

Zacky Kharboutli

# Assignment Report

February 9, 2018

## 1 Introduction

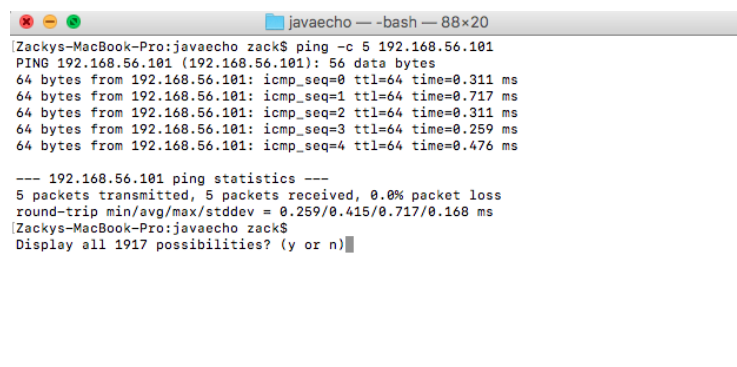
This is the first assignment of the course Computer Networks and it is about creating a UDP/TCP socket in java. The codes is to be tested by creating an environment in a virtual networking.

- 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.

```
javaecho — -bash — 88x20
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!)
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).

[illegible]

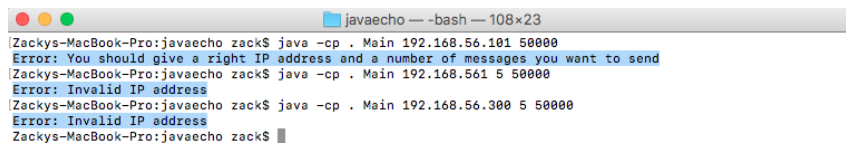
Figure 3.3: The transfer rate is 100. After 1 sec it notifies user how many messages were sent

Finally an abstract class `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.




```
javaecho --bash-- 108x23
Zackys-MacBook-Pro:javaecho zack$ java -cp . Main 192.168.56.101 50000
Error: You should give a right IP address and a number of messages you want to send
Zackys-MacBook-Pro:javaecho zack$ java -cp . Main 192.168.561 5 50000
Error: Invalid IP address
Zackys-MacBook-Pro:javaecho zack$ java -cp . Main 192.168.56.300 5 50000
Error: Invalid IP address
Zackys-MacBook-Pro:javaecho zack$
```

Figure 3.4: Some example of the error messages

## 4 Problem Three:

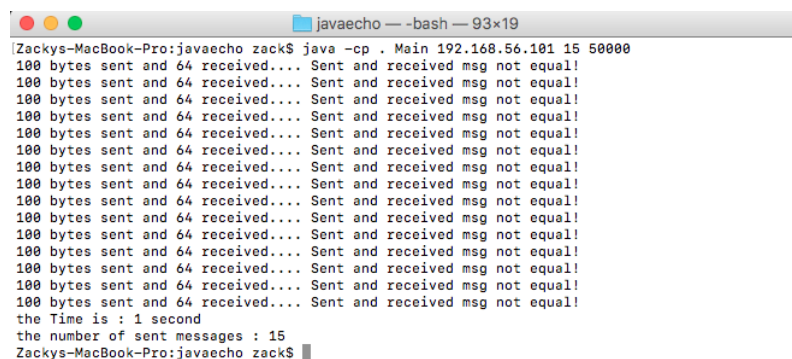
This problem was about implementing a TCP connection between the host machine and the virtualbox machine. I used some codes from the previous problem and needed some new ones. However, the TCPEchoClient class is implementing the abstract class networkingLayer for the thread and the error handling. In figure 4.5 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.

A terminal window titled 'javaecho -- -bash -- 93x19' showing the output of a Java program. The program sends a message of 100 bytes and receives it back. The output shows 15 iterations of '100 byte sent and 100 byte received and the buffer size is 64'. The total time taken is 1001 ms.

```
[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  
the Time is : 1001 ms  
Zackys-MacBook-Pro:javaecho zack$
```

Figure 4.5: In TCP the client received the whole message even if the buffer size is small

While in Figure 4.6 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.

A terminal window titled 'javaecho -- -bash -- 93x19' showing the output of a Java program. The program sends a message of 100 bytes and receives it back. The output shows 15 iterations of '100 bytes sent and 64 received... Sent and received msg not equal!'. The total time taken is 1 second and the number of sent messages is 15.

```
[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 4.6: In UDP the client received the message according to the buffer size

## 5 Problem Four:

### 5.1 UDP traffic capturing

In this problem I run the same code and captured the traffic by Wireshark. In Figure 5.7 I had small buffer size while in Figure 5.8 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.

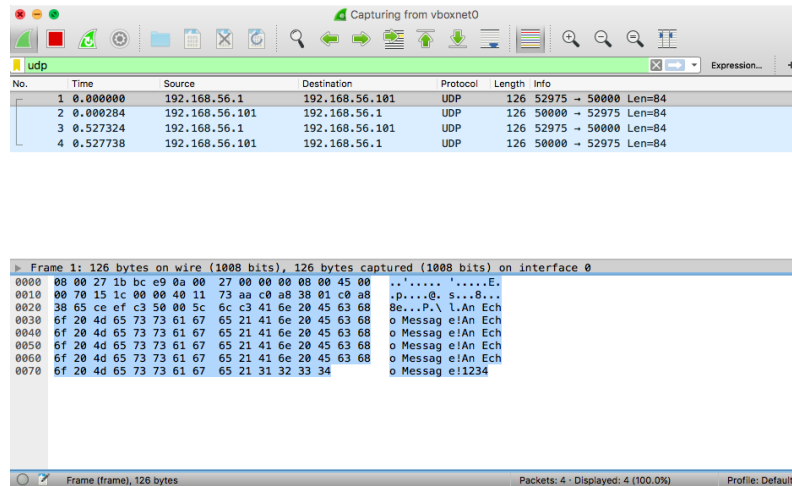


Figure 5.7: The buffer size is set to be enough for the message size

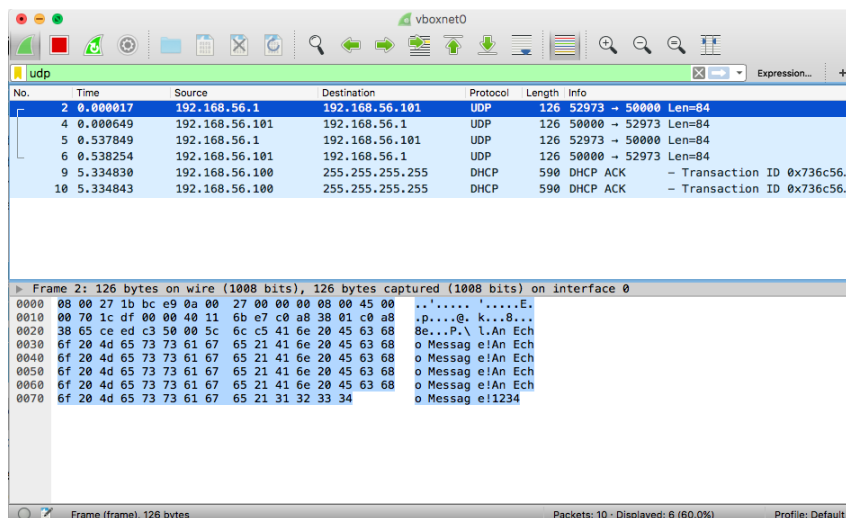


Figure 5.8: The buffer size is set to be smaller than the message size

## 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 5.9 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. Then the flag PSH will be set to start sending the messages. In the TCP there is as well no difference when the buffer size is smaller or bigger than the message size if compare the figure 5.9 and 5.10

The screenshot shows a Wireshark packet capture from a virtual machine. The packet list on the left shows 15 packets. The packet details pane on the right shows the selected packet (No. 14) as a DHCP ACK. The packet bytes pane at the bottom shows the raw data of the selected packet.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.56.1	192.168.56.101	TCP	56257	→ 50000 [SYN, ECN, CWR] Seq=0 Win=65535 Len=0 MSS=1460 WS=32 TSval=974748891 TSecr=13945454
2	0.000232	192.168.56.101	192.168.56.1	TCP	74	50000 → 56257 [SYN, ACK, ECN] Seq=0 Ack=1 Win=28968 Len=0 MSS=1460 SACK_PERM=1 TSval=13945454
3	0.000273	192.168.56.1	192.168.56.101	TCP	66	56257 → 50000 [ACK] Seq=1 Ack=1 Win=131744 Len=0 TSval=974748891 TSecr=13945454
4	0.000492	192.168.56.1	192.168.56.101	TCP	157	56257 → 50000 [PSH, ACK] Seq=92 Ack=92 Win=131648 Len=91 TSval=974748897 TSecr=13945454
5	0.000783	192.168.56.101	192.168.56.1	TCP	66	50000 → 56257 [ACK] Seq=1 Ack=92 Win=29056 Len=0 TSval=13945456 TSecr=974748897
6	0.000946	192.168.56.101	192.168.56.1	TCP	157	50000 → 56257 [PSH, ACK] Seq=1 Ack=92 Win=29056 Len=91 TSval=13945456 TSecr=974748897
7	0.000979	192.168.56.1	192.168.56.101	TCP	66	56257 → 50000 [ACK] Seq=92 Ack=92 Win=131648 Len=0 TSval=974748897 TSecr=13945456
8	0.007940	192.168.56.1	192.168.56.101	TCP	157	56257 → 50000 [PSH, ACK] Seq=92 Ack=92 Win=131648 Len=91 TSval=974748937 TSecr=13945456
9	0.008336	192.168.56.101	192.168.56.1	TCP	157	50000 → 56257 [PSH, ACK] Seq=92 Ack=183 Win=29056 Len=91 TSval=13945581 TSecr=9747493...
10	0.008387	192.168.56.1	192.168.56.101	TCP	66	56257 → 50000 [ACK] Seq=183 Ack=183 Win=131584 Len=0 TSval=974749397 TSecr=13945581
11	1.005991	192.168.56.1	192.168.56.101	TCP	66	56257 → 50000 [FIN, ACK] Seq=183 Ack=183 Win=131584 Len=0 TSval=974749892 TSecr=13945...
12	1.005482	192.168.56.101	192.168.56.1	TCP	66	50000 → 56257 [FIN, ACK] Seq=183 Ack=184 Win=29056 Len=0 TSval=13945786 TSecr=9747498...
13	1.005525	192.168.56.1	192.168.56.101	TCP	66	56257 → 50000 [ACK] Seq=184 Ack=184 Win=131584 Len=0 TSval=974749892 TSecr=13945786
14	53.202512	192.168.56.100	255.255.255.255	DHCP	590	DHCP ACK → Transaction ID 0x736c5636
15	53.202514	192.168.56.100	255.255.255.255	DHCP	590	DHCP ACK → Transaction ID 0x736c5636

Packet details for packet 14:

- Ethernet II, Src: PcsCompu\_c9:27:43 (08:00:27:c9:27:43), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
- Internet Protocol Version 4, Src: 192.168.56.100, Dst: 255.255.255.255
- User Datagram Protocol, Src Port: 67, Dst Port: 68
- Bootstrap Protocol (ACK)

Packet bytes:

```
0000 ff ff ff ff ff ff 00 27 c9 27 43 00 00 45 00 .....'.C..E.
0010 02 40 7d 5e 00 00 ff 11 43 42 c0 a8 38 64 ff ff ..@....CB..d.
0020 ff ff 00 43 00 44 02 2c 78 0f 02 01 00 73 6c ...C.D.,x....sl
0030 56 36 00 00 00 c0 a8 38 64 00 00 00 00 00 00 W6.....Be...e...
0040 00 00 00 00 00 00 00 00 27 1b bc e9 00 00 00 00 .....
0050 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0060 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0070 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
```

Figure 5.9: The buffer size is set to be enough for the message size

The screenshot shows a Wireshark packet capture from a virtual machine. The packet list on the left shows 13 packets. The packet details pane on the right shows the selected packet (No. 13) as a DHCP ACK. The packet bytes pane at the bottom shows the raw data of the selected packet.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.56.1	192.168.56.101	TCP	78	56315 → 50000 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=32 TSval=974954878 TSecr=0 SACK...
2	0.000322	192.168.56.101	192.168.56.1	TCP	74	50000 → 56315 [SYN, ACK] Seq=0 Ack=1 Win=28968 Len=0 MSS=1460 SACK_PERM=1 TSval=13997...
3	0.000364	192.168.56.1	192.168.56.101	TCP	66	56315 → 50000 [ACK] Seq=1 Ack=1 Win=131744 Len=0 TSval=974954878 TSecr=13997826
4	0.000528	192.168.56.1	192.168.56.101	TCP	157	56315 → 50000 [PSH, ACK] Seq=1 Ack=1 Win=131744 Len=91 TSval=974954875 TSecr=13997826
5	0.000596	192.168.56.101	192.168.56.1	TCP	66	50000 → 56315 [ACK] Seq=1 Ack=92 Win=29056 Len=0 TSval=13997827 TSecr=974954875
6	0.000694	192.168.56.101	192.168.56.1	TCP	157	50000 → 56315 [PSH, ACK] Seq=1 Ack=92 Win=29056 Len=91 TSval=13997827 TSecr=974954875
7	0.0006132	192.168.56.1	192.168.56.101	TCP	66	56315 → 50000 [ACK] Seq=92 Ack=92 Win=131648 Len=0 TSval=974954875 TSecr=13997827
8	0.511414	192.168.56.1	192.168.56.101	TCP	157	56315 → 50000 [PSH, ACK] Seq=92 Ack=92 Win=131648 Len=91 TSval=974955378 TSecr=139978...
9	0.511895	192.168.56.101	192.168.56.1	TCP	157	50000 → 56315 [PSH, ACK] Seq=92 Ack=183 Win=29056 Len=91 TSval=13997153 TSecr=9749553...
10	0.511948	192.168.56.1	192.168.56.101	TCP	66	56315 → 50000 [ACK] Seq=183 Ack=183 Win=131584 Len=0 TSval=974955378 TSecr=13997153
11	1.005911	192.168.56.1	192.168.56.101	TCP	66	56315 → 50000 [FIN, ACK] Seq=183 Ack=183 Win=131584 Len=0 TSval=974955869 TSecr=13997...
12	1.005391	192.168.56.101	192.168.56.1	TCP	66	50000 → 56315 [FIN, ACK] Seq=183 Ack=184 Win=29056 Len=0 TSval=13997277 TSecr=9749558...
13	1.005453	192.168.56.1	192.168.56.101	TCP	66	56315 → 50000 [ACK] Seq=184 Ack=184 Win=131584 Len=0 TSval=974955869 TSecr=13997277

Packet details for packet 13:

- TCP Option - Timestamps: TSval 974954878, TSecr 0
- Kind: Time Stamp Option (8)
- Length: 18
- Timestamp value: 974954878

Packet bytes:

```
0000 00 00 27 1b bc e9 0a 00 27 00 00 00 00 45 00 .....'.C..E.
0010 00 40 61 99 40 00 00 e7 67 c0 a8 38 01 c0 a8 ..@.e.g..8...
0020 38 05 0b 7b c3 58 c2 58 1a 2f 00 00 00 b0 02 Be...P.P./.....
0030 ff ff cd cd 00 00 02 04 05 b4 01 03 03 03 03 .....
0040 00 0a 3a 1c a1 76 00 00 00 00 04 02 00 00 .....
```

Figure 5.10: The buffer size is set to be smaller than the message size

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.