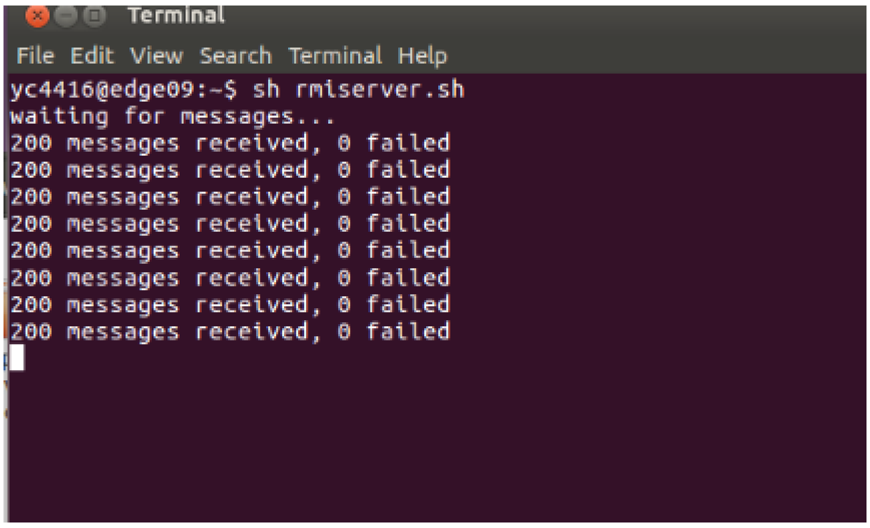


```
yc4416@voxel14:~$ sh rmiclient.sh 146.169.52.14 1500
1500 messages sent
yc4416@voxel14:~$ sh rmiclient.sh 146.169.52.14 1500
1500 messages sent
yc4416@voxel14:~$ sh rmiclient.sh 146.169.52.14 1500
1500 messages sent
yc4416@voxel14:~$ sh rmiclient.sh 146.169.52.14 1500
1500 messages sent
yc4416@voxel14:~$ sh rmiclient.sh 146.169.52.14 1500
1500 messages sent
yc4416@voxel14:~$ sh rmiclient.sh 146.169.52.14 1500
1500 messages sent
yc4416@voxel14:~$ sh rmiclient.sh 146.169.52.14 1500
1500 messages sent
yc4416@voxel14:~$ sh rmiclient.sh 146.169.52.14 1500
1500 messages sent
yc4416@voxel14:~$
```

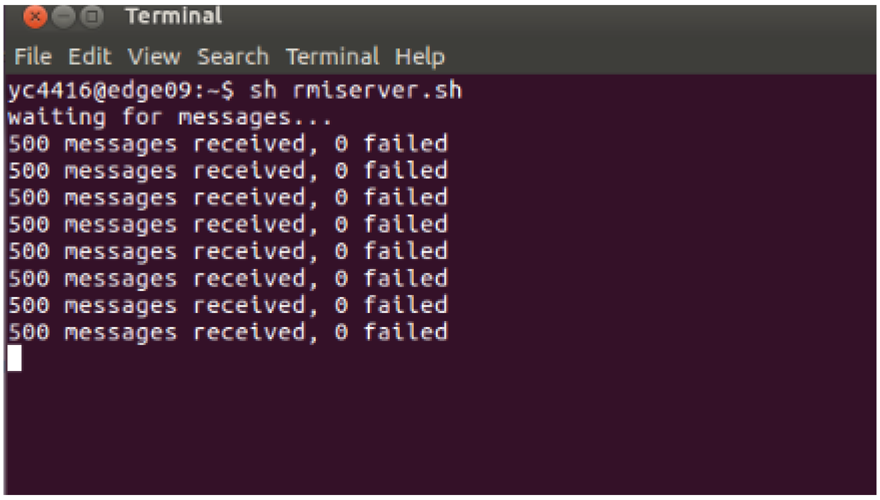


```
yc4416@voxel14:~$ sh rmiclient.sh 146.169.52.14 2000
2000 messages sent
yc4416@voxel14:~$ sh rmiclient.sh 146.169.52.14 2000
2000 messages sent
yc4416@voxel14:~$ sh rmiclient.sh 146.169.52.14 2000
2000 messages sent
yc4416@voxel14:~$ sh rmiclient.sh 146.169.52.14 2000
2000 messages sent
yc4416@voxel14:~$ sh rmiclient.sh 146.169.52.14 2000
2000 messages sent
yc4416@voxel14:~$ sh rmiclient.sh 146.169.52.14 2000
2000 messages sent
yc4416@voxel14:~$ sh rmiclient.sh 146.169.52.14 2000
2000 messages sent
yc4416@voxel14:~$ sh rmiclient.sh 146.169.52.14 2000
2000 messages sent
yc4416@voxel14:~$
```

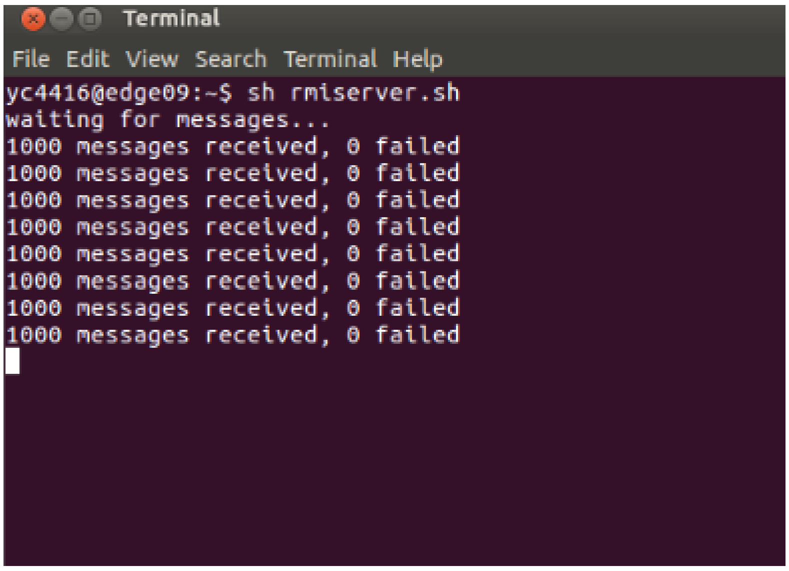
RMI Server:



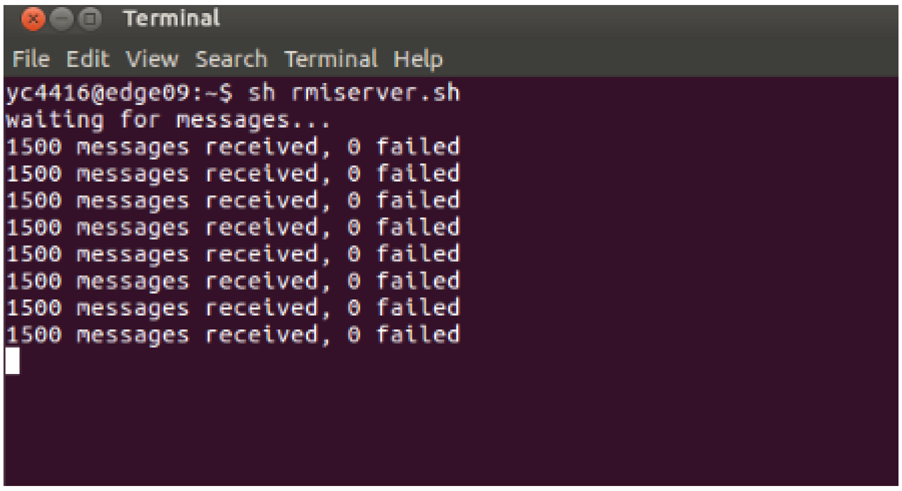
```
yc4416@edge09:~$ sh rmiserver.sh
waiting for messages...
200 messages received, 0 failed
200 messages received, 0 failed
200 messages received, 0 failed
200 messages received, 0 failed
200 messages received, 0 failed
200 messages received, 0 failed
200 messages received, 0 failed
200 messages received, 0 failed
```



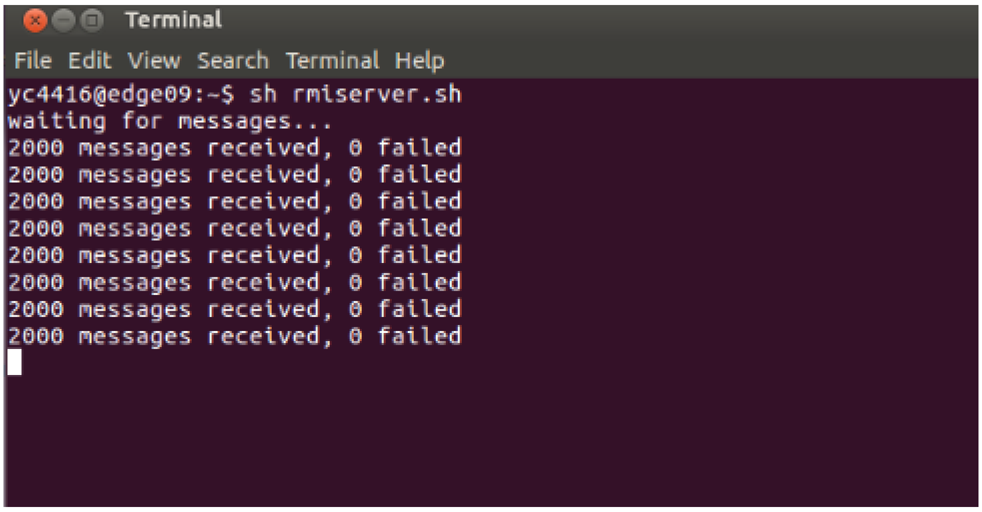
```
yc4416@edge09:~$ sh rmiserver.sh
waiting for messages...
500 messages received, 0 failed
500 messages received, 0 failed
500 messages received, 0 failed
500 messages received, 0 failed
500 messages received, 0 failed
500 messages received, 0 failed
500 messages received, 0 failed
500 messages received, 0 failed
```



```
yc4416@edge09:~$ sh rmiserver.sh
waiting for messages...
1000 messages received, 0 failed
1000 messages received, 0 failed
1000 messages received, 0 failed
1000 messages received, 0 failed
1000 messages received, 0 failed
1000 messages received, 0 failed
1000 messages received, 0 failed
1000 messages received, 0 failed
```

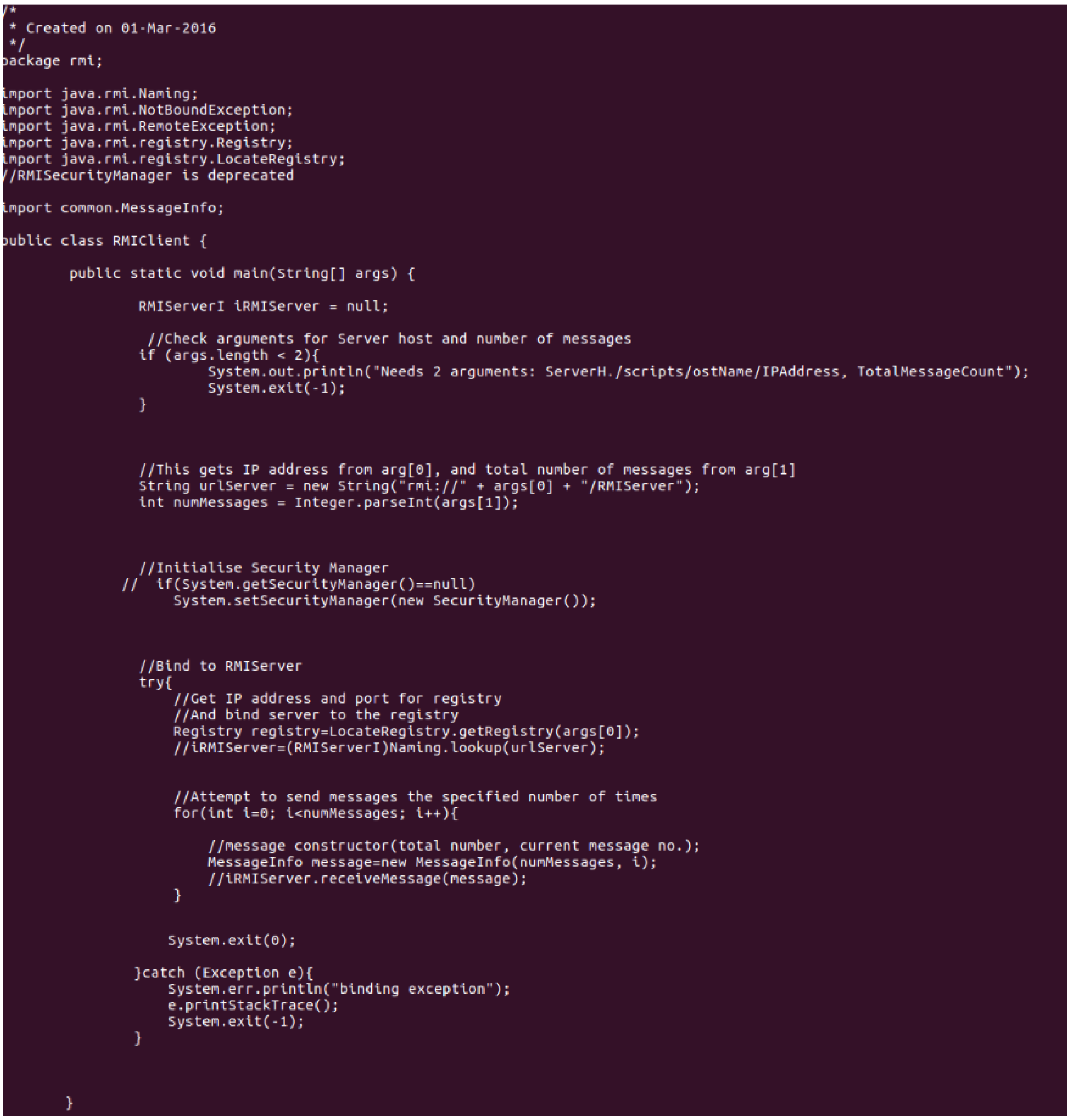


```
yc4416@edge09:~$ sh rmiserver.sh
waiting for messages...
1500 messages received, 0 failed
1500 messages received, 0 failed
1500 messages received, 0 failed
1500 messages received, 0 failed
1500 messages received, 0 failed
1500 messages received, 0 failed
1500 messages received, 0 failed
1500 messages received, 0 failed
```



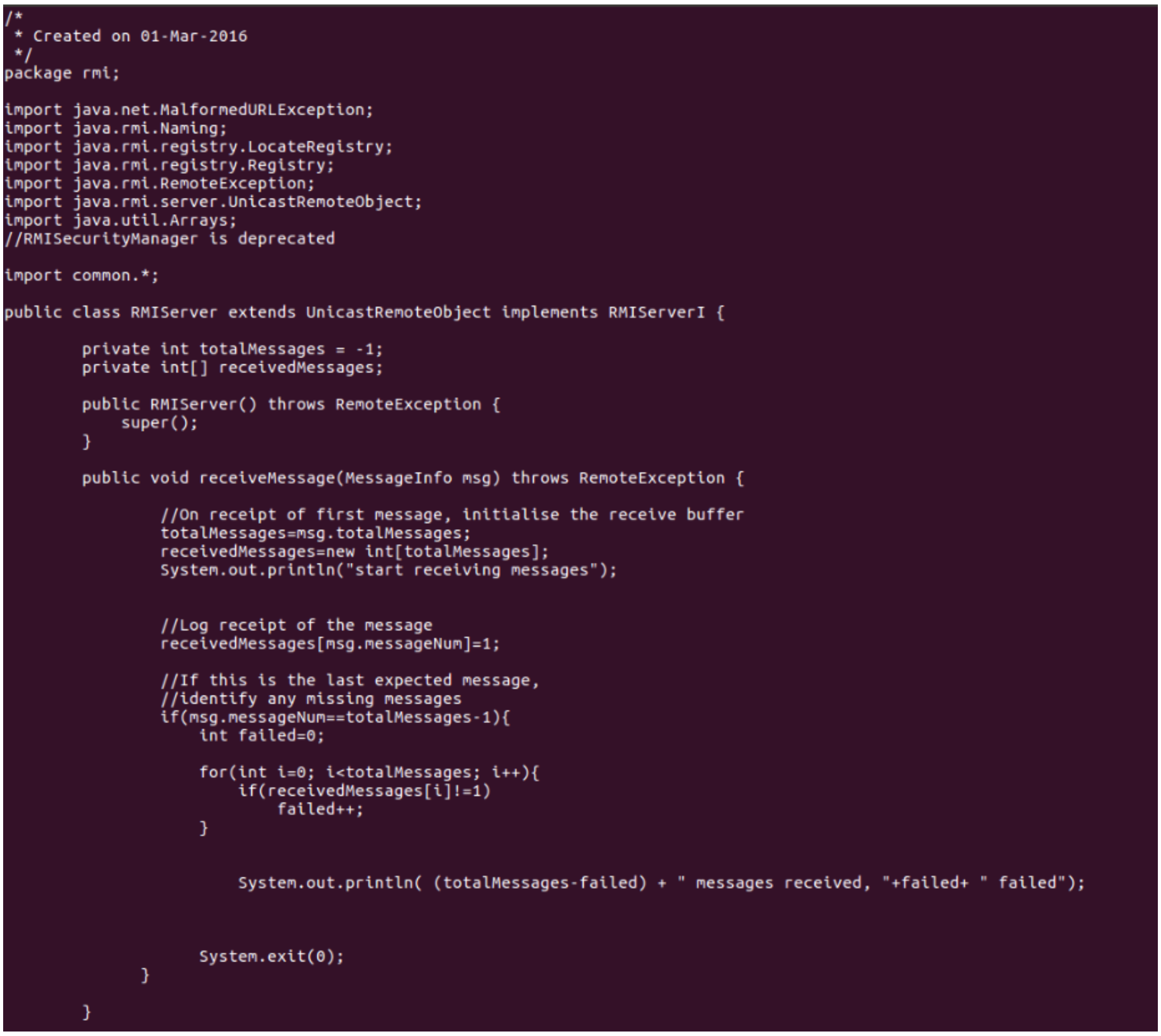
```
yc4416@edge09:~$ sh rmiserver.sh
waiting for messages...
2000 messages received, 0 failed
2000 messages received, 0 failed
2000 messages received, 0 failed
2000 messages received, 0 failed
2000 messages received, 0 failed
2000 messages received, 0 failed
2000 messages received, 0 failed
2000 messages received, 0 failed
```

RMI Client code:

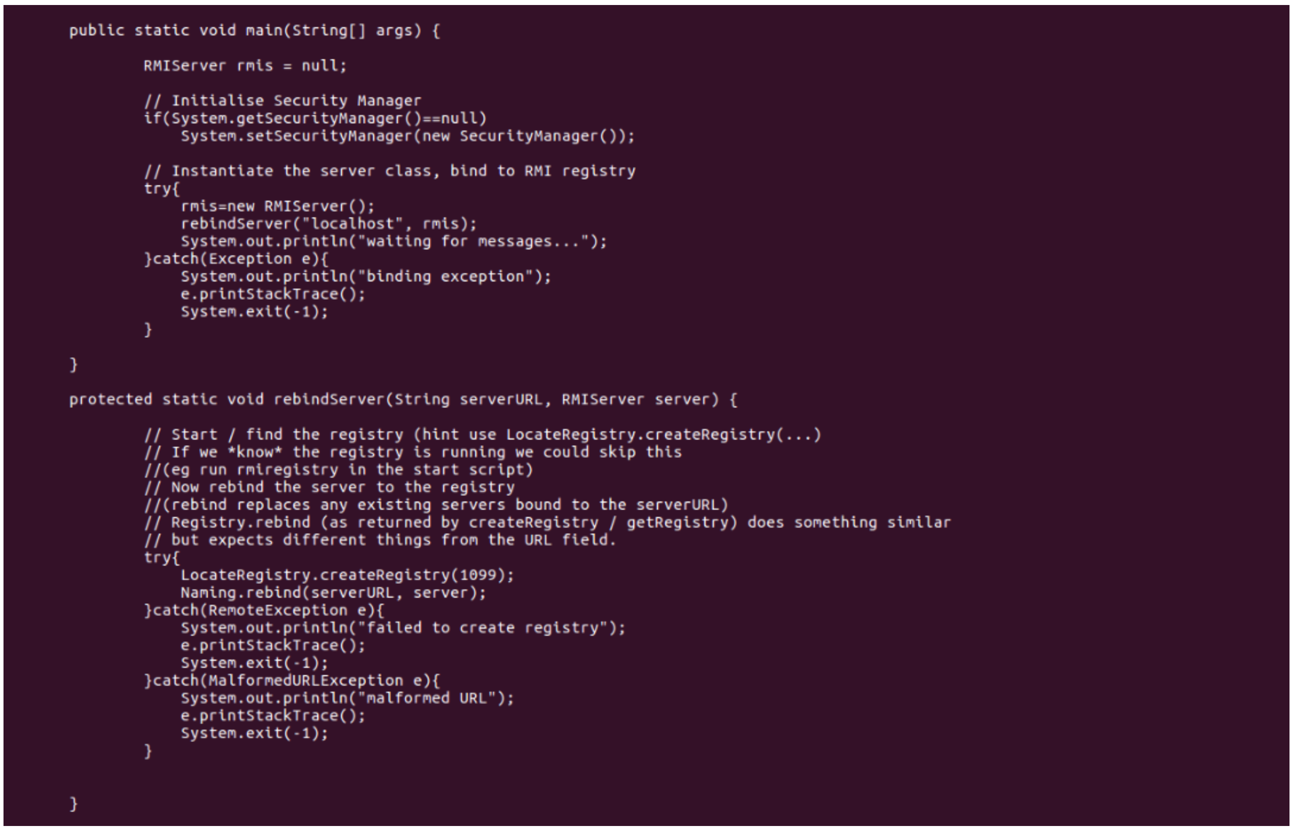


```
/*
 * Created on 01-Mar-2016
 */
package rmi;

import java.rmi.Naming;
import java.rmi.NotBoundException;
import java.rmi.RemoteException;
import java.rmi.registry.Registry;
import java.rmi.registry.LocateRegistry;
//RMISecurityManager is deprecated

import common.MessageInfo;

public class RMIClient {

	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: ServerH./scripts/ostName/IPAddress, TotalMessageCount");
			System.exit(-1);
		}


		//This gets IP address from arg[0], and total number of messages from arg[1]
		String urlServer = new String("rmi://" + args[0] + "/RMIServer");
		int numMessages = Integer.parseInt(args[1]);


		//Initialise Security Manager
	     //  if(System.getSecurityManager()==null)
		    System.setSecurityManager(new SecurityManager());


		//Bind to RMIServer
		try{
		    //Get IP address and port for registry
		    //And bind server to the registry
		    Registry registry=LocateRegistry.getRegistry(args[0]);
		    //iRMIServer=(RMIServerI)Naming.lookup(urlServer);


		    //Attempt to send messages the specified number of times
		    for(int i=0; i<numMessages; i++){

			//message constructor(total number, current message no.);
			MessageInfo message=new MessageInfo(numMessages, i);
			//iRMIServer.receiveMessage(message);
		    }


		   System.exit(0);

		}catch (Exception e){
		   System.err.println("binding exception");
		   e.printStackTrace();
		   System.exit(-1);
		}


	}
```

RMI Server code:

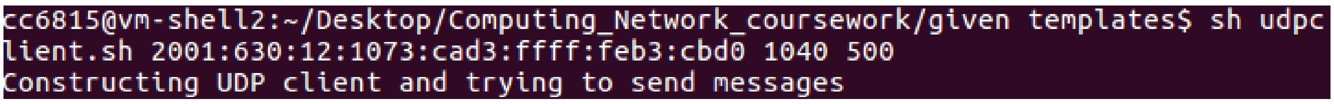


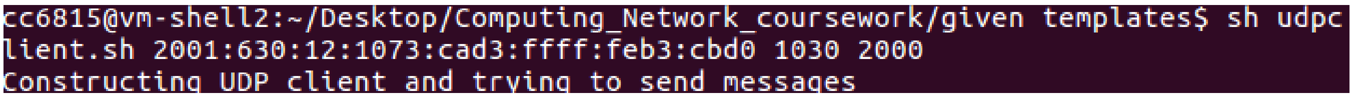
```
/*
 * Created on 01-Mar-2016
 */
package rmi;

import java.net.MalformedURLException;
import java.rmi.Naming;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;
import java.util.Arrays;
//RMISecurityManager is deprecated

import common.*;

public class RMIServer extends UnicastRemoteObject implements RMIServerI {

        private int totalMessages = -1;
        private int[] receivedMessages;

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

        public void receiveMessage(MessageInfo msg) throws RemoteException {

                //On receipt of first message, initialise the receive buffer
                totalMessages=msg.totalMessages;
                receivedMessages=new int[totalMessages];
                System.out.println("start receiving messages");


                //Log receipt of the message
                receivedMessages[msg.messageNum]=1;

                //If this is the last expected message,
                //identify any missing messages
                if(msg.messageNum==totalMessages-1){
                    int failed=0;

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


                        System.out.println( (totalMessages-failed) + " messages received, "+failed+ " failed");



                    System.exit(0);
              }

        }
```


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

            RMIServer rmis = null;

            // Initialise Security Manager
            if(System.getSecurityManager()==null)
                System.setSecurityManager(new SecurityManager());

            // Instantiate the server class, bind to RMI registry
            try{
                rmis=new RMIServer();
                rebindServer("localhost", rmis);
                System.out.println("waiting for messages...");
            }catch(Exception e){
                System.out.println("binding exception");
                e.printStackTrace();
                System.exit(-1);
            }

    }

    protected static void rebindServer(String serverURL, RMIServer server) {

            // 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)
            // Now rebind the server to the registry
            //(rebind replaces any existing servers bound to the serverURL)
            // Registry.rebind (as returned by createRegistry / getRegistry) does something similar
            // but expects different things from the URL field.
            try{
                LocateRegistry.createRegistry(1099);
                Naming.rebind(serverURL, server);
            }catch(RemoteException e){
                System.out.println("failed to create registry");
                e.printStackTrace();
                System.exit(-1);
            }catch(MalformedURLException e){
                System.out.println("malformed URL");
                e.printStackTrace();
                System.exit(-1);
            }


    }
```

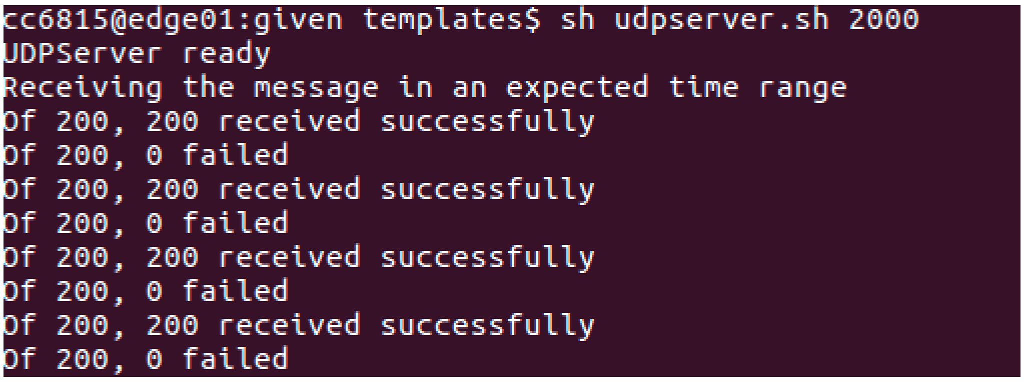
UDP screen shots:

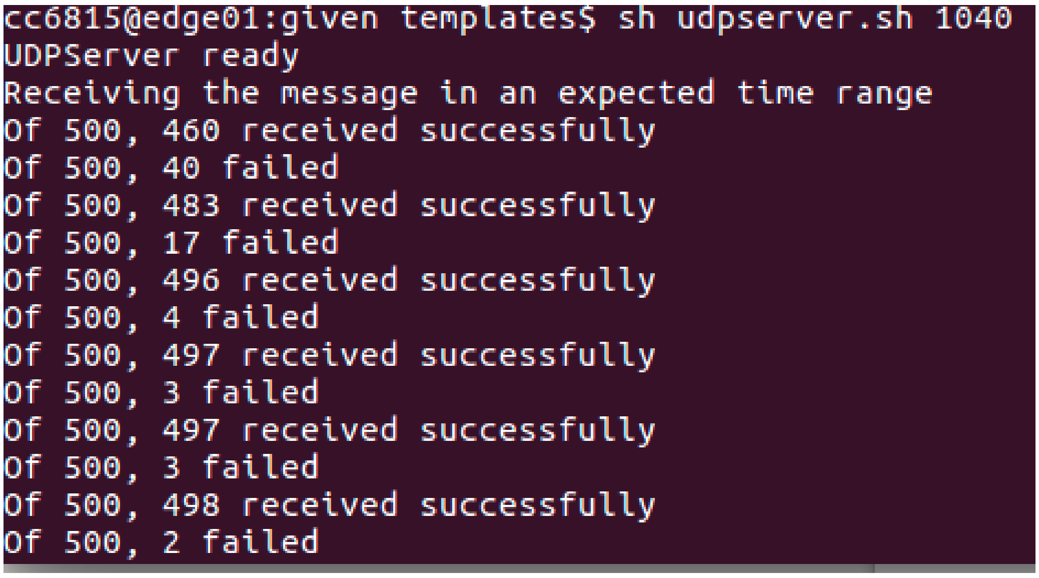
UDP client:


```
cc6815@vm-shell2:~/Desktop/Computing_Network_coursework/given templates$ sh udpc
lient.sh 2001:630:12:1073:cad3:ffff:feb3:cbd0 1040 500
Constructing UDP client and trying to send messages
```

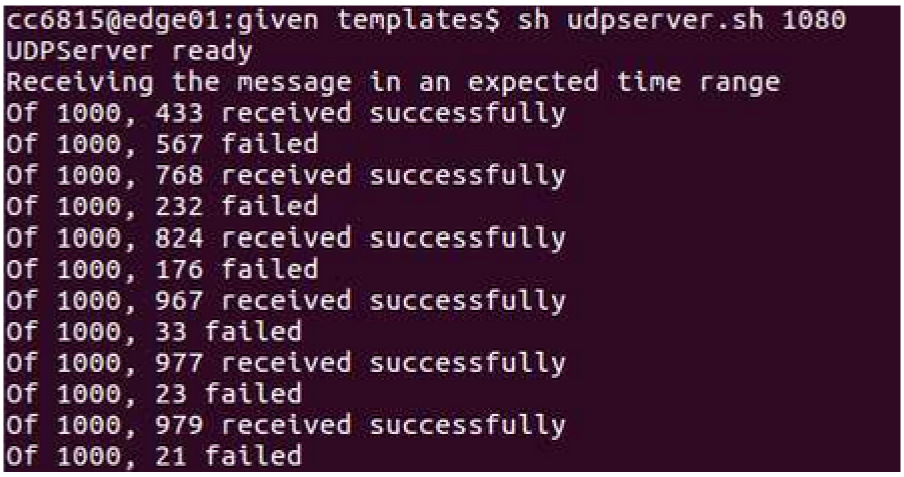

```
cc6815@vm-shell2:~/Desktop/Computing_Network_coursework/given templates$ sh udpc
lient.sh 2001:630:12:1073:cad3:ffff:feb3:cbd0 1030 2000
Constructing UDP client and trying to send messages
```

UDP server:

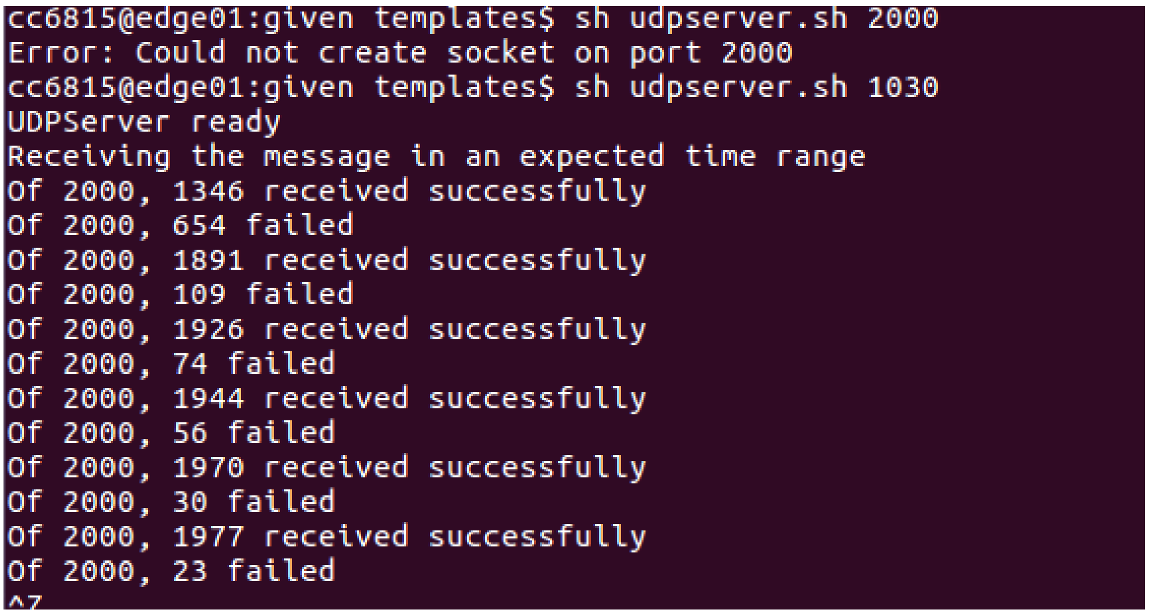

```
cc6815@edge01:given templates$ sh udpserver.sh 2000
UDPServer ready
Receiving the message in an expected time range
Of 200, 200 received successfully
Of 200, 0 failed
Of 200, 200 received successfully
Of 200, 0 failed
Of 200, 200 received successfully
Of 200, 0 failed
Of 200, 200 received successfully
Of 200, 0 failed
```



```
cc6815@edge01:given templates$ sh udpserver.sh 1040
UDPServer ready
Receiving the message in an expected time range
Of 500, 460 received successfully
Of 500, 40 failed
Of 500, 483 received successfully
Of 500, 17 failed
Of 500, 496 received successfully
Of 500, 4 failed
Of 500, 497 received successfully
Of 500, 3 failed
Of 500, 497 received successfully
Of 500, 3 failed
Of 500, 498 received successfully
Of 500, 2 failed
```

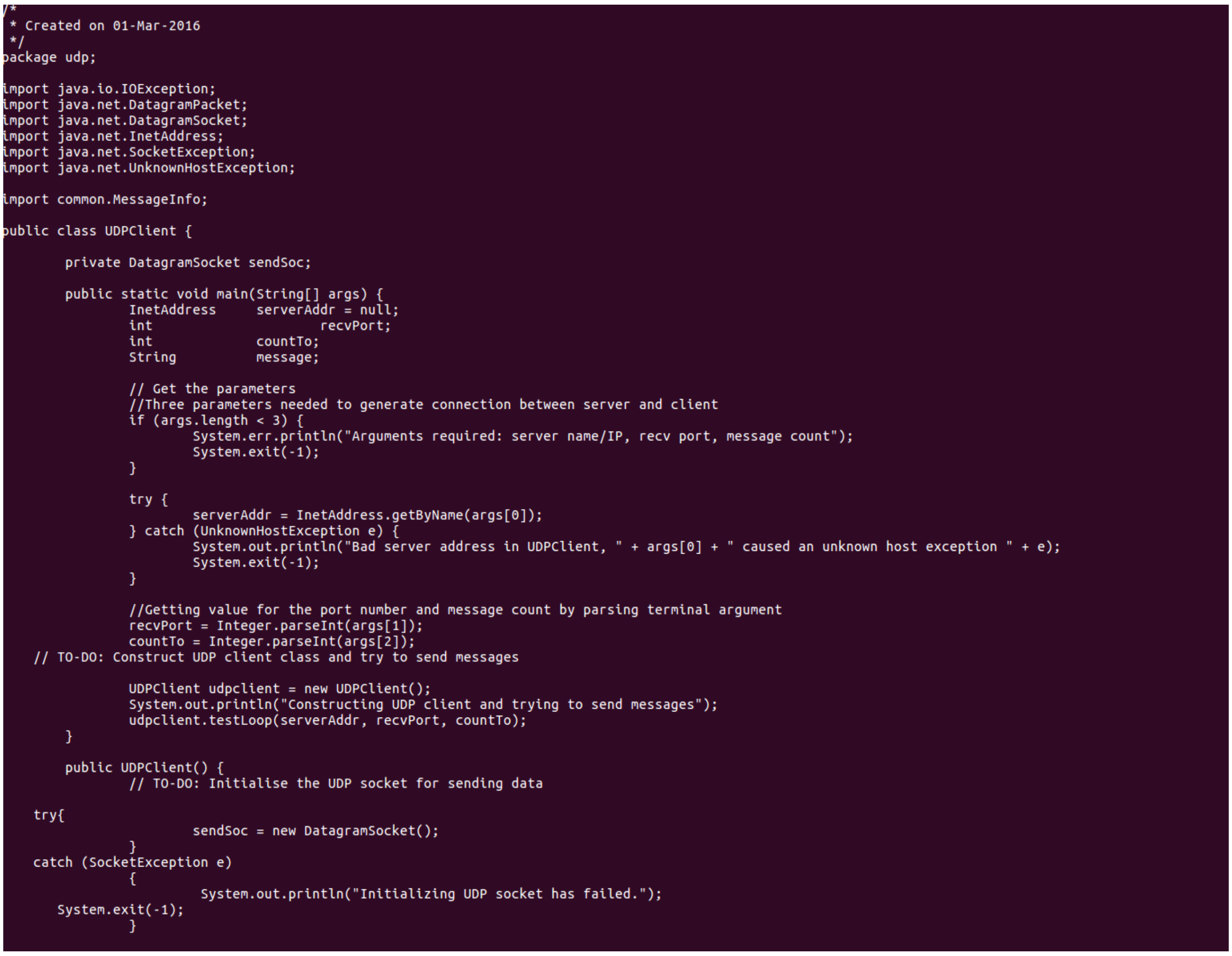


```
cc6815@edge01:given templates$ sh udpserver.sh 1080
UDPServer ready
Receiving the message in an expected time range
Of 1000, 433 received successfully
Of 1000, 567 failed
Of 1000, 768 received successfully
Of 1000, 232 failed
Of 1000, 824 received successfully
Of 1000, 176 failed
Of 1000, 967 received successfully
Of 1000, 33 failed
Of 1000, 977 received successfully
Of 1000, 23 failed
Of 1000, 979 received successfully
Of 1000, 21 failed
```

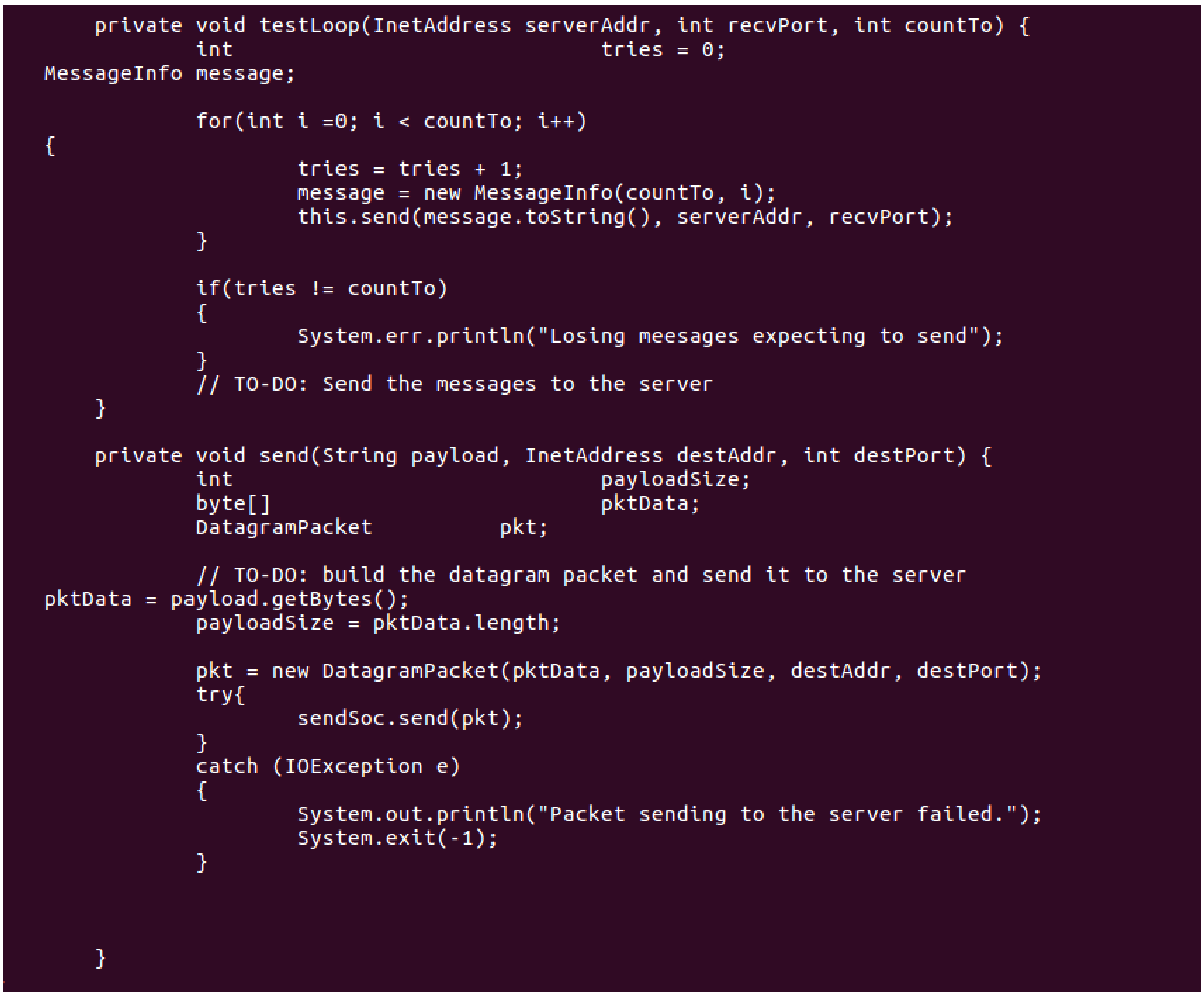


```
cc6815@edge01:given templates$ sh udpserver.sh 2000
Error: Could not create socket on port 2000
cc6815@edge01:given templates$ sh udpserver.sh 1030
UDPServer ready
Receiving the message in an expected time range
Of 2000, 1346 received successfully
Of 2000, 654 failed
Of 2000, 1891 received successfully
Of 2000, 109 failed
Of 2000, 1926 received successfully
Of 2000, 74 failed
Of 2000, 1944 received successfully
Of 2000, 56 failed
Of 2000, 1970 received successfully
Of 2000, 30 failed
Of 2000, 1977 received successfully
Of 2000, 23 failed
^Z
```

UDP client code:

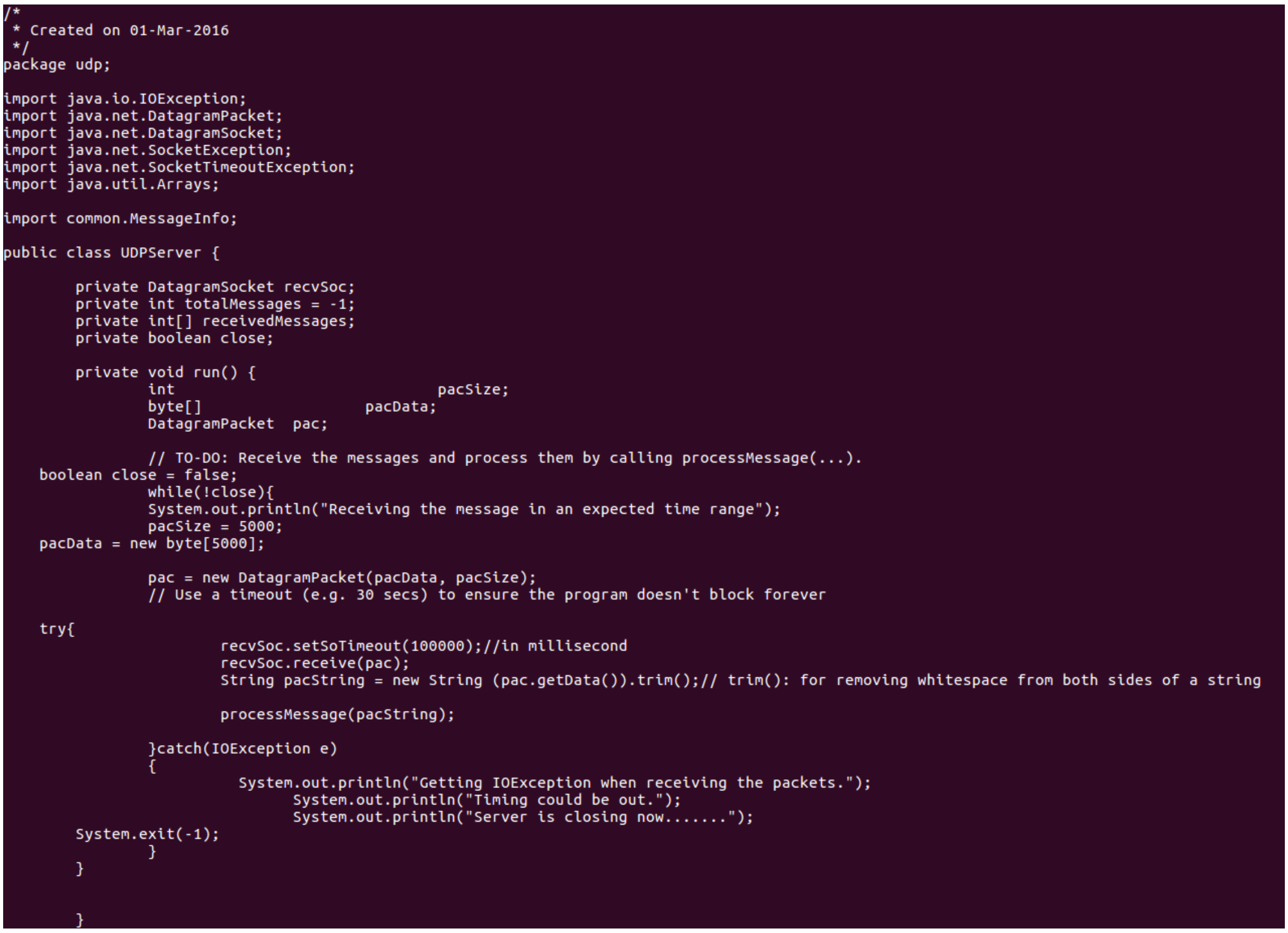


```
/*
 * Created on 01-Mar-2016
 */
package udp;

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

                // Get the parameters
                //Three parameters needed to generate connection between server and client
                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);
                }

                //Getting value for the port number and message count by parsing terminal argument
                recvPort = Integer.parseInt(args[1]);
                countTo = Integer.parseInt(args[2]);
    // TO-DO: Construct UDP client class and try to send messages

                UDPClient udpclient = new UDPClient();
                System.out.println("Constructing UDP client and trying to send messages");
                udpclient.testLoop(serverAddr, recvPort, countTo);
        }

        public UDPClient() {
                // TO-DO: Initialise the UDP socket for sending data

    try{
                        sendSoc = new DatagramSocket();
                }
    catch (SocketException e)
                {
                         System.out.println("Initializing UDP socket has failed.");
       System.exit(-1);
                }
```

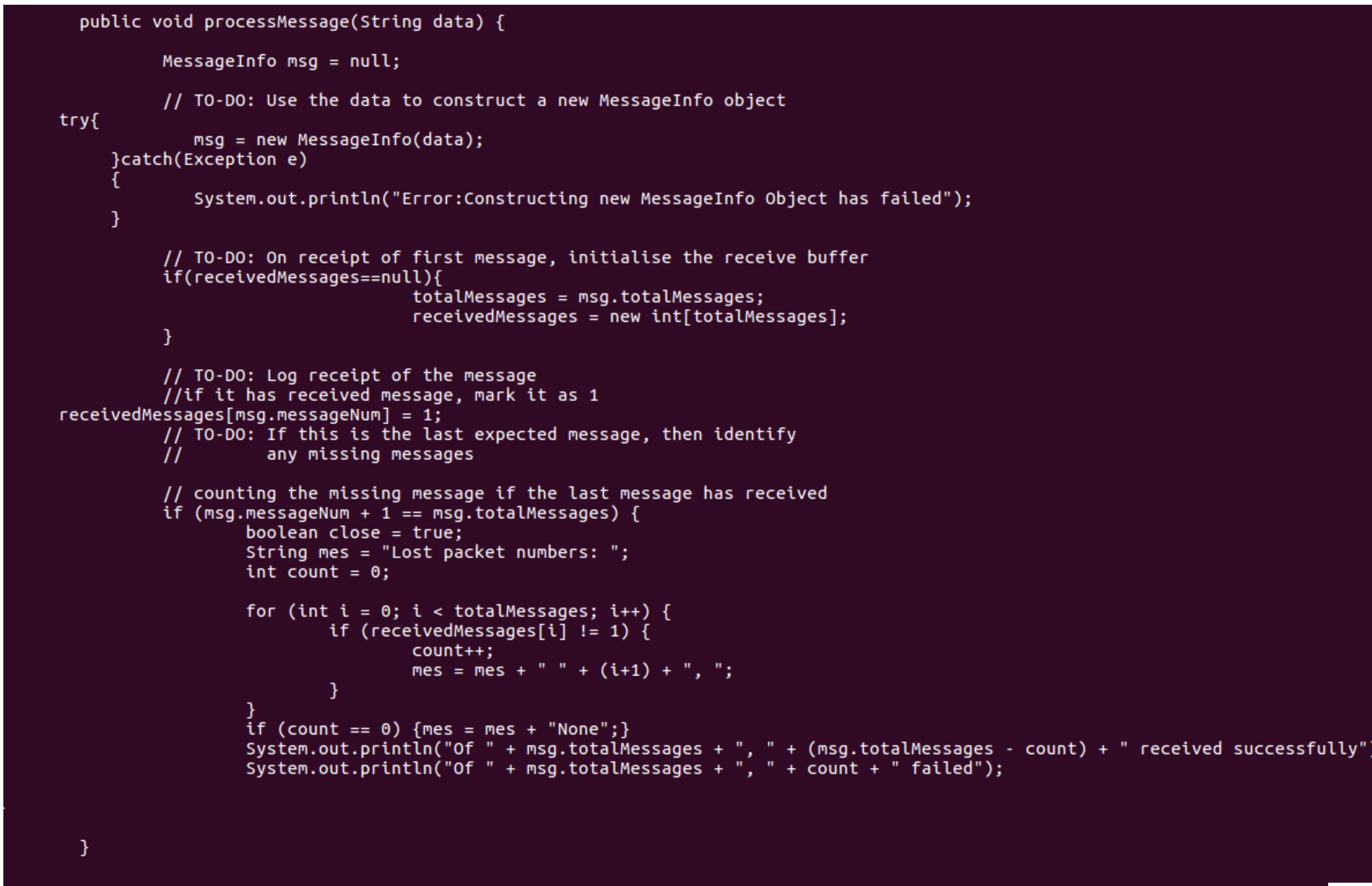


```
    private void testLoop(InetAddress serverAddr, int recvPort, int countTo) {
            int                             tries = 0;
MessageInfo message;

            for(int i =0; i < countTo; i++)
{
                    tries = tries + 1;
                    message = new MessageInfo(countTo, i);
                    this.send(message.toString(), serverAddr, recvPort);
            }

            if(tries != countTo)
            {
                    System.err.println("Losing meesages expecting to send");
            }
            // TO-DO: Send the messages to the server
    }

    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
pktData = payload.getBytes();
            payloadSize = pktData.length;

            pkt = new DatagramPacket(pktData, payloadSize, destAddr, destPort);
            try{
                    sendSoc.send(pkt);
            }
            catch (IOException e)
            {
                    System.out.println("Packet sending to the server failed.");
                    System.exit(-1);
            }



    }
```

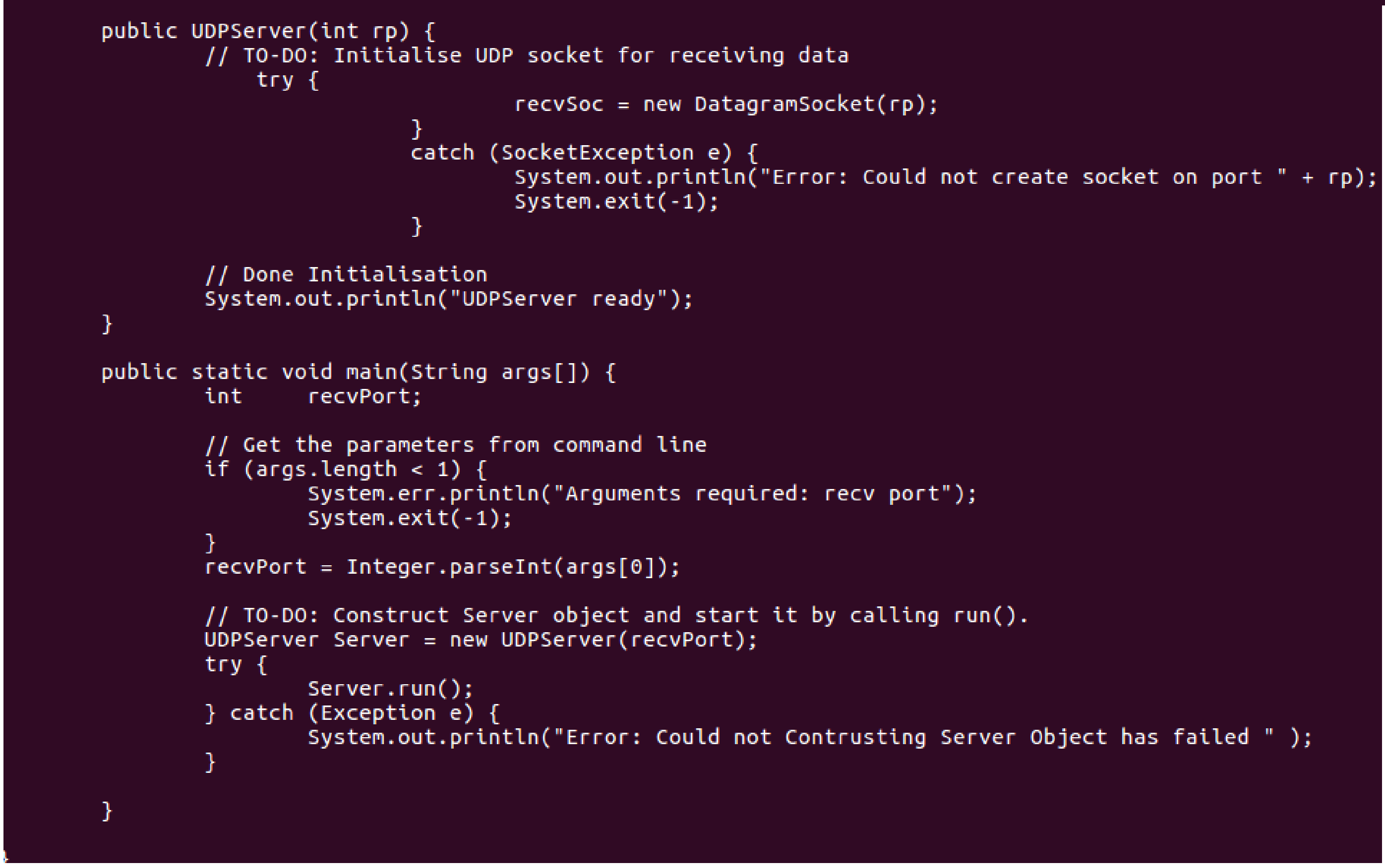
UDP Server Code:



```
/*
 * Created on 01-Mar-2016
 */
package udp;

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 common.MessageInfo;

public class UDPServer {

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

        private void run() {
                int                             pacSize;
                byte[]                  pacData;
                DatagramPacket  pac;

                // TO-DO: Receive the messages and process them by calling processMessage(...).
    boolean close = false;
                while(!close){
                System.out.println("Receiving the message in an expected time range");
                pacSize = 5000;
    pacData = new byte[5000];

                pac = new DatagramPacket(pacData, pacSize);
                // Use a timeout (e.g. 30 secs) to ensure the program doesn't block forever

    try{
                        recvSoc.setSoTimeout(100000);//in millisecond
                        recvSoc.receive(pac);
                        String pacString = new String (pac.getData()).trim();// trim(): for removing whitespace from both sides of a string

                        processMessage(pacString);

                }catch(IOException e)
                {
                          System.out.println("Getting IOException when receiving the packets.");
                                System.out.println("Timing could be out.");
                                System.out.println("Server is closing now.......");
        System.exit(-1);
                }
        }


        }
```



```
      public void processMessage(String data) {

              MessageInfo msg = null;

              // TO-DO: Use the data to construct a new MessageInfo object
    try{
                 msg = new MessageInfo(data);
         }catch(Exception e)
         {
                 System.out.println("Error:Constructing new MessageInfo Object has failed");
         }

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

              // TO-DO: Log receipt of the message
              //if it has received message, mark it as 1
    receivedMessages[msg.messageNum] = 1;
              // TO-DO: If this is the last expected message, then identify
              //        any missing messages

              // counting the missing message if the last message has received
              if (msg.messageNum + 1 == msg.totalMessages) {
                      boolean close = true;
                      String mes = "Lost packet numbers: ";
                      int count = 0;

                      for (int i = 0; i < totalMessages; i++) {
                              if (receivedMessages[i] != 1) {
                                      count++;
                                      mes = mes + " " + (i+1) + ", ";
                              }
                      }
                      if (count == 0) {mes = mes + "None";}
                      System.out.println("Of " + msg.totalMessages + ", " + (msg.totalMessages - count) + " received successfully"
                      System.out.println("Of " + msg.totalMessages + ", " + count + " failed");


      }
```



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

            // Done Initialisation
            System.out.println("UDPServer ready");
    }

    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 (Exception e) {
                    System.out.println("Error: Could not Contrusting Server Object has failed " );
            }

    }
```