

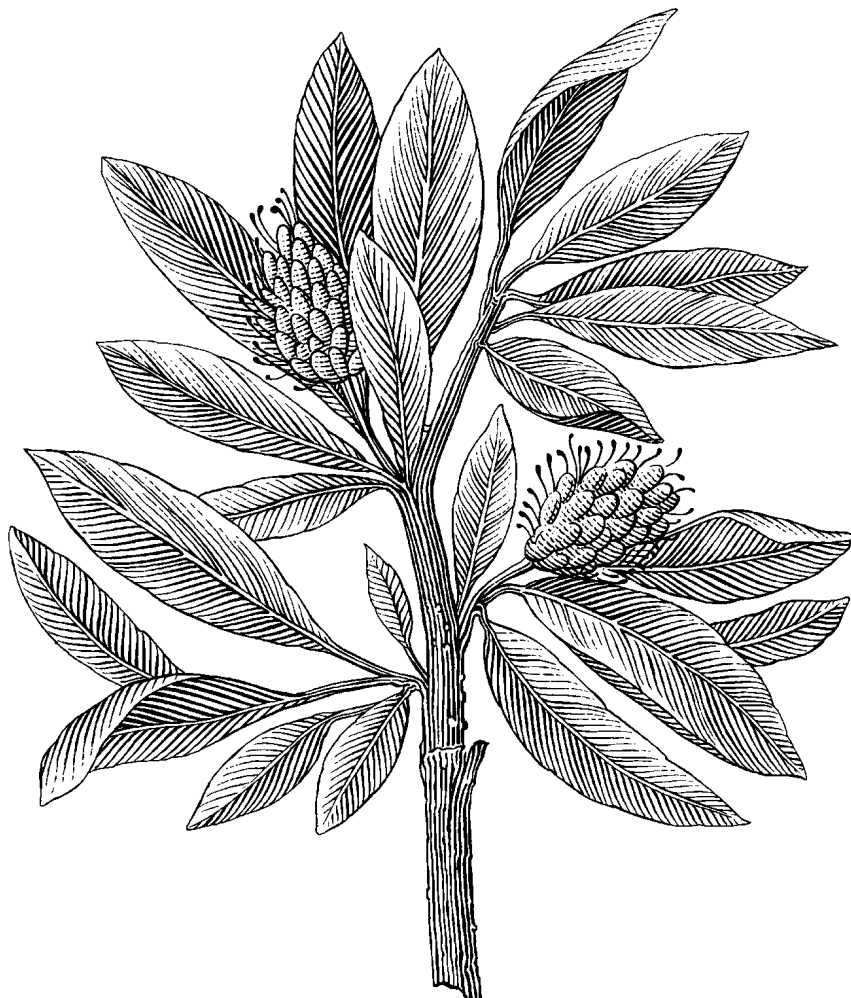


Linnéuniversitetet
Kalmar Våxjö

Report

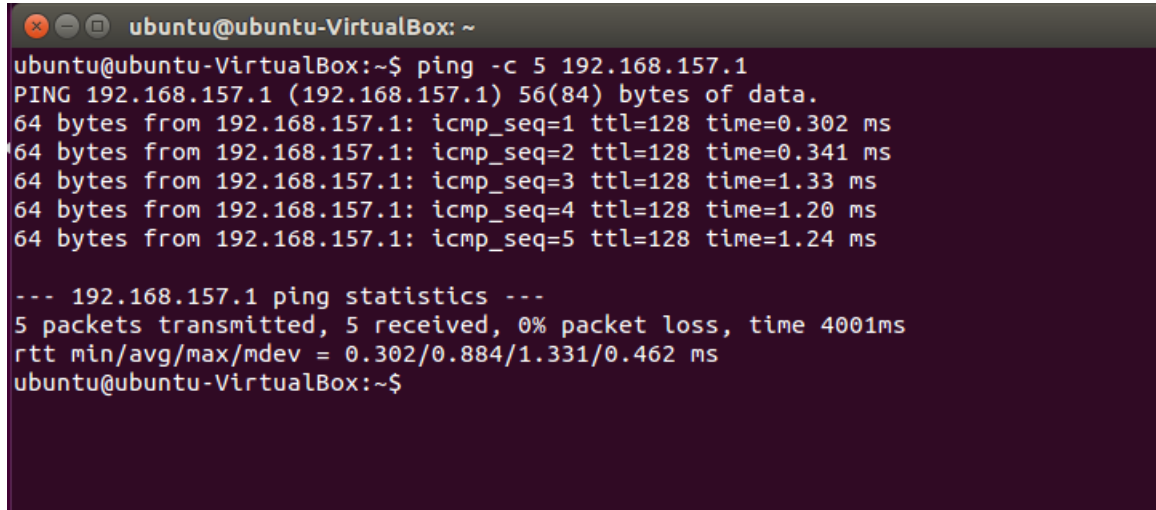
Assignment 1

UDP/TCP socket programming



Author: Mohammed Basel Nasrini
Course: 1DV701
Semester: Spring 2018

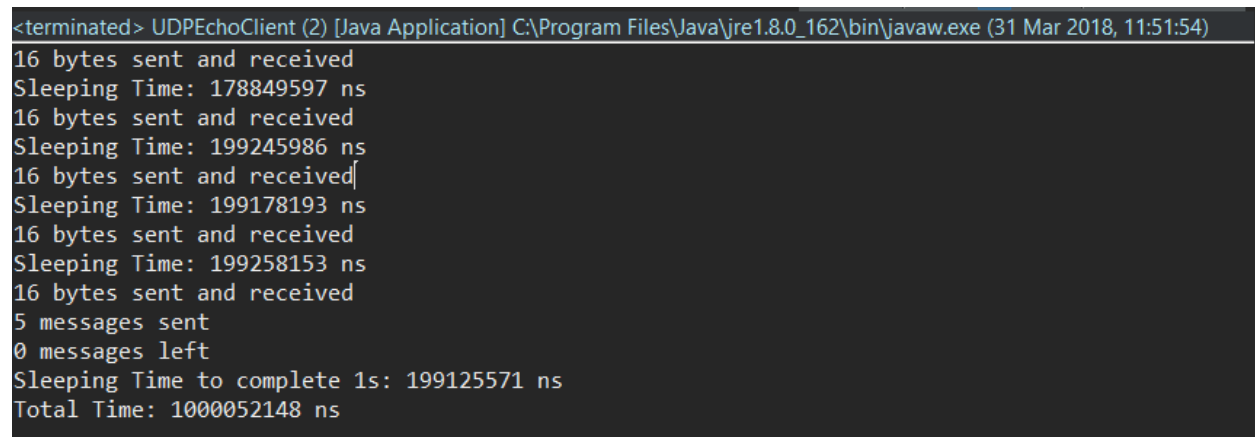
Problem 1



```
ubuntu@ubuntu-VirtualBox: ~  
ubuntu@ubuntu-VirtualBox:~$ ping -c 5 192.168.157.1  
PING 192.168.157.1 (192.168.157.1) 56(84) bytes of data.  
64 bytes from 192.168.157.1: icmp_seq=1 ttl=128 time=0.302 ms  
64 bytes from 192.168.157.1: icmp_seq=2 ttl=128 time=0.341 ms  
64 bytes from 192.168.157.1: icmp_seq=3 ttl=128 time=1.33 ms  
64 bytes from 192.168.157.1: icmp_seq=4 ttl=128 time=1.20 ms  
64 bytes from 192.168.157.1: icmp_seq=5 ttl=128 time=1.24 ms  
  
--- 192.168.157.1 ping statistics ---  
5 packets transmitted, 5 received, 0% packet loss, time 4001ms  
rtt min/avg/max/mdev = 0.302/0.884/1.331/0.462 ms  
ubuntu@ubuntu-VirtualBox:~$
```

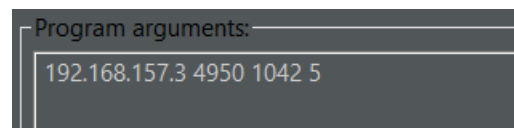
The above picture shows a ping from the virtual machine to the host machine.

Problem 2



```
<terminated> UDPEchoClient (2) [Java Application] C:\Program Files\Java\jre1.8.0_162\bin\javaw.exe (31 Mar 2018, 11:51:54)  
16 bytes sent and received  
Sleeping Time: 178849597 ns  
16 bytes sent and received  
Sleeping Time: 199245986 ns  
16 bytes sent and received  
Sleeping Time: 199178193 ns  
16 bytes sent and received  
Sleeping Time: 199258153 ns  
16 bytes sent and received  
5 messages sent  
0 messages left  
Sleeping Time to complete 1s: 199125571 ns  
Total Time: 1000052148 ns
```

The above picture shows sending 5 messages from the host machine to the virtual machine.



```
Program arguments:  
192.168.157.3 4950 1042 5
```

The above picture shows the used argument to send the messages. Which is [IP] [Port] [Buffer size] [Transfer rate].

List of the exceptions/errors handled by the program:

1. Number of the arguments is not equal to 4.
2. IP doesn't contain three dots "." or four integers between 0 and 255.
3. Port number isn't an integer or between 1 and 65535.

4. Buffer size is less than 1 or too big which make (outOfMemoryError) exception or not an integer.
 5. Transfer rate is less than 0 or not an integer.
 6. Message is empty (or longer than 65507 in the case of UDP connection according to the UDP maximum packet size).
-

VG-task 1

```
<terminated> UDPEchoClient (2) [Java Application] C:\Program Files\Java\jre1.8.0_162\bin\javaw.exe (31 Mar 2018, 13:20:45)
16 bytes sent and received
Sleeping Time: 481056805 ns
16 bytes sent and received
2 messages sent
0 messages left
Sleeping Time to complete 1s: 499324445 ns
Total Time: 1000039506 ns
```

I used (System.nanoTime) to obtain the time in the beginning of the sending and receiving process and stop it after 1 sec. A delay method sleepToTime(long time) used to delay the next sending process until a given time. For example, if the transfer rate is 2, after sending the first message it will sleep about 0,5 second after sending the second message. And in the end, it will sleep until completing 1 sec. The above picture shows the client side sending 2 messages.

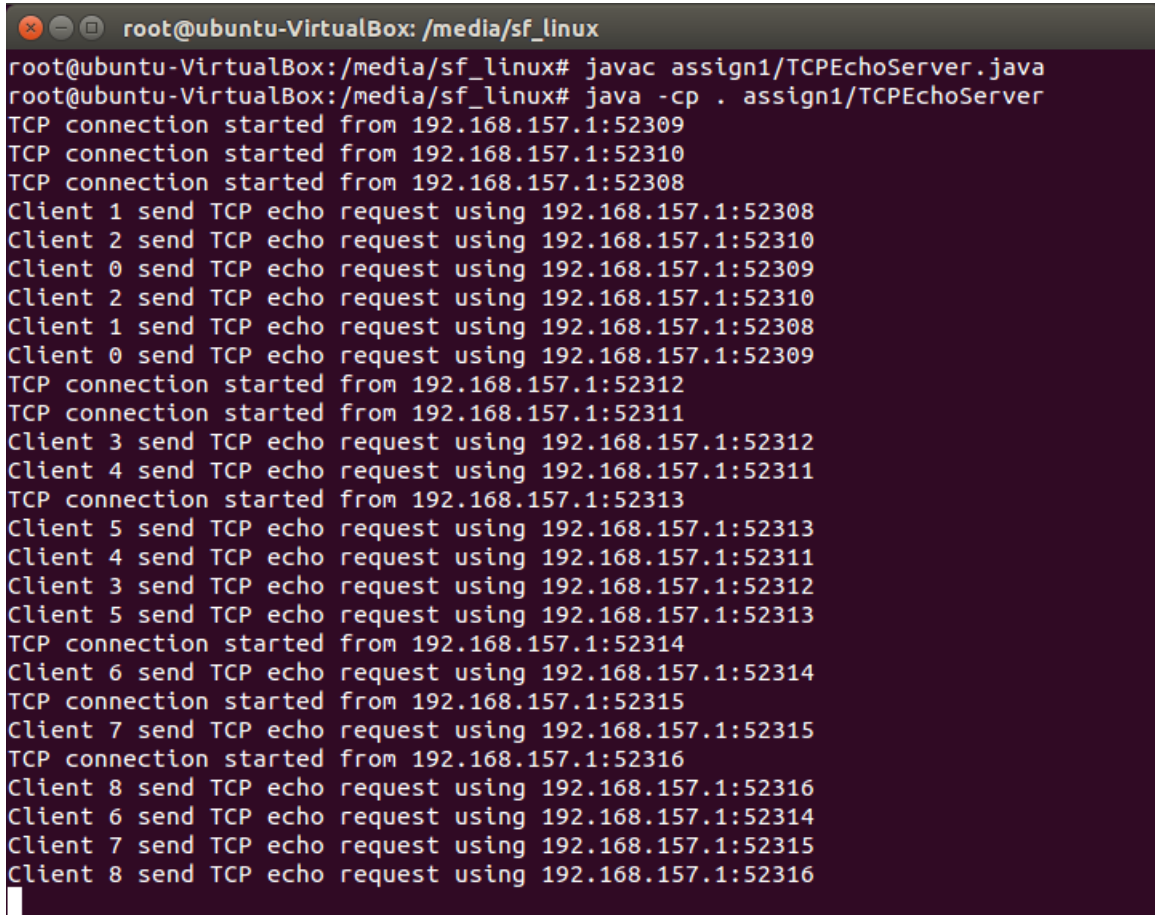
```
<terminated> UDPEchoClient (2) [Java Application] C:\Program Files\Java\jre1.8.0_162\bin\javaw.exe (31 Mar 2018, 13:31:33)
16 bytes sent and received
16 bytes sent and received
16 bytes sent and received
16 bytes sent and received
16 bytes sent and received
16 bytes sent and received
16 bytes sent and received
16 bytes sent and received
16 bytes sent and received
16 bytes sent and received
826 messages sent
174 messages left
Total Time: 1046945938 ns
```

The above picture shows the client side using big transfer rate (here I used 1000/sec). It shows the number message sent within a sec and the left message.

VG-Task 2

An abstract class NetworkingLayer has the shared methods used by both UDP and TCP.

Problem 3



```
root@ubuntu-VirtualBox: /media/sf_linux
root@ubuntu-VirtualBox:/media/sf_linux# javac assign1/TCPEchoServer.java
root@ubuntu-VirtualBox:/media/sf_linux# java -cp . assign1/TCPEchoServer
TCP connection started from 192.168.157.1:52309
TCP connection started from 192.168.157.1:52310
TCP connection started from 192.168.157.1:52308
Client 1 send TCP echo request using 192.168.157.1:52308
Client 2 send TCP echo request using 192.168.157.1:52310
Client 0 send TCP echo request using 192.168.157.1:52309
Client 2 send TCP echo request using 192.168.157.1:52310
Client 1 send TCP echo request using 192.168.157.1:52308
Client 0 send TCP echo request using 192.168.157.1:52309
TCP connection started from 192.168.157.1:52312
TCP connection started from 192.168.157.1:52311
Client 3 send TCP echo request using 192.168.157.1:52312
Client 4 send TCP echo request using 192.168.157.1:52311
TCP connection started from 192.168.157.1:52313
Client 5 send TCP echo request using 192.168.157.1:52313
Client 4 send TCP echo request using 192.168.157.1:52311
Client 3 send TCP echo request using 192.168.157.1:52312
Client 5 send TCP echo request using 192.168.157.1:52313
TCP connection started from 192.168.157.1:52314
Client 6 send TCP echo request using 192.168.157.1:52314
TCP connection started from 192.168.157.1:52315
Client 7 send TCP echo request using 192.168.157.1:52315
TCP connection started from 192.168.157.1:52316
Client 8 send TCP echo request using 192.168.157.1:52316
Client 6 send TCP echo request using 192.168.157.1:52314
Client 7 send TCP echo request using 192.168.157.1:52315
Client 8 send TCP echo request using 192.168.157.1:52316
```

The above screenshot shows the server side of multiple TCP clients connections. Each client sends 2 messages. I used ThreadPool for sending multiple clients at the same time and its in the class multiTCPClient.

The client will stop after sending response, but the server will continue waiting for new requests until manual termination.

TCP with small buffer size

I sent a message of 16 bytes. The client buffer size is 3 bytes and the server buffer size is 1024 bytes.

```
search your computer and online sources
root@ubuntu-virtualbox:/media/sf_linux# java -cp . assign1/TCPEchoServer
TCP connection started from 192.168.157.1:58841
Client 0 send TCP echo request using 192.168.157.1:58841 | Sent and received:16 bytes | Buffer Size: 1024 bytes
Client 0 send TCP echo request using 192.168.157.1:58841 | Sent and received:16 bytes | Buffer Size: 1024 bytes
Client 0 send TCP echo request using 192.168.157.1:58841 | Sent and received:16 bytes | Buffer Size: 1024 bytes
```

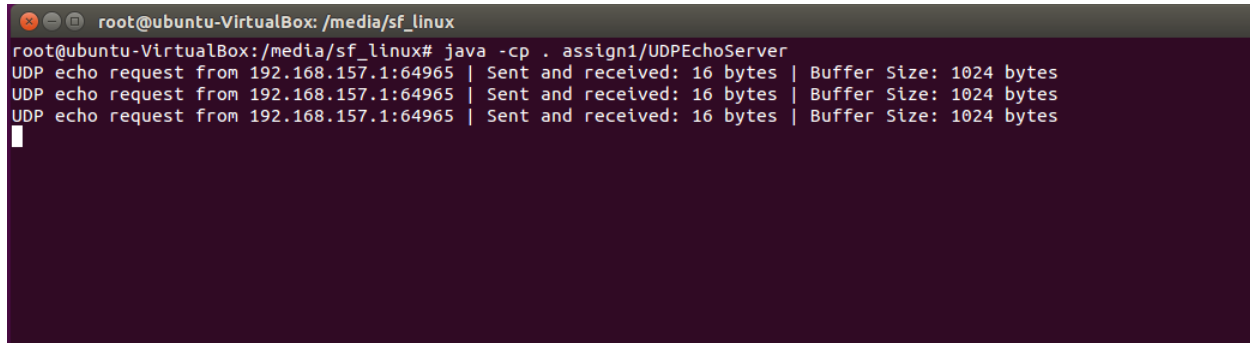
The above screenshot shows the server side of sending. The server received 16 bytes from client and sent 16 bytes to it.

```
<terminated> TCPEchoClient [Java Application] C:\Program Files\Java\jre1.8.0_162\bin\javaw.exe (1 Apr 2018, 18:22:54)
16 bytes sent and received | Buffer size: 3 bytes
Sleeping Time: 309076167 ns
16 bytes sent and received | Buffer size: 3 bytes
Sleeping Time: 332471967 ns
16 bytes sent and received | Buffer size: 3 bytes
3 messages sent
0 messages left
Sleeping Time to complete 1s: 332227688 ns
Total Time: 1000041086 ns
```

Here we can see the client side. It sent and received 16 bytes even if the buffer size is 3 bytes.

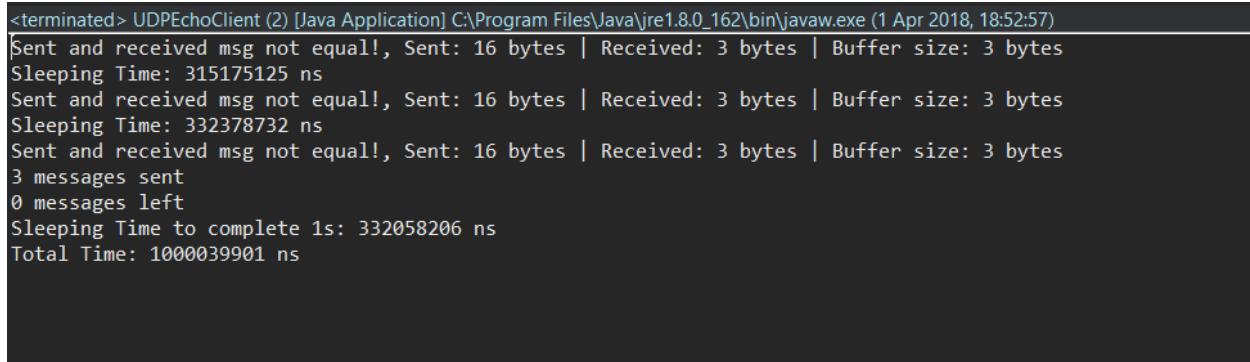
UDP with small buffer size

Now here I did the same thing with UDP client. The UDP client buffer size set to 3 and the server buffer size was 1024.

A screenshot of a terminal window titled 'root@ubuntu-VirtualBox: /media/sf_linux'. The terminal shows the execution of a Java program 'assign1/UDPEchoServer'. The output consists of three lines, each representing a received UDP echo request. Each line reports that 16 bytes were sent and received, and the buffer size is 1024 bytes.

```
root@ubuntu-VirtualBox: /media/sf_linux
root@ubuntu-VirtualBox:/media/sf_linux# java -cp . assign1/UDPEchoServer
UDP echo request from 192.168.157.1:64965 | Sent and received: 16 bytes | Buffer Size: 1024 bytes
UDP echo request from 192.168.157.1:64965 | Sent and received: 16 bytes | Buffer Size: 1024 bytes
UDP echo request from 192.168.157.1:64965 | Sent and received: 16 bytes | Buffer Size: 1024 bytes
```

The above picture is a screenshot of the UDP server. It shows that the server has received and sent 16 bytes and the buffer size is 1024.

A screenshot of a terminal window titled '<terminated> UDPEchoClient (2) [Java Application] C:\Program Files\Java\jre1.8.0_162\bin\javaw.exe (1 Apr 2018, 18:52:57)'. The output shows three iterations of sending and receiving messages. In each iteration, 16 bytes were sent but only 3 bytes were received. The buffer size is consistently 3 bytes. The terminal also shows sleeping times and a total execution time.

```
<terminated> UDPEchoClient (2) [Java Application] C:\Program Files\Java\jre1.8.0_162\bin\javaw.exe (1 Apr 2018, 18:52:57)
Sent and received msg not equal!, Sent: 16 bytes | Received: 3 bytes | Buffer size: 3 bytes
Sleeping Time: 315175125 ns
Sent and received msg not equal!, Sent: 16 bytes | Received: 3 bytes | Buffer size: 3 bytes
Sleeping Time: 332378732 ns
Sent and received msg not equal!, Sent: 16 bytes | Received: 3 bytes | Buffer size: 3 bytes
3 messages sent
0 messages left
Sleeping Time to complete 1s: 332058206 ns
Total Time: 1000039901 ns
```

The above picture is a screenshot of the UDP client. It shows that the client has sent 16 bytes, but it received 3 bytes. The same size of the buffer size.

What is the difference in this case between TCP and UDP?

The difference was in the client side. UDP received 3 bytes while TCP received 16 bytes even if the message size and the buffer size was same in both cases.

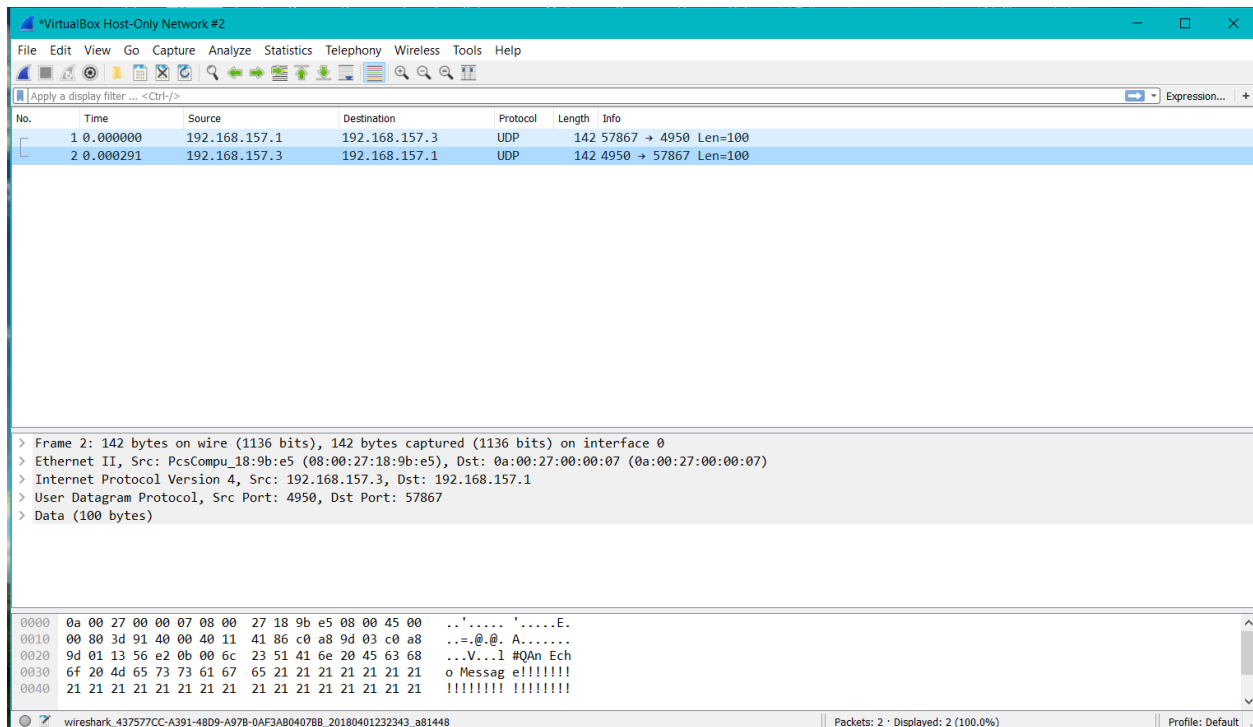
UDP uses datagram packets to receive messages and the size of this packet is the same size of the buffer size which was 3 in the previous case. If it received a message bigger than its size. It will ignore the bytes that are bigger than its size, so it received 3 bytes and lost the remaining 13 bytes and will never get them back.

TCP uses stream (DataInputStream) to receive the message and if the message bigger than the buffer size, it will continuously read the message and append it to a string (or a StringBuilder like I used in my implementation) until the whole response has been read.

So, TCP is more efficient than UDP because it takes all bytes and does not the response even if it's bigger than the client buffer size.

Problem 4

UDP Server Traffic with small buffer size:



The above picture is a screenshot of the UDP server traffic.

Explanation about the traffic from the picture:

Client IP: 192.168.157.1, Port: 57867

Server IP: 192.168.157.3, Port: 4950

The message has been sent 1 time and its size is 100.

The traffic took the following steps:

1. Client sent 142 bytes to the server (100 bytes is the message length + 42 bytes is the header length).
2. The server sent back the same message to the client (100 bytes is the message length + 42 bytes is the header length).

By using UDP the client will receive the message depending on its buffer size and ignore the rest of the message (in my case the buffer size was 3 so the client received 3 bytes and dropped off the rest of it).

TCP Server Traffic with small buffer size:

VirtualBox Host-Only Network #2

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

Apply a display filter ... <Ctrl-/> Expression... +

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.157.1	192.168.157.3	TCP	66	60932 → 4950 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
2	0.000223	192.168.157.3	192.168.157.1	TCP	66	4950 → 60932 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460 SACK_PERM=1 WS=128
3	0.000303	192.168.157.1	192.168.157.3	TCP	54	60932 → 4950 [ACK] Seq=1 Ack=1 Win=525568 Len=0
4	0.002232	192.168.157.1	192.168.157.3	TCP	154	60932 → 4950 [PSH, ACK] Seq=1 Ack=1 Win=525568 Len=100
5	0.002384	192.168.157.3	192.168.157.1	TCP	60	4950 → 60932 [ACK] Seq=1 Ack=101 Win=29312 Len=0
6	0.010841	192.168.157.3	192.168.157.1	TCP	154	4950 → 60932 [PSH, ACK] Seq=1 Ack=101 Win=29312 Len=100
7	0.010928	192.168.157.1	192.168.157.3	TCP	54	60932 → 4950 [ACK] Seq=101 Ack=101 Win=525312 Len=0
8	0.029104	192.168.157.1	192.168.157.3	TCP	54	60932 → 4950 [FIN, ACK] Seq=101 Ack=101 Win=525312 Len=0
9	0.029665	192.168.157.3	192.168.157.1	TCP	60	4950 → 60932 [FIN, ACK] Seq=101 Ack=102 Win=29312 Len=0
10	0.029731	192.168.157.1	192.168.157.3	TCP	54	60932 → 4950 [ACK] Seq=102 Ack=102 Win=525312 Len=0

> Frame 2: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface 0

> Ethernet II, Src: PcsCompu_18:9b:e5 (08:00:27:18:9b:e5), Dst: 0a:00:27:00:00:07 (0a:00:27:00:00:07)

> Internet Protocol Version 4, Src: 192.168.157.3, Dst: 192.168.157.1

> Transmission Control Protocol, Src Port: 4950, Dst Port: 60932, Seq: 0, Ack: 1, Len: 0

0000 0a 00 27 00 00 07 08 00 27 18 9b e5 08 00 45 00 ..'.....'.....E.

0010 00 34 00 00 40 00 04 06 7f 6e c0 a8 9d 03 c0 a8 .4..@..n.....

0020 9d 01 13 56 ee 04 6f 31 0d 5e a7 1c 5e 14 00 12 ...V...o1 ^.^..

0030 72 10 be 7f 00 00 02 04 05 b4 01 01 04 02 01 03 r.....

0040 03 07 ..

Byte 46: Nonce (tcp.flags.ns) | Packets: 10 · Displayed: 10 (100.0%) | Profile: Default

Explanation about the traffic from the picture:

Client IP: 192.168.157.1, Port: 60932

Server IP: 192.168.157.3, Port: 4950

The message has been sent 1 time and its size is 100.

Terms Description:

- Win (Window size): used to inform the other end of the connection about the amount of bytes receiver buffer can hold. And by time, it informs the remaining buffer size. If the receiver window size is exceeded it will send and ACK that Win=0. The sender will not send messages anymore and will wait until the receiver to consume the data and send a positive window size.
- MSS (Maximum segment size): It's found only in the SYN and SYN/ACK packets of the connection which it's the maximum size of data in bytes that a compute or a communication device can handle in a single unfragmented piece.
- SACK_PERM=1: means that the sender knows how to work with Selective Acknowledgments.

The traffic took the following steps:

1. The client sent sync message (SYN) to server with sequence number Seq=0 and windows scaling WS=256.

2. Server replied with a sync and acknowledgment message (SYN, ACK) with Seq=0, Ack=1.
3. Client replied with acknowledgment message (ACK) with Seq=1, Ack=1. Here the connection has been established and that what it's called three-way handshake.
4. Client sent the message that has 100 bytes length (Len=100) with Seq=1, Ack=1. It's also set the push flag (PSH) that tells the server to send the message back to the client immediately without waiting until its buffer to be filled (Server buffer size=1024).
5. Server sent an acknowledgment message (ACK) to the client with Seq=1, Ack=101.
6. Server replied to the client with the message that has 100 bytes length (Len=100) with Seq=1, Ack=101. It's also set the push flag. (PSH)
7. Client received the message and sent an acknowledgment message (ACK) with Seq=101, Ack=101.
8. Client sent a finish message (FIN) that told the server that client finished the sending process with Seq=101, Ack=101.
9. Server replied with an acknowledgment message (ACK) and increasing the value by 1 Ack=101 and with a finish message. Seq=101, Ack=102.
10. Client replied with an acknowledgment message (ACK) with Seq=102, Ack=102. Here the connection is ended.

Differences between TCP and UDP

Transmission Control Protocol (TCP)	User Datagram Protocol (UDP)
Connection-oriented protocol (handshaking)	Connectionless protocol
Used when the sending time is less important (Slow)	Suited when it need fast transmission like in gaming (Fast)
Rearrange the data packets	Doesn't rearrange data packets
Guarantee that the whole sent data will be received (Reliable)	Some data may be lost (Unreliable)
Used by HTTP, HTTPs	Used by DNS, VOIP, TFTP
Acknowledgment Segments	No acknowledgment segment