

## Problem 1

```
C:\Users\Oskar>ping 192.168.56.101

Pinging 192.168.56.101 with 32 bytes of data:
Reply from 192.168.56.101: bytes=32 time<1ms TTL=64
Reply from 192.168.56.101: bytes=32 time<1ms TTL=64
Reply from 192.168.56.101: bytes=32 time<1ms TTL=64
Reply from 192.168.56.101: bytes=32 time<1ms TTL=64

Ping statistics for 192.168.56.101:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 0ms, Maximum = 0ms, Average = 0ms

C:\Users\Oskar>
```

```
ubuntu@ubuntu-VirtualBox: ~
ubuntu@ubuntu-VirtualBox:~$ ping -c 5 192.168.56.1
PING 192.168.56.1 (192.168.56.1) 56(84) bytes of data.
64 bytes from 192.168.56.1: icmp_seq=1 ttl=128 time=0.252 ms
64 bytes from 192.168.56.1: icmp_seq=2 ttl=128 time=0.208 ms
64 bytes from 192.168.56.1: icmp_seq=3 ttl=128 time=0.236 ms
64 bytes from 192.168.56.1: icmp_seq=4 ttl=128 time=0.233 ms
64 bytes from 192.168.56.1: icmp_seq=5 ttl=128 time=0.235 ms

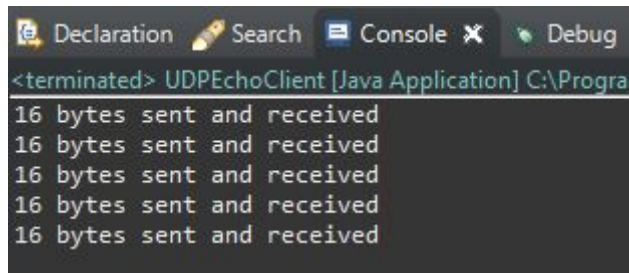
--- 192.168.56.1 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4000ms
rtt min/avg/max/mdev = 0.208/0.232/0.252/0.023 ms
ubuntu@ubuntu-VirtualBox:~$
```

The above pictures shows the output when using the ping command both on the host machine and the virtual machine. The first picture is the host machine where the client application is run and it is pinging the virtual machine used for the server application. The second picture is the virtual machine pinging the host machine. The pings show that no packet loss happen and that all round trips were performed in under 1 millisecond.

## Problem 2



```
root@ubuntu-VirtualBox: /media/sf_linux
root@ubuntu-VirtualBox:/media/sf_linux# java -cp . UDPEchoServer
Server has been started and is now running..
UDP echo request from 192.168.56.1 using port 55948
UDP echo request from 192.168.56.1 using port 55948
UDP echo request from 192.168.56.1 using port 55948
UDP echo request from 192.168.56.1 using port 55948
UDP echo request from 192.168.56.1 using port 55948
```



```
<terminated> UDPEchoClient [Java Application] C:\Program
16 bytes sent and received
16 bytes sent and received
16 bytes sent and received
16 bytes sent and received
16 bytes sent and received
```

The pictures listed above shows the the amount of sent and received data when sending 5 messages from the client to the server. The arguments given to the client applications is in the format of "[IP] [PORT] [BUFFER LENGTH] [TRANSFER RATE]".

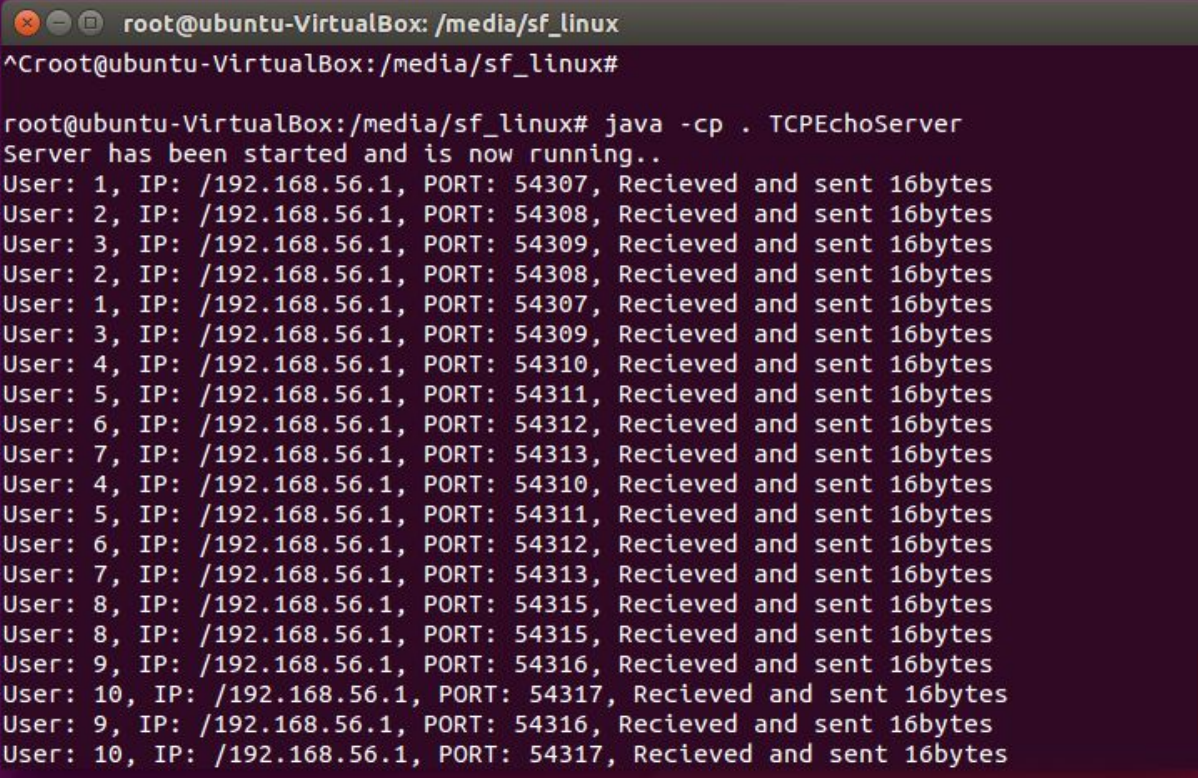
### Handled exceptions / errors:

- Invalid number of arguments. (Not 4 specified arguments)
- Invalid IP address. (Using regex to detect IPv4 address)
- Invalid Port. (Greater than 65535 or less than 1)
- Invalid Buffer size. (Less than 1)
- Invalid transfer rate. (Less than 0)
- Errors when creating and binding socket.
- Errors when sending and receiving packets.

### VG Task 2

I created a class called EchoClient that contains all the shared members and functions that both the UDP and TCP client needs such as initializing all the members, verifying them and creating a local and remote point.

### Problem 3

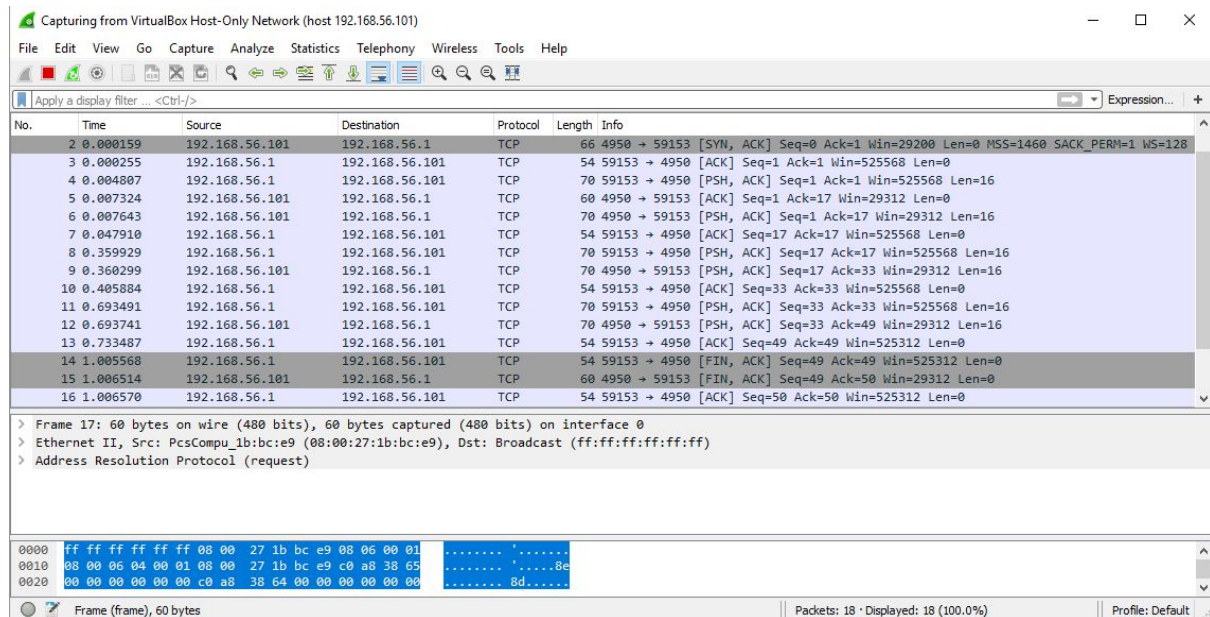
A terminal window titled 'root@ubuntu-VirtualBox: /media/sf\_linux' showing the execution of a Java program. The prompt is '^Croot@ubuntu-VirtualBox: /media/sf\_linux#'. The command 'java -cp . TCPEchoServer' has been executed. The output shows the server starting and then processing 20 requests from 10 users. Each request is logged as 'User: X, IP: /192.168.56.1, PORT: Y, Recieved and sent 16bytes', where X is the user number (1-10), Y is a unique port number (54307-54317), and the text 'Recieved' is misspelled as 'Recieved'.

```
root@ubuntu-VirtualBox: /media/sf_linux
^Croot@ubuntu-VirtualBox: /media/sf_linux#
root@ubuntu-VirtualBox: /media/sf_linux# java -cp . TCPEchoServer
Server has been started and is now running..
User: 1, IP: /192.168.56.1, PORT: 54307, Recieved and sent 16bytes
User: 2, IP: /192.168.56.1, PORT: 54308, Recieved and sent 16bytes
User: 3, IP: /192.168.56.1, PORT: 54309, Recieved and sent 16bytes
User: 2, IP: /192.168.56.1, PORT: 54308, Recieved and sent 16bytes
User: 1, IP: /192.168.56.1, PORT: 54307, Recieved and sent 16bytes
User: 3, IP: /192.168.56.1, PORT: 54309, Recieved and sent 16bytes
User: 4, IP: /192.168.56.1, PORT: 54310, Recieved and sent 16bytes
User: 5, IP: /192.168.56.1, PORT: 54311, Recieved and sent 16bytes
User: 6, IP: /192.168.56.1, PORT: 54312, Recieved and sent 16bytes
User: 7, IP: /192.168.56.1, PORT: 54313, Recieved and sent 16bytes
User: 4, IP: /192.168.56.1, PORT: 54310, Recieved and sent 16bytes
User: 5, IP: /192.168.56.1, PORT: 54311, Recieved and sent 16bytes
User: 6, IP: /192.168.56.1, PORT: 54312, Recieved and sent 16bytes
User: 7, IP: /192.168.56.1, PORT: 54313, Recieved and sent 16bytes
User: 8, IP: /192.168.56.1, PORT: 54315, Recieved and sent 16bytes
User: 8, IP: /192.168.56.1, PORT: 54315, Recieved and sent 16bytes
User: 9, IP: /192.168.56.1, PORT: 54316, Recieved and sent 16bytes
User: 10, IP: /192.168.56.1, PORT: 54317, Recieved and sent 16bytes
User: 9, IP: /192.168.56.1, PORT: 54316, Recieved and sent 16bytes
User: 10, IP: /192.168.56.1, PORT: 54317, Recieved and sent 16bytes
```

This picture displays the server output when it receives requests from multiple clients and includes the data received and sent back to the client. It is implemented using threads as stated as an requirement in the assignment and the picture displays 10 clients sending messages with a transfer rate of 2.

## Problem 4

### TCP connection in wireshark:



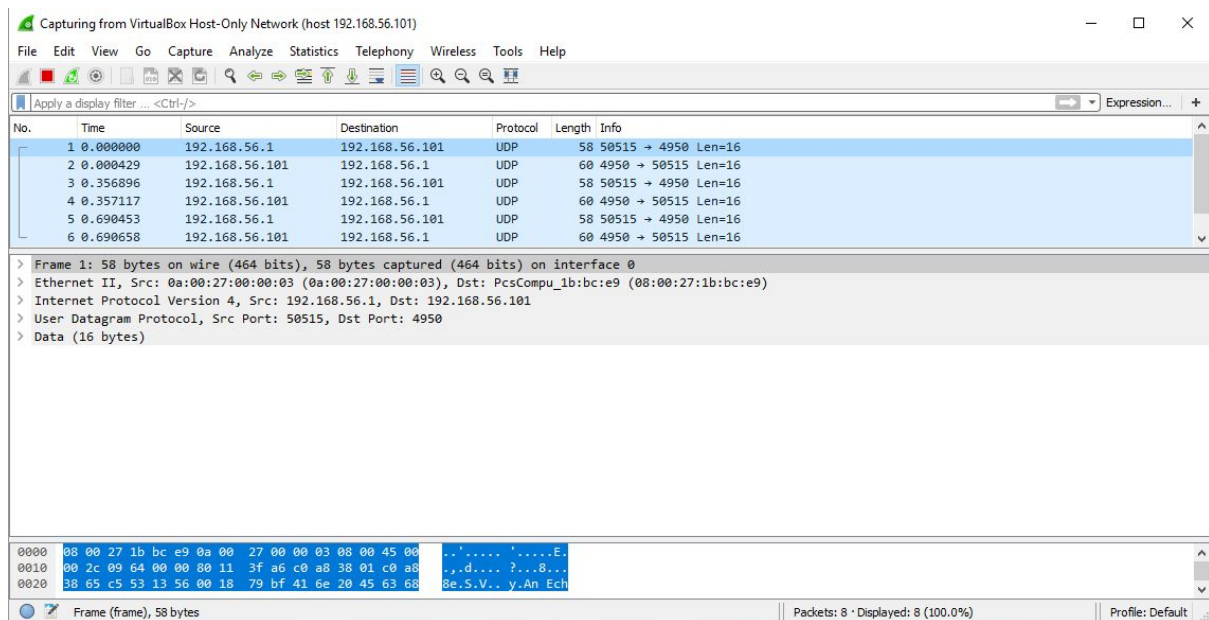
In this picture the TCP client is using a buffer that can hold 1024 bytes and is sending a message to the server. Wireshark displays how the client sent a synchronize(SYN) message with the sequence number 0 to the server. Then the server replied with a synchronize-acknowledgment(SYN-ACK) message where the sequence number is 0 and the Ack is set to 1. Then the client replied with Ack set to 1 and the connection have been established.

The client then proceeds with sending a message to the server of length 16. Here the (PSH) flag is set so the client doesn't wait for the servers buffer to be filled but send the message to the client immediately.

Server then receives the message and responds with Ack value 17. Server then sends the message back to the client with the PSH flag and Ack set to 17. When client received the message it responds with an Ack message set to 17. This process continues for three messages and then the client sends a message with FIN and the last Ack value. FIN means that the client is finished but will still keep listening to the server. Server then responds with the same message but increments the Ack value by one. TCP always expects the same Ack values to be shared by the client and server which makes it more reliable.

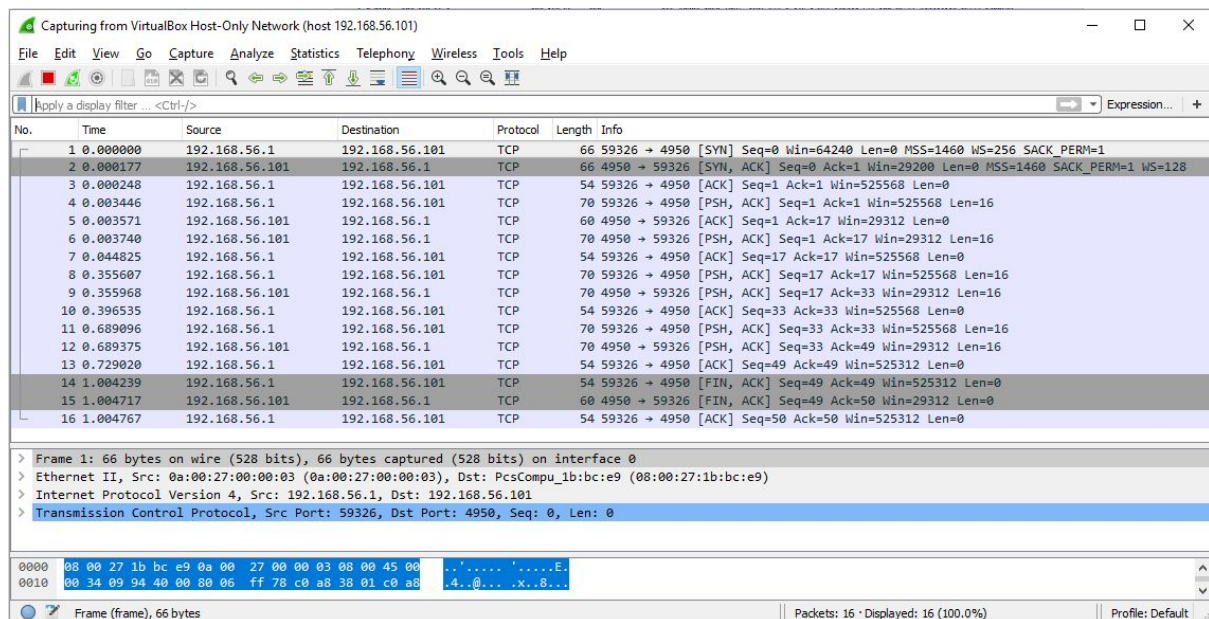


## UDP connection in wireshark:



In this picture the UDP client sends a message with length of 16 bytes to the server and the server receives 58 bytes (16 for the message and 42 for the header). Then the server sends back the same message to the client. This is repeated two more times. In the lower part of the image we can see that there was 58 bytes sent to the server which is the 16 bytes for the message and 42 for the header.

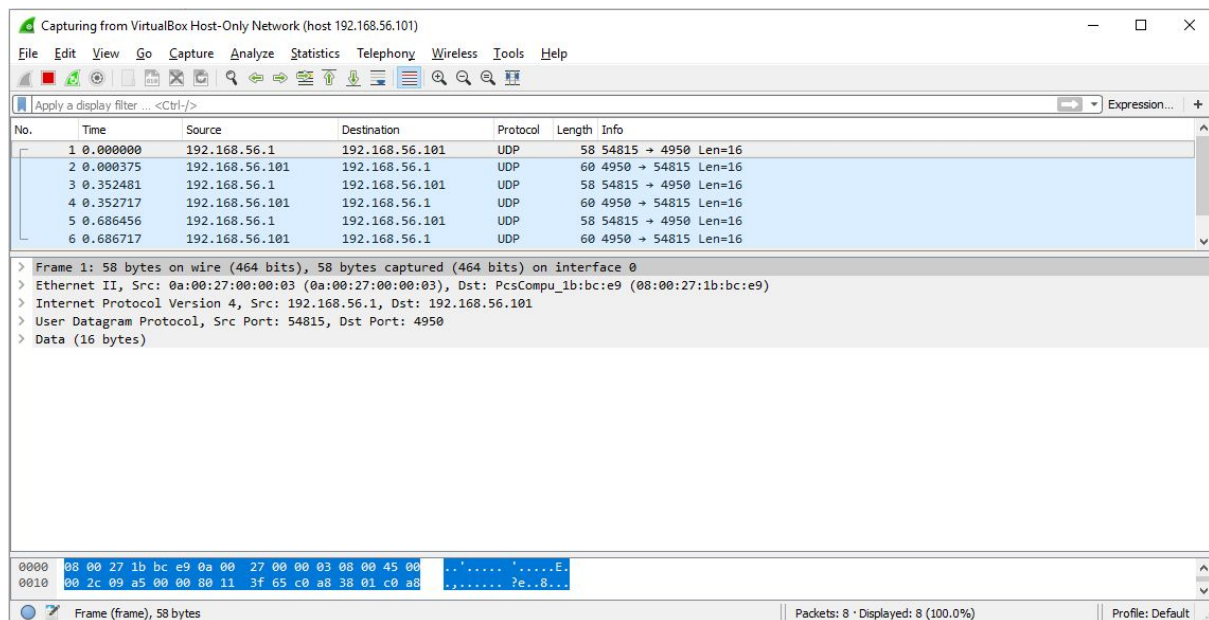
## TCP with small buffer:



Here the TCP client's buffer size is set to hold 8 bytes but the TCP server can still hold 1024 bytes. The client still works even though messages of 16 bytes are sent back to the client since the TCP client is using a stream to read and send the messages to the server. So the TCP client is still functional.

Oskar Mendel  
Student id: om222dq  
Student mail: om222dq@student.lnu.se

## UDP with small buffer:



Now the UDP client have a buffer set to hold 8 bytes and the messages sent between the server and client is of size 16 bytes. The messages are sent to the server but cannot be retrieved as expected since the datagram packet is of a set size. When a message retrieved is of greater length than the specified buffer size it will drop all the bytes beyond its size which will result in that bytes are lost and the messages are not equal. This shows how TCP is more reliable and safe to use because it uses all the bytes.

