Computer Networks and Distributed Systems

Assessed Coursework: RMI and UDP

Yoon Hyuk Chang
CID: 01115052

David Rovick Arrojo
CID:01070823

February 17th 2017

1 Summary

A Possible Causes of Lost Messages

UDP:

As UDP does not guarantee delivery, ordering or duplicate protection of messages, the protocol has a bigger problem with message loss than RMI. One of the causes of messages loss is the failure of the network. The further the client and the server are to each other, the more likely it will be that a message will be dropped. Another possibility, especially in the DoC labs with many users and computers, could be network congestion. Other possibilities could be buffer overflow on the server side, faulty hardware or network drivers. Buffer overflow can occur if the client sends messages at a faster rate then the rate that the server can process them.

RMI:

Since RMI is generally a much safer way of sending messages between client and server, the RMI can really only loose messages if either there is no physical connection between the client and server, or the server itself is down.

B Observed Patterns on Lost Messages

RMI showed no drop in messages in our testings, which is expected, given the high reliability of this protocol.

UDP, on the other hand, showed less reliable behavior. It would always be able to send up to 300 messages successfully, but at higher numbers of messages it would drop messages unpredictably. Sometimes it would send messages successfully at 1500 messages, and loose them at 1300.(See figure 3) At other times it would handle increasing number of messages up to 300 hundred then loose messages at 400. (See figure 2)

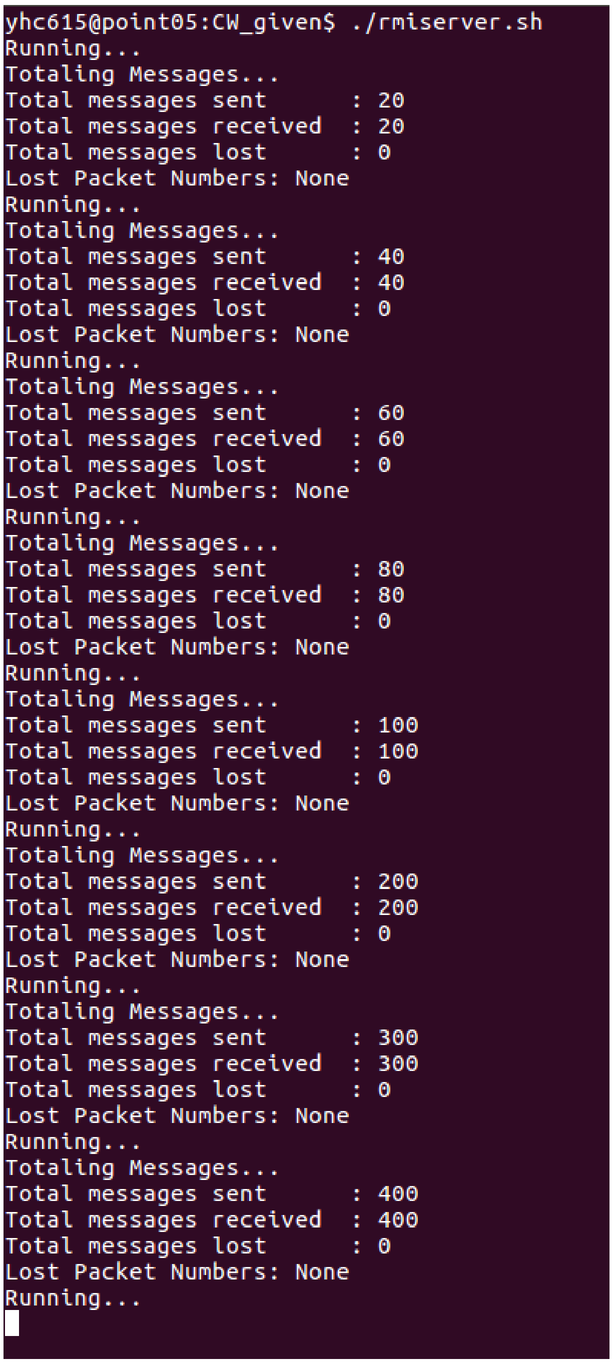
C Relative Reliability

RMI was the significantly more reliable protocol in comparison to UDP. This is because RMI handles the transmission through object implementation and manages any failures that may occur in between, which can be difficult to account for in UDP.

D Programming Difficulty

RMI was easier to program in our group experience. Having recently completed the OOP module, creating object references and implementing methods were familiar for us. The abstraction of the network communication details in RMI also made it much easier to program, such as handing port connections through binding versus creating datagram packets and buffering bytes for UDP.

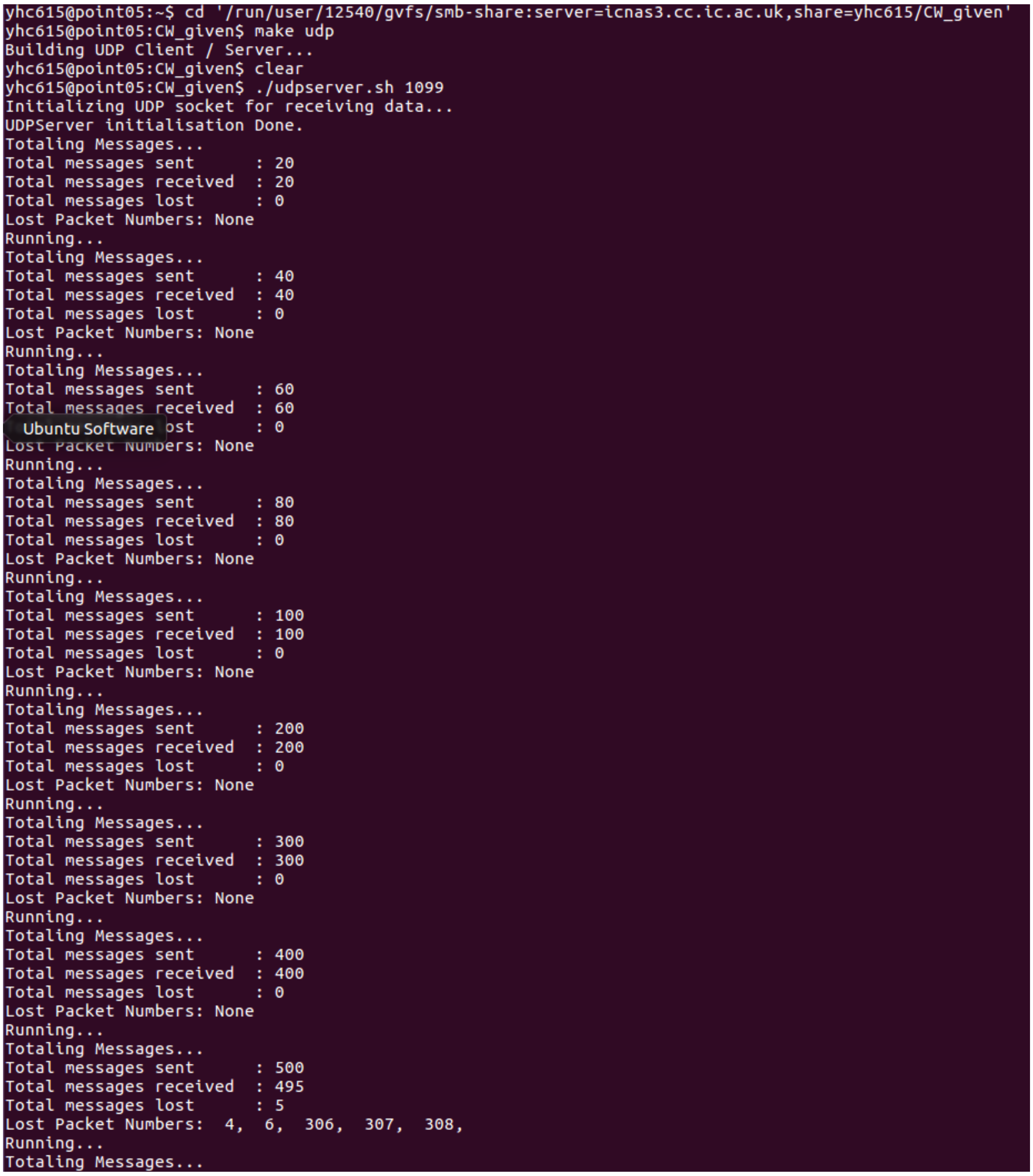
2 RMI Console Log



```
yhc615@point05:CW_given$ ./rmiserver.sh
Running...
Totaling Messages...
Total messages sent      : 20
Total messages received  : 20
Total messages lost      : 0
Lost Packet Numbers: None
Running...
Totaling Messages...
Total messages sent      : 40
Total messages received  : 40
Total messages lost      : 0
Lost Packet Numbers: None
Running...
Totaling Messages...
Total messages sent      : 60
Total messages received  : 60
Total messages lost      : 0
Lost Packet Numbers: None
Running...
Totaling Messages...
Total messages sent      : 80
Total messages received  : 80
Total messages lost      : 0
Lost Packet Numbers: None
Running...
Totaling Messages...
Total messages sent      : 100
Total messages received  : 100
Total messages lost      : 0
Lost Packet Numbers: None
Running...
Totaling Messages...
Total messages sent      : 200
Total messages received  : 200
Total messages lost      : 0
Lost Packet Numbers: None
Running...
Totaling Messages...
Total messages sent      : 300
Total messages received  : 300
Total messages lost      : 0
Lost Packet Numbers: None
Running...
Totaling Messages...
Total messages sent      : 400
Total messages received  : 400
Total messages lost      : 0
Lost Packet Numbers: None
Running...
```

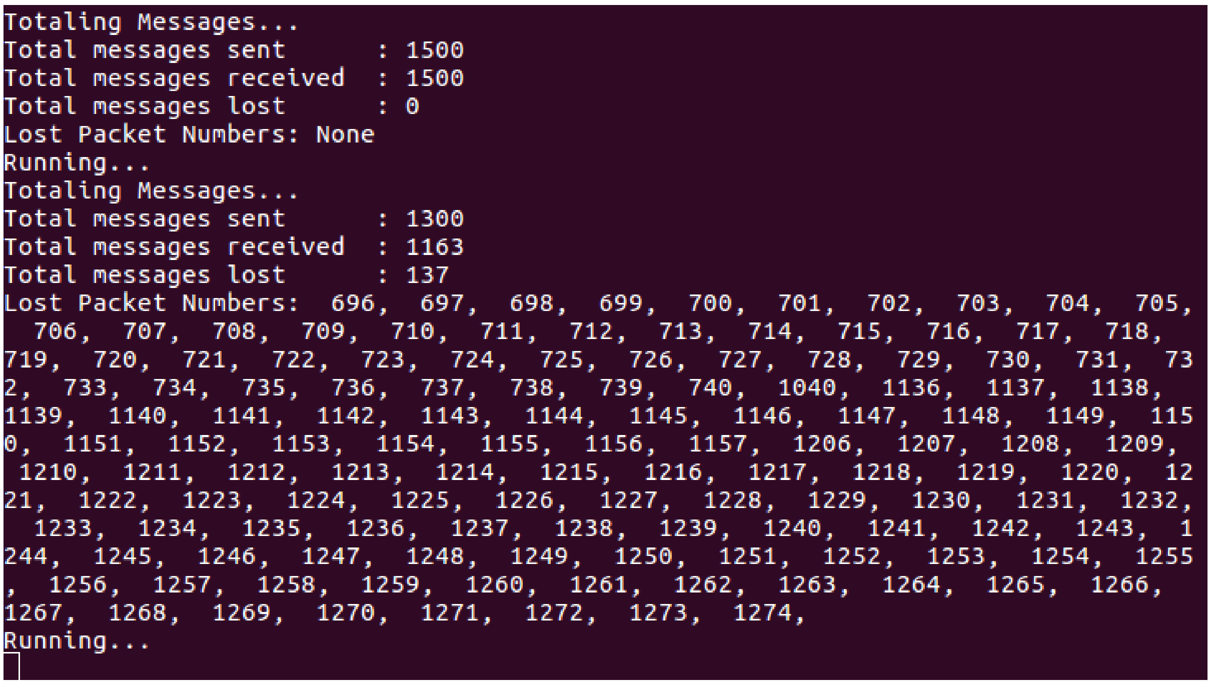
Figure 1: RMI Console Log

3 UDP Console Log



```
yhc615@point05:~$ cd '/run/user/12540/gvfs/smb-share:server=icnas3.cc.ic.ac.uk,share=yhc615/CW_given'
yhc615@point05:CW_given$ make udp
Building UDP Client / Server...
yhc615@point05:CW_given$ clear
yhc615@point05:CW_given$ ./udpserver.sh 1099
Initializing UDP socket for receiving data...
UDPServer initialisation Done.
Totaling Messages...
Total messages sent      : 20
Total messages received  : 20
Total messages lost      : 0
Lost Packet Numbers: None
Running...
Totaling Messages...
Total messages sent      : 40
Total messages received  : 40
Total messages lost      : 0
Lost Packet Numbers: None
Running...
Totaling Messages...
Total messages sent      : 60
Total messages received  : 60
Ubuntu Software ost      : 0
Lost Packet Numbers: None
Running...
Totaling Messages...
Total messages sent      : 80
Total messages received  : 80
Total messages lost      : 0
Lost Packet Numbers: None
Running...
Totaling Messages...
Total messages sent      : 100
Total messages received  : 100
Total messages lost      : 0
Lost Packet Numbers: None
Running...
Totaling Messages...
Total messages sent      : 200
Total messages received  : 200
Total messages lost      : 0
Lost Packet Numbers: None
Running...
Totaling Messages...
Total messages sent      : 300
Total messages received  : 300
Total messages lost      : 0
Lost Packet Numbers: None
Running...
Totaling Messages...
Total messages sent      : 400
Total messages received  : 400
Total messages lost      : 0
Lost Packet Numbers: None
Running...
Totaling Messages...
Total messages sent      : 500
Total messages received  : 495
Total messages lost      : 5
Lost Packet Numbers:  4,  6,  306,  307,  308,
Running...
Totaling Messages...
```

Figure 2: UDP Console Log



```
Totaling Messages...
Total messages sent      : 1500
Total messages received  : 1500
Total messages lost      : 0
Lost Packet Numbers: None
Running...
Totaling Messages...
Total messages sent      : 1300
Total messages received  : 1163
Total messages lost      : 137
Lost Packet Numbers:  696,  697,  698,  699,  700,  701,  702,  703,  704,  705,
  706,  707,  708,  709,  710,  711,  712,  713,  714,  715,  716,  717,  718,
719,  720,  721,  722,  723,  724,  725,  726,  727,  728,  729,  730,  731,  73
2,  733,  734,  735,  736,  737,  738,  739,  740,  1040,  1136,  1137,  1138,
1139,  1140,  1141,  1142,  1143,  1144,  1145,  1146,  1147,  1148,  1149,  115
0,  1151,  1152,  1153,  1154,  1155,  1156,  1157,  1206,  1207,  1208,  1209,
 1210,  1211,  1212,  1213,  1214,  1215,  1216,  1217,  1218,  1219,  1220,  12
21,  1222,  1223,  1224,  1225,  1226,  1227,  1228,  1229,  1230,  1231,  1232,
  1233,  1234,  1235,  1236,  1237,  1238,  1239,  1240,  1241,  1242,  1243,  1
244,  1245,  1246,  1247,  1248,  1249,  1250,  1251,  1252,  1253,  1254,  1255
,  1256,  1257,  1258,  1259,  1260,  1261,  1262,  1263,  1264,  1265,  1266,
1267,  1268,  1269,  1270,  1271,  1272,  1273,  1274,
Running...
```

Figure 3: UDP Console Log2

4 RMI Client Code

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

import java.rmi.Naming;
import java.rmi.NotBoundException;
import java.rmi.RemoteException;
import java.net.MalformedURLException;
import java.rmi.RMISecurityManager;

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("No arguments given: ServerHostName/IPAddress, TotalMessageCount");
                        System.exit(-1);
                }

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

                // TO-DO: Initialise Security Manager
                if(System.getSecurityManager()==null){
                            System.setSecurityManager(new SecurityManager());
                }
                // TO-DO: Bind to RMIServer

                try {
                                                iRMIServer = (RMIServerI) Naming.lookup(urlServer);
                                                for(int i = 0; i < numMessages; i++) {
                                                 MessageInfo message = new MessageInfo(numMessages,i);
                                                 iRMIServer.receiveMessage(message);
                                                }
                } catch (MalformedURLException e) {
                                                 System.out.println("Error: Malformed URL Exception.");
                } catch (RemoteException e) {
                                                 System.out.println("Error: Remote Exception");
                } catch (NotBoundException e) {
                                                 System.out.println("Error: Not Bound Exception.");
                }
        }
}
```

5 RMI Server Code

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

import java.net.MalformedURLException;
import java.rmi.Naming;
import java.rmi.registry.LocateRegistry;
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;
import java.util.Arrays;

import common.*;

public class RMIServer extends UnicastRemoteObject implements RMIServerI {

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

        public RMIServer() throws RemoteException {
                //lkjkjklkjl
                System.setProperty("java.rmi.server.hostname", "192.168.1.199.");
        }

        public void receiveMessage(MessageInfo msg) throws RemoteException {

                // TO-DO: On receipt of first message, initialise the receive buffer
                if ((receivedMessages == null) || (msg.totalMessages != totalMessages)) {
                        totalMessages = msg.totalMessages;
                        receivedMessages = new int[msg.totalMessages];
                }
                // 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 + 1 == totalMessages) {
                        System.out.println("Totaling Messages...");

                        String str = "Lost Packet Numbers: ";
                        int count = 0;
                        for (int i = 0; i < totalMessages; i++) {
                                if (receivedMessages[i] != 1) {
                                        count++;
                                        str = str + " " + (i+1) + ", ";
                                }
                        }

                        if (count == 0) str = str + "None";

                        System.out.println("Total messages sent      : " + totalMessages);
                        System.out.println("Total messages received  : " + (totalMessages - count));
                        System.out.println("Total messages lost      : " + count);
                        System.out.println(str);
                        System.out.println("Running...");
                        //System.exit(0);
                }

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

		RMIServer rmis = null;

		// TO-DO: Initialise Security Manager
		if(System.getSecurityManager()==null){
				System.setSecurityManager(new SecurityManager());
		}
		// TO-DO: Instantiate the server class
		try { rmis = new RMIServer();
		} catch(RemoteException e) {
			System.out.println("Error instantiating server class.");
			System.exit(-1);
		}

		// TO-DO: Bind to RMI registry

		rebindServer("RMIServer", rmis );

		System.out.println("Running...");

	}

	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)
		try{ LocateRegistry.createRegistry( 1099 );
		} catch (RemoteException e) {
			System.out.println("Error initializing registry.");
			System.exit(-1);
		}

		// 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{ Naming.rebind( serverURL , server );
		} catch (RemoteException e) {
			System.out.println("Error binding server.");
			System.exit(-1);
		} catch (MalformedURLException e) {
			System.out.println("Could not bind server to defined registry as the URL was malformed.");
			System.exit(-1);
		}
	}
}
```

6 UDP Client Code

```java
/*
 * 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
                if (args.length < 3) {
                        System.err.println("No arguments given: server name/IP, recv port, message count");
                        System.exit(-1);
                }

                try {
                        serverAddr = InetAddress.getByName(args[0]);
                } catch (UnknownHostException e) {
                        System.out.println("Bad server name/IP : " + args[0] + " , 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
                System.out.println("Constructing UDP client class...");
                UDPClient udp = new UDPClient();
                System.out.println("Attempting to send messages...");
                for (int i = 0; i < countTo ; i++){
                                message = Integer.toString(countTo) + ";" + Integer.toString(i);
                                udp.send( message, serverAddr, recvPort);
                }

        }

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

        }


        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("Error transmitting packet over network.");
                        System.exit(-1);
                }
        }
}
```

7 UDP Server Code

```java
/*
 * 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(...).
                //        Use a timeout (e.g. 30 secs) to ensure the program doesn't block forever
                while(!close){
                        //initialise buffer
                        pacSize = 3000;
                        pacData = new byte [pacSize];

                        pac = new DatagramPacket(pacData, pacSize);
                        try{
                                recvSoc.setSoTimeout(30000);
                                recvSoc.receive(pac);
                        } catch (SocketTimeoutException e){
                                System.out.println("Error: Socket Exception, timeout may have occured (30sec)");
                                System.out.println("Closing server...");
                                System.exit(-1);
                        } catch (IOException e){
                                System.out.println("Error: IOException");
                                System.out.println("Closing server...");
                                System.exit(-1);
                        }
                        processMessage(new String(pac.getData()));
                }
        }

        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(e);
                }
        // TO-DO: On receipt of first message, initialise the receive buffer
        if ((receivedMessages == null) || (msg.totalMessages != totalMessages)) {
            totalMessages = msg.totalMessages;
            receivedMessages = new int[msg.totalMessages];
        }
        // 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 + 1 == totalMessages) {
            System.out.println("Totaling Messages...");

            String str = "Lost Packet Numbers: ";
            int count = 0;
            for (int i = 0; i < totalMessages; i++) {
                if (receivedMessages[i] != 1) {
                    count++;
                    str = str + " " + (i+1) + ", ";
                }
            }

            if (count == 0) str = str + "None";

            System.out.println("Total messages sent      : " + totalMessages);
            System.out.println("Total messages received  : " + (totalMessages - count));
            System.out.println("Total messages lost      : " + count);
            System.out.println(str);
            System.out.println("Running...");
            }
        }

    public UDPServer(int rp) {
        // TO-DO: Initialise UDP socket for receiving data
        System.out.println("Initializing UDP socket for receiving data...");
        try{
            recvSoc = new DatagramSocket(rp);
        } catch (SocketException e){
            System.out.println("Error: SocketException");
            System.out.println("Closing Server...");
            System.exit(-1);
        }
        // Done Initialisation
        System.out.println("UDPServer initialisation Done.");
    }

    public static void main(String args[]) {
        int         recvPort;
        // Get the parameters from command line
        if (args.length < 1) {
            System.err.println("No agrument given: recv port");
            System.exit(-1);
        }
        recvPort = Integer.parseInt(args[0]);

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

}
```