

Assignment Report

February 8, 2018

1 Introduction

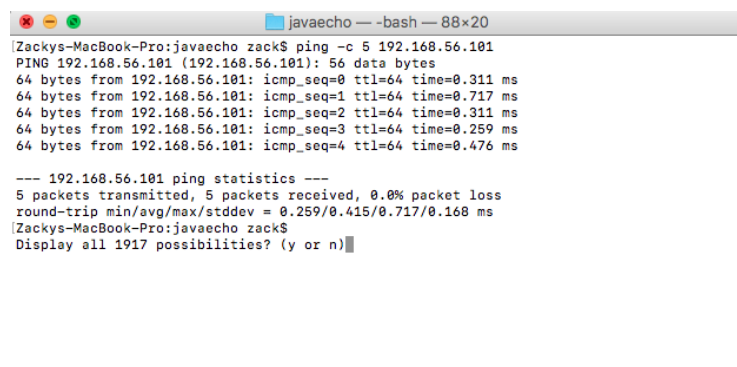
This is the first assignment of the course Computer Networks and it is about creating a UDP/TCP socket in java. The codes is to be tested by creating an environment in a virtual networking.

- The first part is about creating the connection and setting up the virtual network environment.
- The Second part is to implement the UDP connection
- The third part is to implement the TCP connection
- The last part is about capturing the traffic by using the software Wireshark.

The report will address the way the problems were solved and the some pictures were add to show the results.

2 Problem One :

This problem was about setting up the virtual machine environment. By following the guidance everything went smooth. One kind of trouble was to set up the Host-only adapter as in the beginning there was no default vboxnet(). The next screenshot I took after finish the set up and calling ping -c 5 of the IP address of the virtual machine from the host machine. We can see in the picture the Time to Live which is 64. Moreover, in the statistics, we can see the minimum round trip, the average and the max.



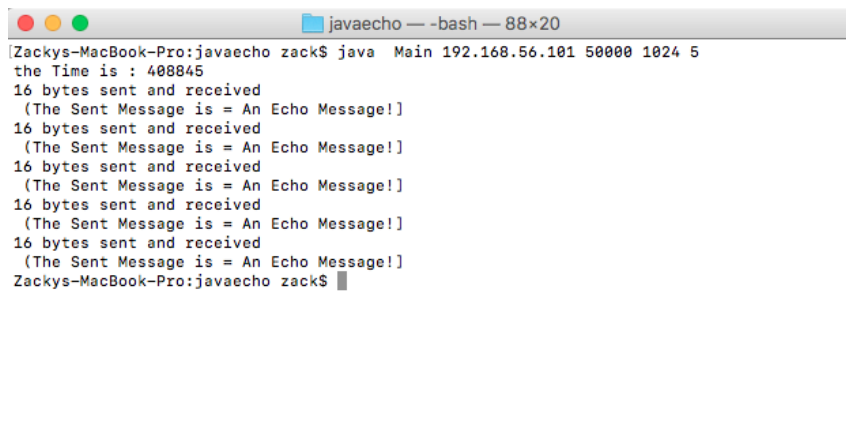
```
javaecho -- -bash -- 88x20
Zackys-MacBook-Pro:javaecho zack$ ping -c 5 192.168.56.101
PING 192.168.56.101 (192.168.56.101): 56 data bytes
64 bytes from 192.168.56.101: icmp_seq=0 ttl=64 time=0.311 ms
64 bytes from 192.168.56.101: icmp_seq=1 ttl=64 time=0.717 ms
64 bytes from 192.168.56.101: icmp_seq=2 ttl=64 time=0.311 ms
64 bytes from 192.168.56.101: icmp_seq=3 ttl=64 time=0.259 ms
64 bytes from 192.168.56.101: icmp_seq=4 ttl=64 time=0.476 ms

--- 192.168.56.101 ping statistics ---
5 packets transmitted, 5 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 0.259/0.415/0.717/0.168 ms
Zackys-MacBook-Pro:javaecho zack$
Display all 1917 possibilities? (y or n)
```

Figure 2.1: pinging the IP address of the server from the client machine

3 Problem Two

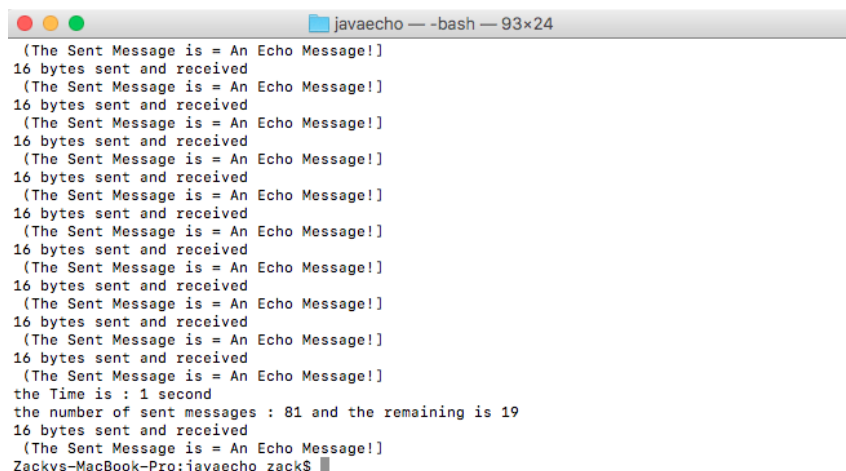
This problem was about implementing the configuration of the buffer size and the message rate.

A terminal window titled 'javaecho --bash -- 88x20' showing the output of a Java program. The program sends and receives 16 bytes of data (the message 'An Echo Message!') five times. The output shows the time as 408845 and the IP address as 192.168.56.101. The transfer rate is 50000 messages per second and the port address is 1024. The program is run from the 'Main' class.

```
Zackys-MacBook-Pro:javaecho zack$ java Main 192.168.56.101 50000 1024 5
the Time is : 408845
16 bytes sent and received
(The Sent Message is = An Echo Message!)
16 bytes sent and received
(The Sent Message is = An Echo Message!)
16 bytes sent and received
(The Sent Message is = An Echo Message!)
16 bytes sent and received
(The Sent Message is = An Echo Message!)
16 bytes sent and received
(The Sent Message is = An Echo Message!)
16 bytes sent and received
(The Sent Message is = An Echo Message!)
Zackys-MacBook-Pro:javaecho zack$
```

Figure 3.2: Sending messages from client to server by UDP

As we can see in the figure 3.2, By running the main class in terminal and providing the IP of the server machine and the transfer rate which is the desired number of messages per seconds and the port address. In the next example I tried 5 messages per second. By using a thread, I tended to stop the simulation after 1 second as it is asked in the first VG task to make the mechanism works properly. In case the transfer rate is so high the client will send as many messages as it can in 1 second and print the number of the sent messages and the number of the ones that were not able to be send (as you can see in figure 3.3).

A terminal window titled 'javaecho --bash -- 93x24' showing the output of a Java program. The program sends and receives 16 bytes of data (the message 'An Echo Message!') 81 times. The output shows the time as 1 second and the number of sent messages as 81 and the remaining as 19. The program is run from the 'Main' class.

```
(The Sent Message is = An Echo Message!)
16 bytes sent and received
(The Sent Message is = An Echo Message!)
16 bytes sent and received
(The Sent Message is = An Echo Message!)
16 bytes sent and received
(The Sent Message is = An Echo Message!)
16 bytes sent and received
(The Sent Message is = An Echo Message!)
16 bytes sent and received
(The Sent Message is = An Echo Message!)
16 bytes sent and received
(The Sent Message is = An Echo Message!)
16 bytes sent and received
(The Sent Message is = An Echo Message!)
16 bytes sent and received
(The Sent Message is = An Echo Message!)
16 bytes sent and received
(The Sent Message is = An Echo Message!)
16 bytes sent and received
(The Sent Message is = An Echo Message!)
16 bytes sent and received
(The Sent Message is = An Echo Message!)
16 bytes sent and received
16 bytes sent and received
the Time is : 1 second
the number of sent messages : 81 and the remaining is 19
16 bytes sent and received
(The Sent Message is = An Echo Message!)
Zackys-MacBook-Pro:javaecho zack$
```

Figure 3.3: The transfer rate is 100. After 1 sec it notifies user how many messages were sent

Finally an abstract networkLayering was implemented for the VG-task 2 with some methods that are being used in UDPEchoClient class and are going to be used in TCPEchoClient. In the abstract there is Run method as the abstract implements Runnable class. Moreover, some other methods such as RunClient, Delay, RunTheThread and ErrorHandler.


3.1 Handling the error:

I implemented an error handling method in the networkLayer class. It gives messages for some error that might occur while the user enters the command to send messages from the client server.

- First a method that checks if the IP is valid or in a right format. That is the IP address should be consisted of 4 parts and the numbers between 0-255.
- A function that checks if the port number is valid. Since the port number is an unsigned 16-bit integer so it should be in the range (1-65535)
- A function to check the transfer rate. The transfer rate should not be less than 1 as if it is zero so no messages will be sent and it cannot defiantly be less than zero.
- In the UDPEchoClient class, there is a function to check the message length. The longest a message can be that does not cause a failure in the program is 65507. That is because the IP header is 20 and the UDP header is 8, thus $65535 - 20 - 8 = 65507$ byte. The function checks as well if the message is empty and notify the user.
- Lastly, there is a function that checks if the number of arguments that entered by the user is right. The program asks the user to give an IP address, Transfer rate and a port number.

4 Problem Three:

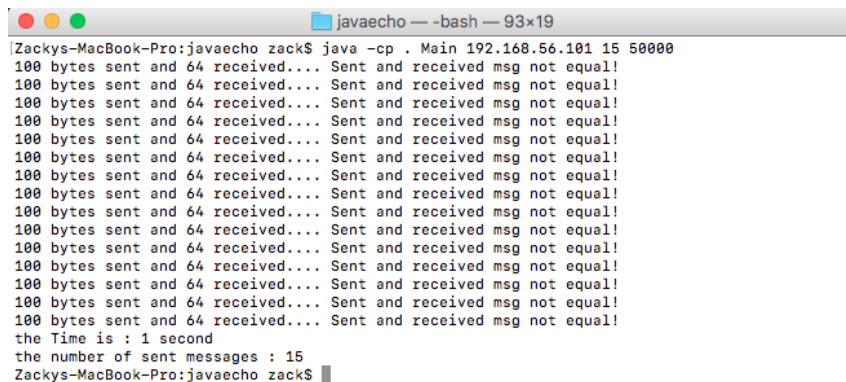
This problem was about implementing a TCP connection between the host machine and the virtualbox machine. I used some codes from the previous problem and needed some new ones. However, the TCPEchoClient class is implementing the abstract class networkingLayer for the thread and the error handling. In figure 4.4 we can see that I sent a message with a size 100 while the buffer size in the abstract class was set to 64 and the client still sent and received the whole message.



```
Zackys-MacBook-Pro:javaecho zack$ java -cp . Main 192.168.56.101 16 50000
100 byte sent and 100 byte received and the buffer size is 64
100 byte sent and 100 byte received and the buffer size is 64
100 byte sent and 100 byte received and the buffer size is 64
100 byte sent and 100 byte received and the buffer size is 64
100 byte sent and 100 byte received and the buffer size is 64
100 byte sent and 100 byte received and the buffer size is 64
100 byte sent and 100 byte received and the buffer size is 64
100 byte sent and 100 byte received and the buffer size is 64
100 byte sent and 100 byte received and the buffer size is 64
100 byte sent and 100 byte received and the buffer size is 64
100 byte sent and 100 byte received and the buffer size is 64
100 byte sent and 100 byte received and the buffer size is 64
100 byte sent and 100 byte received and the buffer size is 64
100 byte sent and 100 byte received and the buffer size is 64
100 byte sent and 100 byte received and the buffer size is 64
100 byte sent and 100 byte received and the buffer size is 64
the Time is : 1001 ms
Zackys-MacBook-Pro:javaecho zack$
```

Figure 4.4:

While in Figure 4.5 and by repeating the same procedure but on the UDP connection we see that 100 bytes were sent but 64 was received which is the size of the buffer. That is explained in the definition of stream-oriented connection. In other words, TCP works on gathering the byte contiguously by streaming them and putting them into one segment or more. That is why TCP keep track of the whole message boundaries and send it all.



```
Zackys-MacBook-Pro:javaecho zack$ java -cp . Main 192.168.56.101 15 50000
100 bytes sent and 64 received.... Sent and received msg not equal!
100 bytes sent and 64 received.... Sent and received msg not equal!
100 bytes sent and 64 received.... Sent and received msg not equal!
100 bytes sent and 64 received.... Sent and received msg not equal!
100 bytes sent and 64 received.... Sent and received msg not equal!
100 bytes sent and 64 received.... Sent and received msg not equal!
100 bytes sent and 64 received.... Sent and received msg not equal!
100 bytes sent and 64 received.... Sent and received msg not equal!
100 bytes sent and 64 received.... Sent and received msg not equal!
100 bytes sent and 64 received.... Sent and received msg not equal!
100 bytes sent and 64 received.... Sent and received msg not equal!
100 bytes sent and 64 received.... Sent and received msg not equal!
100 bytes sent and 64 received.... Sent and received msg not equal!
100 bytes sent and 64 received.... Sent and received msg not equal!
100 bytes sent and 64 received.... Sent and received msg not equal!
the Time is : 1 second
the number of sent messages : 15
Zackys-MacBook-Pro:javaecho zack$
```

Figure 4.5:

5 Problem Four:

5.1 UDP traffic capturing

In this problem I run the same code and captured the traffic by Wireshark. In Figure 5.6 I had small buffer size while in Figure 5.7 I had a buffer size that fit the message length. There was no different in the length as the wireshark shows that the length of the sent and the received message is the same.

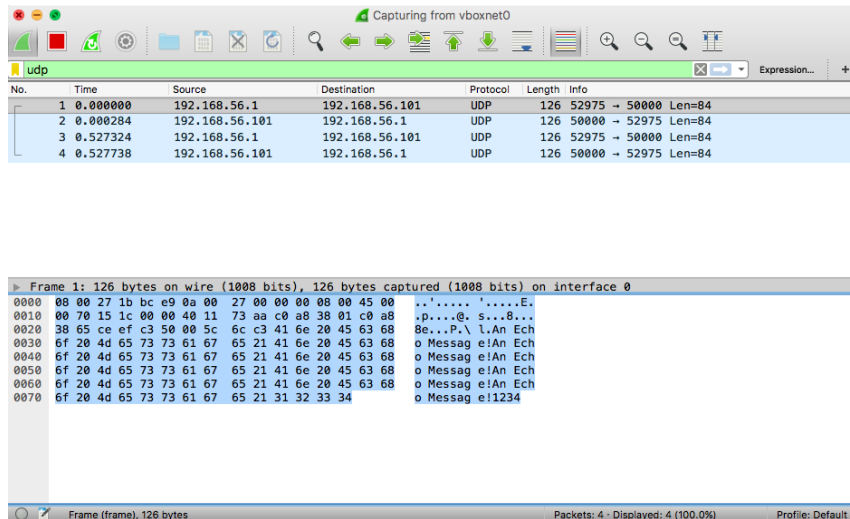


Figure 5.6:

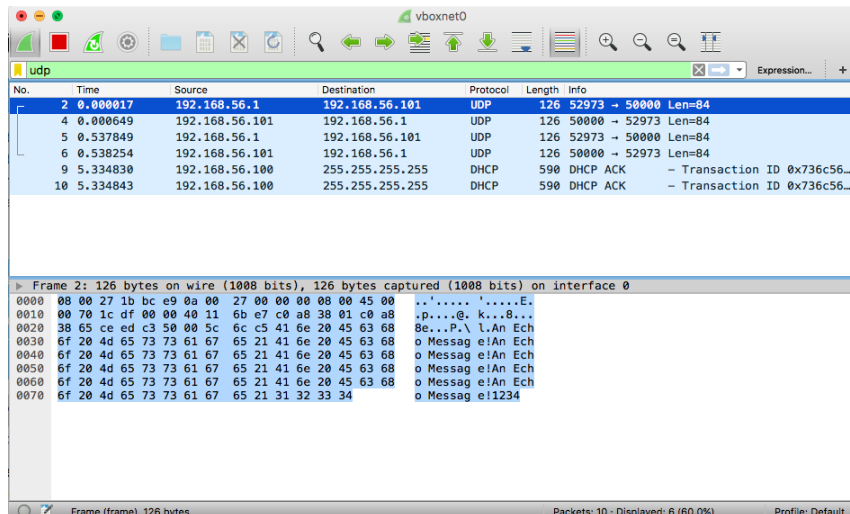


Figure 5.7:

5.2 TCP traffic capturing

In TCP the server and the client create a connection so even though I am sending the same number of messages which is 2, we see that there is more traffic when it comes for TCP and that is because the three way shake. In the first line in Figure 6.1 there is a request from the host machine with sequence number 0 and here the host machine is asking to create a connection by synchronise request. In the second line, the virtual machine is replying and accepting by sending (SYN - ACK) message. While in third line, the host machine is replying by ACK message and this means that the host machine acknowledges the request and the connection is created. After establishing the connection, the client will be able to send messages through the connection by setting the flag PSH. In the TCP there is as well no difference when the buffer size is smaller or bigger than the message size.

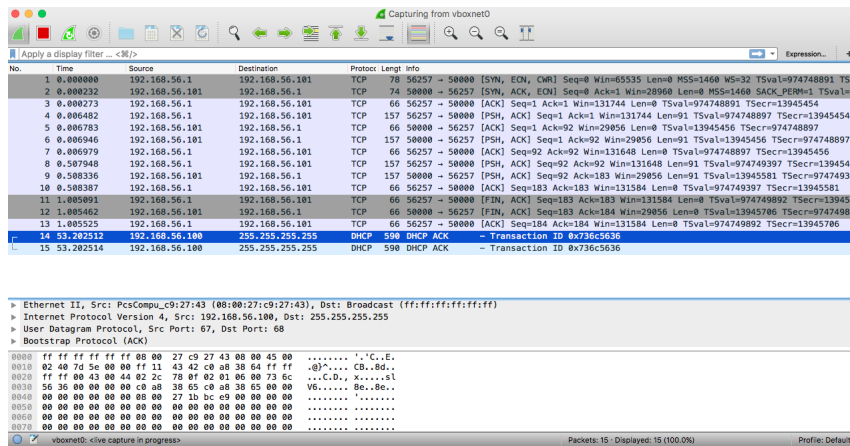


Figure 5.8:

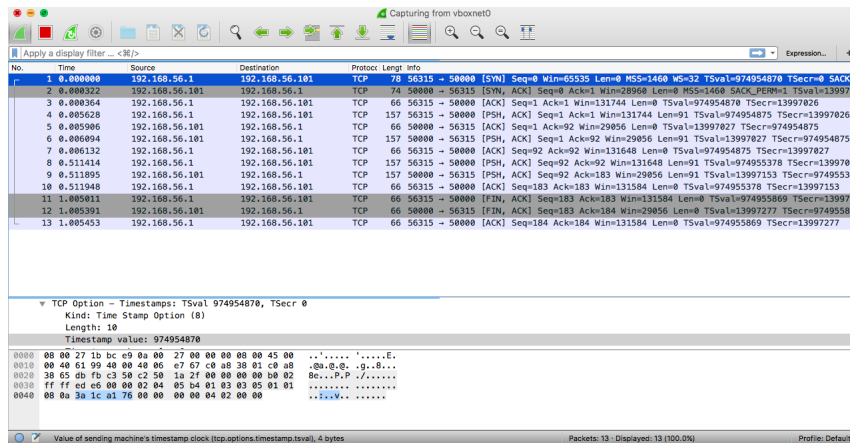


Figure 5.9:

Finally, and as a conclusion of the experiment, we can see that the UDP is connectionless while the TCP use connection to transmit messages. TCP is slower and that is because the time establishing the connection take. One more thing to be noticed in the pictures is that the final message size after appending the header is smaller in UDP and that is because the TCP header is 20 bytes while the UDP header is 8 bytes. The way TCP streams bytes make it reliable and here we mean that it guarantees that there will be no lost data while transmitting it. On the other hand, UDP does not guarantee that and not even getting the data in a right order.