



# Report

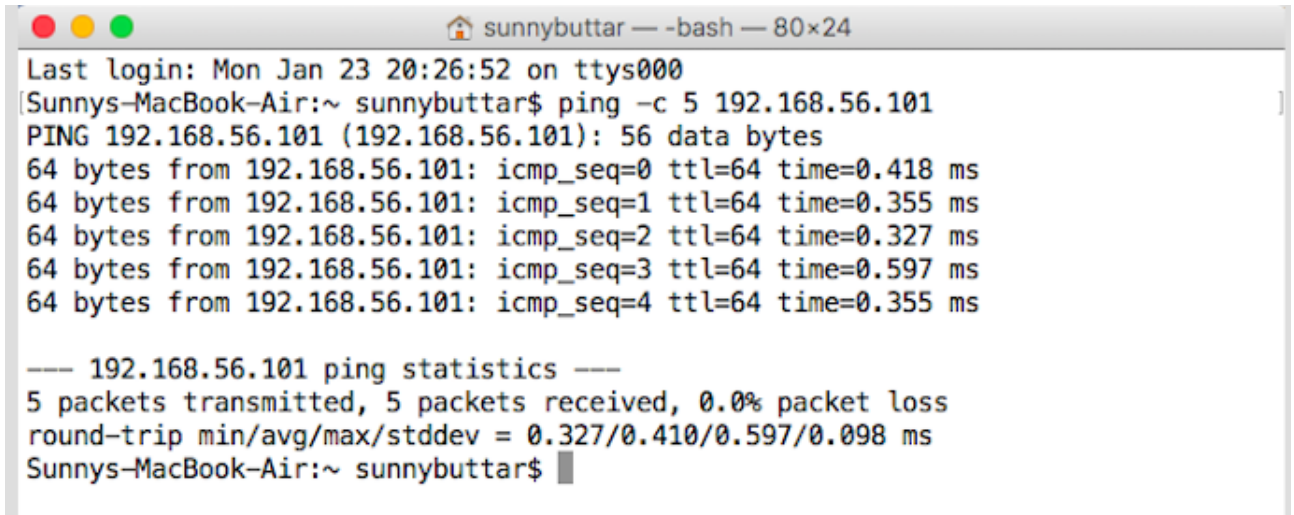
## Assignment 1 - *UDP / TCP socket programming*



11 Feb, 2017

*Semester:* Spring 2017  
*Course:* Computer Networks  
*Author:* Sarpreet Singh Buttar

## Problem 1

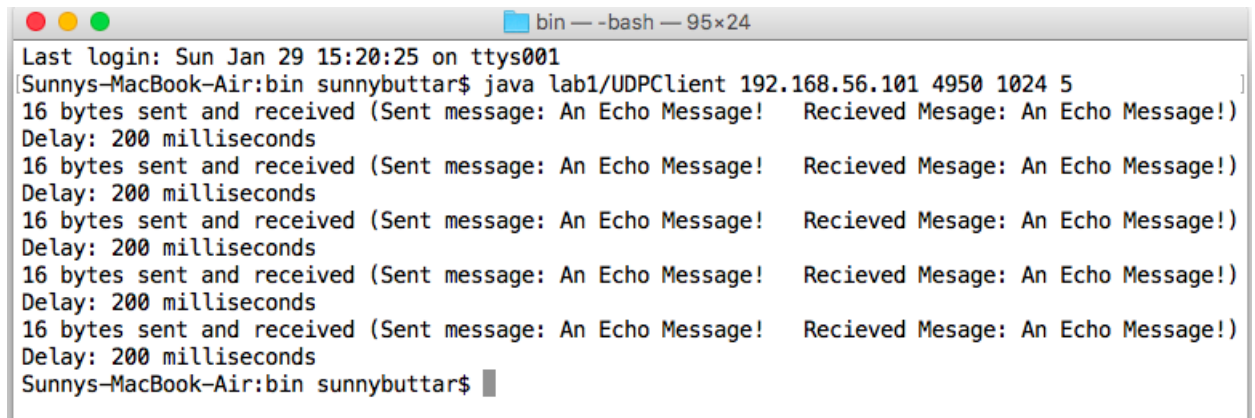
A screenshot of a terminal window titled 'sunnybuttar — -bash — 80x24'. The window shows the output of a ping command. The prompt is 'Sunnys-MacBook-Air:~ sunnybuttar\$'. The command entered is 'ping -c 5 192.168.56.101'. The output shows five successful ping probes with their respective round-trip times. Below the probes, there is a summary line: '--- 192.168.56.101 ping statistics ---'. This is followed by statistics: '5 packets transmitted, 5 packets received, 0.0% packet loss' and 'round-trip min/avg/max/stddev = 0.327/0.410/0.597/0.098 ms'. The prompt returns to 'Sunnys-MacBook-Air:~ sunnybuttar\$'.

```
Last login: Mon Jan 23 20:26:52 on ttys000
Sunnys-MacBook-Air:~ sunnybuttar$ 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=0.418 ms
64 bytes from 192.168.56.101: icmp_seq=1 ttl=64 time=0.355 ms
64 bytes from 192.168.56.101: icmp_seq=2 ttl=64 time=0.327 ms
64 bytes from 192.168.56.101: icmp_seq=3 ttl=64 time=0.597 ms
64 bytes from 192.168.56.101: icmp_seq=4 ttl=64 time=0.355 ms

--- 192.168.56.101 ping statistics ---
5 packets transmitted, 5 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 0.327/0.410/0.597/0.098 ms
Sunnys-MacBook-Air:~ sunnybuttar$
```

The above picture shows the output of running ping on host machine for sending five probes to the virtual machine target. It also include the result obtained from each probe. The shortest round trip time is 0.327 ms, the longest round trip time is 0.597 ms and the average round trip time is 1.768 ms.

## Problem 2



```
bin - bash - 95x24
Last login: Sun Jan 29 15:20:25 on ttys001
Sunnys-MacBook-Air:bin sunnybuttar$ java lab1/UDPCClient 192.168.56.101 4950 1024 5
16 bytes sent and received (Sent message: An Echo Message!  Recieved Mesage: An Echo Message!)
Delay: 200 milliseconds
16 bytes sent and received (Sent message: An Echo Message!  Recieved Mesage: An Echo Message!)
Delay: 200 milliseconds
16 bytes sent and received (Sent message: An Echo Message!  Recieved Mesage: An Echo Message!)
Delay: 200 milliseconds
16 bytes sent and received (Sent message: An Echo Message!  Recieved Mesage: An Echo Message!)
Delay: 200 milliseconds
16 bytes sent and received (Sent message: An Echo Message!  Recieved Mesage: An Echo Message!)
Delay: 200 milliseconds
Sunnys-MacBook-Air:bin sunnybuttar$
```

The above picture shows the amount of sent and received data during the process of sending 5 messages from host machine to virtual machine. In addition, we can also see the amount of delay between each message. In the picture, the given argument is in [IP] [port] [buffer size] [message transfer rate] format. For testing the program, please follow this pattern for providing the arguments.

### List of handled exceptions/error

1. When number of arguments is not equal to 4.
2. When port, buffer size and message transfer rate are not integers.
3. When IP is invalid or contains characters other than dot(.).
4. When port is greater than 65535 or less than 1.
5. When buffer size is less than 1.
6. When buffer size is too big and cause out of memory exception.
7. When message transfer rate is less than 0.
8. When message size is above than 65507 (according to UDP packet size).
9. When message is empty.

An IP address can be considered as invalid if:

1. it does not contain four '.' characters.
2. it does not contain four integers between those four '.' characters.
3. those integers are more than 255 or less than 0.
4. Eg: 256.123.234.0, asdd.fff.8fgg.93d, 255.255.123.-1 etc are some examples of invalid IP address.

Here is the example of valid IP address 192.168.56.101

## **VG Tasks**

### **VG Task 1**

For sending as many messages as possible in 1 second, I used thread. UDP and TCP classes implemented a method called run which execute the thread for approximately 1 second. For executing the thread, I use thread pool. There is one inbuilt method in thread pool called 'await' which runs the thread according to the given number of milliseconds. After spending some time on checking the average time, I decided to provide 995 ms rather than 1000 ms in order to stop the process within 1 sec. However, sometimes program runs above or below one second depending on the message transfer rate and the amount of delay created by it.

### **VG Task 2**

There is one abstract class called Network layer. This class perform all the shared responsibilities among UDP and TCP such as creating local and remote point, validating IP, port, buffer size, message transfer rate. In addition, it has some method which is used by both protocols such as delay, run the process for 1 second, display process time, compare sent and receive message and display process time. This class implements Runnable interface and the run method is implemented in UDP and TCP client classes separately in order to send as many messages as possible in 1 second.

## Problem 3

### TCP with mutiple clients

```
root@ubuntu-VirtualBox:/media/sf_Assignments# java -cp . TCPEchoServer
Server Running
[User 1] TCP echo request from /192.168.56.1 using port 50891 (Received [16bytes], Sent [16 bytes], Buffer size = 1024)
[User 2] TCP echo request from /192.168.56.1 using port 50892 (Received [16bytes], Sent [16 bytes], Buffer size = 1024)
[User 1] TCP echo request from /192.168.56.1 using port 50891 (Received [16bytes], Sent [16 bytes], Buffer size = 1024)
[User 3] TCP echo request from /192.168.56.1 using port 50893 (Received [16bytes], Sent [16 bytes], Buffer size = 1024)
[User 2] TCP echo request from /192.168.56.1 using port 50892 (Received [16bytes], Sent [16 bytes], Buffer size = 1024)
[User 4] TCP echo request from /192.168.56.1 using port 50894 (Received [16bytes], Sent [16 bytes], Buffer size = 1024)
[User 5] TCP echo request from /192.168.56.1 using port 50895 (Received [16bytes], Sent [16 bytes], Buffer size = 1024)
[User 3] TCP echo request from /192.168.56.1 using port 50891 (Received [16bytes], Sent [16 bytes], Buffer size = 1024)
[User 6] TCP echo request from /192.168.56.1 using port 50896 (Received [16bytes], Sent [16 bytes], Buffer size = 1024)
[User 4] TCP echo request from /192.168.56.1 using port 50894 (Received [16bytes], Sent [16 bytes], Buffer size = 1024)
[User 7] TCP echo request from /192.168.56.1 using port 50897 (Received [16bytes], Sent [16 bytes], Buffer size = 1024)
[User 5] TCP echo request from /192.168.56.1 using port 50895 (Received [16bytes], Sent [16 bytes], Buffer size = 1024)
[User 8] TCP echo request from /192.168.56.1 using port 50898 (Received [16bytes], Sent [16 bytes], Buffer size = 1024)
[User 6] TCP echo request from /192.168.56.1 using port 50896 (Received [16bytes], Sent [16 bytes], Buffer size = 1024)
[User 7] TCP echo request from /192.168.56.1 using port 50897 (Received [16bytes], Sent [16 bytes], Buffer size = 1024)
[User 9] TCP echo request from /192.168.56.1 using port 50899 (Received [16bytes], Sent [16 bytes], Buffer size = 1024)
[User 8] TCP echo request from /192.168.56.1 using port 50898 (Received [16bytes], Sent [16 bytes], Buffer size = 1024)
[User 9] TCP echo request from /192.168.56.1 using port 50899 (Received [16bytes], Sent [16 bytes], Buffer size = 1024)
```

The above picture belong to server side and it shows the requests of multiple clients including the amount of data they sent to and received from TCP server.

The image displays a 3x3 grid of console window screenshots, each representing a different client instance. Each window shows the following output:

```
<terminated> Main (9) [Java Application] [Library\Java\JavaVirtualMachines\jdk1.8.0_102...]
16 bytes sent and received (Buffer Size = 100)
16 bytes sent and received (Buffer Size = 100)
Time: 996 ms
Message left: 0
```

The above picture belongs to client side and it shows the amount of data sent to and received from TCP server by each client.

## TCP with small buffer size

Now, TCP client buffer size is set to 64 and message size is 100. On the other hand TCP server buffer size is still set to default (1024).

[illegible]

The above picture shows the TCP server side. We can see that client is sending 100 bytes to server and also receiving same amount of bytes from server.

[illegible]

The above picture shows the TCP client side. We can see that client is sending 100 bytes to server and receiving 100 bytes back even if buffer size is 64.



### UDP with small buffer size

Now, similarly UDP client buffer size is set to 64 and message size is 100. On the other hand UDP server buffer size is still set to default (1024).

[illegible]

The above picture shows the UDP server side. We can see that client is sending 100 bytes to server and also receiving same amount of bytes from server.

[illegible]

The above picture shows the UDP client side. We can see that client is sending 100 bytes to server and but only receiving 64 bytes back which is same as its buffer size.

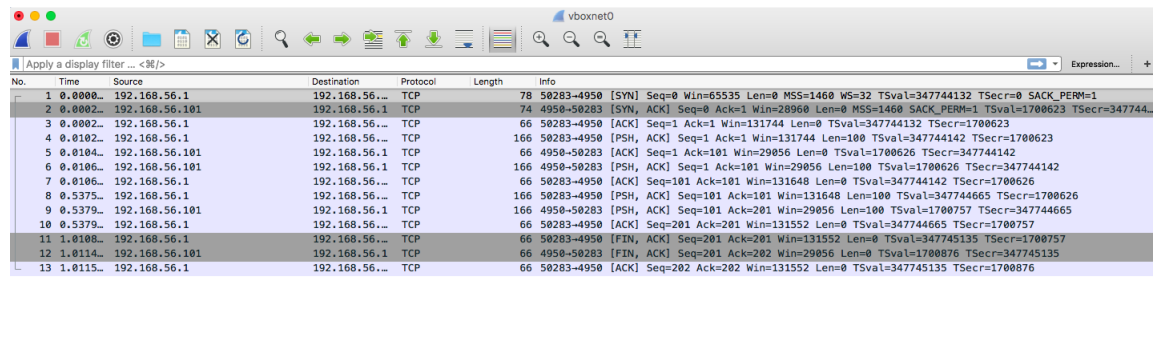
### **What is the difference and why?**

The difference is the amount of bytes receiving from server. We have clearly seen that TCP got 100 bytes back and UDP only got 64 bytes back. This happen because TCP use stream to read and send the messages and it continues to read until there is a message. In each iteration it creates a new buffer and save the message inside the same string. When message ends, it remove all the empty spaces and compare the receieved message with sent one. On the other hand, UDP use datagram packets to send and receive the message. Datagram packets have some fixed length which we specify with the buffer size. So when a message comes whose length is greater than packet length, the packet drop all the bytes which are beyond its own size. In result, those bytes are lost and UDP never try to get them back. That is why TCP is more reliable than UDP because it takes all the bytes.



## Problem 4

### TCP traffic with smaller buffer size



No.	Time	Source	Destination	Protocol	Length	Info
1	0.0000	192.168.56.1	192.168.56.101	TCP	78	50283->4950 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=32 TSval=347744132 TSecr=0 SACK_PERM=1
2	0.0002	192.168.56.101	192.168.56.1	TCP	74	4950->50283 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460 SACK_PERM=1 TSval=1700623 TSecr=347744132
3	0.0002	192.168.56.1	192.168.56.101	TCP	66	50283->4950 [ACK] Seq=1 Ack=1 Win=131744 Len=0 TSval=347744132 TSecr=1700623
4	0.0102	192.168.56.1	192.168.56.101	TCP	166	50283->4950 [PSH, ACK] Seq=1 Ack=1 Win=131744 Len=100 TSval=347744142 TSecr=1700623
5	0.0104	192.168.56.101	192.168.56.1	TCP	66	4950->50283 [ACK] Seq=1 Ack=101 Win=29056 Len=0 TSval=1700626 TSecr=347744142
6	0.0106	192.168.56.101	192.168.56.1	TCP	166	4950->50283 [PSH, ACK] Seq=1 Ack=101 Win=29056 Len=100 TSval=1700626 TSecr=347744142
7	0.0106	192.168.56.1	192.168.56.101	TCP	66	50283->4950 [ACK] Seq=101 Ack=101 Win=131648 Len=0 TSval=347744142 TSecr=1700626
8	0.5375	192.168.56.1	192.168.56.101	TCP	166	50283->4950 [PSH, ACK] Seq=101 Ack=101 Win=131648 Len=100 TSval=347744665 TSecr=1700626
9	0.5379	192.168.56.101	192.168.56.1	TCP	166	4950->50283 [PSH, ACK] Seq=101 Ack=201 Win=29056 Len=100 TSval=1700757 TSecr=347744665
10	0.5379	192.168.56.1	192.168.56.101	TCP	66	50283->4950 [ACK] Seq=201 Ack=201 Win=131552 Len=0 TSval=347744665 TSecr=1700757
11	1.0108	192.168.56.1	192.168.56.101	TCP	66	50283->4950 [FIN, ACK] Seq=201 Ack=201 Win=131552 Len=0 TSval=347745135 TSecr=1700757
12	1.0114	192.168.56.101	192.168.56.1	TCP	66	4950->50283 [FIN, ACK] Seq=201 Ack=202 Win=29056 Len=0 TSval=1700876 TSecr=347745135
13	1.0115	192.168.56.1	192.168.56.101	TCP	66	50283->4950 [ACK] Seq=202 Ack=202 Win=131552 Len=0 TSval=347745135 TSecr=1700876

Following steps explains what happened in the above picture.

1. Host machine (client) sent a synchronize (SYN) message with sequence number (Seq) = 0 to virtual machine (server)
2. Server replied with a synchronize-acknowledgment (SYN-ACK) message where Seq = 0 and Ack = 1
3. Client replied with an acknowledgment message with ACK = 1. At this point connection is established
4. Client sent a message to server whose length is 100. In this step Push (PSH) flag sets which means do not wait until the server buffer size (1024) is full, send the message to client immediately.
5. Server received the message and sent back ACK with value 101
6. Server sent message to client with PSH flag and ACK = 101
7. Client received the message and sent back ACK with value 101
8. Client sent message to server with PSH flag and same ACK value
9. Server sent message to client with PSH flag and ACK = 201
10. Client received the message and sent back ACK with same value
11. Client replied with FIN and same ACK value. FIN means client finished with sending the messages but will still listen to server
12. Server also replied with same message but increasing the ACK value by 1 (ACK = 202). In TCP ACK value always increase with +1. Whatever value server gets from client it add 1 into it and then waits for the client response with same ACK value. This makes TCP more reliable than UDP.
13. Client received the message and sent back ACK with same value to inform the server that message has been received

## UDP traffic with smaller buffer size

No.	Time	Source	Destination	Protocol	Length	Info
1	0.0000...	192.168.56.1	192.168.56.1	UDP	142	50160→4950 Len=100
2	0.0023...	192.168.56.101	192.168.56.1	UDP	142	4950→50160 Len=100
3	0.5075...	192.168.56.1	192.168.56.1	UDP	142	50160→4950 Len=100
4	0.5078...	192.168.56.101	192.168.56.1	UDP	142	4950→50160 Len=100

► Frame 4: 142 bytes on wire (1136 bits), 142 bytes captured (1136 bits) on interface 0
► Ethernet II, Src: PcsSyste_1b:bcb:e9 (08:00:27:1b:bcb:e9), Dst: 0a:00:27:00:00:00 (0a:00:27:00:00:00)
► Internet Protocol Version 4, Src: 192.168.56.101, Dst: 192.168.56.1
► User Datagram Protocol, Src Port: 4950, Dst Port: 50160
► Data (100 bytes)

0000	0a 00 27 00 00 00 00 00	27 1b bc e9 00 00 45 00	..'......'.....E.
0010	00 00 50 c4 40 00 40 11	ef f1 c9 a8 38 65 c0 a8	..X.@.@: ....Be..
0020	38 01 13 56 c3 f0 00 6c	0b 0b 41 6e 20 45 63 68	8..V...l ..An Ech
0030	6f 20 40 65 73 73 61 67	65 21 21 21 21 21 21 21	o Messag e!!!!!!!
0040	21 21 21 21 21 21 21 21	21 21 21 21 21 21 21 21	!!!!!!!
0050	21 21 21 21 21 21 21 21	21 21 21 21 21 21 21 21	!!!!!!!
0060	21 21 21 21 21 21 21 21	21 21 21 21 21 21 21 21	!!!!!!!
0070	21 21 21 21 21 21 21 21	21 21 21 21 21 21 21 21	!!!!!!!
0080	21 21 21 21 21 21 21 21	21 21 21 21 21 21 21 21	!!!!!!!

Following steps explains what happened in the above picture.

1. Client sent a message to server whose length is 100 and server received 142 bytes (100 from message + 42 from header)
2. Server sent the same message back to client
3. Client again sent the same message to server
4. Server again sent it back to client

In the bottom of the picture we can see the complete message from client but client can only receive 64 bytes because of the packet size which is depended on the buffer size (64 bytes).

## Difference between UDP and TCP

UDP	TCP
It is connection less	It is connection oriented (three way handshake)
It is fast	It is slow
It is unreliable	It is reliable
It cannot bring lost data	It brings lost data
Arrived data is unordered	Arrived data is ordered
It send and receive data in packets. Only one packet can be send and received in one call	Multiple packets can be send and received at once because it use stream to receive and send data
It is used by DNS, streaming media applications etc	It is used by World Wide Web, email etc