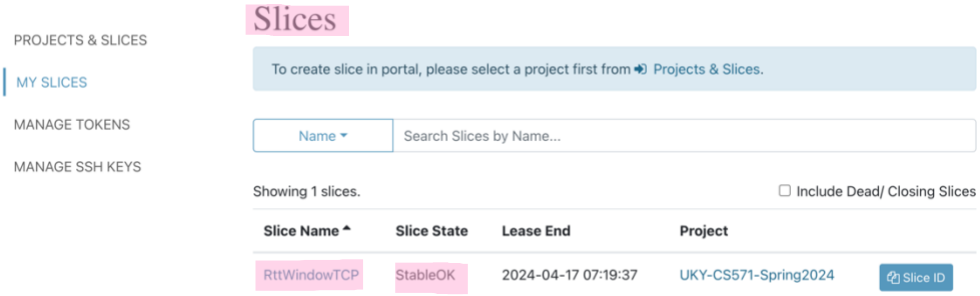
Here are what you will need to submit:

- Screenshots about the slice created. Include the four tables displayed after you run the cell submitting your slice. (20') **(DONE)**
- Screenshots of what you did and the corresponding output in the client terminal and server terminal for the first part of the assignment (RTT and window size). (20') **(DONE)**
- Screenshots of four plots show the number of packets transmitted over time. (20')**(DONE)**
- Modified UDPEchoServer.c and UDPEchoClient-Timeout.c code. (20') **(DONE)**
- Screenshots of what you did and the corresponding output in the client and server terminals when running UDPEchoServer and UDPEchoClient-Timeout. (20') **(DONE)**

1. Showing the information about the slice created:



Including the four tables that were displayed after I ran the cell submitting the slice:

- 1.

Slice

ID	c9ac3862-c2af-4fba-af71-92a9133edce5
Name	RttWindowTCP
Lease Expiration (UTC)	2024-04-11 11:19:37 +0000
Lease Start (UTC)	2024-04-10 11:19:38 +0000
Project ID	bcbbcfe6-a8c3-49c7-91ab-132f960945dd
State	StableOK

- 2.

Nodes

ID	Name	Cores	RAM	Disk	Image	Image Type	Host	Site	Username	Management IP	State	Error	SSH Command	Public SSH Key File	Private SSH Key File
14a48a87-59d5-4958-90be-4ab1de4e568d	Client	1	2	10	default_ubuntu_20	qcow2	eduky-w12.fabric-testbed.net	EDUKY	ubuntu	2610:1e0:1700:206:f816:3eff:fe71:7c5a	Active		ssh -i /home/fabric/work/fabric_config/slice_key -F /home/fabric/work/fabric_config/ssh_config ubuntu@2610:1e0:1700:206:f816:3eff:fe71:7c5a	/home/fabric/work/fabric_config/slice_key.pub	/home/fabric/work/fabric_config/slice_key
bc87dbf6-03c4-4955-8aa7-69a899bdf8ab	Server	1	2	10	default_ubuntu_20	qcow2	eduky-w12.fabric-testbed.net	EDUKY	ubuntu	2610:1e0:1700:206:f816:3eff:fea8:cd70	Active		ssh -i /home/fabric/work/fabric_config/slice_key -F /home/fabric/work/fabric_config/ssh_config ubuntu@2610:1e0:1700:206:f816:3eff:fea8:cd70	/home/fabric/work/fabric_config/slice_key.pub	/home/fabric/work/fabric_config/slice_key

- 3.

Networks

ID	Name	Layer	Type	Site	Subnet	Gateway	State	Error
cf26210b-410c-47ad-ae71-9eec0018714d	LAN_CONN	L2	L2Bridge	EDUKY	None	None	Active	

- 4.

Interfaces

Name	Short Name	Node	Network	Bandwidth	Mode	VLAN	MAC	Physical Device	Device	IP Address	Numa Node
Server-Server_P-p1	p1	Server	LAN_CONN	100	config		1E:D0:CA:80:12:10	enp7s0	enp7s0	None	1
Client-Client_P-p1	p1	Client	LAN_CONN	100	config		32:95:C0:CD:71:75	enp7s0	enp7s0	None	1

2. Showing what I did and the corresponding output in the client terminal and server terminal for the first part of the assignment (RTT and window size):

2.1:

When I ran ifconfig on the terminal Server:

```
ubuntu@Server:~$ ifconfig
enp3s0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 9000
        inet 10.30.8.26  netmask 255.255.224.0  broadcast 10.30.31.255
        inet6 2610:1e0:1700:206:f816:3eff:fea8:cd70  prefixlen 64  scopeid 0x0<global>
        inet6 fe80::f816:3eff:fea8:cd70  prefixlen 64  scopeid 0x20<link>
        ether fa:16:3e:a8:cd:70  txqueuelen 1000  (Ethernet)
        RX packets 66909  bytes 309472442 (309.4 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 49057  bytes 4884167 (4.8 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

enp7s0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.1.1.1  netmask 255.255.255.0  broadcast 0.0.0.0
        inet6 fe80::1cd0:caff:fe80:1210  prefixlen 64  scopeid 0x20<link>
        ether 1e:d0:ca:80:12:10  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 11  bytes 866 (866.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 258  bytes 24004 (24.0 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 258  bytes 24004 (24.0 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

ubuntu@Server:~$
```

This will be the interface that is able to connect the server and the client.

Then adjusting the MTU size for the interface enp7s0, we adjusted the MTU in sudo ifconfig <interface name> mtu # to # = 1400:

```
ubuntu@Server:~$ sudo ifconfig enp7s0 mtu 1400
ubuntu@Server:~$
```

Then I repeated the same thing for the client, so I ran ifconfig on the terminal Client:

```
ubuntu@Client:~$ ifconfig
enp3s0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 9000
        inet 10.30.6.103  netmask 255.255.224.0  broadcast 10.30.31.255
        inet6 fe80::f816:3eff:fe71:7c5a  prefixlen 64  scopeid 0x20<link>
        inet6 2610:1e0:1700:206:f816:3eff:fe71:7c5a  prefixlen 64  scopeid 0x0<global>
        ether fa:16:3e:71:7c:5a  txqueuelen 1000  (Ethernet)
        RX packets 68337  bytes 309603661 (309.6 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 49676  bytes 4939409 (4.9 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

enp7s0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.1.1.2  netmask 255.255.255.0  broadcast 0.0.0.0
        inet6 fe80::3095:c0ff:fecd:7175  prefixlen 64  scopeid 0x20<link>
        ether 32:95:c0:cd:71:75  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 13  bytes 1006 (1.0 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 258  bytes 24004 (24.0 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 258  bytes 24004 (24.0 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

ubuntu@Client:~$
```

Then adjusting the MTU size for the interface enp7s0, we adjusted the MTU in sudo ifconfig <interface name> mtu # to # = 1400:

```
ubuntu@Client:~$ sudo ifconfig enp7s0 mtu 1400
ubuntu@Client:~$
```

2.2:

To find the approximate base RTT between the nodes, we are using the ping tool:

```
ubuntu@Server:~$ ping 10.1.1.2 -c 10
PING 10.1.1.2 (10.1.1.2) 56(84) bytes of data.
64 bytes from 10.1.1.2: icmp_seq=1 ttl=64 time=0.201 ms
64 bytes from 10.1.1.2: icmp_seq=2 ttl=64 time=0.079 ms
64 bytes from 10.1.1.2: icmp_seq=3 ttl=64 time=0.077 ms
64 bytes from 10.1.1.2: icmp_seq=4 ttl=64 time=0.076 ms
64 bytes from 10.1.1.2: icmp_seq=5 ttl=64 time=0.070 ms
64 bytes from 10.1.1.2: icmp_seq=6 ttl=64 time=0.064 ms
64 bytes from 10.1.1.2: icmp_seq=7 ttl=64 time=0.101 ms
64 bytes from 10.1.1.2: icmp_seq=8 ttl=64 time=0.085 ms
64 bytes from 10.1.1.2: icmp_seq=9 ttl=64 time=0.092 ms
64 bytes from 10.1.1.2: icmp_seq=10 ttl=64 time=0.087 ms

--- 10.1.1.2 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9201ms
rtt min/avg/max/mdev = 0.064/0.093/0.201/0.037 ms
ubuntu@Server:~$
```

Then we use the Linux tool called tc; traffic control to add a fixed amount of delay to the packets leaving the interface, and so the delay will be 100ms and using the command sudo tc qdisc add dev <interface name> root netem delay 100ms:

```
ubuntu@Server:~$ sudo tc qdisc add dev enp7s0 root netem delay 100ms
ubuntu@Server:~$
```

Then run the ping again to see the RTT average:

```
ubuntu@Server:~$ ping 10.1.1.2 -c 10
PING 10.1.1.2 (10.1.1.2) 56(84) bytes of data.
64 bytes from 10.1.1.2: icmp_seq=1 ttl=64 time=100 ms
64 bytes from 10.1.1.2: icmp_seq=2 ttl=64 time=100 ms
64 bytes from 10.1.1.2: icmp_seq=3 ttl=64 time=100 ms
64 bytes from 10.1.1.2: icmp_seq=4 ttl=64 time=100 ms
64 bytes from 10.1.1.2: icmp_seq=5 ttl=64 time=100 ms
64 bytes from 10.1.1.2: icmp_seq=6 ttl=64 time=100 ms
64 bytes from 10.1.1.2: icmp_seq=7 ttl=64 time=100 ms
64 bytes from 10.1.1.2: icmp_seq=8 ttl=64 time=100 ms
64 bytes from 10.1.1.2: icmp_seq=9 ttl=64 time=100 ms
64 bytes from 10.1.1.2: icmp_seq=10 ttl=64 time=100 ms

--- 10.1.1.2 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9010ms
rtt min/avg/max/mdev = 100.080/100.092/100.114/0.011 ms
ubuntu@Server:~$
```

It does show that the RTT average is 100ms longer

Then we are able to remove the delay using the following command sudo tc qdisc del dev <interface name> root:

```
ubuntu@Server:~$ sudo tc qdisc del dev enp7s0 root
ubuntu@Server:~$
```

Then finally, we will be pinging the client should now be the same as it was before adding the delay ping <Client ip addr> -c 10:

```
ubuntu@Server:~$ ping 10.1.1.2 -c 10
PING 10.1.1.2 (10.1.1.2) 56(84) bytes of data.
64 bytes from 10.1.1.2: icmp_seq=1 ttl=64 time=0.082 ms
64 bytes from 10.1.1.2: icmp_seq=2 ttl=64 time=0.071 ms
64 bytes from 10.1.1.2: icmp_seq=3 ttl=64 time=0.078 ms
64 bytes from 10.1.1.2: icmp_seq=4 ttl=64 time=0.077 ms
64 bytes from 10.1.1.2: icmp_seq=5 ttl=64 time=0.072 ms
64 bytes from 10.1.1.2: icmp_seq=6 ttl=64 time=0.058 ms
64 bytes from 10.1.1.2: icmp_seq=7 ttl=64 time=0.068 ms
64 bytes from 10.1.1.2: icmp_seq=8 ttl=64 time=0.075 ms
64 bytes from 10.1.1.2: icmp_seq=9 ttl=64 time=0.075 ms
64 bytes from 10.1.1.2: icmp_seq=10 ttl=64 time=0.071 ms

--- 10.1.1.2 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9224ms
rtt min/avg/max/mdev = 0.058/0.072/0.082/0.006 ms
ubuntu@Server:~$
```

2.3:

Now we will be using the iperf is a tool for measuring TCP and UDP bandwidth performance; this tool will allow us to change the window size, so we'll be using the command (also -s will run iperf in server mode and will allow it to receive iperf traffic), and so in the Server terminal we'll use this command iperf -s
And, on the Client terminal, we'll use the command iperf -c <server ip addr> -t 10:

Server terminal

```
ubuntu@Server:~$ iperf -s
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size:  128 KByte (default)
------------------------------------------------------------
[  4] local 10.1.1.1 port 5001 connected with 10.1.1.2 port 38298
[ ID] Interval       Transfer     Bandwidth
[  4]  0.0-10.0 sec  22.1 GBytes  18.9 Gbits/sec
^Cubuntu@Server:~$
```

Client terminal

```
ubuntu@Client:~$ iperf -c 10.1.1.1 -t 10
------------------------------------------------------------
Client connecting to 10.1.1.1, TCP port 5001
TCP window size: 2.30 MByte (default)
------------------------------------------------------------
[  3] local 10.1.1.2 port 38298 connected with 10.1.1.1 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-10.0 sec  22.1 GBytes  19.0 Gbits/sec
ubuntu@Client:~$
```

Now the same procedure will be repeated, but we'll be limiting the window size on both the client and server using this command iperf -s -w 2KB in the Server terminal:
And, using this command iperf -c <server ip addr> -t 10 -w 2KB in the Client terminal:

Server terminal

```
ubuntu@Server:~$ iperf -s -w 2KB
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 4.00 KByte (WARNING: requested 2.00 KByte)
------------------------------------------------------------
[  4] local 10.1.1.1 port 5001 connected with 10.1.1.2 port 42632
[ ID] Interval       Transfer     Bandwidth
[  4]  0.0-10.0 sec   252 MBytes   211 Mbits/sec
^Cubuntu@Server:~$
```

Client terminal

```
ubuntu@Client:~$ iperf -c 10.1.1.1 -t 10 -w 2KB
------------------------------------------------------------
Client connecting to 10.1.1.1, TCP port 5001
TCP window size: 4.50 KByte (WARNING: requested 2.00 KByte)
------------------------------------------------------------
[  3] local 10.1.1.2 port 42632 connected with 10.1.1.1 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-10.0 sec   252 MBytes   211 Mbits/sec
ubuntu@Client:~$
```

When comparing the average bandwidth value, we can notice that it's significantly less than when we used the default window size

3. **Graph 1 (4KB and 50ms) → Step 1:** Set up the intended time delay using the command that was learned from section 2.2:

```
ubuntu@Server:~$ ping 10.1.1.2 -c 10
PING 10.1.1.2 (10.1.1.2) 56(84) bytes of data.
64 bytes from 10.1.1.2: icmp_seq=1 ttl=64 time=0.125 ms
64 bytes from 10.1.1.2: icmp_seq=2 ttl=64 time=0.077 ms
64 bytes from 10.1.1.2: icmp_seq=3 ttl=64 time=0.064 ms
64 bytes from 10.1.1.2: icmp_seq=4 ttl=64 time=0.071 ms
64 bytes from 10.1.1.2: icmp_seq=5 ttl=64 time=0.067 ms
64 bytes from 10.1.1.2: icmp_seq=6 ttl=64 time=0.048 ms
64 bytes from 10.1.1.2: icmp_seq=7 ttl=64 time=0.064 ms
64 bytes from 10.1.1.2: icmp_seq=8 ttl=64 time=0.064 ms
64 bytes from 10.1.1.2: icmp_seq=9 ttl=64 time=0.064 ms
64 bytes from 10.1.1.2: icmp_seq=10 ttl=64 time=0.067 ms

--- 10.1.1.2 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9203ms
rtt min/avg/max/mdev = 0.048/0.071/0.125/0.019 ms
ubuntu@Server:~$
ubuntu@Server:~$
ubuntu@Server:~$ sudo tc qdisc add dev enp7s0 root netem delay 50ms
ubuntu@Server:~$
ubuntu@Server:~$
ubuntu@Server:~$ ping 10.1.1.2 -c 10
PING 10.1.1.2 (10.1.1.2) 56(84) bytes of data.
64 bytes from 10.1.1.2: icmp_seq=1 ttl=64 time=50.1 ms
64 bytes from 10.1.1.2: icmp_seq=2 ttl=64 time=50.1 ms
64 bytes from 10.1.1.2: icmp_seq=3 ttl=64 time=50.1 ms
64 bytes from 10.1.1.2: icmp_seq=4 ttl=64 time=50.1 ms
64 bytes from 10.1.1.2: icmp_seq=5 ttl=64 time=50.1 ms
64 bytes from 10.1.1.2: icmp_seq=6 ttl=64 time=50.1 ms
64 bytes from 10.1.1.2: icmp_seq=7 ttl=64 time=50.1 ms
64 bytes from 10.1.1.2: icmp_seq=8 ttl=64 time=50.1 ms
64 bytes from 10.1.1.2: icmp_seq=9 ttl=64 time=50.1 ms
64 bytes from 10.1.1.2: icmp_seq=10 ttl=64 time=50.1 ms

--- 10.1.1.2 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9010ms
rtt min/avg/max/mdev = 50.066/50.081/50.114/0.013 ms
ubuntu@Server:~$
```

Step 2: Start the server side iperf using the command iperf -s -w 4KB in the Server terminal:

Before

```
ubuntu@Server:~$ iperf -s -w 4KB
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 8.00 KByte (WARNING: requested 4.00 KByte)
------------------------------------------------------------
```

After

```
ubuntu@Server:~$ iperf -s -w 4KB
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 8.00 KByte (WARNING: requested 4.00 KByte)
------------------------------------------------------------
[  4] local 10.1.1.1 port 5001 connected with 10.1.1.2 port 36636
[ ID] Interval       Transfer     Bandwidth
[  4]  0.0-12.1 sec   640 KBytes   433 Kbits/sec
^Cubuntu@Server:~$
```

Step 3: On the window to the Client (Terminal A), we will start the pre-loaded script with the command ./RT-output.sh:

```
ubuntu@Client:~$ ./RT-output.sh
tcpdump: listening on enp7s0, link-type EN10MB (Ethernet), capture size 262144 bytes
```

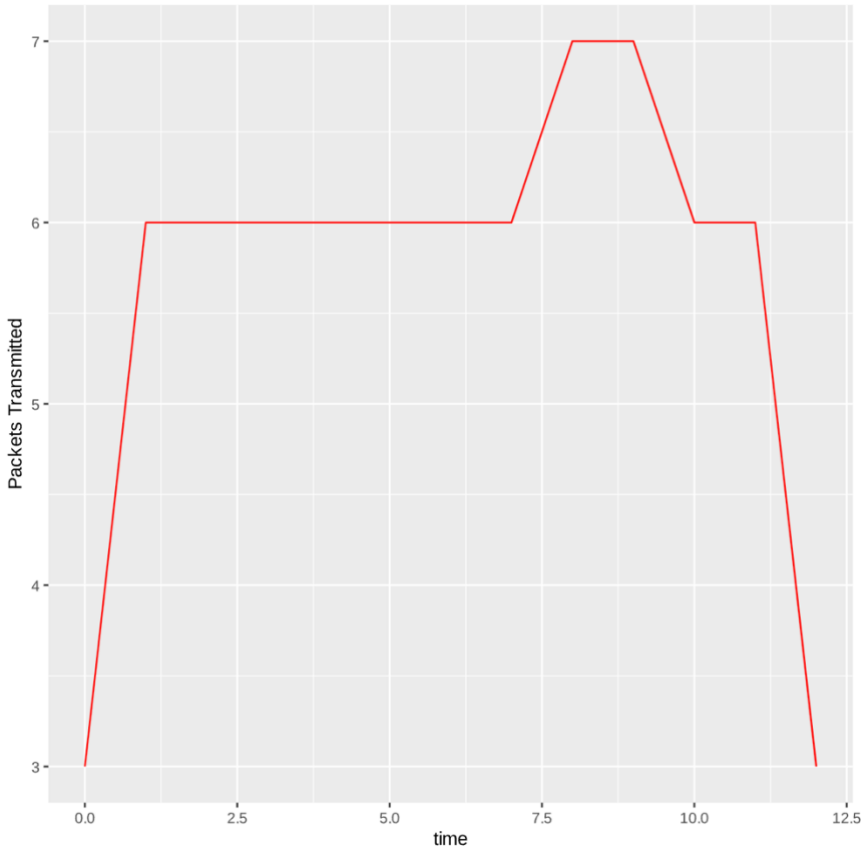
Step 4: On window to Client Terminal B, we will start the client connection to the server using the command iperf -c 10.1.1.1 -t 10 -w 4KB in the Client terminal:

```
ubuntu@Client:~$ iperf -c 10.1.1.1 -t 10 -w 4KB
------------------------------------------------------------
Client connecting to 10.1.1.1, TCP port 5001
TCP window size: 8.00 KByte (WARNING: requested 4.00 KByte)
------------------------------------------------------------
[  3] local 10.1.1.2 port 36636 connected with 10.1.1.1 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-12.1 sec   640 KBytes   434 Kbits/sec
ubuntu@Client:~$
```

Step 5: Once completed on Client Terminal B, we will hit Ctrl-C on Client Terminal A:

```
ubuntu@Client:~$ ./RT-output.sh
tcpdump: listening on enp7s0, link-type EN10MB (Ethernet), capture size 262144 bytes
^C787 packets captured
787 packets received by filter
0 packets dropped by kernel
^Xreading from file packets.pcap, link-type EN10MB (Ethernet)
ubuntu@Client:~$
```

Step 6: Ran the code and then opened the '.sgv' file to look at the graph for the following combination of 4KB and 50ms:



Graph 2 (4KB and 250ms) → Step 1: Set up the intended time delay using the command that was learned from section 2.2:

```
ubuntu@Server:~$ ping 10.1.1.2 -c 10
PING 10.1.1.2 (10.1.1.2) 56(84) bytes of data.
64 bytes from 10.1.1.2: icmp_seq=1 ttl=64 time=0.093 ms
64 bytes from 10.1.1.2: icmp_seq=2 ttl=64 time=0.065 ms
64 bytes from 10.1.1.2: icmp_seq=3 ttl=64 time=0.060 ms
64 bytes from 10.1.1.2: icmp_seq=4 ttl=64 time=0.055 ms
64 bytes from 10.1.1.2: icmp_seq=5 ttl=64 time=0.061 ms
64 bytes from 10.1.1.2: icmp_seq=6 ttl=64 time=0.059 ms
64 bytes from 10.1.1.2: icmp_seq=7 ttl=64 time=0.059 ms
64 bytes from 10.1.1.2: icmp_seq=8 ttl=64 time=0.053 ms
64 bytes from 10.1.1.2: icmp_seq=9 ttl=64 time=0.058 ms
64 bytes from 10.1.1.2: icmp_seq=10 ttl=64 time=0.059 ms

--- 10.1.1.2 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9212ms
rtt min/avg/max/mdev = 0.053/0.062/0.093/0.010 ms
ubuntu@Server:~$
ubuntu@Server:~$
ubuntu@Server:~$ sudo tc qdisc add dev enp7s0 root netem delay 250ms
ubuntu@Server:~$
ubuntu@Server:~$
ubuntu@Server:~$ ping 10.1.1.2 -c 10
PING 10.1.1.2 (10.1.1.2) 56(84) bytes of data.
64 bytes from 10.1.1.2: icmp_seq=1 ttl=64 time=250 ms
64 bytes from 10.1.1.2: icmp_seq=2 ttl=64 time=250 ms
64 bytes from 10.1.1.2: icmp_seq=3 ttl=64 time=250 ms
64 bytes from 10.1.1.2: icmp_seq=4 ttl=64 time=250 ms
64 bytes from 10.1.1.2: icmp_seq=5 ttl=64 time=250 ms
64 bytes from 10.1.1.2: icmp_seq=6 ttl=64 time=250 ms
64 bytes from 10.1.1.2: icmp_seq=7 ttl=64 time=250 ms
64 bytes from 10.1.1.2: icmp_seq=8 ttl=64 time=250 ms
64 bytes from 10.1.1.2: icmp_seq=9 ttl=64 time=250 ms
64 bytes from 10.1.1.2: icmp_seq=10 ttl=64 time=250 ms

--- 10.1.1.2 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9009ms
rtt min/avg/max/mdev = 250.063/250.072/250.090/0.008 ms
ubuntu@Server:~$
```

Step 2: Start the server side iperf using the command iperf -s -w 4KB in the Server terminal:

Before

```
ubuntu@Server:~$ iperf -s -w 4KB
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 8.00 KByte (WARNING: requested 4.00 KByte)
------------------------------------------------------------
```

After

```
ubuntu@Server:~$ iperf -s -w 4KB
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 8.00 KByte (WARNING: requested 4.00 KByte)
------------------------------------------------------------
[  4] local 10.1.1.1 port 5001 connected with 10.1.1.2 port 57260
[ ID] Interval       Transfer     Bandwidth
[  4]  0.0-10.3 sec   109 KBytes  87.3 Kbits/sec
^Cubuntu@Server:~$
```

Step 3: On the window to the Client (Terminal A), we will start the pre-loaded script with the command ./RT-output.sh:

```
ubuntu@Client:~$ ./RT-output.sh
tcpdump: listening on enp7s0, link-type EN10MB (Ethernet), capture size 262144 bytes
```

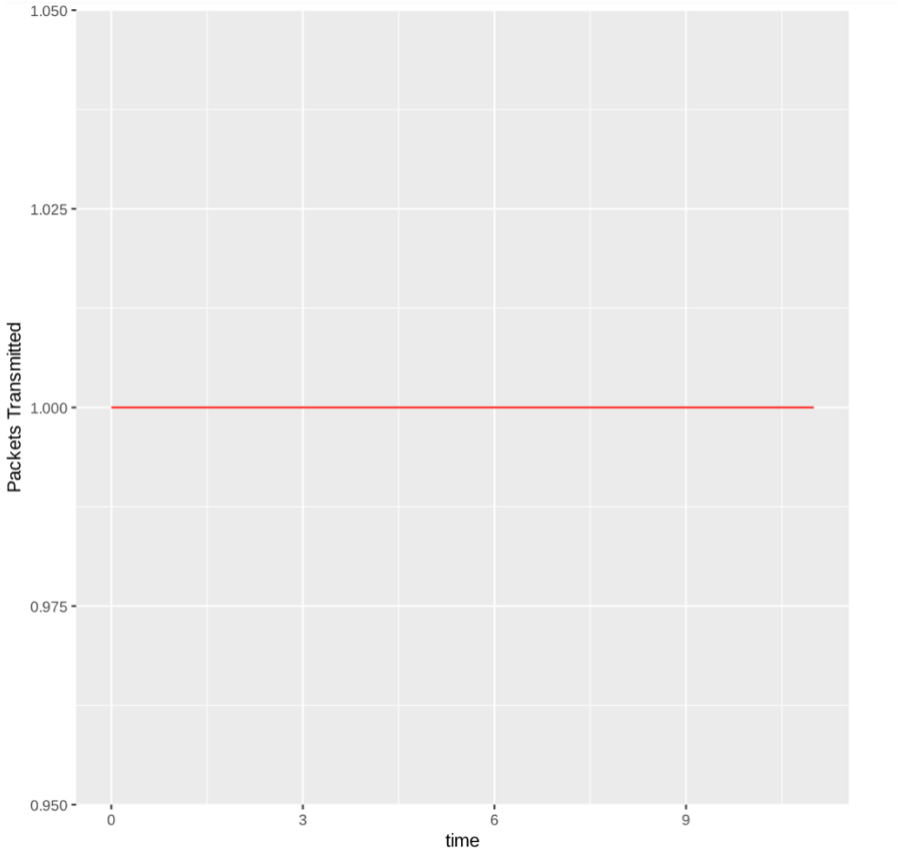
Step 4: On window to Client Terminal B, we will start the client connection to the server using the command iperf -c 10.1.1.1 -t 10 -w 4KB in the Client terminal:

```
ubuntu@Client:~$ iperf -c 10.1.1.1 -t 10 -w 4KB
------------------------------------------------------------
Client connecting to 10.1.1.1, TCP port 5001
TCP window size: 8.00 KByte (WARNING: requested 4.00 KByte)
------------------------------------------------------------
[  3] local 10.1.1.2 port 57260 connected with 10.1.1.1 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-10.0 sec   109 KBytes  89.5 Kbits/sec
ubuntu@Client:~$
```

Step 5: Once completed on Client Terminal B, we will hit Ctrl-C on Client Terminal A:

```
ubuntu@Client:~$ ./RT-output.sh
tcpdump: listening on enp7s0, link-type EN10MB (Ethernet), capture size 262144 bytes
^C141 packets captured
141 packets received by filter
0 packets dropped by kernel
reading from file packets.pcap, link-type EN10MB (Ethernet)
ubuntu@Client:~$
```

Step 6: Ran the code and then opened the '.sgv' file to look at the graph for the following combination of 4KB and 250ms:



Graph 3 (32KB and 50ms) → Step 1: Set up the intended time delay using the command that was learned from section 2.2:

```
ubuntu@Server:~$ ping 10.1.1.2 -c 10
PING 10.1.1.2 (10.1.1.2) 56(84) bytes of data.
64 bytes from 10.1.1.2: icmp_seq=1 ttl=64 time=0.144 ms
64 bytes from 10.1.1.2: icmp_seq=2 ttl=64 time=0.057 ms
64 bytes from 10.1.1.2: icmp_seq=3 ttl=64 time=0.074 ms
64 bytes from 10.1.1.2: icmp_seq=4 ttl=64 time=0.058 ms
64 bytes from 10.1.1.2: icmp_seq=5 ttl=64 time=0.057 ms
64 bytes from 10.1.1.2: icmp_seq=6 ttl=64 time=0.052 ms
64 bytes from 10.1.1.2: icmp_seq=7 ttl=64 time=0.062 ms
64 bytes from 10.1.1.2: icmp_seq=8 ttl=64 time=0.065 ms
64 bytes from 10.1.1.2: icmp_seq=9 ttl=64 time=0.062 ms
64 bytes from 10.1.1.2: icmp_seq=10 ttl=64 time=0.063 ms

--- 10.1.1.2 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9209ms
rtt min/avg/max/mdev = 0.052/0.069/0.144/0.025 ms
ubuntu@Server:~$
ubuntu@Server:~$
ubuntu@Server:~$ sudo tc qdisc add dev enp7s0 root netem delay 50ms
ubuntu@Server:~$
ubuntu@Server:~$
ubuntu@Server:~$ ping 10.1.1.2 -c 10
PING 10.1.1.2 (10.1.1.2) 56(84) bytes of data.
64 bytes from 10.1.1.2: icmp_seq=1 ttl=64 time=50.1 ms
64 bytes from 10.1.1.2: icmp_seq=2 ttl=64 time=50.1 ms
64 bytes from 10.1.1.2: icmp_seq=3 ttl=64 time=50.1 ms
64 bytes from 10.1.1.2: icmp_seq=4 ttl=64 time=50.1 ms
64 bytes from 10.1.1.2: icmp_seq=5 ttl=64 time=50.1 ms
64 bytes from 10.1.1.2: icmp_seq=6 ttl=64 time=50.1 ms
64 bytes from 10.1.1.2: icmp_seq=7 ttl=64 time=50.1 ms
64 bytes from 10.1.1.2: icmp_seq=8 ttl=64 time=50.2 ms
64 bytes from 10.1.1.2: icmp_seq=9 ttl=64 time=50.1 ms
64 bytes from 10.1.1.2: icmp_seq=10 ttl=64 time=50.1 ms

--- 10.1.1.2 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9011ms
rtt min/avg/max/mdev = 50.065/50.094/50.208/0.040 ms
ubuntu@Server:~$
```

Step 2: Start the server side iperf using the command iperf -s -w 32KB in the Server terminal:

Before

```
ubuntu@Server:~$ iperf -s -w 32KB
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 64.0 KByte (WARNING: requested 32.0 KByte)
------------------------------------------------------------
```

After

```
ubuntu@Server:~$ iperf -s -w 32KB
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 64.0 KByte (WARNING: requested 32.0 KByte)
------------------------------------------------------------
[  4] local 10.1.1.1 port 5001 connected with 10.1.1.2 port 35440
[ ID] Interval       Transfer     Bandwidth
[  4]  0.0-10.1 sec  5.88 MBytes  4.89 Mbits/sec
^Cubuntu@Server:~$
```

Step 3: On the window to the Client (Terminal A), we will start the pre-loaded script with the command ./RT-output.sh:

```
ubuntu@Client:~$ ./RT-output.sh
tcpdump: listening on enp7s0, link-type EN10MB (Ethernet), capture size 262144 bytes
```

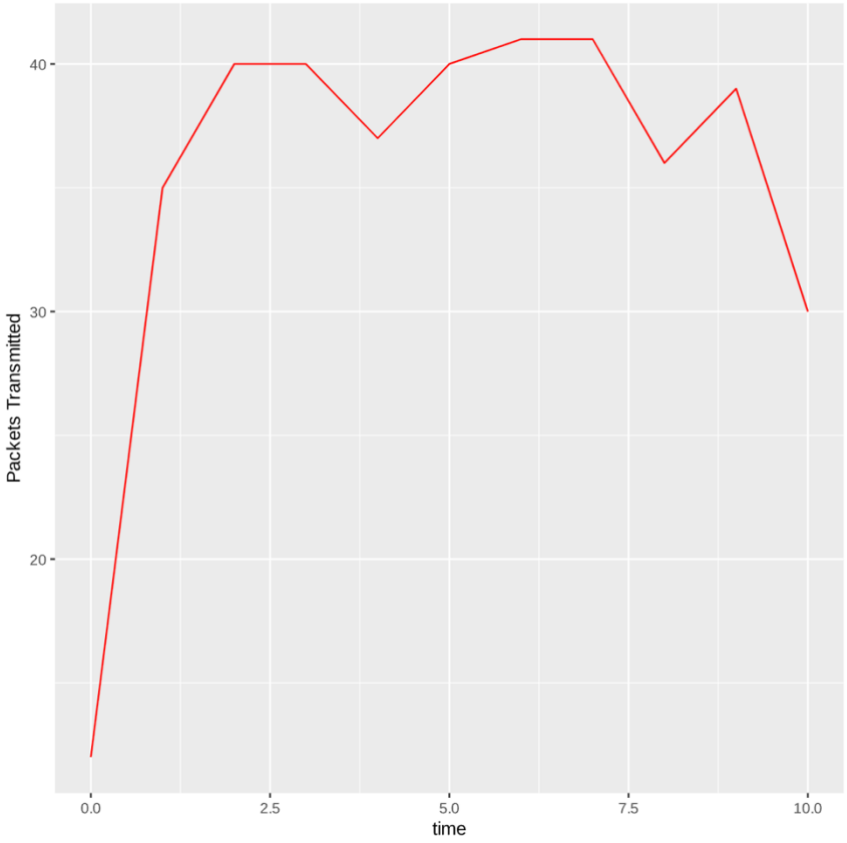
Step 4: On window to Client Terminal B, we will start the client connection to the server using the command iperf -c 10.1.1.1 -t 10 -w 32KB in the Client terminal:

```
ubuntu@Client:~$ iperf -c 10.1.1.1 -t 10 -w 32KB
------------------------------------------------------------
Client connecting to 10.1.1.1, TCP port 5001
TCP window size: 64.0 KByte (WARNING: requested 32.0 KByte)
------------------------------------------------------------
[  3] local 10.1.1.2 port 35440 connected with 10.1.1.1 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-10.1 sec  5.88 MBytes  4.89 Mbits/sec
ubuntu@Client:~$
```

Step 5: Once completed on Client Terminal B, we will hit Ctrl-C on Client Terminal A:

```
ubuntu@Client:~$ ./RT-output.sh
tcpdump: listening on enp7s0, link-type EN10MB (Ethernet), capture size 262144 bytes
^C3965 packets captured
3965 packets received by filter
0 packets dropped by kernel
reading from file packets.pcap, link-type EN10MB (Ethernet)
ubuntu@Client:~$
```

Step 6: Ran the code and then opened the '.sgv' file to look at the graph for the following combination of 32KB and 50ms:



Graph 4 (32KB and 250ms) → Step 1: Set up the intended time delay using the command that was learned from section 2.2:

```
ubuntu@Server:~$ ping 10.1.1.2 -c 10
PING 10.1.1.2 (10.1.1.2) 56(84) bytes of data.
64 bytes from 10.1.1.2: icmp_seq=1 ttl=64 time=0.107 ms
64 bytes from 10.1.1.2: icmp_seq=2 ttl=64 time=0.102 ms
64 bytes from 10.1.1.2: icmp_seq=3 ttl=64 time=0.103 ms
64 bytes from 10.1.1.2: icmp_seq=4 ttl=64 time=0.070 ms
64 bytes from 10.1.1.2: icmp_seq=5 ttl=64 time=0.069 ms
64 bytes from 10.1.1.2: icmp_seq=6 ttl=64 time=0.072 ms
64 bytes from 10.1.1.2: icmp_seq=7 ttl=64 time=0.077 ms
64 bytes from 10.1.1.2: icmp_seq=8 ttl=64 time=0.074 ms
64 bytes from 10.1.1.2: icmp_seq=9 ttl=64 time=0.078 ms
64 bytes from 10.1.1.2: icmp_seq=10 ttl=64 time=0.081 ms

--- 10.1.1.2 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9215ms
rtt min/avg/max/mdev = 0.069/0.083/0.107/0.014 ms
ubuntu@Server:~$
ubuntu@Server:~$
ubuntu@Server:~$ sudo tc qdisc add dev enp7s0 root netem delay 250ms
ubuntu@Server:~$
ubuntu@Server:~$
ubuntu@Server:~$ ping 10.1.1.2 -c 10
PING 10.1.1.2 (10.1.1.2) 56(84) bytes of data.
64 bytes from 10.1.1.2: icmp_seq=1 ttl=64 time=250 ms
64 bytes from 10.1.1.2: icmp_seq=2 ttl=64 time=250 ms
64 bytes from 10.1.1.2: icmp_seq=3 ttl=64 time=250 ms
64 bytes from 10.1.1.2: icmp_seq=4 ttl=64 time=250 ms
64 bytes from 10.1.1.2: icmp_seq=5 ttl=64 time=250 ms
64 bytes from 10.1.1.2: icmp_seq=6 ttl=64 time=250 ms
64 bytes from 10.1.1.2: icmp_seq=7 ttl=64 time=250 ms
64 bytes from 10.1.1.2: icmp_seq=8 ttl=64 time=250 ms
64 bytes from 10.1.1.2: icmp_seq=9 ttl=64 time=250 ms
64 bytes from 10.1.1.2: icmp_seq=10 ttl=64 time=250 ms

--- 10.1.1.2 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9001ms
rtt min/avg/max/mdev = 250.062/250.111/250.166/0.032 ms
ubuntu@Server:~$
```

Step 2: Start the server side iperf using the command iperf -s -w 32KB in the Server terminal:

Before

```
ubuntu@Server:~$ iperf -s -w 32KB
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 64.0 KByte (WARNING: requested 32.0 KByte)
------------------------------------------------------------
```

After

```
ubuntu@Server:~$ iperf -s -w 32KB
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 64.0 KByte (WARNING: requested 32.0 KByte)
------------------------------------------------------------
[  4] local 10.1.1.1 port 5001 connected with 10.1.1.2 port 33706
[ ID] Interval       Transfer     Bandwidth
[  4]  0.0-10.8 sec  1.25 MBytes   971 Kbits/sec
^Cubuntu@Server:~$
```

Step 3: On the window to the Client (Terminal A), we will start the pre-loaded script with the command ./RT-output.sh:

```
ubuntu@Client:~$ ./RT-output.sh
tcpdump: listening on enp7s0, link-type EN10MB (Ethernet), capture size 262144 bytes
```

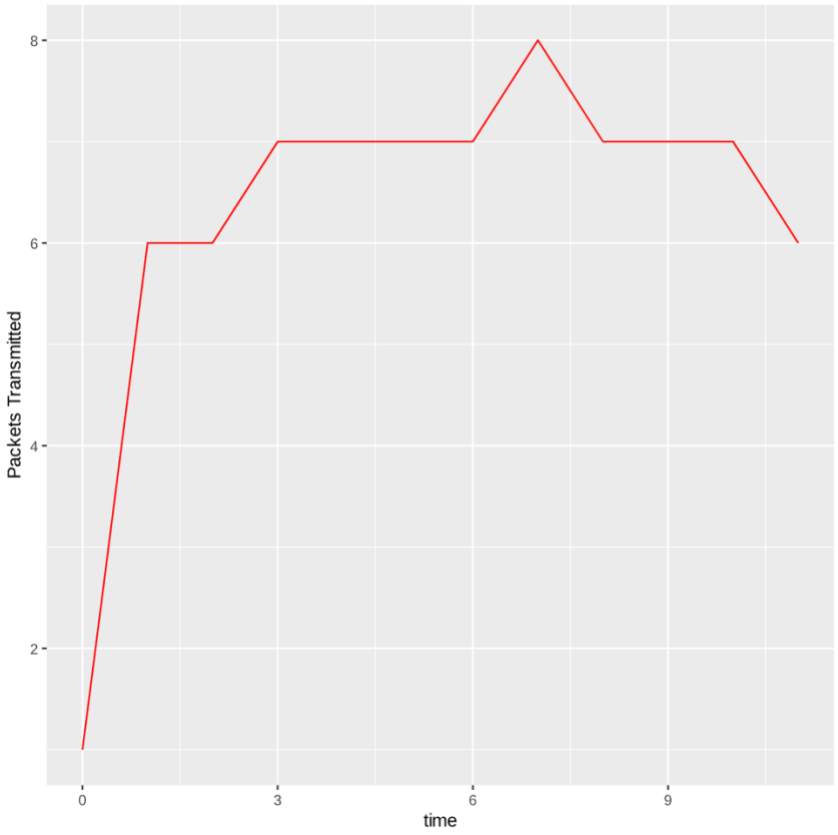
Step 4: On window to Client Terminal B, we will start the client connection to the server using the command iperf -c 10.1.1.1 -t 10 -w 32KB in the Client terminal:

```
ubuntu@Client:~$ iperf -c 10.1.1.1 -t 10 -w 32KB
------------------------------------------------------------
Client connecting to 10.1.1.1, TCP port 5001
TCP window size: 64.0 KByte (WARNING: requested 32.0 KByte)
------------------------------------------------------------
[  3] local 10.1.1.2 port 33706 connected with 10.1.1.1 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-10.8 sec  1.25 MBytes   971 Kbits/sec
ubuntu@Client:~$
```

Step 5: Once completed on Client Terminal B, we will hit Ctrl-C on Client Terminal A:

```
ubuntu@Client:~$ ./RT-output.sh
tcpdump: listening on enp7s0, link-type EN10MB (Ethernet), capture size 262144 bytes
^C806 packets captured
806 packets received by filter
0 packets dropped by kernel
reading from file packets.pcap, link-type EN10MB (Ethernet)
ubuntu@Client:~$
```

Step 6: Ran the code and then opened the '.sgv' file to look at the graph for the following combination of 32KB and 250ms:



4. Modifying UDPEchoServer.c and UDPEchoClient-Timeout.c code:

In the Server terminal, we will be downloading the programs UDPEchoServer.c and DieWithError.c by running:
wget –no-check-certificate
https://cs.baylor.edu/~donahoo/practical/CSockets/code/UDPEchoServer.c
and
wget –no-check-certificate
https://cs.baylor.edu/~donahoo/practical/CSockets/code/DieWithError.c

```
ubuntu@Server:~$ wget --no-check-certificate https://cs.baylor.edu/~donahoo/practical/C
Sockets/code/UDPEchoServer.c
--2024-04-11 03:06:27--  https://cs.baylor.edu/~donahoo/practical/CSockets/code/UDPEcho
Server.c
Resolving cs.baylor.edu (cs.baylor.edu)... 2600:2701:5000:5001::813e:940a, 129.62.148.1
0
Connecting to cs.baylor.edu (cs.baylor.edu)|2600:2701:5000:5001::813e:940a|:443... conn
ected.
WARNING: cannot verify cs.baylor.edu's certificate, issued by 'CN=InCommon RSA Server C
A 2,O=Internet2,C=US':
  Unable to locally verify the issuer's authority.
HTTP request sent, awaiting response... 200 OK
Length: 2574 (2.5K) [text/plain]
Saving to: 'UDPEchoServer.c'

UDPEchoServer.c       100%[=======================>]   2.51K  --.-KB/s    in 0s

2024-04-11 03:06:28 (88.6 MB/s) - 'UDPEchoServer.c' saved [2574/2574]

ubuntu@Server:~$
ubuntu@Server:~$
ubuntu@Server:~$ wget --no-check-certificate https://cs.baylor.edu/~donahoo/practical/C
Sockets/code/DieWithError.c
--2024-04-11 03:07:00--  https://cs.baylor.edu/~donahoo/practical/CSockets/code/DieWith
Error.c
Resolving cs.baylor.edu (cs.baylor.edu)... 2600:2701:5000:5001::813e:940a, 129.62.148.1
0
Connecting to cs.baylor.edu (cs.baylor.edu)|2600:2701:5000:5001::813e:940a|:443... conn
ected.
WARNING: cannot verify cs.baylor.edu's certificate, issued by 'CN=InCommon RSA Server C
A 2,O=Internet2,C=US':
  Unable to locally verify the issuer's authority.
HTTP request sent, awaiting response... 200 OK
Length: 158 [text/plain]
Saving to: 'DieWithError.c'

DieWithError.c        100%[=======================>]     158  --.-KB/s    in 0s

2024-04-11 03:07:01 (23.6 MB/s) - 'DieWithError.c' saved [158/158]

ubuntu@Server:~$
```

Now making the changes to the code so that it will print out the message received from the client. Make sure to clean up the buffer every time; otherwise, the previous content will stay in the buffer, and we will be only overwriting the first part of the buffer:

Modification to the UDPEchoServer.c file:

```c
#include <stdio.h>      /* for printf() and fprintf() */
#include <sys/socket.h> /* for socket() and bind() */
#include <arpa/inet.h>  /* for sockaddr_in and inet_ntoa() */
#include <stdlib.h>     /* for atoi() and exit() */
#include <string.h>     /* for memset() */
#include <unistd.h>     /* for close() */

#define ECHOMAX 255     /* Longest string to echo */

void DieWithError(char *errorMessage);  /* External error handling function */

int main(int argc, char *argv[])
{
    int sock;                        /* Socket */
    struct sockaddr_in echoServAddr; /* Local address */
    struct sockaddr_in echoCltntAddr; /* Client address */
    unsigned int cliAddrLen;         /* Length of incoming message */
    char echoBuffer[ECHOMAX];        /* Buffer for echo string */
    unsigned short echoServPort;     /* Server port */
    int recvMsgSize;                 /* Size of received message */

    if (argc != 2)         /* Test for correct number of parameters */
    {
        fprintf(stderr,"Usage:  %s <UDP SERVER PORT>\n", argv[0]);
        exit(1);
    }

    echoServPort = atoi(argv[1]);  /* First arg:  local port */

    /* Create socket for sending/receiving datagrams */
    if ((sock = socket(PF_INET, SOCK_DGRAM, IPPROTO_UDP)) < 0)
        DieWithError("socket() failed");

    /* Construct local address structure */
    memset(&echoServAddr, 0, sizeof(echoServAddr));   /* Zero out structure */
    echoServAddr.sin_family = AF_INET;                /* Internet address family */
    echoServAddr.sin_addr.s_addr = htonl(INADDR_ANY); /* Any incoming interface */
    echoServAddr.sin_port = htons(echoServPort);      /* Local port */

    /* Bind to the local address */
    if (bind(sock, (struct sockaddr *) &echoServAddr, sizeof(echoServAddr)) < 0)
        DieWithError("bind() failed");

    for (;;) /* Run forever */
    {
        /* Set the size of the in-out parameter */
        cliAddrLen = sizeof(echoCltntAddr);

        /* Block until receive message from a client */
        if ((recvMsgSize = recvfrom(sock, echoBuffer, ECHOMAX, 0,
            (struct sockaddr *) &echoCltntAddr, &cliAddrLen)) < 0)
            DieWithError("recvfrom() failed");

        printf("Handling client %s\n", inet_ntoa(echoCltntAddr.sin_addr));
        printf("The recieved message:  %s\n", echoBuffer); // This is to print the received message from the client

        /* Send received datagram back to the client */
        if (sendto(sock, echoBuffer, recvMsgSize, 0,
             (struct sockaddr *) &echoCltntAddr, sizeof(echoCltntAddr)) != recvMsgSize)
            DieWithError("sendto() sent a different number of bytes than expected");

        memset(echoBuffer, 0, ECHOMAX); // This is to make sure to clean up the buffer after sending the message back to the client
    }
    /* NOT REACHED */
}
```

In the Client terminal, we will be downloading the programs UDPEchoClient-Timeout.c and DieWithError.c by running:
wget --no-check-certificate https://cs.baylor.edu/~donahoo/practical/CSockets/code/UDPEchoClient-Timeout.c

and

wget --no-check-certificate
https://cs.baylor.edu/~donahoo/practical/CSockets/code/DieWithError.c

```
ubuntu@Client:~$ wget --no-check-certificate https://cs.baylor.edu/~donahoo/practical/CSockets/code/UDPEchoClient-Timeout.c
--2024-04-11 03:59:16--  https://cs.baylor.edu/~donahoo/practical/CSockets/code/UDPEchoClient-Timeout.c
Resolving cs.baylor.edu (cs.baylor.edu)... 2600:2701:5000:5001::813e:940a, 129.62.148.10
Connecting to cs.baylor.edu (cs.baylor.edu)|2600:2701:5000:5001::813e:940a|:443... connected.
WARNING: cannot verify cs.baylor.edu's certificate, issued by 'CN=InCommon RSA Server CA 2,O=Internet2,C=US':
  Unable to locally verify the issuer's authority.
HTTP request sent, awaiting response... 200 OK
Length: 4545 (4.4K) [text/plain]
Saving to: 'UDPEchoClient-Timeout.c'

UDPEchoClient-Timeout.c          100%[=====================================================>]   4.44K  --.-KB/s    in 0.001s

2024-04-11 03:59:17 (3.78 MB/s) - 'UDPEchoClient-Timeout.c' saved [4545/4545]

ubuntu@Client:~$
ubuntu@Client:~$
ubuntu@Client:~$ wget --no-check-certificate https://cs.baylor.edu/~donahoo/practical/CSockets/code/DieWithError.c
--2024-04-11 03:59:57--  https://cs.baylor.edu/~donahoo/practical/CSockets/code/DieWithError.c
Resolving cs.baylor.edu (cs.baylor.edu)... 2600:2701:5000:5001::813e:940a, 129.62.148.10
Connecting to cs.baylor.edu (cs.baylor.edu)|2600:2701:5000:5001::813e:940a|:443... connected.
WARNING: cannot verify cs.baylor.edu's certificate, issued by 'CN=InCommon RSA Server CA 2,O=Internet2,C=US':
  Unable to locally verify the issuer's authority.
HTTP request sent, awaiting response... 200 OK
Length: 158 [text/plain]
Saving to: 'DieWithError.c'

DieWithError.c                   100%[=====================================================>]     158  --.-KB/s    in 0s

2024-04-11 03:59:57 (49.8 MB/s) - 'DieWithError.c' saved [158/158]

ubuntu@Client:~$
```

Now making the changes to the code so that 1) the client always explicitly specifies the port number of the server this client will connect to as an argument and 2) the default timeout will be 2 seconds if we don't add an extra argument when running the program, but we can add an extra argument that specifies the timeout value at the client side.

```c
#include <stdio.h>      /* for printf() and fprintf() */
#include <sys/socket.h> /* for socket(), connect(), sendto(), and recvfrom() */
#include <arpa/inet.h>  /* for sockaddr_in and inet_addr() */
#include <stdlib.h>     /* for atoi() and exit() */
#include <string.h>     /* for memset() */
#include <unistd.h>     /* for close() and alarm() */
#include <errno.h>      /* for errno and EINTR */
#include <signal.h>     /* for sigaction() */

#define ECHOMAX         255     /* Longest string to echo */
// #define TIMEOUT_SECS    2       /* Seconds between retransmits */
#define MAXTRIES        5       /* Tries before giving up */

int tries=0;   /* Count of times sent - GLOBAL for signal-handler access */

void DieWithError(char *errorMessage);   /* Error handling function */
void CatchAlarm(int ignored);            /* Handler for SIGALRM */

int main(int argc, char *argv[])
{
    int sock;                        /* Socket descriptor */
    struct sockaddr_in echoServAddr; /* Echo server address */
    struct sockaddr_in fromAddr;     /* Source address of echo */
    unsigned short echoServPort;     /* Echo server port */
    unsigned int fromSize;           /* In-out of address size for recvfrom() */
    struct sigaction myAction;       /* For setting signal handler */
    char *servIP;                    /* IP address of server */
    char *echoString;                /* String to send to echo server */
    char echoBuffer[ECHOMAX+1];      /* Buffer for echo string */
    int echoStringLen;               /* Length of string to echo */
    int respStringLen;               /* Size of received datagram */
    int Timeout = 2;                 /* Timeout in seconds, default value */  // Modified

    if ((argc < 4) || (argc > 5))    /* Test for correct number of arguments */
    {
        fprintf(stderr,"Usage: %s <Server IP> <Echo Word> [<Echo Port>]\n", argv[0]);
        exit(1);
    }

    servIP = argv[1];           /* First arg:  server IP address (dotted quad) */
    echoString = argv[2];       /* Second arg: string to echo */
    echoServPort = atoi(argv[3]);  // Third arg: The use of the given port (server port)

    if ((echoStringLen = strlen(echoString)) > ECHOMAX)
        DieWithError("Echo word too long");

    if (argc == 5)  // Modified
            Timeout = atoi(argv[4]);  /* Use given port, if any */  // Fourth arg: Updated the timeout value

    /* Create a best-effort datagram socket using UDP */
    if ((sock = socket(PF_INET, SOCK_DGRAM, IPPROTO_UDP)) < 0)
        DieWithError("socket() failed");

    /* Set signal handler for alarm signal */
    myAction.sa_handler = CatchAlarm;
    if (sigfillset(&myAction.sa_mask) < 0) /* block everything in handler */
        DieWithError("sigfillset() failed");
    myAction.sa_flags = 0;

    if (sigaction(SIGALRM, &myAction, 0) < 0)
        DieWithError("sigaction() failed for SIGALRM");

    /* Construct the server address structure */
    memset(&echoServAddr, 0, sizeof(echoServAddr));    /* Zero out structure */
    echoServAddr.sin_family = AF_INET;
    echoServAddr.sin_addr.s_addr = inet_addr(servIP);  /* Server IP address */
    echoServAddr.sin_port = htons(echoServPort);       /* Server port */

    /* Send the string to the server */
    if (sendto(sock, echoString, echoStringLen, 0, (struct sockaddr *)
               &echoServAddr, sizeof(echoServAddr)) != echoStringLen)
        DieWithError("sendto() sent a different number of bytes than expected");

    /* Get a response */

    fromSize = sizeof(fromAddr);
    alarm(Timeout);        /* Set the timeout */  // Modified the setting of the timeout
    while ((respStringLen = recvfrom(sock, echoBuffer, ECHOMAX, 0,
           (struct sockaddr *) &fromAddr, &fromSize)) < 0)
        if (errno == EINTR)     /* Alarm went off  */
        {
            if (tries < MAXTRIES)      /* incremented by signal handler */
            {
                printf("timed out, %d more tries...\n", MAXTRIES-tries);
                if (sendto(sock, echoString, echoStringLen, 0, (struct sockaddr *)
                           &echoServAddr, sizeof(echoServAddr)) != echoStringLen)
                    DieWithError("sendto() failed");
                alarm(Timeout); // Modified
            }
            else
                DieWithError("No Response");
        }
        else
            DieWithError("recvfrom() failed");

    /* recvfrom() got something --  cancel the timeout */
    alarm(0);

    /* null-terminate the received data */
    echoBuffer[respStringLen] = '\0';
    printf("Received: %s\n", echoBuffer);    /* Print the received data */

    close(sock);
    exit(0);
}

void CatchAlarm(int ignored)     /* Handler for SIGALRM */
{
    tries += 1;
}
```

5. Screenshots of what I did and the corresponding output in the Client and Server terminal when running UDPEchoServer and UDPEchoClient-Timeout:

Compiling the code by running gcc:

```
ubuntu@Server:~$
ubuntu@Server:~$ gcc UDPEchoServer.c DieWithError.c -o UDPEchoServer
ubuntu@Server:~$
ubuntu@Server:~$
```

Compiling the code by running gcc:

```
ubuntu@Client:~$
ubuntu@Client:~$ gcc UDPEchoClient-Timeout.c DieWithError.c -o UDPEchoClient-Timeout
ubuntu@Client:~$
ubuntu@Client:~$
```

Step 1:

Ran the Client code in the Client terminal with the server port number equal to 8000, and this is what happened:

```
ubuntu@Client:~$ ./UDPEchoClient-Timeout 10.1.1.1 "Hello, again!" 8000
timed out, 4 more tries...
timed out, 3 more tries...
timed out, 2 more tries...
timed out, 1 more tries...
No Response: Interrupted system call
ubuntu@Client:~$
```

Step 2:

I ran the Server code in the Server terminal on port number 8000:

```
ubuntu@Server:~$ ./UDPEchoServer 8000
```

This shows that the received message “Hello, again!” hasn’t been received.

Step 3:

Then I repeated the step of running the Client code in the Client terminal with the server port number equal to 8000:

```
ubuntu@Client:~$ ./UDPEchoClient-Timeout 10.1.1.1 "Hello, again!" 8000
Received: Hello, again!
ubuntu@Client:~$
```

```
ubuntu@Server:~$ ./UDPEchoServer 8000
Handling client 10.1.1.2
The recieved message:  Hello, again!
```

This shows that the received message “Hello, again!” has been received.

Step 4:
Then I ran the Client code in the client window with the server port number equal to 9000, and this is what happened.

```
ubuntu@Client:~$ ./UDPEchoClient-Timeout 10.1.1.1 "Hello, again!" 9000
timed out, 4 more tries...
timed out, 3 more tries...
timed out, 2 more tries...
timed out, 1 more tries...
No Response: Interrupted system call
ubuntu@Client:~$
```

```
ubuntu@Server:~$ ./UDPEchoServer 8000
Handling client 10.1.1.2
The recieved message:  Hello, again!
```

This shows that the received message "Hello, again!" hasn't been received.

6. Then, lastly, I cleaned up the resource.