ANKERS, Harrison (haa516)

Imperial College London

Department of Computing
Academic Year **2017-2018**
Page created Mon Feb 12 12:43:30 GMT 2018

CATE

527 ecl1 1
i2 haa516 v3

SAO date stamp here

Exercise Information

Module:	527 Computer Networks and Distributed Systems	**Issued:**	Tue - 23 Jan 2018
Exercise:	1 (CW)	**Due:**	Mon - 12 Feb 2018
Title:	Coursework	**Assessment:**	Group
FAO:	Lupu, Emil (ecl1)	**Submission:**	Hardcopy AND Electronic

Student Declaration - Version 3

- I acknowledge the following people for help through our original discussions:

Harrison Ankers (haa516) Keerthanen Ravichandran (kr1116)

Signed: _____ Date: _____

For Markers only: (circle appropriate grade)

ANKERS, Harrison (haa516) LEADER	01211208	i2	2018-02-11 19:54:45	A*	A	B	C	D	E	F
RAVICHANDRAN, Keerthanen (kr1116)	01195170	i2	2018-02-11 19:57:32	A*	A	B	C	D	E	F

527 — Computer Networks and Distributed Systems — Assessed Coursework: RMI and UDP

User Datagram Protocol

UDP mechanism involves sending information in the form of packets called Datagrams. In this example, we were sending information contained in the common class called MessageInfo, which contains an integer of the number of messages to be sent and a message number identifier.

The mechanism was used to send information of increasing size, starting from 20 messages up to 2000. The results showed a clear increase in lost packets after 300 messages. Messages up to the 300 point were safely delivered. The pattern formed following that was messages were lost in sequential blocks. For example, messages of size 500 occurred to have an instance of sequential losses, while 2000 had about 2-3 blocks of sequential losses.

The main cause of the loss of messages in a UDP mechanism is the lack of overheads. This means messages are simply sent however, there are no checks made to ensure they were safely transmitted in order or possibly duplicated. Furthermore, the more messages sent, the greater the congestion. UDP's lack of congestion control, means if the port is overloaded, more packets are lost. No mechanism exists to retransmit lost data, so the chances of losing data is greater, as seen in the results. It was suggested to check the last message to confirm all messages had been sent however, this resulted in the server timing out if the last message did not arrive. This was a significant weakness of using UDP, as it caused often caused the server to timeout and the need for several attempts in sending before obtaining results.

Remote Method Invocation

RMI allows an object on a client machine to invoke remote method on another object running on a difference Java Virtual Machine, in this context, our server machine. It was found with RMI mechanism that there were no messages lost, even when sending a significant number of messages.

This is because RMI transmits and checks whether a message has been sent, then retransmit it to ensure all messages are sent. This means problems such as buffer overflows and network congestion at the receiver end are taken care of. The only downside was an increased time as more messages were sent due to more overheads as messages had to be checked for receipt.

Conclusion

From the results, RMI was far more reliable than UDP. At significantly high levels of messages, there were no lost messages, while UDP failed to send several messages after 300. Furthermore, the process of coding RMI was easier as sending and receiving information was dynamically handled by encapsulated marshalling arguments, while in UDP, the process of sending packets and ports had to be manually configured. There was no need to understand how the client communicated with the server, due to the abstraction of the RMI. Although, it was found running UDP was often faster than RMI due to the lack of overheads.

UDP			
No. Messages Sent	Average No. Received	Average No. Lost	Percentage loss
20	20	0	0
40	40	0	0
60	60	0	0
80	80	0	0
100	100	0	0
200	200	0	0
300	300	0	0
400	350	50	12.5
500	376	124	24.8
1000	788	212	21.2
2000	1134	866	43.3

Figure 1: UDP results taken over an average of ten readings

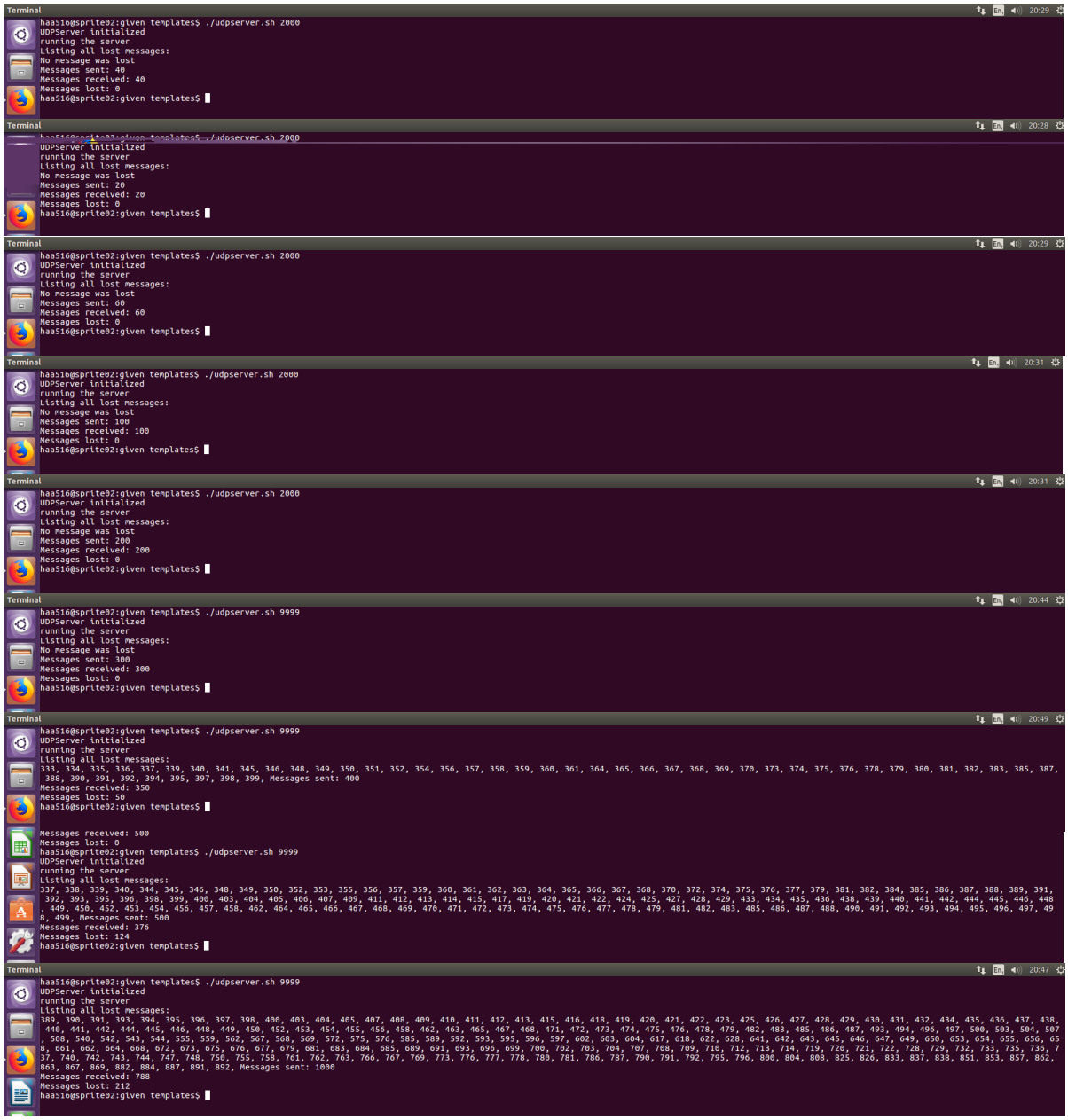
RMI		
No. Messages Sent	Average No. Received	Average No. Lost
20	20	0
40	40	0
60	60	0
80	80	0
100	100	0
200	200	0
300	300	0
400	400	0
500	500	0
1000	1000	0

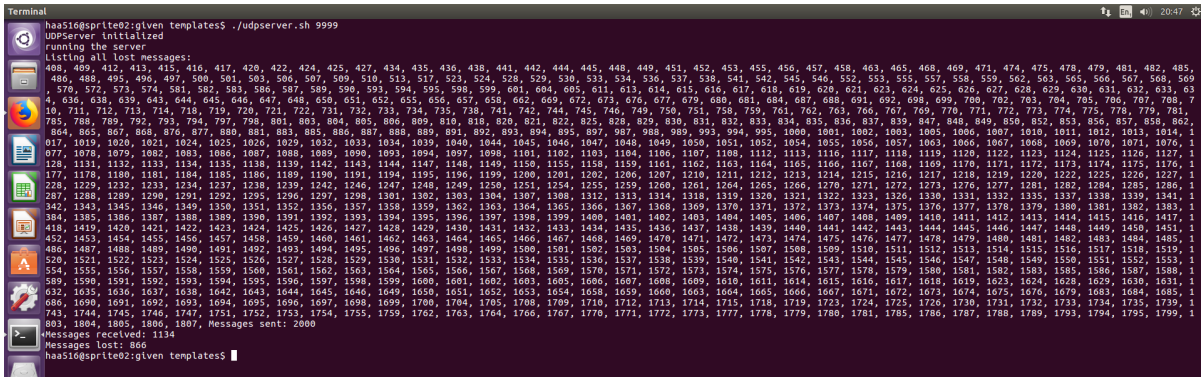
2000	2000	0

Figure 2: RMI results taken over an average of ten readings

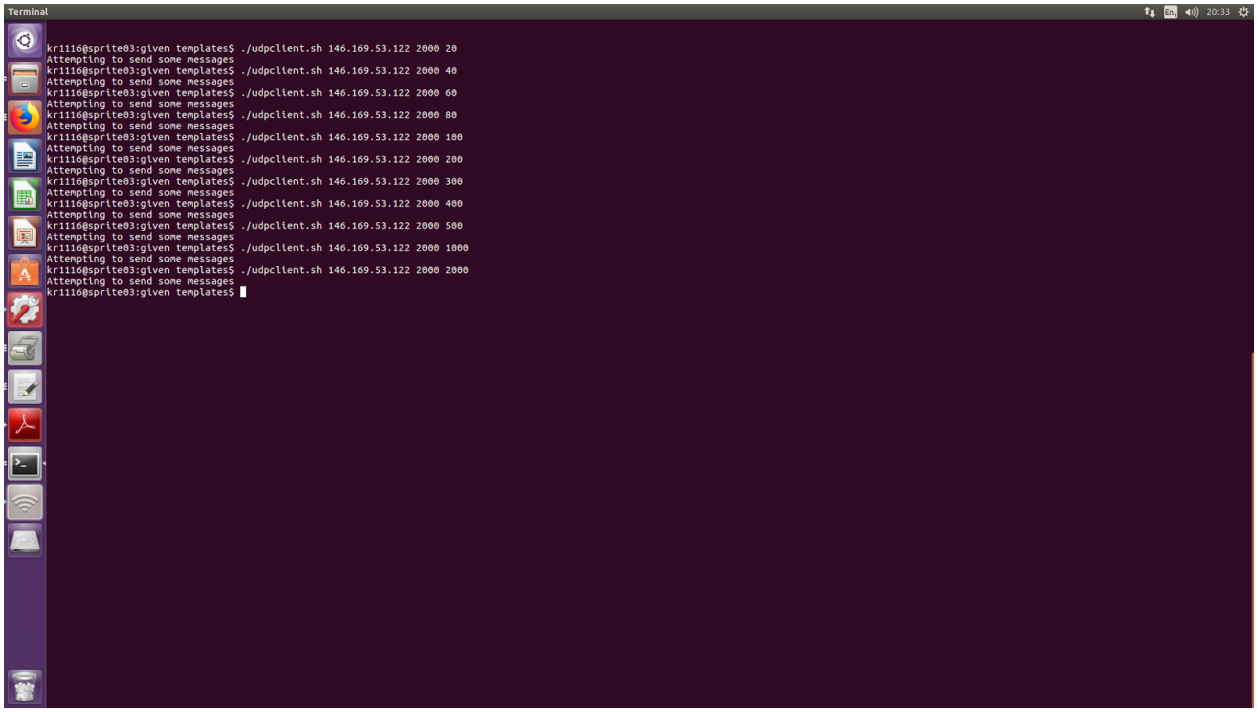
Logs

UDP Server

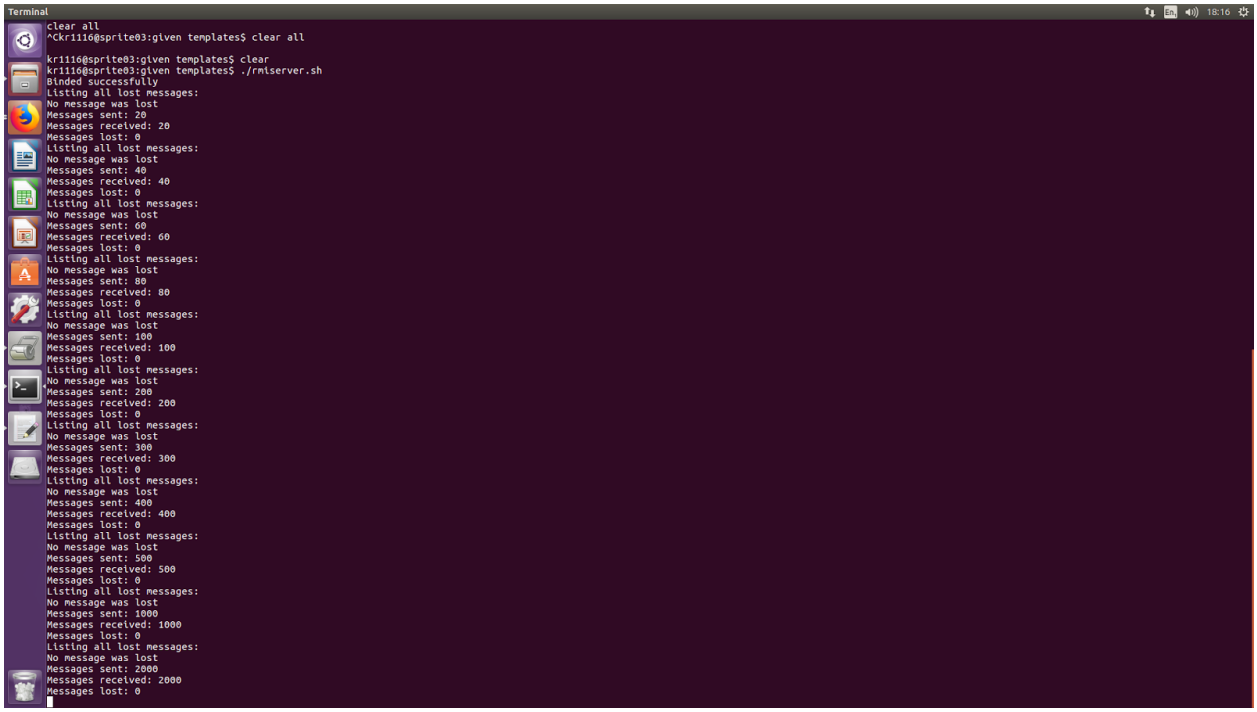



Terminal
haa516@sprite02:given templates$ ./udpserver.sh 9999
UDPServer initialized
running the server
Listing all lost messages:
Messages received: 1134
Messages lost: 866
haa516@sprite02:given templates$

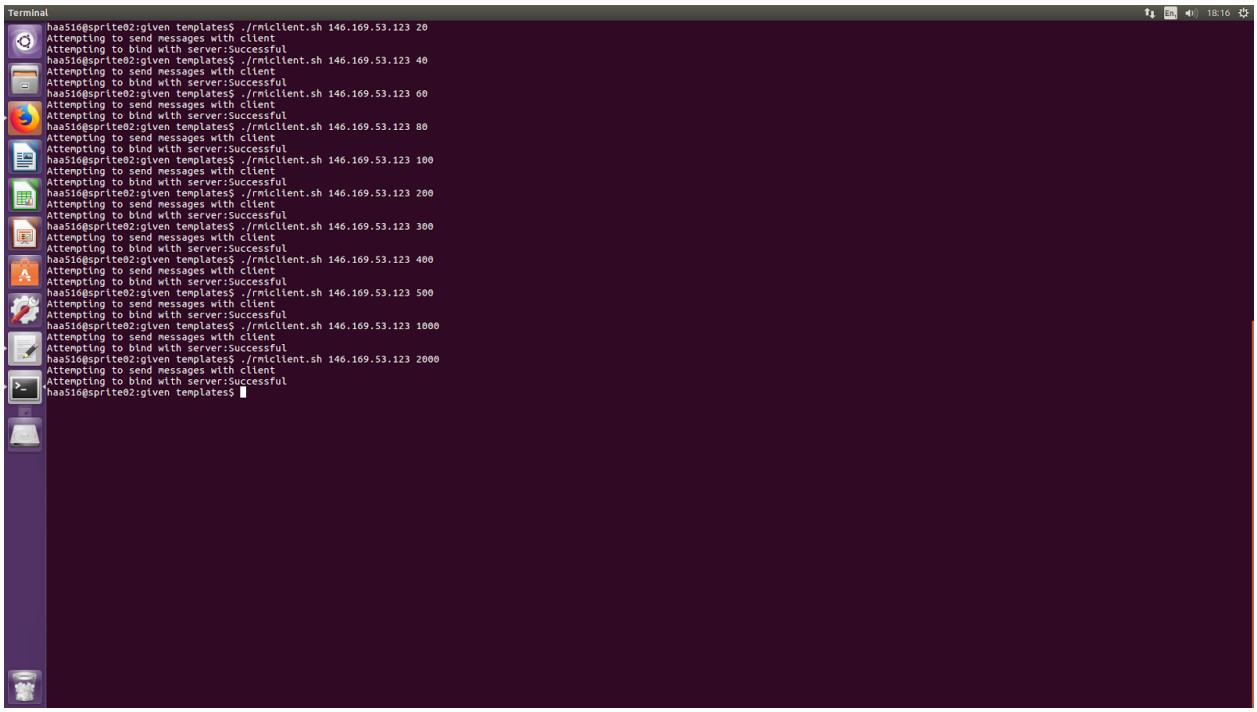
UDP Client


Terminal
kr1116@sprite03:given templates$ ./udpclient.sh 146.169.53.122 2000 20
Attempting to send some messages
kr1116@sprite03:given templates$ ./udpclient.sh 146.169.53.122 2000 40
Attempting to send some messages
kr1116@sprite03:given templates$ ./udpclient.sh 146.169.53.122 2000 60
Attempting to send some messages
kr1116@sprite03:given templates$ ./udpclient.sh 146.169.53.122 2000 80
Attempting to send some messages
kr1116@sprite03:given templates$ ./udpclient.sh 146.169.53.122 2000 100
Attempting to send some messages
kr1116@sprite03:given templates$ ./udpclient.sh 146.169.53.122 2000 200
Attempting to send some messages
kr1116@sprite03:given templates$ ./udpclient.sh 146.169.53.122 2000 300
Attempting to send some messages
kr1116@sprite03:given templates$ ./udpclient.sh 146.169.53.122 2000 400
Attempting to send some messages
kr1116@sprite03:given templates$ ./udpclient.sh 146.169.53.122 2000 500
Attempting to send some messages
kr1116@sprite03:given templates$ ./udpclient.sh 146.169.53.122 2000 1000
Attempting to send some messages
kr1116@sprite03:given templates$ ./udpclient.sh 146.169.53.122 2000 2000
Attempting to send some messages
kr1116@sprite03:given templates$

RMI Server



RMI client



Completed Code:

1. RMI Client

```
package rmi;

import java.rmi.Naming;
import java.rmi.NotBoundException;
import java.rmi.RemoteException;

import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;

import java.rmi.RMISecurityManager;

import common.MessageInfo;

public class RMIClient {

	public static void main(String[] args) {
		System.out.println("Attempting to send messages with client");
		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]);

		// TO-DO: Initialise Security Manager
	    //sets security manager
	 System.setSecurityManager(new RMISecurityManager());

	 try{
	System.out.print("Attempting to bind with server:");
		// TO-DO: Bind to RMIServer
	    //checks the url for exceptions and bings the lookup server
	 iRMIServer = (RMIServerI) Naming.lookup(urlServer);
	System.out.println("Successful");


		// TO-DO: Attempt to send messages the specified number of times
	 for( int i=0; i<numMessages; i++){
	    //creates a new message and calls RMI server receive this message
	    MessageInfo msg = new MessageInfo(numMessages,i);
	    iRMIServer.receiveMessage(msg);
	 }
} catch (Exception e) {
System.out.println("Error in client");
}

	}
}
```

2. RMI Server

```
/*
 * 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 java.rmi.registry.Registry;

import common.*;

public class RMIServer extends UnicastRemoteObject implements RMIServerI {

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

	public RMIServer() throws RemoteException {
	}

	public void receiveMessage(MessageInfo msg) throws RemoteException {

		// TO-DO: On receipt of first message, initialise the receive buffer
	    //initialises the array
	 if(receivedMessages == null){
	    totalMessages = msg.totalMessages;
	    receivedMessages = new int[msg.totalMessages];
	 }
	System.out.println("Recieving messages");

		// TO-DO: Log receipt of the message
	    //verifies a recepit of a message in the array
	 receivedMessages[msg.messageNum] = 1;

		// TO-DO: If this is the last expected message, then identify
		//        any missing messages
	    //note: that if it didn't receive the last message all the messages would
not be logged
	 if(msg.messageNum + 1 == totalMessages){
	    System.out.println("Listing all lost messages: ");
	    int count = 0;
	    for(int i=0; i<totalMessages; i++) {
	        if (receivedMessages[i] != 1) {
	            count++;
	            System.out.print(i+1 + ", ");
	        }
	    }

	    if (count == 0){
	        System.out.println("No message was lost");
	    }
	    System.out.println("Messages sent: " + totalMessages);
	    System.out.println("Messages received: " + (totalMessages - count));
	    System.out.println("Messages lost: " + count);
	   receivedMessages = null;
```

```
        }
       }


       public static void main(String[] args) {

              RMIServer rmiserv = null;

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

       try{
              // TO-DO: Instantiate the server class
        rmiserv = new RMIServer();

              // TO-DO: Bind to RMI registry
        rebindServer("RMIServer", rmiserv);
       System.out.println("Binded successfully");
       } catch(Exception e) {
              System.out.println("Error in RMISERV");
       }
       }

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

        LocateRegistry.createRegistry(1099);

              // 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.

        Naming.rebind(serverURL, server);
       } catch (Exception e){
              System.out.println("Error binding");
       }
       }
}
```

3. UDP Client

```
/*
 * 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 java.io.*;

import common.MessageInfo;

public class UDPClient {

	private DatagramSocket sendSoc;

	public static void main(String[] args) {
		InetAddress   serverAddr = null;
		//Inetaddress class representing an IP address
		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]);
		//gets host's ip address given the host name
		} 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]);
		//parseInt--string to integer variable

		// TO-DO: Construct UDP client class and try to send messages
		UDPClient client = new UDPClient();
		//UDP client is a class and initialised here
		System.out.println("Attempting to send some messages");
		client.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("Socket error");
}
	}

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

		// TO-DO: Send the messages to the server
```

```
                MessageInfo mess;
                ByteArrayOutputStream byteStream;
                ObjectOutputStream oos;

                for(int i = 0; i < countTo; i++) {
                        mess = new MessageInfo(countTo,i);

                        //thought 2000 would be enough
                        byteStream = new ByteArrayOutputStream(60000);
                        try {
                                oos = new ObjectOutputStream(new
BufferedOutputStream(byteStream));
                                oos.writeObject(mess);
                                oos.flush();
                        } catch (Exception e) {
                                System.out.println("Error doing the client stream");
                                System.exit(-1);
                        }

                        byte[] sendByteArr = byteStream.toByteArray();
                        send(sendByteArr, serverAddr, recvPort);
                }
        }

        private void send(byte[] data, InetAddress destAddr, int destPort) {
                DatagramPacket       pkt;

                // TO-DO: build the datagram packet and send it to the server
                pkt = new DatagramPacket(data, data.length, destAddr, destPort);
                try {
                        sendSoc.send(pkt);
                } catch (IOException e) {
                        System.out.println("Error sending");
                        System.exit(-1);
}
        }
}
```

4. UDP Server

```
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 java.io.*;

import common.MessageInfo;

public class UDPServer {

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

	private void run() throws SocketTimeoutException{
		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){
			//pacsize is randomly chosen has to match client.java we thought
2000 would be enough
			pacSize = 60000;
			pacData = new byte[60000];

			pac = new DatagramPacket(pacData, pacSize);
			try {
				//thought this would be 30 seconds as it's in milliseconds
				recvSoc.setSoTimeout(30000);
				recvSoc.receive(pac);
			} catch (IOException e) {
				System.out.println("Error receiving packet");
				System.exit(-1);
			}
			processMessage(pac.getData());
		}
	}

	public void processMessage(byte[] data) {

		MessageInfo msg = null;

		// TO-DO: Use the data to construct a new MessageInfo object
		ByteArrayInputStream byteStream = new ByteArrayInputStream(data);
		ObjectInputStream ois;

		try {
			ois = new ObjectInputStream(new BufferedInputStream(byteStream));
			msg = (MessageInfo) ois.readObject();
			ois.close();
		} catch (Exception e) {
			System.out.println("Error creating the stream");
			System.exit(-1);
		}

		// 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
                receivedMessages[msg.messageNum] = 1;

                // TO-DO: If this is the last expected message, then identify
                //        any missing messages
                //note won't print if the last message is lost
                if (msg.messageNum + 1 == msg.totalMessages) {
                        close = true;

                        System.out.println("Listing all lost messages: ");
                        int count = 0;
                        for (int i = 0; i < totalMessages; i++) {
                                if (receivedMessages[i] != 1) {
                                        count++;
                                        System.out.print(i+1 + ", ");
                                }
                        }

                        if (count == 0){
                                System.out.println("No message was lost");
                        }

                        System.out.println("Messages sent: " + totalMessages);
                        System.out.println("Messages received: " + (totalMessages -
count));
                        System.out.println("Messages lost: " + count);
                        receivedMessages = null;
                }

        }


        public UDPServer(int portno) {
                // TO-DO: Initialise UDP socket for receiving data
                try {
                        recvSoc = new DatagramSocket(portno);
                } catch (SocketException e) {
                        System.out.println("Error creating the server port");
                        System.exit(-1);
                }
                System.out.println("UDPServer initialized");
        }

        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 udpserv = new UDPServer(recvPort);
                try {
                        System.err.println("running the server");
                        udpserv.run();
                } catch (SocketTimeoutException e) {
                        System.err.println("Error Socket Timeout");
                }
        }

}
```