

COURSEWORK

IMPERIAL COLLEGE LONDON

DEPARTMENT OF COMPUTING

Networks Coursework

Author:

Mohamed Abdelmagid, Mostafa Hagra (CID: 01050081, 01053843)

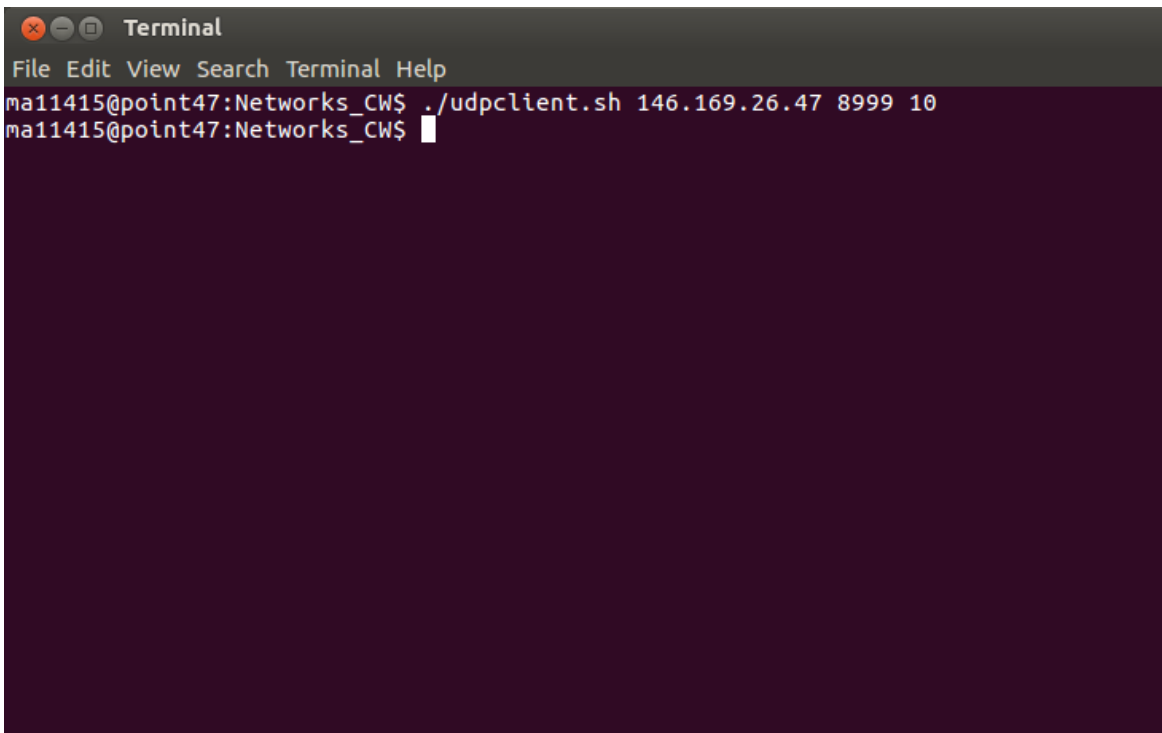
Date: February 16, 2017

1 UDP

UDP experiences message loss during transmission. Due to this reason, we feel that RMI is a more reliable transmission process. However, UDP has some benefits, one being that it is a quicker transmission method with a larger throughput and a lower latency. The packet sizes are smaller as they have less overhead due to the absence of the acknowledgements, method used in RMI.

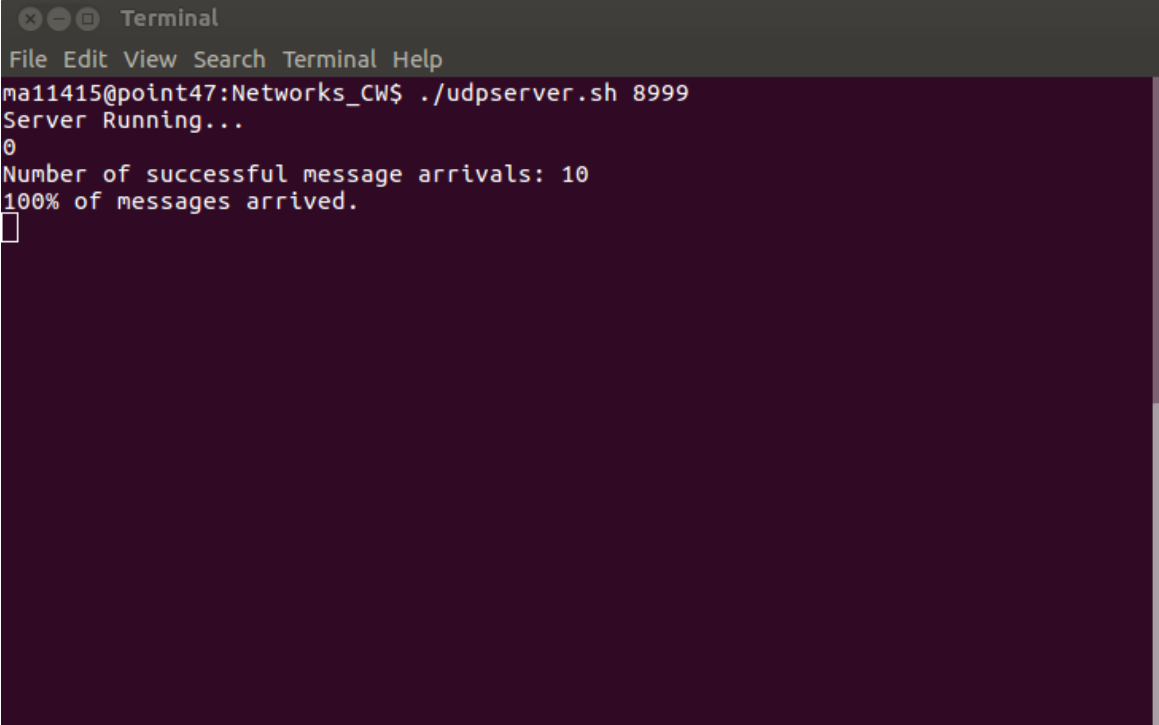
The main reason for packet loss in UDP is network congestion. Due to it being a faster method of transmission, if the data arrives at a rate that is faster than it is possible to send them to their destination, then the packets have to be dropped, and with the speed of UDP, this is quite common hence the packets loss we experience once we send a large number of messages

We sent messages across computers and found UDP loses more messages the further the computers were away. Anything more than 400 messages and UDP will nearly always lose some messages but the number of messages it loses is never the same.

A screenshot of a terminal window titled "Terminal". The terminal has a menu bar with "File", "Edit", "View", "Search", "Terminal", and "Help". The prompt is "ma11415@point47:Networks_CW\$". The command entered is "./udpclient.sh 146.169.26.47 8999 10". The prompt is now on the next line, indicating the command has been executed.

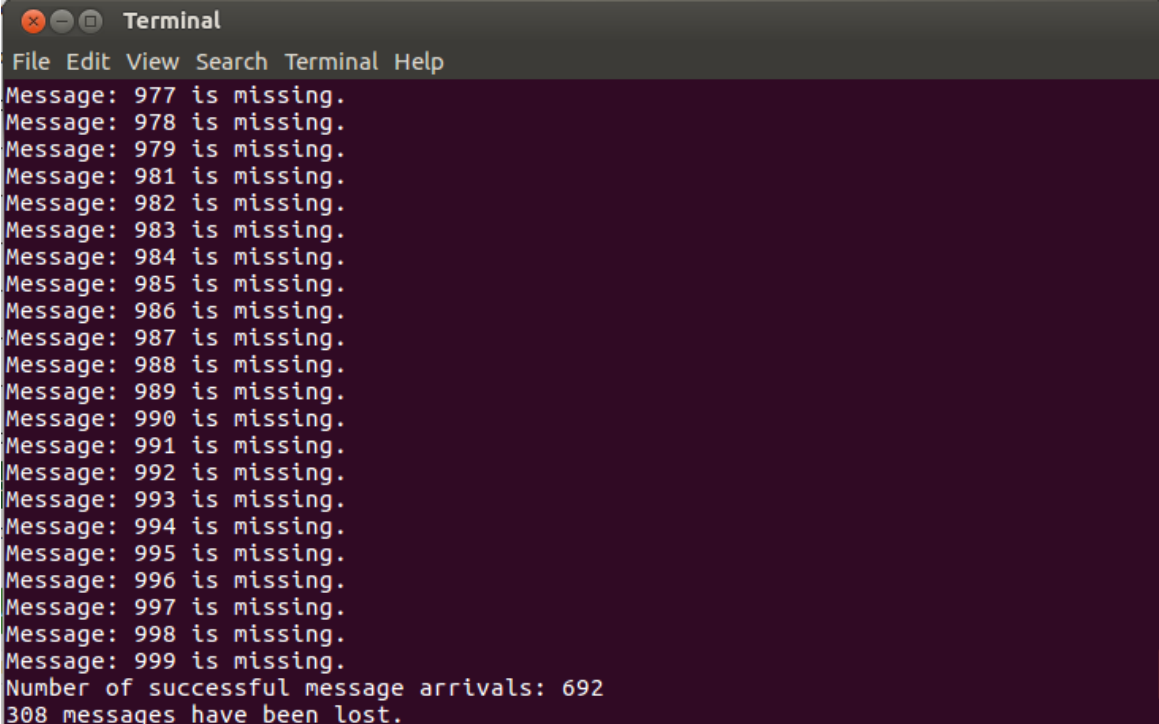
```
Terminal
File Edit View Search Terminal Help
ma11415@point47:Networks_CW$ ./udpclient.sh 146.169.26.47 8999 10
ma11415@point47:Networks_CW$
```

Figure 1: Client side UDP sending 10 messages to server located in IPv4 146.169.26.47 and port 8999



```
Terminal
File Edit View Search Terminal Help
ma11415@point47:Networks_CW$ ./udpserver.sh 8999
Server Running...
0
Number of successful message arrivals: 10
100% of messages arrived.
```

Figure 2: Server-side UDP receiving the 10 messages sent.



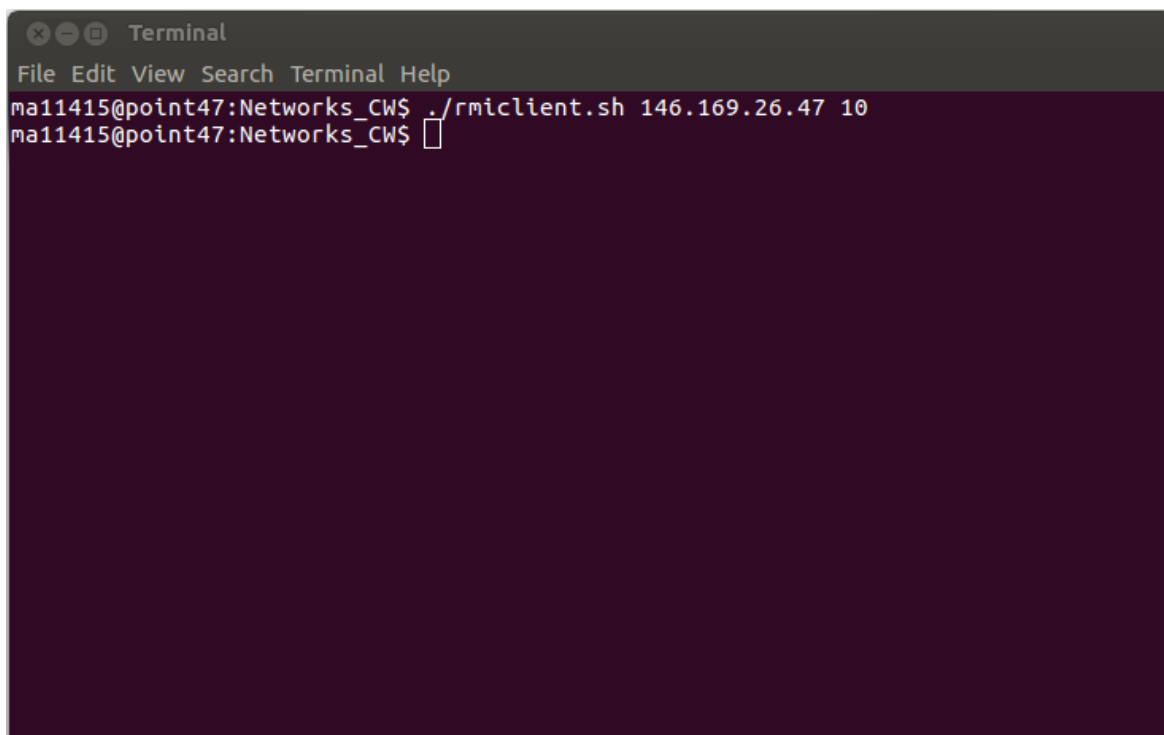
```
Terminal
File Edit View Search Terminal Help
Message: 977 is missing.
Message: 978 is missing.
Message: 979 is missing.
Message: 981 is missing.
Message: 982 is missing.
Message: 983 is missing.
Message: 984 is missing.
Message: 985 is missing.
Message: 986 is missing.
Message: 987 is missing.
Message: 988 is missing.
Message: 989 is missing.
Message: 990 is missing.
Message: 991 is missing.
Message: 992 is missing.
Message: 993 is missing.
Message: 994 is missing.
Message: 995 is missing.
Message: 996 is missing.
Message: 997 is missing.
Message: 998 is missing.
Message: 999 is missing.
Number of successful message arrivals: 692
308 messages have been lost.
```

Figure 3: Server-side UDP receiving the 1000 messages sent and outputting successful arrivals and which messages lost.

2 RMI

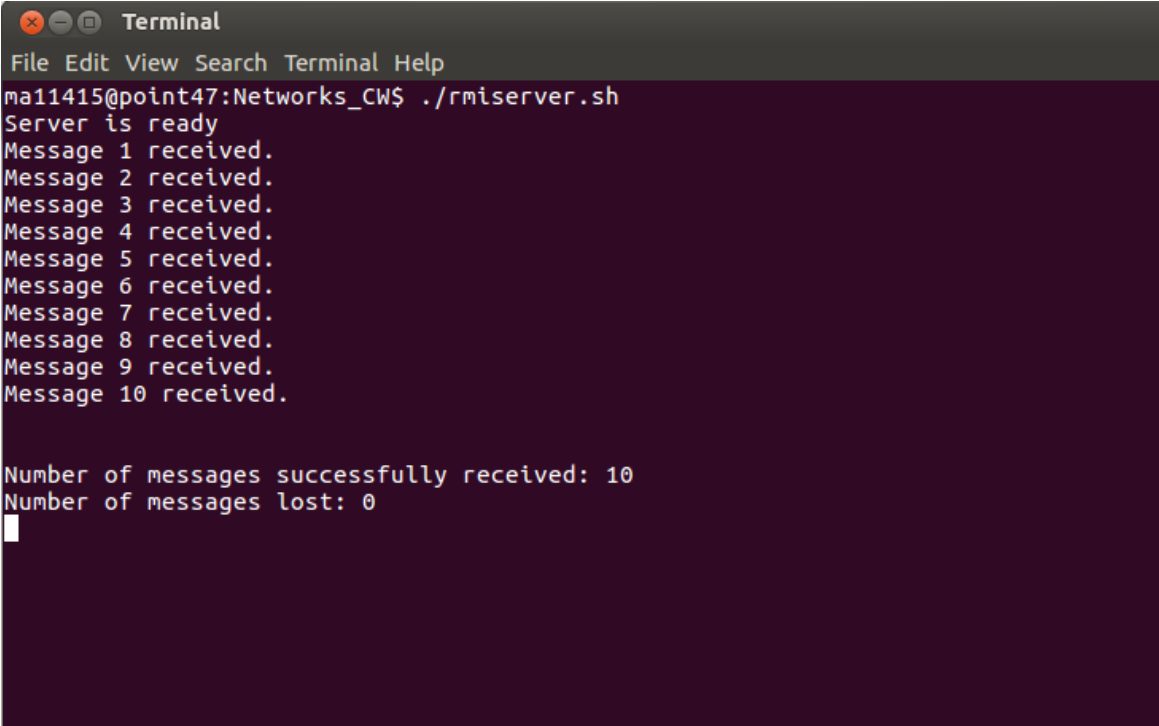
Coding RMI was more difficult than UDP because we had issues connecting to the correct port. We initially attempted to use *Naming* instead of *LocateRegistry* which was unsuccessful. We test RMI code by connecting to the IP the server is located in - we use the same port 8090 for all our RMI connections and is hard coded into both the client and server-side.

RMI has displayed zero message loss even when sending as many as 2000 messages as seen in Figure 6 and across different lab computers. RMI experiences no loss of messages due to its use of the TCP/IP protocol. The reason for the reliability of this protocol is that there is constant communication between server and client. Upon receipt of a packet or message, the client returns a signal to the server called an **Acknowledgement**. If this Acknowledgement is not received within a certain span of time, or the server receives a duplicate acknowledgement, then the server retransmits the message. For these reasons, the packets are never lost in RMI.

A screenshot of a terminal window titled "Terminal" with a dark background. The terminal shows a command prompt "ma11415@point47:Networks_CW\$" followed by the command "./rmiclient.sh 146.169.26.47 10". The command has been executed, and the prompt is now on a new line, indicating the command completed successfully.

```
Terminal
File Edit View Search Terminal Help
ma11415@point47:Networks_CW$ ./rmiclient.sh 146.169.26.47 10
ma11415@point47:Networks_CW$
```

Figure 4: Client side RMI sending 10 messages to server located in IPv4 146.169.26.47

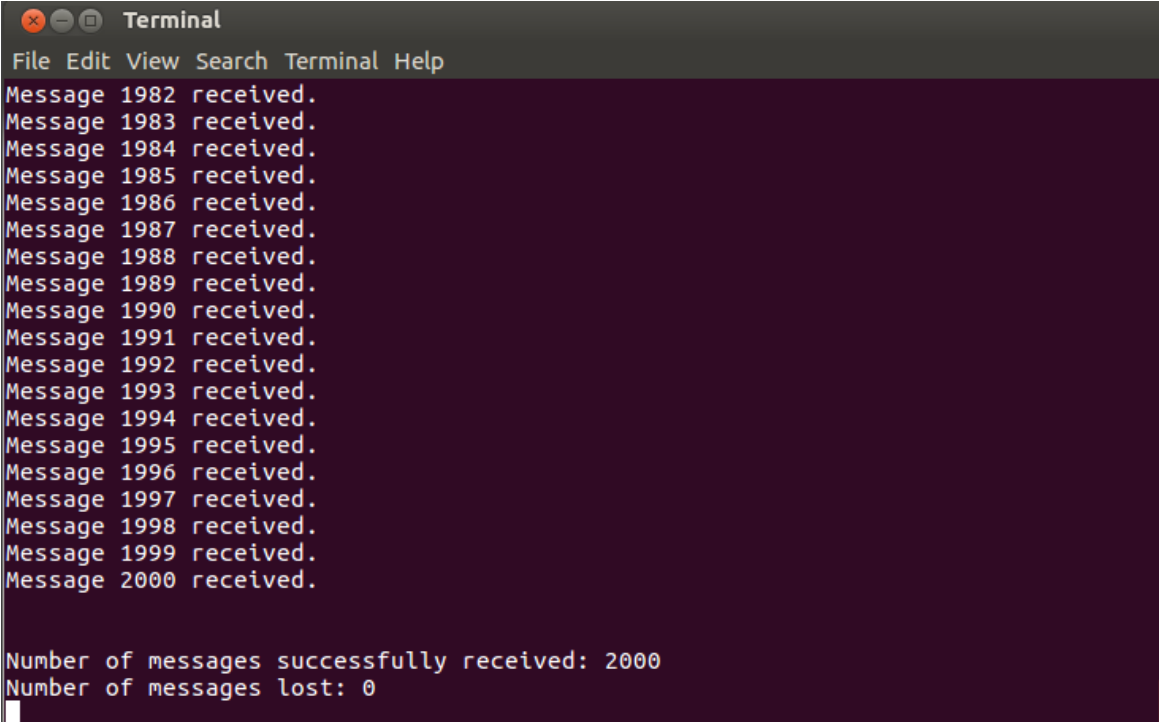
A terminal window titled "Terminal" with a menu bar (File, Edit, View, Search, Terminal, Help). The prompt is "ma11415@point47:Networks_CW\$". The command executed is "./rmiserver.sh". The output shows the server is ready and receives 10 messages, numbered 1 to 10. At the bottom, it reports "Number of messages successfully received: 10" and "Number of messages lost: 0".

```
Terminal
File Edit View Search Terminal Help
ma11415@point47:Networks_CW$ ./rmiserver.sh
Server is ready
Message 1 received.
Message 2 received.
Message 3 received.
Message 4 received.
Message 5 received.
Message 6 received.
Message 7 received.
Message 8 received.
Message 9 received.
Message 10 received.

Number of messages successfully received: 10
Number of messages lost: 0

```

Figure 5: RMI Server receiving the 10 messages sent with zero messages lost.

A terminal window titled "Terminal" with a menu bar (File, Edit, View, Search, Terminal, Help). The prompt is "ma11415@point47:Networks_CW\$". The command executed is "./rmiserver.sh". The output shows the server is ready and receives 2000 messages, numbered 1982 to 2000. At the bottom, it reports "Number of messages successfully received: 2000" and "Number of messages lost: 0".

```
Terminal
File Edit View Search Terminal Help
ma11415@point47:Networks_CW$ ./rmiserver.sh
Server is ready
Message 1982 received.
Message 1983 received.
Message 1984 received.
Message 1985 received.
Message 1986 received.
Message 1987 received.
Message 1988 received.
Message 1989 received.
Message 1990 received.
Message 1991 received.
Message 1992 received.
Message 1993 received.
Message 1994 received.
Message 1995 received.
Message 1996 received.
Message 1997 received.
Message 1998 received.
Message 1999 received.
Message 2000 received.

Number of messages successfully received: 2000
Number of messages lost: 0

```

Figure 6: RMI Server receiving 2000 messages with zero messages lost.

3 UDP Java Code

3.0.1 Client-side Code

```

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 rcvPort;
        int countTo;
        String message;

        // Get the parameters
        if (args.length < 3) {
            System.err.println("Arguments required: server name/IP, rcv 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);
        }
        rcvPort = Integer.parseInt(args[1]);
        countTo = Integer.parseInt(args[2]);

        // TO-DO: Construct UDP client class and try to send messages

        UDPClient udp_client = new UDPClient();
        udp_client.testLoop(serverAddr, rcvPort, countTo);
    }

    public UDPClient() {
        // TO-DO: Initialise the UDP socket for sending data
        try {
            sendSoc = new DatagramSocket();
        } catch (SocketException e) {
    
```

```
        e.printStackTrace();
    }
}

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

    // TO-DO: Send the messages to the server
    for (tries = 0; tries < countTo; tries++){
        MessageInfo mailInfo = new MessageInfo(countTo, tries);
        try{
            send(mailInfo.toString(), serverAddr, recvPort);
        } catch (IOException e){
            e.printStackTrace();
        }
    }
}

private void send(String payload, InetAddress destAddr, int destPort)
    throws IOException {
    int payloadSize = payload.length();
    byte[] pktData = payload.getBytes();
    DatagramPacket pkt;

    // TO-DO: build the datagram packet and send it to the server

    pkt = new DatagramPacket(pktData, payloadSize, destAddr, destPort);
    sendSoc.send(pkt);
}
}
```

Listing 1: UDP Client-side Java code.

3.0.2 Server-side Code

```

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() throws SocketException, IOException{
        int pacSize = 2048; // Try different sizes?
        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
        close = false;
        while (!close){

            pacData = new byte[pacSize];

            pac = new DatagramPacket(pacData, pacSize);

            try {
                recvSoc.setSoTimeout(45000); //45 seconds
                recvSoc.receive(pac);
                String pac_message = new String(pac.getData());
                try{
                    processMessage(pac_message);
                }catch(Exception e){
                    e.printStackTrace();
                }
            }catch(SocketTimeoutException e){
                int lost_messages = 0;
                // If there are no messages to begin with, exit
                if (receivedMessages == null){
                    return;
                }

                for (int i = 0; i < receivedMessages.length; i++){
                    if (receivedMessages[i] != 1){ //missing if the message is not
                        equal to one?
                        System.out.println("Message: " + i + " is missing.");
                    }
                }
            }
        }
    }
}

```



```
        lost_messages++;
    }
}

System.out.println("Number of successful message arrivals: " + (
    totalMessages - lost_messages));
if (lost_messages > 0){
    System.out.println(lost_messages + " messages have been lost.")
;
} else{
    System.out.println("100% of messages arrived.");
}

receivedMessages = null;
totalMessages = -1;
System.out.println("Socket Timeout");
close = true;
}
}

}

public void processMessage(String data) throws Exception{

    MessageInfo msg = new MessageInfo(data.trim());

    // TO-DO: Use the data to construct a new MessageInfo object

    // TO-DO: On receipt of first message, initialise the receive buffer
    if ((receivedMessages == null) || (msg.totalMessages !=
        receivedMessages.length)){
        receivedMessages = new int[msg.totalMessages];
        totalMessages = 0;
    }
    // TO-DO: Log receipt of the message

    totalMessages++;

    if (msg.messageNum % 100 == 0 ){
        System.out.println(msg.messageNum);
    }
    receivedMessages[msg.messageNum] = 1;
    // TO-DO: If this is the last expected message, then identify
    //          any missing messages

    if (totalMessages == msg.totalMessages){
        int lost_messages = 0;
        // If there are no messages to begin with, exit
        if (receivedMessages == null){
            return;
        }

        for (int i = 0; i < receivedMessages.length; i++){
```

```

        if (receivedMessages[i] != 1){ //missing if the message is not
            equal to one?
            System.out.println("Message: " + i + " is missing.");
            lost_messages++;
        }
    }

    System.out.println("Number of successful message arrivals: " + (
        totalMessages - lost_messages));
    if (lost_messages > 0){
        System.out.println(lost_messages + " messages have been lost.");
    }else{
        System.out.println("100% of messages arrived.");
    }

    receivedMessages = null;
    totalMessages = -1;
}

}

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

public static void main(String args[]) {

    // Get the parameters from command line
    if (args.length < 1) {
        System.err.println("Arguments required: recv port");
        System.exit(-1);
    }
    int recvPort = Integer.parseInt(args[0]);

    // TO-DO: Construct Server object and start it by calling run().
    UDPServer udp_server = new UDPServer(recvPort);
    try{
        udp_server.run();
    }catch (Exception e){
        e.printStackTrace();
    }
}
}

```

Listing 2: UDP Server-side Java code

4 RMI Java Code

4.0.1 Client-side Code

```
package rmi;

import java.rmi.Naming;
import java.rmi.registry.*;
import java.rmi.NotBoundException;
import java.rmi.RemoteException;
import java.net.MalformedURLException;

import common.MessageInfo;

public class RMIClient {

    public static void main(String[] args) throws MalformedURLException,
        RemoteException, NotBoundException {

        RMIServerI iRMIServer = null;

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

        // Locate and lookup registry for port 8090
        Registry reg = LocateRegistry.getRegistry(args[0], 8090);
        iRMIServer = (RMIServerI) reg.lookup("Mostafa");

        // Send messages
        for(int i = 0; i < numMessages; i++){
            message = new MessageInfo(numMessages, i);
            iRMIServer.receiveMessage(message);
        }
    }
}
```

Listing 3: RMI Client-side Java code.

4.0.2 Server-side Code

```

package rmi;

import java.rmi.registry.Registry;
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 {
        super();
    }

    // Used for debugging
    public String helloTo(String name) throws RemoteException{
        System.err.println(name + " is trying to contact!");
        return "Server says hello to " + name;
    }

    public void receiveMessage(MessageInfo msg) throws RemoteException {
        // TO-DO: On receipt of first message, initialise the receive
        //        buffer

        if ((totalMessages == -1 ) || (totalMessages != msg.totalMessages
        )){
            totalMessages = msg.totalMessages;
            receivedMessages = new int[totalMessages];
        }

        // TO-DO: Log receipt of the message

        receivedMessages[msg.messageNum] = 1;

        System.out.println("Message " + (msg.messageNum + 1) + " received
        .");

        // TO-DO: If this is the last expected message, then identify
        //        any missing messages
    }
}

```

```
int lostMessages = 0;

    if(msg.messageNum == totalMessages-1){
        System.out.println(" ");
        System.out.println(" ");
        for (int i = 0; i < totalMessages; i++){
            if(receivedMessages[i] == 0){
                lostMessages++;
                System.out.println("Message number " + i+ " was not
                    received");
            }
        }
        System.out.println("Number of messages successfully received: " + (
            totalMessages - lostMessages));
        System.out.println("Number of messages lost: " + lostMessages);
    }
}

public static void main(String[] args) throws Exception {

    RMIServer rmis = new RMIServer();

    Registry registry;
    // Bind to registry 8090
    try{
        registry = LocateRegistry.createRegistry(8090);
    }catch (RemoteException e){
        registry = LocateRegistry.getRegistry();
    }
    registry.bind("Mostafa",rmis);
    System.out.println("Server is ready");
}
}
```

Listing 4: RMI Server-side Java code

```
package rmi;

import java.rmi.Remote;
import java.rmi.RemoteException;

import common.*;

public interface RMIServerI extends Remote {
    public void receiveMessage(MessageInfo msg) throws
        RemoteException;
    public String helloTo(String name) throws RemoteException;
}
```

Listing 5: RMI Server-side Java Interface code.