CO527 Networks and Distributed Systems

Coursework

Yiming Wang 00934203

1. Short Summary

1.1 Messages lost

- RMI: There shouldn't have data loss. Possible reasons for this to happen, however, would be incidents in the physical layer, I.e. the disconnection between client and server.
- UDP: Data loss could be caused by properties of network and connection. The probability of packet loss will increase when increasing the distance between client and server, or when the router is overloaded. Another reason for packet loss is the receive buffer overflow. When messages come at a way faster pace than they are processed, messages would be dropped due to the capacity of the buffer. Last is when packets are larger than the MTU size of the network. These packets will be automatically fragmented. And only if we receive all the sub-packets, will the message be reconstructed.

1.2 Patterns

- RMI: No message lost. Reliable.
- UDP: Message loss starting to appear between 300~400 messages per session. Most messages lost have index 300+. Between multiple losses, a message would be received. Hence I believe the message loss is mostly due to buffer overflow: heavy loss starts at an approximately fixed index, meaning that the buffer overflows at that index; between packet drops (due to full buffer), a message is sometimes received (when the receiver just finish processing a packet).

1.3 Relative reliability

- RMI: RMI wields a higher level of abstraction: details of communication should be handled by itself, I.e. flow control.
- UDP: UDP is theoretically unreliable because of its high loss rate, due to the reasons explained above. However UDP is most widely used in areas where packet loss is not considered to be a severe problem, like live video streaming.

1.4 Programing easiness

- Personally I believe UDP implements a more straightforward algorithm than RMI, and usually takes less time to debug a certain error. RMI is more abstract, and hence will be easier to implement. Hence programming RMI should become easier as the transmission gets more and more complex, since RMI deals with lower-level problems itself.

2. Console Logs & Outputs

2.1. UDP

- 2.1.1 Input

```
yw11614@eews305-052:~/Documents/given$ ./udpclient.sh eews305-056 20
Arguments required: server name/IP, recv port, message count
yw11614@eews305-052:~/Documents/given$ ./udpclient.sh eews305-056 1111 20
UDPClient ready
All messages sent
yw11614@eews305-052:~/Documents/given$ ./udpclient.sh eews305-056 1111 20
UDPClient ready
All messages sent
yw11614@eews305-052:~/Documents/given$ ./udpclient.sh eews305-056 1111 40
UDPClient ready
All messages sent
yw11614@eews305-052:~/Documents/given$ ./udpclient.sh eews305-056 1111 60
UDPClient ready
All messages sent
yw11614@eews305-052:~/Documents/given$ ./udpclient.sh eews305-056 1111 80
UDPClient ready
All messages sent
yw11614@eews305-052:~/Documents/given$ ./udpclient.sh eews305-056 1111 100
UDPClient ready
All messages sent
yw11614@eews305-052:~/Documents/given$ ./udpclient.sh eews305-056 1111 200
UDPClient ready
All messages sent
yw11614@eews305-052:~/Documents/given$ ./udpclient.sh eews305-056 1111 300
UDPClient ready
All messages sent
yw11614@eews305-052:~/Documents/given$ ./udpclient.sh eews305-056 1111 400
UDPClient ready
All messages sent
yw11614@eews305-052:~/Documents/given$ ./udpclient.sh eews305-056 1111 500
UDPClient ready
All messages sent
yw11614@eews305-052:~/Documents/given$ ./udpclient.sh eews305-056 1111 600
UDPClient ready
All messages sent
yw11614@eews305-052:~/Documents/given$ ./udpclient.sh eews305-056 1111 800
UDPClient ready
All messages sent
yw11614@eews305-052:~/Documents/given$ ./udpclient.sh eews305-056 1111 400
UDPClient ready
All messages sent
```

- 2.1.2 Output

```
UDPServer ready

*****All messages processed!*****

20 of 20 successfully recieved

Lost packets: None

******Test finished.******

UDPServer ready

*****All messages processed!*****

40 of 40 successfully recieved

Lost packets: None

******Test finished.******

UDPServer ready

*****All messages processed!*****

60 of 60 successfully recieved

Lost packets: None

******Test finished.******

UDPServer ready

*****All messages processed!*****

80 of 80 successfully recieved

Lost packets: None

******Test finished.******

UDPServer ready

*****All messages processed!*****

100 of 100 successfully recieved

Lost packets: None

******Test finished.******

UDPServer ready

*****All messages processed!*****

200 of 200 successfully recieved
```

```
UDPServer ready

*****All messages processed!*****

300 of 300 successfully recieved

Lost packets: None

******Test finished.******

UDPServer ready

*****All messages processed!*****

400 of 400 successfully recieved

Lost packets: None

******Test finished.******

UDPServer ready

*****All messages processed!*****

380 of 500 successfully recieved

Lost packets:  347,  349,  350,  351,
         352,  353,  355,  356,  358,  359,
         360,  361,  362,  363,  365,  366,
         368,  369,  370,  371,  372,  373,
         375,  376,  377,  378,  380,  381,
         383,  384,  385,  386,  387,  389,
         390,  392,  393,  394,  395,  396,
         398,  399,  401,  402,  403,  404,
         405,  407,  408,  409,  410,  412,
         413,  415,  416,  417,  418,  419,
         421,  423,  424,  425,  426,  427,
         428,  429,  431,  432,  433,  435,
         436,  438,  439,  440,  441,  442,
         444,  445,  446,  447,  448,  449,
         451,  452,  453,  455,  456,  457,
         458,  460,  461,  463,  464,  466,
         467,  468,  469,  470,  472,  473,
         475,  476,  478,  479,  480,  481,
         482,  483,  485,  486,  488,  489,
         490,  491,  492,  493,  495,  496,
         497,  498,

******Test finished.******
```

```
UDPServer ready

*****All messages processed!*****

360 of 400 successfully recieved

Lost packets:  349,  351,  352,  354,
         355,  357,  358,  359,  360,  361,
         362,  364,  365,  367,  368,  369,
         370,  371,  373,  374,  376,  377,
         378,  379,  380,  382,  383,  384,
         386,  388,  389,  390,  391,  392,
         394,  396,  397,  398,  399,  400,

******Test finished.******
```

2.2 RMI

- 2.2.1 Input

```
yw11614@eews305-052:~/Documents/given$ ./rmiclient.sh eews305-056 20
Server Exception:java.rmi.UnmarshalException: Error unmarshaling return header;
nested exception is:
      java.io.EOFException
yw11614@eews305-052:~/Documents/given$ ./rmiclient.sh eews305-056 40
Server Exception:java.rmi.UnmarshalException: Error unmarshaling return header;
nested exception is:
      java.io.EOFException
yw11614@eews305-052:~/Documents/given$ ./rmiclient.sh eews305-056 60
Server Exception:java.rmi.UnmarshalException: Error unmarshaling return header;
nested exception is:
      java.io.EOFException
yw11614@eews305-052:~/Documents/given$ ./rmiclient.sh eews305-056 80
Server Exception:java.rmi.UnmarshalException: Error unmarshaling return header;
nested exception is:
      java.io.EOFException
yw11614@eews305-052:~/Documents/given$ ./rmiclient.sh eews305-056 100
Server Exception:java.rmi.UnmarshalException: Error unmarshaling return header;
nested exception is:
      java.io.EOFException
yw11614@eews305-052:~/Documents/given$ ./rmiclient.sh eews305-056 200
Server Exception:java.rmi.UnmarshalException: Error unmarshaling return header;
nested exception is:
      java.io.EOFException
yw11614@eews305-052:~/Documents/given$ ./rmiclient.sh eews305-056 300
Server Exception:java.rmi.UnmarshalException: Error unmarshaling return header;
nested exception is:
      java.io.EOFException
yw11614@eews305-052:~/Documents/given$ ./rmiclient.sh eews305-056 400
Server Exception:java.rmi.UnmarshalException: Error unmarshaling return header;
nested exception is:
      java.io.EOFException
yw11614@eews305-052:~/Documents/given$ ./rmiclient.sh eews305-056 500
Server Exception:java.rmi.UnmarshalException: Error unmarshaling return header;
nested exception is:
      java.io.EOFException
```

Note: The exceptions here are due to the server shutting down via System.exit(0) after all messages are received

- 2.2.2 Output

```
******All messages processed!*****
20 of 20 successfully recieved
Lost packets: None
*****Test finished.*****
Server Class Instantiate Success
Rebind Success
******All messages processed!*****
40 of 40 successfully recieved
Lost packets: None
*****Test finished.*****
Server Class Instantiate Success
Rebind Success
******All messages processed!*****
60 of 60 successfully recieved
Lost packets: None
*****Test finished.*****
Server Class Instantiate Success
Rebind Success
******All messages processed!*****
80 of 80 successfully recieved
Lost packets: None
*****Test finished.*****
Server Class Instantiate Success
Rebind Success
******All messages processed!*****
100 of 100 successfully recieved
Lost packets: None
```

```
Server Class Instantiate Success
Rebind Success
******All messages processed!*****
200 of 200 successfully recieved
Lost packets: None
*****Test finished.*****
Server Class Instantiate Success
Rebind Success
******All messages processed!*****
300 of 300 successfully recieved
Lost packets: None
*****Test finished.*****
Server Class Instantiate Success
Rebind Success
******All messages processed!*****
400 of 400 successfully recieved
Lost packets: None
*****Test finished.*****
Server Class Instantiate Success
Rebind Success
******All messages processed!*****
500 of 500 successfully recieved
Lost packets: None
*****Test finished.*****
```

3. Java Code

Code Modified to fit in lines, but not tested. The original code is in the digital hand-in on CATE.

3.1 RMI

- 3.1.1 RMI Client

```
package rmi;

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

import java.rmi.registry.Registry;
import java.rmi.registry.LocateRegistry;
import java.rmi.server.UnicastRemoteObject;
import java.net.MalformedURLException;
import java.rmi.RMISecurityManager;

import common.MessageInfo;

/**
 * @author bandara
 *
 */
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());
                }
                // TO-DO: Bind to RMIServer
                try{
                        Registry reg = 
LocateRegistry.getRegistry(args[0],1099);
                        iRMIServer = (RMIServerI) reg.lookup("RMIServer");
                }catch (NotBoundException e) {
                        System.out.println("Not Bound Exception");
                }catch (RemoteException e) {
                        System.out.println("Remote Exception When Binding");
                }

                // TO-DO: Attempt to send messages the specified number of 
times
                try{
                         for(int i = 0; i < numMessages; i++) {
                                MessageInfo msg = new 
MessageInfo(numMessages,i);
                                iRMIServer.receiveMessage(msg);
                        }
                } catch (Exception e) {
                        System.out.println("Server Exception:" + e);
                }
        }
}
```

- 3.1.2 RMI Server

```
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 java.rmi.RMISecurityManager;
import common.*;

/**
 * @author bandara
 *
 */
public class RMIServer extends UnicastRemoteObject implements RMIServerI {

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

        public RMIServer() throws RemoteException {
                super();
        }

        public void receiveMessage(MessageInfo msg) throws RemoteException {

                // TO-DO: On receipt of first message, initialise the receive
buffer
                if(totalMessages == -1){
                        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
                if (msg.messageNum + 1 == totalMessages)
                        last();
        }
        public void last(){
```

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

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

        System.out.println("******All messages processed!*****");
        System.out.println((totalMessages - count) + " of ");
        System.out.println(totalMessages);
        System.out.println(" successfully recieved");
        System.out.println(lostPac);
        System.out.println("*****Test finished.*****");
        totalMessages = -1;
        System.exit(0);

}

public static void main(String[] args) {

        RMIServer rmis = null;

        // TO-DO: Initialise Security Manager
        if(System.getSecurityManager()==null){
        System.setSecurityManager(new RMISecurityManager());
        }
        // TO-DO: Instantiate the server class
        try{
                rmis = new RMIServer();
                System.out.println("Server Class Instantiate Success");
        }catch (RemoteException e){
                System.out.println("Server Instantiate Failed" + e);
                System.exit(-1);
        }
        // TO-DO: Bind to RMI registry
        rebindServer("RMIServer",rmis);

}

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

```
                try{
                        Registry reg = LocateRegistry.createRegistry(1099);
                        reg.rebind(serverURL,server);
                }catch (RemoteException e){
                        System.out.println("Rebind Failed" + e);
                }
        }
}
```

3.2 UDP

- 3.2.1 UDP Client

```
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("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();
                        client.testLoop(serverAddr,recvPort,countTo);
                        System.out.println("All messages sent");
        }

        public UDPClient() {
                // TO-DO: Initialise the UDP socket for sending data
                try{
                        sendSoc = new DatagramSocket();
                }catch (SocketException e){
                        System.out.println("Failed to init client");
                        System.exit(-1);
                }
                System.out.println("UDPClient ready");
        }

        private void testLoop(InetAddress serverAddr, int recvPort, int
countTo) {
                int                             tries = 0;
                MessageInfo                     temp;
                // TO-DO: Send the messages to the server
                for(int i = 0; i < countTo; i++){
                        temp = new MessageInfo(countTo,i);
                        send(temp.toString(),serverAddr,recvPort);
                }

        }

        private void send(String payload, InetAddress destAddr, int destPort)
{
                int                             payloadSize;
                byte[]                          pktData;
                DatagramPacket          pkt;

                pktData = payload.getBytes();
                payloadSize = pktData.length;

                // TO-DO: build the datagram packet and send it to the server
                pkt = new DatagramPacket(pktData, payloadSize, destAddr,
destPort);
                try {
                        sendSoc.send(pkt);
                } catch (IOException e) {
                        System.out.println("Host fucked up.");
                }
        }
}
```

- 3.2.2 UDP Server

```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 common.MessageInfo;

public class UDPServer {

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

	private void run() throws IOException{
		String			strData;
		byte[]			pacData;
		int			pacSize;
		DatagramPacket		pac;

		// TO-DO: Receive the messages and process them by calling
processMessage(...).

		while(!close){
			byte[]  buffer = new byte[1000];
			pac = new DatagramPacket(buffer,buffer.length);
			try{
				recvSoc.setSoTimeout(15000);
				recvSoc.receive(pac);
			}catch (SocketTimeoutException e){
				if (totalMessages == -1){
					System.out.println("Timeout!");
					System.exit(-1);}
				else last();
			}catch (SocketException e){
				System.out.println("socket exception" + e);
				System.exit(-1);
			}
			pacData = pac.getData();
			pacSize = pacData.length;
```

```
				strData = new String(pacData,0,pac.getLength()-1);
				processMessage(strData);

			}


		}

		public void processMessage(String data) {

			MessageInfo msg = null;
			// TO-DO: Use the data to construct a new MessageInfo object
			try{
				msg = new MessageInfo(data);
			}catch (Exception e) {
				return;
			}
			// TO-DO: On receipt of first message, initialise the receive
	buffer
			if (totalMessages == -1) {
				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
			if (totalMessages == msg.messageNum + 1)
				last();

		}

		public void last(){
			close = true;

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

```
                if (count == 0) lostPac = lostPac + "None";

                System.out.println("******All messages processed!*****");
                System.out.println((totalMessages - count) + " of ");
                System.out.println(totalMessages);
                System.out.println(" successfully recieved");
                System.out.println(lostPac);
                System.out.println("*****Test finished.*****");
                totalMessages = -1;
        }

        public UDPServer(int rp) {
                // TO-DO: Initialise UDP socket for receiving data
                try{
                        recvSoc = new DatagramSocket(rp);
                }catch (SocketException e){
                        System.out.println("Failed to init server, now shutting
down");
                        System.exit(-1);
                }
                // 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{
                        server.run();
                }catch (IOException e){}
        }

}
```