**RMI and UDP Coursework**

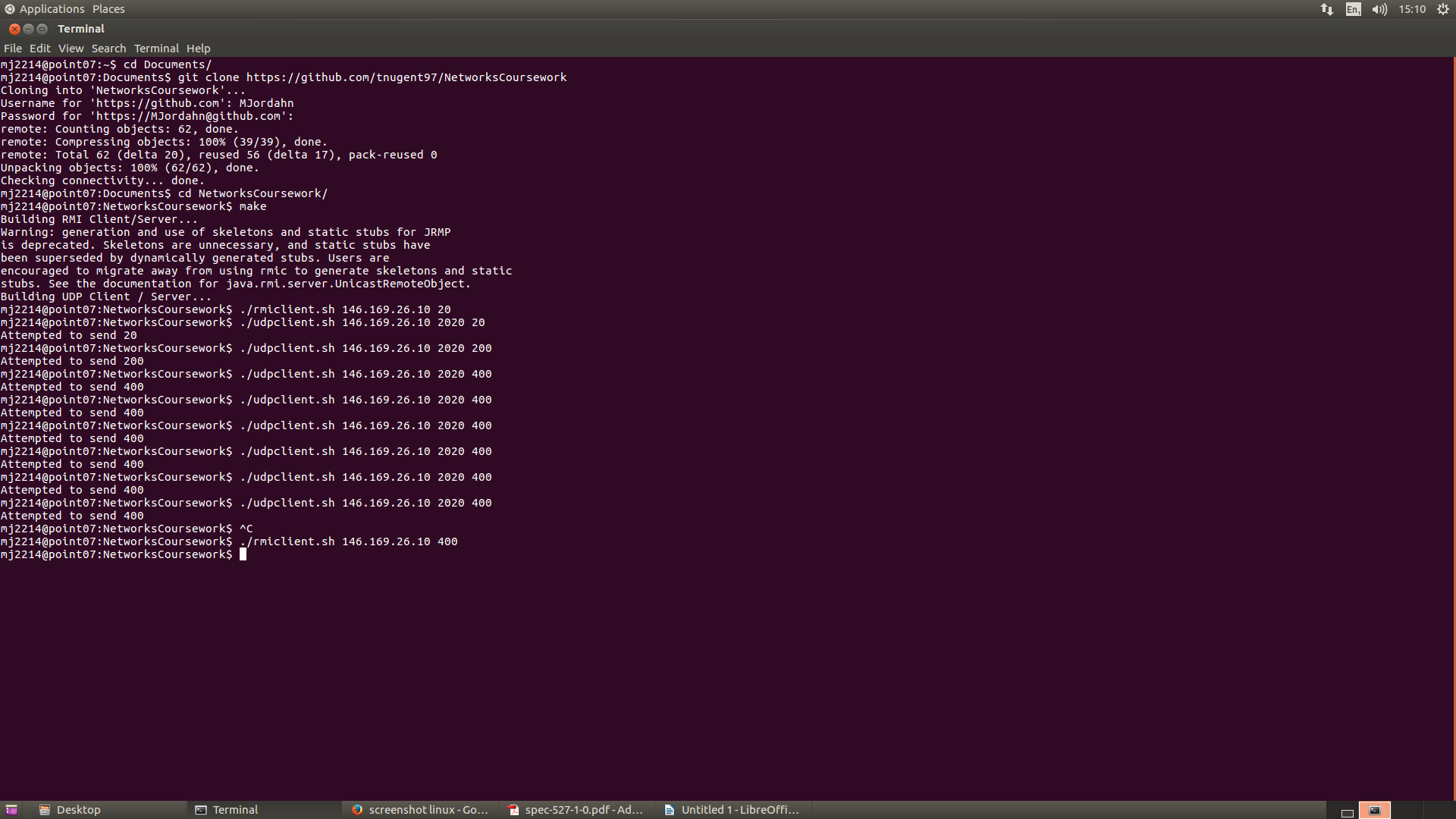
*Thomas Nugent (tn1115) & Mikkel Jordahn (mj2214)*

For UDP there may be two reasons for message loss. One reason is that noise may distort packets when sent, and thus make it difficult to reconstruct the original packet so that it is lost. The other reason that UDP may have message loss is because the client does not care whether or not the server is ready to receive. The server has a receive buffer, which will hold messages if the server is not listening at the time the messages are sent. However, if the client sends more messages than the buffer can hold, before the server can process them, the messages that exceed the buffer will then be dropped and lost. In RMI on the other hand messages are not lost, because TCP is used in this case. This means that the client is waiting to receive back a “received message” reply from the server. If it doesn’t receive this, the client will then resend the same message. This ensures messages are never lost.

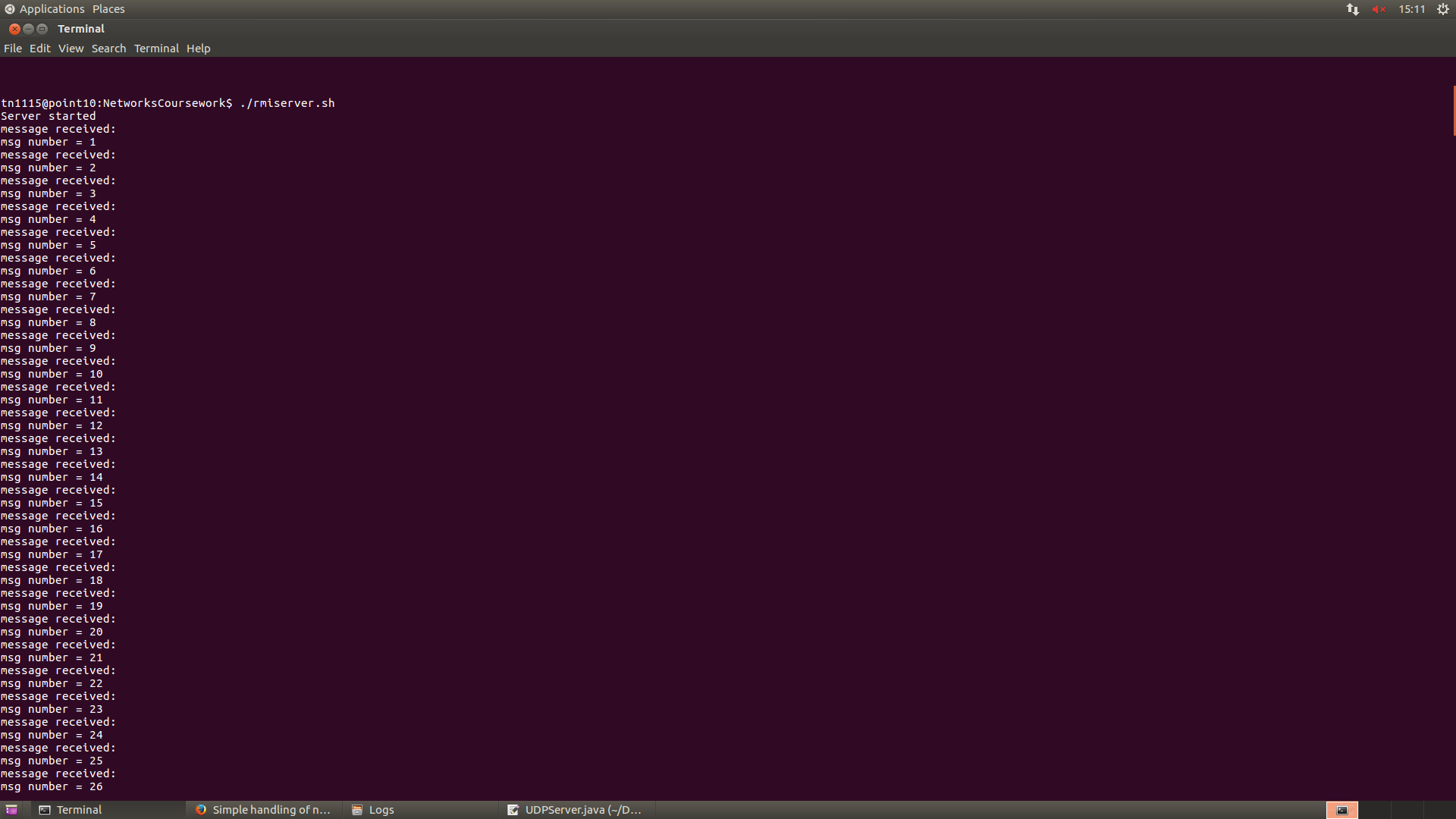
Generally speaking, the only issue we encountered was that the receiving buffer was filled, and therefore could not receive any more messages. As can be seen in the screenshots below, it was typically somewhere after about 300 messages that the buffer was filled, and from then on the rest of the messages were dropped. We were sending messages within the lab, where it appears that the internet connections are too fast and therefore no messages were dropped due to connection issues.

RMI is incredibly more reliable than UDP. This is, as mentioned earlier, because RMI utilizes TCP which means that should messages not be received by the server, then the client will resend the message. In addition, TCP also has a sequence number which is used to identify the order of the bytes when received. UDP on the other hand, does not utilize any reliability mechanisms to ensure that the data is received accurately. The cost of this reliability is the speed of the mechanism, meaning that RMI is slower than UDP. It is noted that we did not experience any significant effect of this in the lab because the college network is very sophisticated.

UDP was significantly easier to program than RMI. This is mainly because UDP is a much simpler concept and so was easier for us to understand. It relies on sockets, meaning you can easily send data using a datagram packet, on a socket that you have set up prior using the address and port. RMI however relies on the client binding to the registry and then waiting for a reply from the server to say that a message has been received, otherwise it will send the message again. UDP is very much a ‘send and forget’ mechanism, so we just keep sending messages until there are no more to send. On the other hand, RMI needs extra code to check for a reply and deal with each case accordingly.



**Screenshot 1: A screenshot showing the client running both RMI and UDP.**



**Screenshot 2: A screenshot from the RMI Server showing messages being received.**

**RMI SERVER COMMAND LINE LOG**

tn1115@point10:NetworksCoursework$ ./rmiserver.sh

Server started

message received:

msg number = 1

message received:

msg number = 2

message received:

msg number = 3

message received:

msg number = 4

message received:

msg number = 5

message received:

msg number = 6

message received:

msg number = 7

message received:

msg number = 8

message received:

msg number = 9

message received:

msg number = 10

message received:

msg number = 11

message received:

msg number = 12

message received:

msg number = 13

message received:

msg number = 14

message received:

msg number = 15

message received:

msg number = 16

message received:

msg number = 17

message received:

msg number = 18

message received:

msg number = 19

message received:

msg number = 20

message received:

msg number = 21

message received:

msg number = 22

message received:

msg number = 23

message received:

msg number = 24

message received:

msg number = 25

message received:

msg number = 26

message received:

msg number = 27

message received:

msg number = 28

message received:

msg number = 29

message received:

msg number = 30

message received:

msg number = 31

…

Continues in this fashion

…

message received:

msg number = 363

message received:

msg number = 364

message received:

msg number = 365

message received:

msg number = 366

message received:

msg number = 367

message received:

msg number = 368

message received:

msg number = 369

message received:

msg number = 370

message received:

msg number = 371

message received:

msg number = 372

message received:

msg number = 373

message received:

msg number = 374

message received:

msg number = 375

message received:

msg number = 376

message received:

msg number = 377

message received:

msg number = 378

message received:

msg number = 379

message received:

msg number = 380

message received:

msg number = 381

message received:

msg number = 382

message received:

msg number = 383

message received:

msg number = 384

message received:

msg number = 385

message received:

msg number = 386

message received:

msg number = 387

message received:

msg number = 388

message received:

msg number = 389

message received:

msg number = 390

message received:

msg number = 391

message received:

msg number = 392

message received:

msg number = 393

message received:

msg number = 394

message received:

msg number = 395

message received:

msg number = 396

message received:

msg number = 397

message received:

msg number = 398

message received:

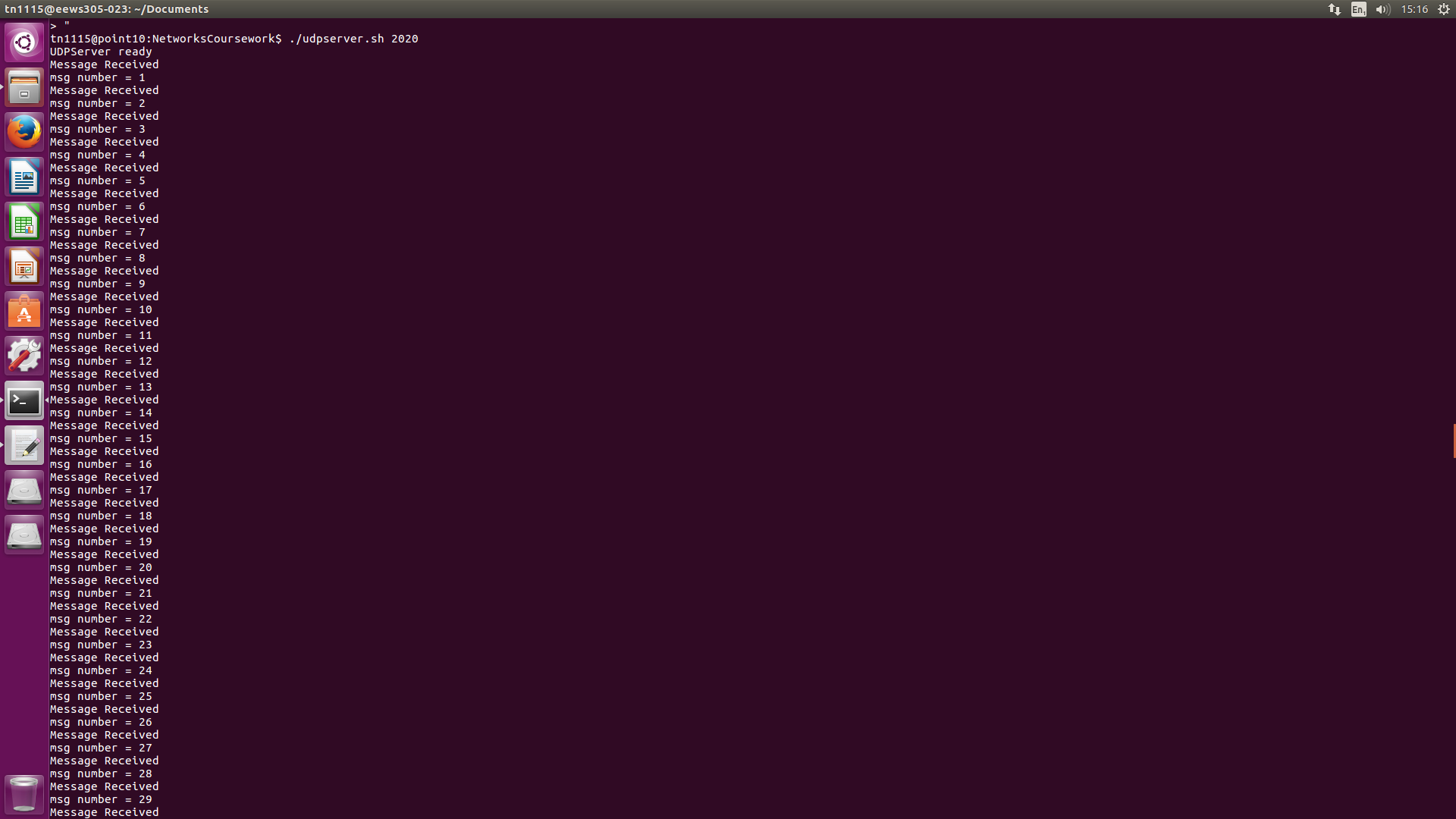
msg number = 399

message received:

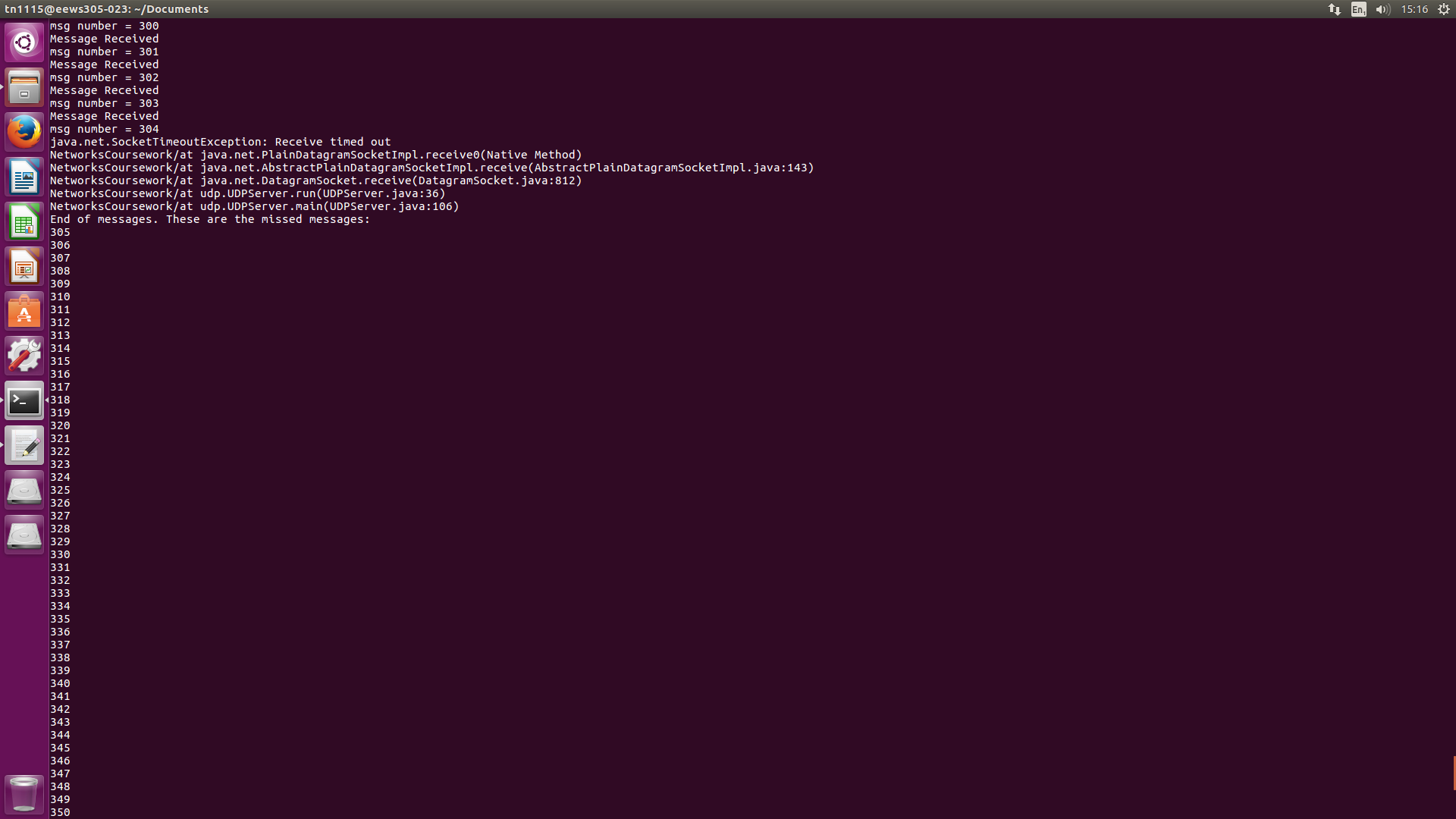
msg number = 400

End of messages. These are the missed messages:

**UDP client screenshot is in the same screenshot as RMI client**

****

**Screenshot 3: A screenshot showing the UDP server receiving messages**

****

**Screenshot 4: A screenshot showing the UDP server at the point when messages stop being recieved.**

**RMI CLIENT CODE**

**package** rmi**;**

**import** java.rmi.Naming**;**

**import** java.rmi.NotBoundException**;**

**import** java.rmi.registry.\***;**

**import** java.rmi.RemoteException**;**

**import** common.MessageInfo**;**

**public** **class** RMIClient **{**

**public** **static** **void** main**(**String**[]** args**)** **{**

RMIServerI iRMIServer **=** **null;**

String hostName **=** args**[0];**

**int** messagesSent **=** **1;**

*// 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**(**"/RMIServer"**);**

**int** numMessages **=** Integer**.**parseInt**(**args**[1]);**

*// TO-DO: Initialise Security Manager*

**if** **(**System**.**getSecurityManager**()** **==** **null){**

System**.**setSecurityManager**(new** SecurityManager**());**

**}**

*// TO-DO: Bind to RMIServer*

**try** **{**

Registry registry **=** LocateRegistry**.**getRegistry**(**args**[0],** **2020);**

iRMIServer **=** **(**RMIServerI**)** **(**registry**.**lookup**(**urlServer**));**

*// TO-DO: Attempt to send messages the specified number of times*

**while** **(**numMessages **>=** messagesSent**){**

MessageInfo msg **=** **new** MessageInfo**(**numMessages**,** messagesSent**);**

iRMIServer**.**receiveMessage**(**msg**);**

messagesSent**++;**

**}**

**}** **catch** **(**Exception e**){**

e**.**printStackTrace**();**

**}**

**}**

**}**

**RMI SERVER CODE**

**package** rmi**;**

**import** java.net.MalformedURLException**;**

**import** java.rmi.Naming**;**

**import** java.rmi.registry.\***;**

**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 **=** **20;**

**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*

**if(**msg**.**messageNum **==** **1){**

receivedMessages **=** **new** **int[**msg**.**totalMessages**];**

**}**

*// TO-DO: Log receipt of the message*

receivedMessages**[**msg**.**messageNum**-1]** **=** msg**.**messageNum**;**

System**.**out**.**println**(**"message received:"**);**

System**.**out**.**println**(**"msg number = " **+** msg**.**messageNum**);**

*// TO-DO: If this is the last expected message, then identify*

*// any missing messages*

**if(**msg**.**messageNum **==** msg**.**totalMessages**){**

System**.**out**.**println**(**"End of messages. These are the missed messages:"**);**

**for(int** i **=** **0;** i**<**msg**.**totalMessages**;** i**++){**

**if(**receivedMessages**[**i**]** **!=** i**+1){**

System**.**out**.**println**(**i**+1);**

**}**

**}**

**}**

**}**

**public** **static** **void** main**(**String**[]** args**)** **{**

RMIServer rmis **=** **null;**

*// TO-DO: Initialise Security Manager*

**if** **(**System**.**getSecurityManager**()** **==** **null)** **{**

System**.**setSecurityManager**(new** SecurityManager**());**

**}**

**try{**

*// TO-DO: Instantiate the server class*

rmis **=** **new** RMIServer**();**

*// TO-DO: Bind to RMI registry*

rmis**.**rebindServer**(**"/RMIServer"**,** rmis**);**

**}** **catch(**Exception e**){**

e**.**printStackTrace**();**

**}**

**}**

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

**try{**

*// TO-DO:*

*// Start / find the registry (hint use LocateRegistry.createRegistry(...)*

Registry registry **=** LocateRegistry**.**createRegistry**(2020);**

*// TO-DO:*

*// Now rebind the server to the registry (rebind replaces any existing servers bound to the serverURL)*

registry**.**rebind**(**serverURL**,** server**);**

System**.**out**.**println**(**"Server started"**);**

**}catch(**Exception e**){**

e**.**printStackTrace**();**

**}**

**}**

**}**

**UDP CLIENT 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** 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**);**

**}**

**public** UDPClient**()** **{**

*// TO-DO: Initialise the UDP socket for sending data*

DatagramSocket sendSoc**;**

**}**

**private** **void** testLoop**(**InetAddress serverAddr**,** **int** recvPort**,** **int** countTo**)** **{**

**int** tries **=** **0;**

**try{**

sendSoc **=** **new** DatagramSocket**(**recvPort**);**

**}** **catch** **(**Exception e**){**

e**.**printStackTrace**();**

**}**

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

**for** **(int** i **=** **0;** i **<** countTo**;** i**++){**

String msg **=** countTo **+** ";" **+** **(**i**+1);**

send**(**msg**,** serverAddr**,** recvPort**);**

tries**++;**

**}**

System**.**out**.**println**(**"Attempted to send " **+** tries**);**

**}**

**private** **void** send**(**String payload**,** InetAddress destAddr**,** **int** destPort**)** **{**

**int** payloadSize**;**

**byte[]** pktData**;**

DatagramPacket pkt**;**

payloadSize **=** payload**.**length**();**

pktData **=** **new** **byte[**payloadSize**];**

pktData **=** payload**.**getBytes**();**

pkt **=** **new** DatagramPacket**(**pktData**,** payloadSize**,** destAddr**,** destPort**);**

**try{**

sendSoc**.**send**(**pkt**);**

**}** **catch** **(**Exception e**){**

e**.**printStackTrace**();**

**}**

**}**

**}**

**UDP SERVER 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**;**

**boolean** timeout **=** **false;**

**boolean** end **=** **false;**

**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*

pacData **=** **new** **byte[1000];**

pac **=** **new** DatagramPacket**(**pacData**,** pacData**.**length**);**

**while(true){**

**try{**

recvSoc**.**setSoTimeout**(30000);**

recvSoc**.**receive**(**pac**);**

**}** **catch(**Exception e**){**

e**.**printStackTrace**();** timeout **=** **true;**

**}**

String data **=** **new** String**(**pac**.**getData**(),** **0** **,** pac**.**getLength**());**

processMessage**(**data**);**

**}**

**}**

**public** **void** processMessage**(**String data**)** **{**

**if(!**timeout**){**

MessageInfo msg **=** **null;**

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

**try{**

msg **=** **new** MessageInfo**(**data**);**

**}** **catch(**Exception e**){** e**.**printStackTrace**();** **}**

*// TO-DO: On receipt of first message, initialise the receive buffer*

**if(**msg**.**messageNum **==** **1){**

receivedMessages **=** **new** **int[**msg**.**totalMessages**];**

totalMessages **=** msg**.**totalMessages**;**

**}**

*// TO-DO: Log receipt of the message*

receivedMessages**[**msg**.**messageNum**-1]** **=** msg**.**messageNum**;**

System**.**out**.**println**(**"Message Received"**);**

System**.**out**.**println**(**"msg number = " **+** msg**.**messageNum**);**

**if(**msg**.**messageNum **==** msg**.**totalMessages**){**

end **=** **true;**

**}**

**}**

*// TO-DO: If this is the last expected message, then identify*

*// any missing messages*

**if((**end**)** **||** **(**timeout**)){**

System**.**out**.**println**(**"End of messages. These are the missed messages:"**);**

**for(int** i **=** **0;** i**<**totalMessages**;** i**++){**

**if(**receivedMessages**[**i**]** **!=** i**+1){**

System**.**out**.**println**(**i**+1);**

**}**

**}**

**}**

**}**

**public** UDPServer**(int** rp**)** **{**

*// TO-DO: Initialise UDP socket for receiving data*

**try{**

recvSoc **=** **new** DatagramSocket**(**rp**);**

**}** **catch** **(**Exception e**){** 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**);**

server**.**run**();**

**}**

**}**