



527 – Computer Networks and Distributed Systems

Imperial College London

Department of Computing

RMI and UDP Networks

FEBRUARY 12, 2018

Muhamad Gafar

1. Program's Working Mechanism

a.) For each mechanism, what are the possible causes, if any, of messages being lost?

In the case of Remote Method Invocation (RMI) the protocol employs a Connection-Oriented Communication mechanism with a bi-directional stream of data between the client and the server. RMI is implemented a level over Transmission Control Protocol (TCP), this abstraction manages the retransmissions, ordering of messages, and any congestion experienced in the buffer making it a very reliable mechanism that doesn't lose messages. Nevertheless, RMI based networks could lose messages if the connection is interrupted unexpectedly.

On the other hand, in UDP connections there is no built-in mechanism for retransmission of data, acknowledgement of receipt, or management of congestion and ordering as it is a Connectionless Communication. This means that unless the application is written specifically to overcome these problems there won't be a guarantee of delivery as IP packets could be dropped due to network errors or hardware/software malfunction or congestion/overflowing buffers.

b.) Are there any patterns in the way messages are lost?

All test carried out with the RMI server and client show that for all the number of message tested (below 2000) not a single message was dropped.

Figures 6-8 show the behaviors observed when testing the UDP connection. It performs modestly well when 304 messages or less are sent but occasionally some messages are dropped. No specific pattern was observed for the lost messages as rerunning the commands show that the message number of the lost messages continuously changes on every run. When sending over 304 messages there are times where no message is dropped but the buffer is congested and so the server stops processing incoming messages.

c.) What is the relative reliability of the different communication mechanisms?

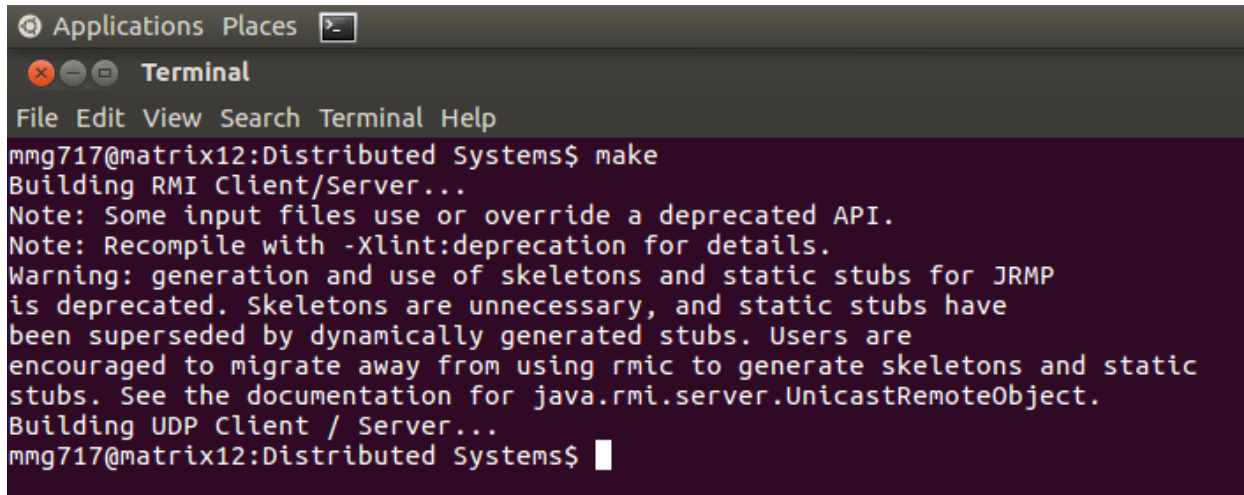
The RMI system has shown to be vastly superior in terms of its reliability. Despite testing this connection multiple times with varying amounts of transferred messages the system still received all messages. On the other hand, the UDP system wasn't very reliable particularly when the total number of messages to be sent were 300 and above. This disparity can be explained by the coupling between the server and provider in the RMI protocol. UDP systems require implementation of safety mechanisms to be done at the application for managing congestion and lost packets.

d.) Which was easier to program and why?

The only difficulty experienced was with the syntax as I had to migrate from C++ and Python environments. However, the programs themselves were not challenging to program since the lectures were straightforward and there are numerous examples of both implementations online. Overall RMI was slightly easier because of how well it was laid out and explained in the "Head First Java" book as well as the course recommended book "Distributed Systems – Concepts and Design" by Coulouris G.

2. Terminal Logs

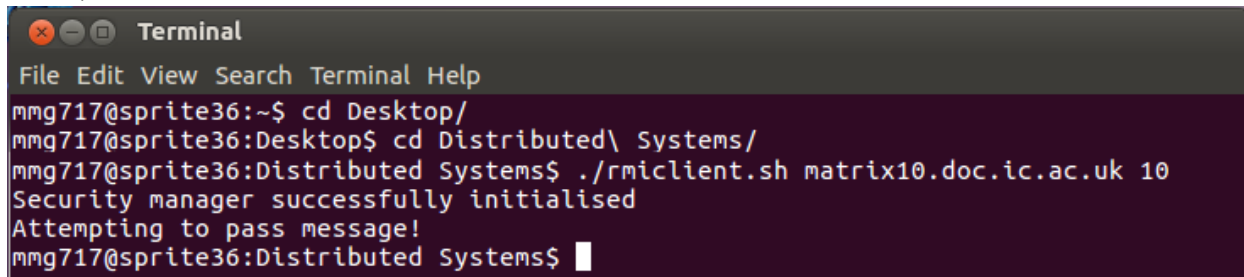
a.) Compilation of RMI and UDP Server and Client

A terminal window titled 'Terminal' with a menu bar (File, Edit, View, Search, Terminal, Help). The prompt is 'mmg717@matrix12:Distributed Systems\$'. The user enters 'make'. The output shows 'Building RMI Client/Server...' followed by several deprecation warnings from the Java compiler. The final output is 'Building UDP Client / Server...' and the prompt returns.

```
mmg717@matrix12:Distributed Systems$ make
Building RMI Client/Server...
Note: Some input files use or override a deprecated API.
Note: Recompile with -Xlint:deprecation for details.
Warning: generation and use of skeletons and static stubs for JRMP
is deprecated. Skeletons are unnecessary, and static stubs have
been superseded by dynamically generated stubs. Users are
encouraged to migrate away from using rmic to generate skeletons and static
stubs. See the documentation for java.rmi.server.UnicastRemoteObject.
Building UDP Client / Server...
mmg717@matrix12:Distributed Systems$
```

Figure 1: Compilation of the RMI Client/Server and UDP Client/Server

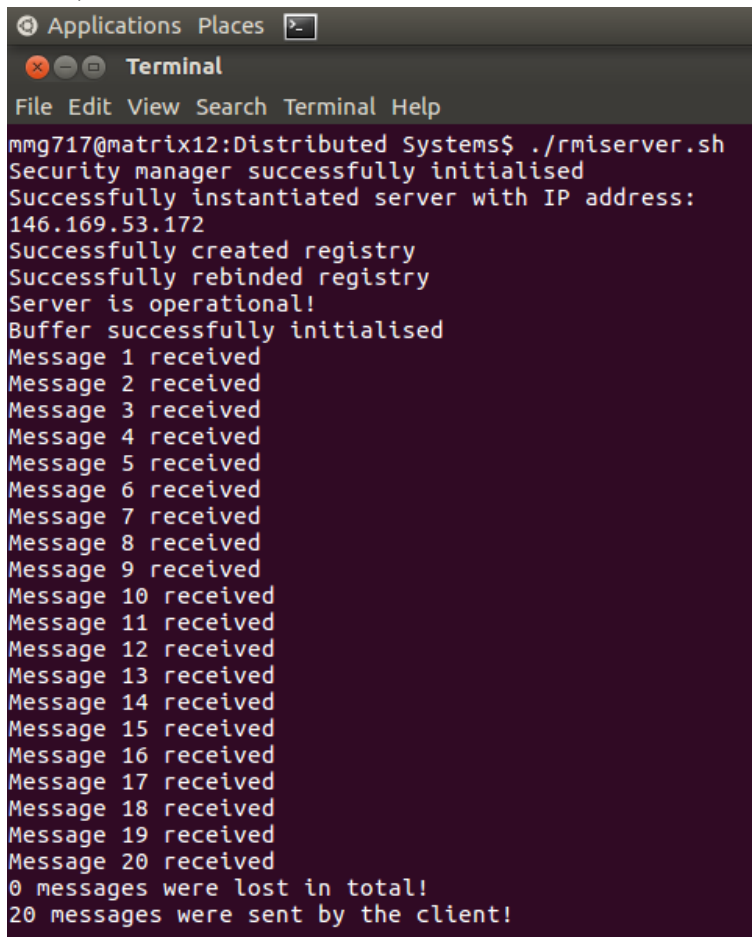
b.) RMI Client

A terminal window titled 'Terminal' with a menu bar (File, Edit, View, Search, Terminal, Help). The prompt is 'mmg717@sprite36:~\$'. The user enters 'cd Desktop/'. The prompt changes to 'mmg717@sprite36:Desktop\$'. The user enters 'cd Distributed\ Systems/'. The prompt changes to 'mmg717@sprite36:Distributed Systems\$'. The user enters './rmiclient.sh matrix10.doc.ic.ac.uk 10'. The output shows 'Security manager successfully initialised' and 'Attempting to pass message!'. The prompt returns.

```
mmg717@sprite36:~$ cd Desktop/
mmg717@sprite36:Desktop$ cd Distributed\ Systems/
mmg717@sprite36:Distributed Systems$ ./rmiclient.sh matrix10.doc.ic.ac.uk 10
Security manager successfully initialised
Attempting to pass message!
mmg717@sprite36:Distributed Systems$
```

Figure 2: Console output when running RMI Client successfully connects and passes messages to a server.

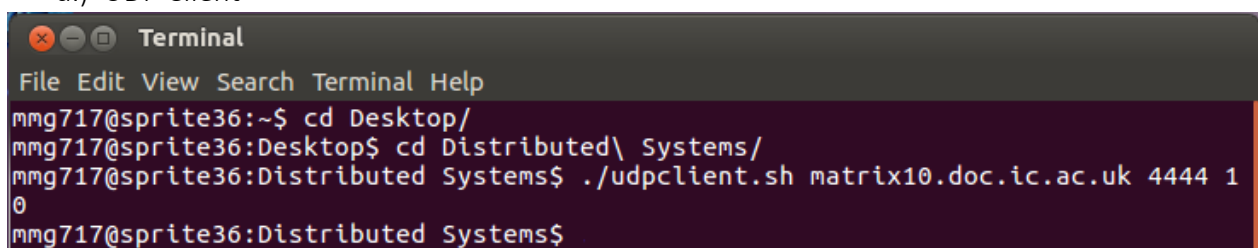
c.) RMI Server



```
Applications Places [icon]
Terminal
File Edit View Search Terminal Help
mmg717@matrix12:Distributed Systems$ ./rmiserver.sh
Security manager successfully initialised
Successfully instantiated server with IP address:
146.169.53.172
Successfully created registry
Successfully rebounded registry
Server is operational!
Buffer successfully initialised
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
Message 11 received
Message 12 received
Message 13 received
Message 14 received
Message 15 received
Message 16 received
Message 17 received
Message 18 received
Message 19 received
Message 20 received
0 messages were lost in total!
20 messages were sent by the client!
```

Figure 3: Successful initialization and message processing of RMI server.

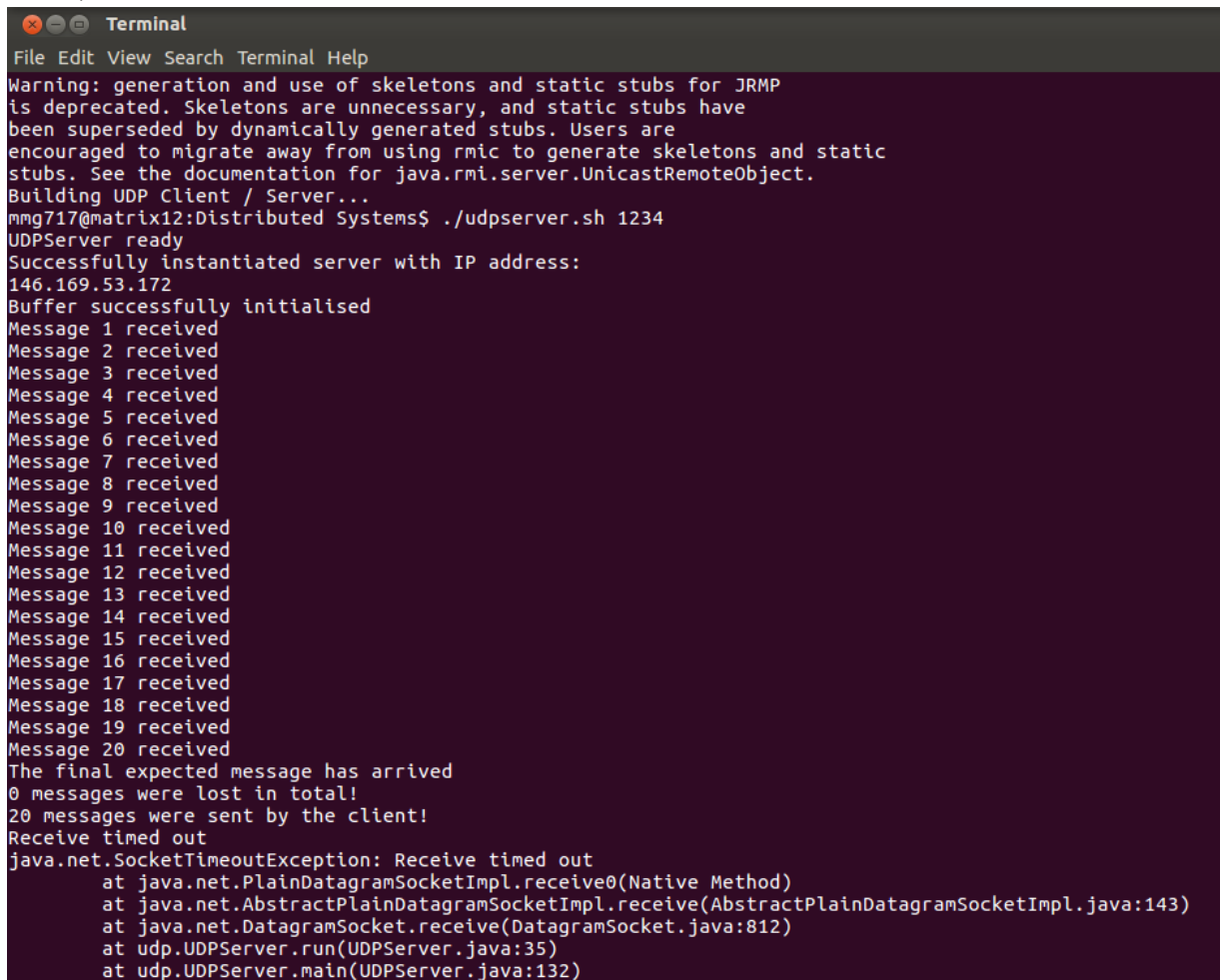
d.) UDP Client



```
Terminal
File Edit View Search Terminal Help
mmg717@sprite36:~$ cd Desktop/
mmg717@sprite36:Desktop$ cd Distributed\ Systems/
mmg717@sprite36:Distributed Systems$ ./udpclient.sh matrix10.doc.ic.ac.uk 4444 1
0
mmg717@sprite36:Distributed Systems$
```

Figure 4: Console output when running UDP Client successfully connects and passes messages to a server.

e.) UDP Server

A terminal window titled "Terminal" with a menu bar (File, Edit, View, Search, Terminal, Help). The output shows a warning about deprecated JRMP, followed by the command to build the UDP client/server. The user runs a script to start the server on port 1234. The server reports its IP address and successfully initializes the buffer. It then receives 20 messages, one by one, from "Message 1 received" to "Message 20 received". After the 20th message, it reports that the final expected message has arrived, that 0 messages were lost, and that 20 messages were sent by the client. Finally, it reports a "Receive timed out" error with a full Java stack trace for java.net.SocketTimeoutException.

```
Warning: generation and use of skeletons and static stubs for JRMP
is deprecated. Skeletons are unnecessary, and static stubs have
been superseded by dynamically generated stubs. Users are
encouraged to migrate away from using rmic to generate skeletons and static
stubs. See the documentation for java.rmi.server.UnicastRemoteObject.
Building UDP Client / Server...
mmg717@matrix12:Distributed Systems$ ./udpserver.sh 1234
UDPServer ready
Successfully instantiated server with IP address:
146.169.53.172
Buffer successfully initialised
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
Message 11 received
Message 12 received
Message 13 received
Message 14 received
Message 15 received
Message 16 received
Message 17 received
Message 18 received
Message 19 received
Message 20 received
The final expected message has arrived
0 messages were lost in total!
20 messages were sent by the client!
Receive timed out
java.net.SocketTimeoutException: Receive timed out
    at java.net.PlainDatagramSocketImpl.receive0(Native Method)
    at java.net.AbstractPlainDatagramSocketImpl.receive(AbstractPlainDatagramSocketImpl.java:143)
    at java.net.DatagramSocket.receive(DatagramSocket.java:812)
    at udp.UDPServer.run(UDPServer.java:35)
    at udp.UDPServer.main(UDPServer.java:132)
```

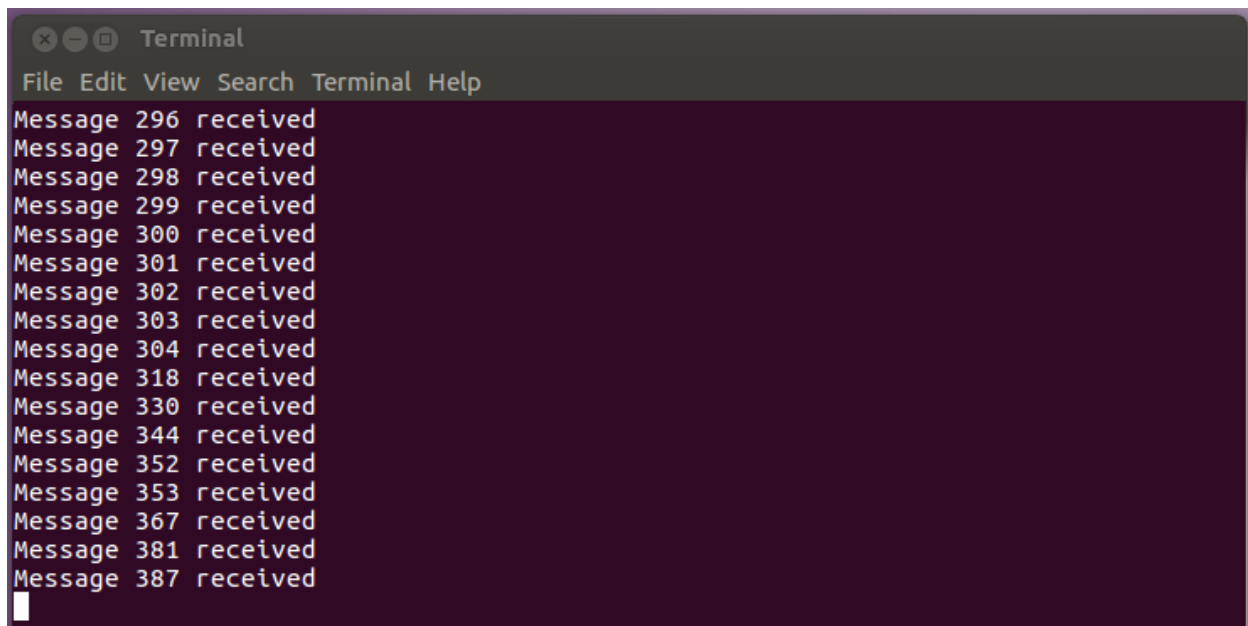
Figure 5: Successful initialization and message processing of UDP server until the socket timeout.

```

Message 282 received
Message 283 received
Message 284 received
Message 285 received
Message 286 received
Message 287 received
Message 288 received
Message 289 received
Message 290 received
Message 291 received
Message 292 received
Message 293 received
Message 294 received
Message 295 received
Message 296 received
Message 297 received
Message 298 received
Message 299 received
Message 300 received
The final expected message has arrived
0 messages were lost in total!
300 messages were sent by the client!
Receive timed out
java.net.SocketTimeoutException: Receive timed out
    at java.net.PlainDatagramSocketImpl.receive0(Native Method)
    at java.net.AbstractPlainDatagramSocketImpl.receive(AbstractPlainDatagramSocketImpl.java:143)
    at java.net.DatagramSocket.receive(DatagramSocket.java:812)
    at udp.UDPServer.run(UDPServer.java:35)
    at udp.UDPServer.main(UDPServer.java:132)
mmg717@matrix12:Distributed Systems$ make
Building RMI Client/Server...
Note: Some input files use or override a deprecated API.
Note: Recompile with -Xlint:deprecation for details.
Warning: generation and use of skeletons and static stubs for JRMP
is deprecated. Skeletons are unnecessary, and static stubs have
been superseded by dynamically generated stubs. Users are
encouraged to migrate away from using rmic to generate skeletons and static
stubs. See the documentation for java.rmi.server.UnicastRemoteObject.
Building UDP Client / Server...
mmg717@matrix12:Distributed Systems$ make
Building RMI Client/Server...
Note: Some input files use or override a deprecated API.
Note: Recompile with -Xlint:deprecation for details.
Warning: generation and use of skeletons and static stubs for JRMP
is deprecated. Skeletons are unnecessary, and static stubs have

```

Figure 6: Largest number of messages the UDP server receives successfully.

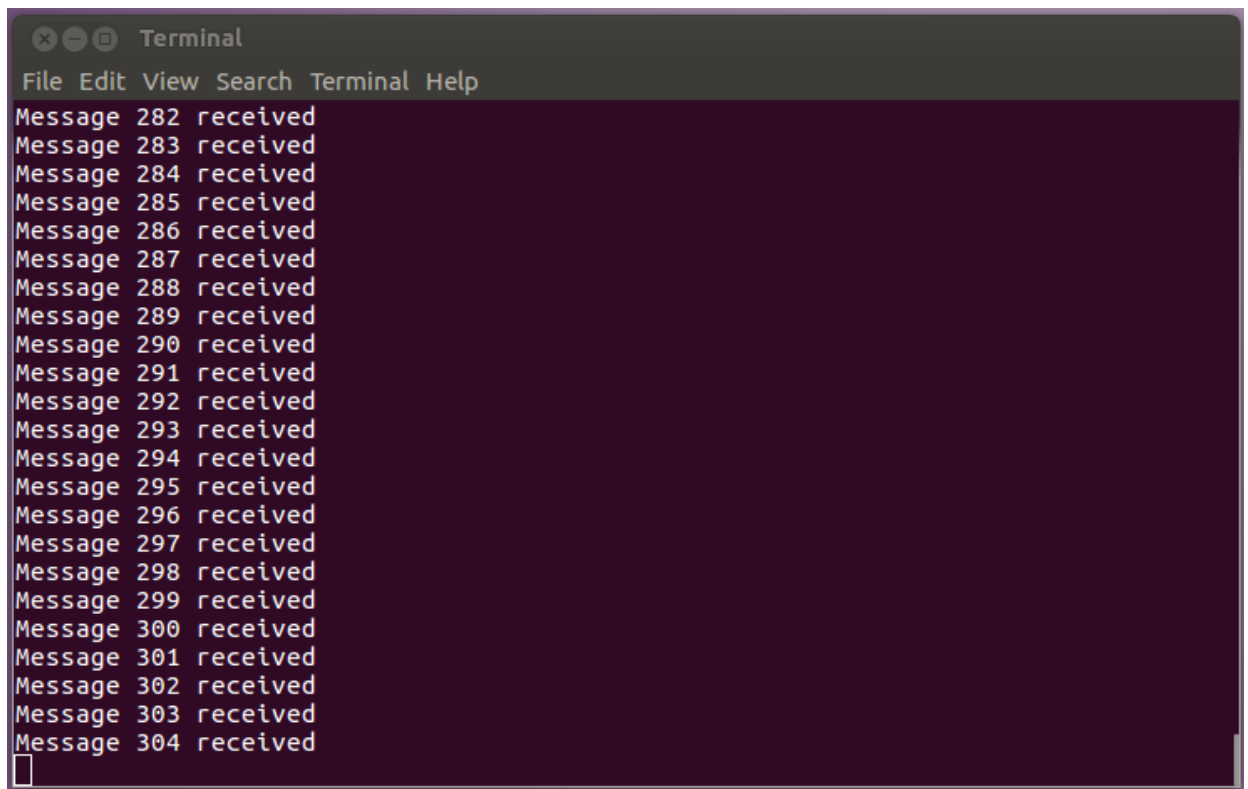


A terminal window titled "Terminal" with a menu bar containing "File", "Edit", "View", "Search", "Terminal", and "Help". The terminal displays a list of received messages. The messages are numbered 296 through 387, with some gaps (e.g., 304 is missing, 318 is present). The list ends with a cursor on the line "Message 387 received".

```
Message 296 received
Message 297 received
Message 298 received
Message 299 received
Message 300 received
Message 301 received
Message 302 received
Message 303 received
Message 304 received
Message 318 received
Message 330 received
Message 344 received
Message 352 received
Message 353 received
Message 367 received
Message 381 received
Message 387 received

```

Figure 7: Unpredictable message losses with UDP protocol.



A terminal window titled "Terminal" with a menu bar containing "File", "Edit", "View", "Search", "Terminal", and "Help". The terminal displays a list of received messages. The messages are numbered 282 through 304, with no gaps. The list ends with a cursor on the line "Message 304 received".

```
Message 282 received
Message 283 received
Message 284 received
Message 285 received
Message 286 received
Message 287 received
Message 288 received
Message 289 received
Message 290 received
Message 291 received
Message 292 received
Message 293 received
Message 294 received
Message 295 received
Message 296 received
Message 297 received
Message 298 received
Message 299 received
Message 300 received
Message 301 received
Message 302 received
Message 303 received
Message 304 received

```

Figure 8: RMI Server doesn't lose any message but stops after 304 messages are received.

3.) Source Code Listings

RMIClient.java

```
package rmi;

import java.net.MalformedURLException;
import java.rmi.Naming;
import java.rmi.NotBoundException;
import java.rmi.RemoteException;
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("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
        if(System.getSecurityManager() == null)
        {
            System.setSecurityManager(new RMISecurityManager());
            System.out.println("Security manager successfully initialised");
        } else
            System.out.println("A security manager already exists");

        // TO-DO: Bind to RMIServer
        boolean server_is_active = false;
        try
        {
            iRMIServer = (RMIServerI) Naming.lookup(urlServer);
            server_is_active = true;
        } catch (MalformedURLException e)
        {
            System.out.println(e.getMessage());
            e.printStackTrace();
        } catch (NotBoundException e)
        {
            System.out.println(e.getMessage());
            e.printStackTrace();
        } catch (RemoteException e)
        {
            System.out.println(e.getMessage());
            e.printStackTrace();
        }

        // TO-DO: Attempt to send messages the specified number of times
        if(server_is_active)
        {
            System.out.println("Attempting to pass message!");
            try
```



```
{
    for(int i = 0; i < numMessages; i++)
    {
        MessageInfo message = new MessageInfo(numMessages, i);
        iRMIServer.receiveMessage(message);
    }
} catch (RemoteException e)
{
    System.out.println(e.getMessage());
    e.printStackTrace();
} finally
{
    System.exit(0);
}
}
```

RMIserver.java

```
package rmi;

import java.rmi.Naming;
import java.rmi.registry.Registry;
import java.rmi.registry.LocateRegistry;
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;
import java.util.Arrays;
import java.rmi.RMISecurityManager;
import java.net.InetAddress;

import common.*;

public class RMIserver extends UnicastRemoteObject implements RMIserverI {

    private int totalMessages = -1;
    private int[] receivedMessages;
    private static int port = 1099;
    private static String hostname = "146.169.53.13";

    public RMIserver() throws RemoteException {
    }

    public void receiveMessage(MessageInfo msg) throws RemoteException
    {
        // TO-DO: On receipt of first message, initialise the receive buffer
        if(msg.messageNum == 0)
        {
            totalMessages = msg.totalMessages;
            receivedMessages = new int[totalMessages];
            System.out.println("Buffer successfully initialised");
        }

        // 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
        if(msg.messageNum == totalMessages - 1)
        {
            int missing_messages = 0;
            for(int i = 0; i < totalMessages; i++)
            {
                if(receivedMessages[i] != 1)
                {
                    System.out.println("Message " + (i+1) + "was lost!");
                    missing_messages++;
                }
            }
            System.out.println(missing_messages + " messages were lost in total!");
            System.out.println(msg.totalMessages + " messages were sent by the
client!");
        }
    }

    public static void main(String[] args)
    {
        RMIserver rmis = null;
```

```

// TO-DO: Initialise Security Manager
if(System.getSecurityManager() == null)
{
    System.setSecurityManager(new RMISecurityManager());
    System.out.println("Security manager successfully initialised");
} else
    System.out.println("A security manager already exists");

// TO-DO: Instantiate the server class
try
{
    rmis = new RMIServer();
    System.out.println("Successfully instantiated server with IP address: ");
    System.out.println(InetAddress.getLocalHost().getHostAddress());
} catch (Exception e)
{
    System.out.println(e.getMessage());
    e.printStackTrace();
}

// TO-DO: Bind to RMI registry
String urlServer = new String("rmi://" + hostname + ":" + port +
"/RMIServer");
rebindServer(port, rmis);
}

protected static void rebindServer(int port, 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)
    boolean already_created = false;
    try
    {
        LocateRegistry.createRegistry(port);
        System.out.println("Successfully created registry");
    } catch (Exception e)
    {
        already_created = true;
        System.out.println(e.getMessage());
        e.printStackTrace();
    }

    try
    {
        if(already_created)
        {
            System.out.println("Attempting to get existing registry");
            LocateRegistry.getRegistry(port);
        }

        // 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("RMIServer", server);
        System.out.println("Successfully rebinded registry");
    }
}

```

```
        System.out.println("Server is operational!");  
    } catch (Exception e)  
    {  
        System.out.println(e.getMessage());  
        e.printStackTrace();  
    }  
}
```

UDPCClient.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 UDPCClient {

    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("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 UDPCClient, " + 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
        UDPCClient client = new UDPCClient();
        client.testLoop(serverAddr, recvPort, countTo);
    }

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

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

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

```

        while(tries < countTo)
        {
            MessageInfo msg = new MessageInfo(countTo, tries);
            send(msg.toString(), serverAddr, recvPort);
            tries++;
        }
    }

    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(e.getMessage());
            e.printStackTrace();
        }
    }
}

```

UDPServer.java

```
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.net.InetAddress;

import common.MessageInfo;

public class UDPServer {

    private DatagramSocket recvSoc = null;
    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
        try
        {
            pacSize = 1024;
            while(true)
            {
                pacData = new byte[pacSize];
                pac = new DatagramPacket(pacData, pacSize);
                recvSoc.setSoTimeout(30000);
                recvSoc.receive(pac);
                processMessage(new String(pac.getData()));
            }
        } catch (SocketException e)
        {
            System.out.println(e.getMessage());
            e.printStackTrace();
        } catch (IOException e)
        {
            System.out.println(e.getMessage());
            e.printStackTrace();
        }
    }

    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.getMessage());
            e.printStackTrace();
        }
    }
}
```

```

    }

    // TO-DO: On receipt of first message, initialise the receive buffer
    if(msg.messageNum == 0)
    {
        totalMessages = msg.totalMessages;
        receivedMessages = new int[totalMessages];
        System.out.println("Buffer successfully initialised");
    }

    // 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
    if(msg.messageNum == totalMessages - 1)
    {
        System.out.println("The final expected message has arrived");
        int missing_messages = 0;
        for(int i = 0; i < totalMessages; i++)
        {
            if(receivedMessages[i] != 1)
            {
                System.out.println("Message " + (i+1) + " was lost!");
                missing_messages++;
            }
        }
        System.out.println(missing_messages + " messages were lost in total!");
        System.out.println(msg.totalMessages + " messages were sent by the
client!");
    }
}

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

    // 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

```



```
    {
        System.out.println("Successfully instantiated server with IP address: ");
        System.out.println(InetAddress.getLocalHost().getHostAddress());
    } catch (Exception e)
    {
        System.out.println(e.getMessage());
        e.printStackTrace();
    }

    server.run();
}
}
```