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

An RMI (Remote Method Invocation) and a UDP (User Datagram Protocol) Client and Server were sucessfully written and ran in JAVA. The findings are outlined.
Messages were sent between the client and the server, over a local area network on two different computers which were not physically close to each other.
Messages were sent in increments of 100, starting from 10, up until 1000 (the initial increment was from 10 to 100).

UDP mechanism:

Sending 10-300 messages resulted in no message loss. Any more than 300 had some message loss, which increased at each 100 message increment.

Possible causes include: ***network congestion*** *which causes the 30ms timeout to be reached,* ***attenuation*** *of the packet signals due to interference,* ***network hardware failure*** *or* ***network drivers failure****. UDP does not guarantee receipt of the message being sent (no compensation for lost packets) and the ordering of message receival is not defined.*

RMI mechanism:

There was no packet loss at any number of messages sent from 10-1000.

Reason: *The RMI uses the TCP/IP protocol. Therefore a connection is established between client and server with a 3-way handshake. If a message is not received by the server, a request is made to the client to be re-sent. Therefore no messages were lost.*

RMI vs. UDP mechanism:

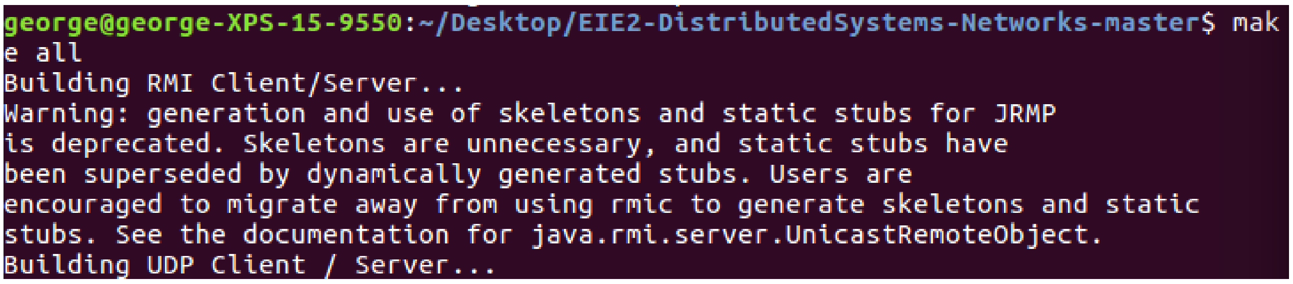
RMI: *Guaranteed receipt of all messages, messages are received in the same order they were sent. Dropped packets are re-transmitted. It uses the security manager defined to protect systems from hostile applets. It is multi-threaded, allowing the server to exploit Java threads for better concurrent processing of client requests. It can pass full objects as arguments and return values, not just predefined data types. It is more reliable than UDP when data loss cannot be tolerated and in-order delivery is required (eg. establishing an SSH connection).*

UDP: *It does not need to retransmit lost packets and does not do any connection establishment, therefore sending data has less delay than RMI. Much better than RMI for multicasting since it does not have to keep track of retransmissions/sending rate for multiple receivers. It is more reliable than RMI when data loss can be tolerated and in-order delivery of messages is not needed (eg. Domain Name Server Lookups).*

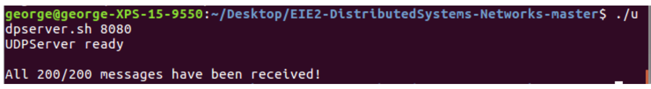
Programming Competence:

The easiest to program was the RMI mechanism. The reason being that, RMI is mostly already prepared code in the readily available JAVA libraries, and it only requires to create a new instance of the RMIServer and bind it to the NameServer, as well as defining the method that will allow receipt of the message by being invoked by the client. On the client side it is only required to initilaize the security managerr, to lookup the NameServer and invoke the available method. However in UDP, it is required to define a Socket Timeout, Create a socket for communication between Server and client that must match, define a method to send data on the client side and a method to receive data on the server side. On top of that, a check needs to be made in every loop iteration if the socket timeout was reached in order to stop the server from keep running.

Proof that both RMI and UDP programs actually ran:

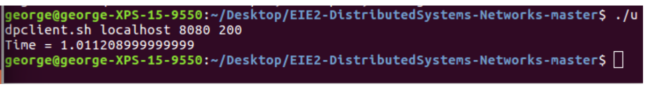


```
george@george-XPS-15-9550:~/Desktop/EIE2-DistributedSystems-Networks-master$ mak
e all
Building RMI Client/Server...
Warning: generation and use of skeletons and static stubs for JRMP
is deprecated. Skeletons are unnecessary, and static stubs have
been superseded by dynamically generated stubs. Users 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...
```

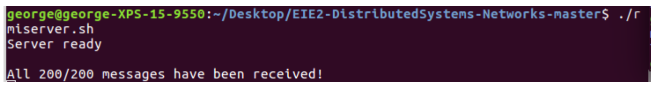


```
george@george-XPS-15-9550:~/Desktop/EIE2-DistributedSystems-Networks-master$ ./u
dpserver.sh 8080
UDPServer ready

All 200/200 messages have been received!
```

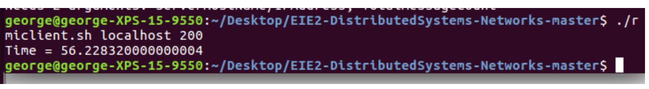


```
george@george-XPS-15-9550:~/Desktop/EIE2-DistributedSystems-Networks-master$ ./u
dpclient.sh localhost 8080 200
Time = 1.011208999999999
george@george-XPS-15-9550:~/Desktop/EIE2-DistributedSystems-Networks-master$
```



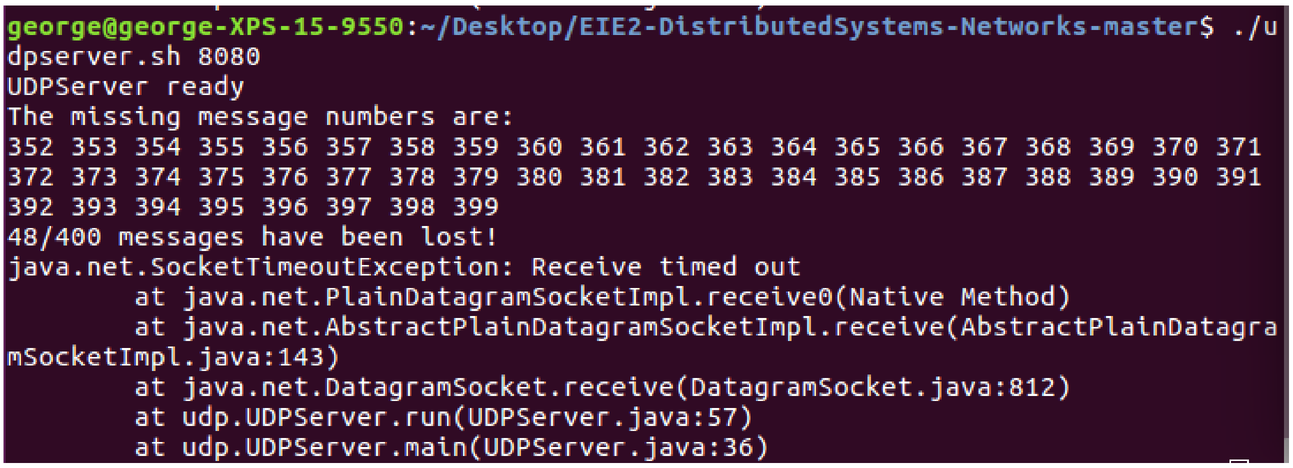
```
george@george-XPS-15-9550:~/Desktop/EIE2-DistributedSystems-Networks-master$ ./r
miserver.sh
Server ready

All 200/200 messages have been received!
```



```
george@george-XPS-15-9550:~/Desktop/EIE2-DistributedSystems-Networks-master$ ./r
miclient.sh localhost 200
Time = 56.228320000000004
george@george-XPS-15-9550:~/Desktop/EIE2-DistributedSystems-Networks-master$
```

Indication of which message number were lost when 1000 messages were transmitted using UDP mechanism:



```
george@george-XPS-15-9550:~/Desktop/EIE2-DistributedSystems-Networks-master$ ./u
dpserver.sh 8080
UDPServer ready
The missing message numbers are:
352 353 354 355 356 357 358 359 360 361 362 363 364 365 366 367 368 369 370 371
372 373 374 375 376 377 378 379 380 381 382 383 384 385 386 387 388 389 390 391
392 393 394 395 396 397 398 399
48/400 messages have been lost!
java.net.SocketTimeoutException: Receive timed out
        at java.net.PlainDatagramSocketImpl.receive0(Native Method)
        at java.net.AbstractPlainDatagramSocketImpl.receive(AbstractPlainDatagra
mSocketImpl.java:143)
        at java.net.DatagramSocket.receive(DatagramSocket.java:812)
        at udp.UDPServer.run(UDPServer.java:57)
        at udp.UDPServer.main(UDPServer.java:36)
```