Computer Networks and Distributed Systems - UDP & RMI

Aufar Laksana

February 13th 2017

1   Introduction

The purpose of this coursework was to understand how RMI and UDP work, by coding a server and client for each method.

2   User Datagram Protocol (UDP)

UDP provides an unreliable service. There are no guarantees for the delivery of the messages or prevention for duplication of the data. This reduces the overhead of using the protocol, at the cost of reliability.

By testing the UDP server and client on separate machines in the computer labs, the following results were obtained:

No. Messages Sent	No. Received	No. Lost
10	10	0
100	100	0
200	179	21
250	250	0
350	350	0
500	307	193
1000	790	210
1500	703	797
2000	1143	857

As we can see from the data, initially, from 10 to 350 messages, most of the packets are received by the server. There is one anomalous piece of data at 200 messages. However, when more than 500 messages are sent, the number of lost messages increases.

This may be due to the fact that UDP is very efficient and the lack of overhead means it can send lots of packets very quickly, especially on a LAN. However, due to its lack of congestion control, the packets are easily lost during transmission due to interference. Furthermore, since it does not retransmit any lost messages, there is an increased chance of loss of data.

3   Remote Method Invocation (RMI)

RMI allows an object on a client machine to invoke methods on an object running on a different JVM, in our case, on the server machine.

The *stub* is an object, and acts as a gateway for the client side and represents the remote object. When a caller invokes a method on the *stub* object, it does:

1. Initiates a connection with remote JVM.

2. Writes and transmits parameters to remote JVM.

3. Waits for results.

4. Reads return value.

5. Returns value to caller.

The *skeleton* is an object which acts as a gateway for the server side object. When the *skeleton* receives an incoming request, it does:

1. Reads parameters for the remote method.

2. Invokes method on remote object.

3. Writes and transmits the result to caller.

In order for the client to be able to invoke the remote methods, the server must register/bind the service with the RMI Registry. The client then calls the registry to obtain a reference to the remote object and it will then be able to call its methods. A Security Manager was also implemented for both the server and the client, which is used when RMI downloads code from a remote machine.

When we tested the RMI server and client on different machines in the labs, we obtained:

No. Messages Sent	No. Received	No. Lost
100	100	0
300	300	0
500	500	0
1000	1000	0
1500	1500	0
2000	2000	0

As we can see from the results, RMI is very reliable, even when sending thousands of messages, not a single message was lost. However, as the number of messages being sent increased, the time between the client sending the message aand the server confirming it had received all the messages increased. This may be due to the fact that there is a lot of overhead, such as the server binding the service to the RMI Registry and *stub* and *skeleton* structure. There are a lot of steps which must be completed before calling a remote method.

4 Conclusion: RMI vs UDP

From the tests, we can see that RMI is much more reliable than UDP, since it did not lose a single message. It was also easier to program. However, it is much slower and would not be suitable for a system which requires a lot of exchanging of messages. UDP on the other hand, was much faster, at the cost of reliability, and from the perspective of a new Java programmer, was much harder to implement.
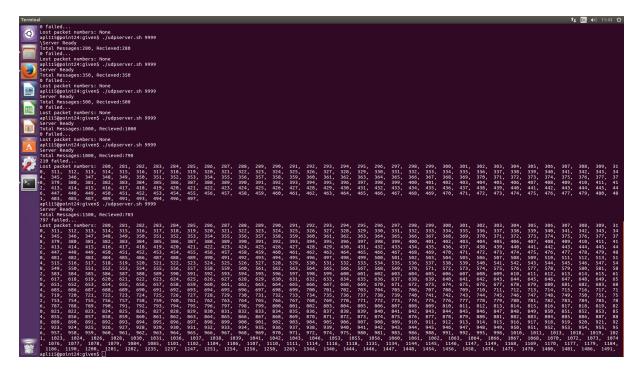
5 Screendumps

Figure 1: A console showing transmission of UDP. Some messages are lost.

Figure 2: Another console showing loss of messages through UDP

3

Figure 3: A console showing successful transmission of RMI. No messages lost

Figure 4: Sending UDP from the client side

6 Program Listings

6.1 RMI Client

```java
/*
 * Created on 13−Feb−2017
 */
package rmi;

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

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

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
                System.setSecurityManager(new RMISecurityManager());

                try{    // TO−DO: Bind to RMIServer

                        //Registry reg = 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++) {
                         MessageInfo msg = new MessageInfo(numMessages,i);
                         iRMIServer.receiveMessage(msg);
                        }
                }
                catch(Exception e){
                        System.out.println("err: " + e.getMessage());
                e.printStackTrace();
```

```
            }
        }
    }
```

6.2 RMI Server

```java
/*
 * Created on 13−Feb−2017
 */
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

                // TO−DO: Log receipt of the message

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

                if(receivedMessages == null){
                        totalMessages = msg.totalMessages;
                        receivedMessages = new int[msg.totalMessages];
                }
                // TO−DO: Log receipt of the message
                receivedMessages[msg.messageNum] = 1;

                if (msg.messageNum + 1 == totalMessages) {
                        System.out.println("Messages being totaled....");

                        String lostmes = "Lost messages: ";
                        int count = 0;
                        for (int i = 0; i < totalMessages; i++) {
                                if (receivedMessages[i] != 1) {
                                        count++;
```

```java
                                    lostmes = lostmes + " " + (i+1) + " , "
                                        ;
                            }
                    }

                    if (count == 0) {
                            lostmes = lostmes + "None";
                    }
                    System.out.println("Messages sent: " + totalMessages);
                    System.out.println("Messages received: " + (
                        totalMessages − count));
                    System.out.println("Messages lost: " + count);
                    System.out.println(lostmes);
                    }
    }

    public static void main(String[] args) {

            System.setProperty("RMIServer","129.31.219.40");

            RMIServer rmis = null;

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

            try{
                    rmis = new RMIServer();
                    //RMIServerI stub = (RMIServerI) UnicastRemoteObject.
                        exportObject(rmis, 0);

                    rebindServer("RMIServer", rmis);

                    System.out.println("Server Ready");
            }
            catch(Exception e){
        System.out.println("RMIServer err: " + e.getMessage());
        e.printStackTrace();
            }

            // TO−DO: Instantiate the server class

            // TO−DO: Bind to RMI registry

    }

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

            // TO−DO:
```

```java
// 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(8080);
        Naming.rebind(serverURL, server);
}
catch(Exception e){
        System.out.println("RMIServer err: " + e.getMessage())
            ;
        e.printStackTrace();
}

// 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.

    }
}
```

6.3 UDP Client

```java
package udp;


import java.io.IOException;
import java.io.ObjectOutputStream;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.net.SocketException;
import java.net.UnknownHostException;
import java.io.BufferedOutputStream;
import java.io.ByteArrayOutputStream;

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("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);
                }
                recvPort = Integer.parseInt(args[1]);
                countTo = Integer.parseInt(args[2]);


                // TO-DO: Construct UDP client class and try to send messages
                UDPClient client = new UDPClient();
                System.out.println("Sending messages");
                client.testLoop(serverAddr, recvPort, countTo);
        }
```

```java
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 testLoop(InetAddress serverAddr, int recvPort, int
    countTo) {
        MessageInfo m;
        ByteArrayOutputStream byteStream;
        ObjectOutputStream os;
        // TO-DO: Send the messages to the server
        for(int i = 0; i < countTo; i++) {
                m = new MessageInfo(countTo, i);

                byteStream = new ByteArrayOutputStream(5000);
                try {
                        os = new ObjectOutputStream(new
                            BufferedOutputStream(byteStream));
                        os.flush();
                        os.writeObject(m);
                        os.flush();
                } catch (IOException e) {
                        System.out.println("Error serializing object")
                            ;
                        System.exit(-1);
                }

                //retrieves byte array
                byte[] sendBuf = byteStream.toByteArray();
                send(sendBuf, 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("Err transmitting packet");
                System.exit(-1);
        }
    }
}
```

6.4　UDP Server

```java
package udp;


import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.SocketException;
import java.net.SocketTimeoutException;
import java.io.*;
import common.MessageInfo;


public class UDPServer {
        private DatagramSocket socket;
        private int port;
        private int totalMessages = -1;
        private int[] receivedMessages;
        private boolean close;

        private void run() throws SocketTimeoutException {
                int                                  pacSize;
                byte[]                     pacData;
                DatagramPacket   packet;

                System.out.println("Server_Ready");
                while(!close){

                    //Receive request from client
                        pacSize = 5000;
                        pacData = new byte[5000];

                        packet = new DatagramPacket(pacData, pacSize);
                        try {
                          socket.setSoTimeout(10000);
                          socket.receive(packet);
                        } catch (IOException e) {
                                System.out.println("Error_IO_exception_
                                    receiving_packet.");
                                System.exit(-1);
                        }

                        processMessage(packet.getData());

                }

        }

        public void processMessage(byte[] data) {

                MessageInfo msg = null;
```

```java
        // Use the data to construct a new MessageInfo object
ByteArrayInputStream byteStream = new ByteArrayInputStream(data);
ObjectInputStream is;

    try {
            is = new ObjectInputStream(new BufferedInputStream(
                byteStream));
            msg = (MessageInfo) is.readObject();
            is.close();
    } catch (ClassNotFoundException e) {
            System.out.println("Could not find class match");
    } catch (IOException e) {
            System.out.println("IOexception: ObjectInputStream.");
    }

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

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

    // If this is the last expected message, then identify
    //       any missing messages
    if (msg.messageNum + 1 == msg.totalMessages) {
            close = true;

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

            if (count == 0){
            lostmes = lostmes + "None";
            }

            System.out.println("Total Messages:" + msg.
                totalMessages + ", Recieved:" + (msg.totalMessages
                 - count));
            System.out.println(count + " failed");
            System.out.println(lostmes);
    }
```

```java
        }
        public UDPServer(int rp) {
                // Initialize UDP socket for receiving data
                try {
                        port = rp;
                        socket = new DatagramSocket(port);
                } catch (SocketException e) {
                        System.out.println("Error:_Could_not_create_socket_on_
                            port_" + port);
                        System.exit(-1);
                }
                // Make it so the server can run.
                close = false;
        }

        public static void main(String args[]) {
                int     recvPort;

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

                // Initialize Server object and start it by calling run()
                UDPServer udpsrv = new UDPServer(recvPort);
                try {
                        udpsrv.run();
                } catch (SocketTimeoutException e) {}
        }

}
```