

---

# UM-SJTU JOINT INSTITUTE

COMPUTER NETWORKS

(VE489)

---

## TERM PROJECT REPORT

Name: Sun Yiwen      ID: 517370910213  
Date: August 1 2020

# 1 Mininet and Socket Programming

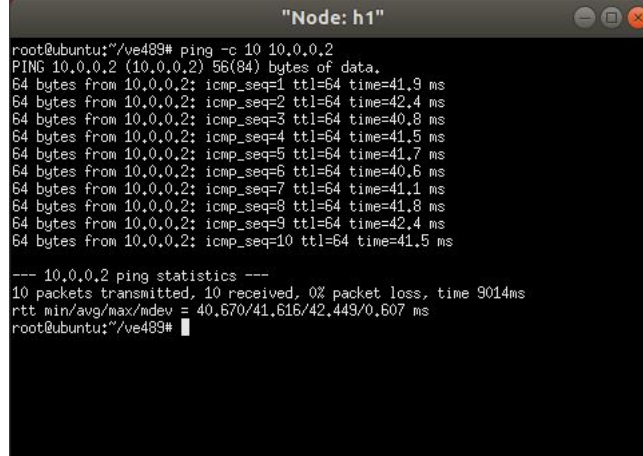
## 1.1 Simple Experiments

### 1. Link latency using ping.

The average round-trip time (RTT) between ( $h_1$  and  $h_2$ ) is

$$\frac{41.9 + 42.4 + 40.8 + 41.5 + 41.7 + 40.6 + 41.1 + 41.8 + 42.4 + 41.5}{10} = 41.57ms.$$

Link  $L_1$  is used and the link latency of  $L_1$  is  $20ms$ . The round-trip time is close to 2 times link latency.



```
"Node: h1"
root@ubuntu:~/ve489# ping -c 10 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data:
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=41.9 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=42.4 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=40.8 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=41.5 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=41.7 ms
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=40.6 ms
64 bytes from 10.0.0.2: icmp_seq=7 ttl=64 time=41.1 ms
64 bytes from 10.0.0.2: icmp_seq=8 ttl=64 time=41.8 ms
64 bytes from 10.0.0.2: icmp_seq=9 ttl=64 time=42.4 ms
64 bytes from 10.0.0.2: icmp_seq=10 ttl=64 time=41.5 ms

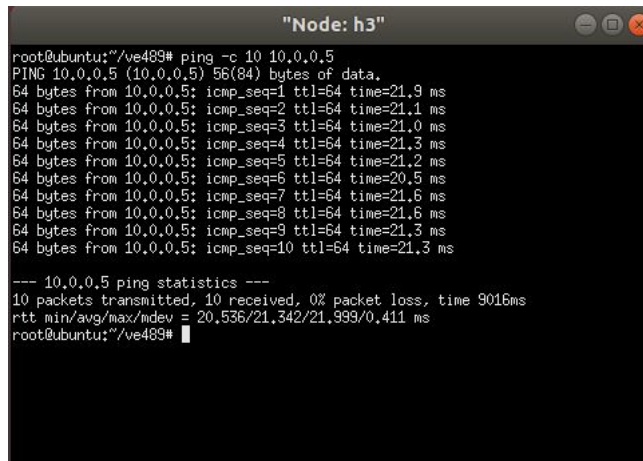
--- 10.0.0.2 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9014ms
rtt min/avg/max/mdev = 40.670/41.616/42.449/0.607 ms
root@ubuntu:~/ve489#
```

Figure 1: Screenshot of pinging 10 times from  $h_1$  to  $h_2$ .

The average round-trip time (RTT) between ( $h_3$  and  $h_5$ ) is

$$\frac{21.9 + 21.1 + 21.0 + 21.3 + 21.2 + 20.5 + 21.6 + 21.6 + 21.3 + 21.3}{10} = 21.28ms.$$

Link  $L_4$  is used and the link latency of  $L_4$  is  $10ms$ . The round-trip time is close to 2 times link latency.



```
"Node: h3"
root@ubuntu:~/ve489# ping -c 10 10.0.0.5
PING 10.0.0.5 (10.0.0.5) 56(84) bytes of data:
64 bytes from 10.0.0.5: icmp_seq=1 ttl=64 time=21.9 ms
64 bytes from 10.0.0.5: icmp_seq=2 ttl=64 time=21.1 ms
64 bytes from 10.0.0.5: icmp_seq=3 ttl=64 time=21.0 ms
64 bytes from 10.0.0.5: icmp_seq=4 ttl=64 time=21.3 ms
64 bytes from 10.0.0.5: icmp_seq=5 ttl=64 time=21.2 ms
64 bytes from 10.0.0.5: icmp_seq=6 ttl=64 time=20.5 ms
64 bytes from 10.0.0.5: icmp_seq=7 ttl=64 time=21.6 ms
64 bytes from 10.0.0.5: icmp_seq=8 ttl=64 time=21.6 ms
64 bytes from 10.0.0.5: icmp_seq=9 ttl=64 time=21.3 ms
64 bytes from 10.0.0.5: icmp_seq=10 ttl=64 time=21.3 ms

--- 10.0.0.5 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9016ms
rtt min/avg/max/mdev = 20.536/21.342/21.939/0.411 ms
root@ubuntu:~/ve489#
```

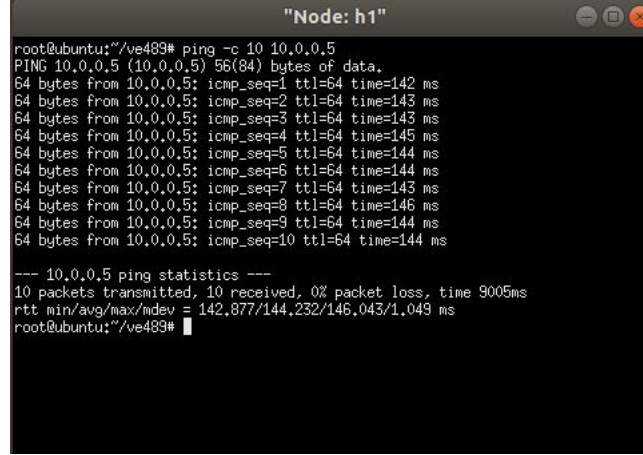
Figure 2: Screenshot of pinging 10 times from  $h_3$  to  $h_5$ .

## 2. Path latency using ping.

The average round-trip time (RTT) between ( $h_1$  and  $h_5$ ) is

$$\frac{142 + 143 + 143 + 145 + 144 + 144 + 143 + 146 + 144 + 144}{10} = 143.8ms.$$

Link  $L_1$ ,  $L_2$  and  $L_4$  are used and according to the given parameters, the theoretical delay between  $h_1$  and  $h_5$  is  $(20 + 40 + 10) \times 2 = 140ms$ . The round-trip time is close to this theoretical delay.



```

"Node: h1"
root@ubuntu:~/ve489# ping -c 10 10.0.0.5
PING 10.0.0.5 (10.0.0.5) 56(84) bytes of data:
64 bytes from 10.0.0.5: icmp_seq=1 ttl=64 time=142 ms
64 bytes from 10.0.0.5: icmp_seq=2 ttl=64 time=143 ms
64 bytes from 10.0.0.5: icmp_seq=3 ttl=64 time=143 ms
64 bytes from 10.0.0.5: icmp_seq=4 ttl=64 time=145 ms
64 bytes from 10.0.0.5: icmp_seq=5 ttl=64 time=144 ms
64 bytes from 10.0.0.5: icmp_seq=6 ttl=64 time=144 ms
64 bytes from 10.0.0.5: icmp_seq=7 ttl=64 time=143 ms
64 bytes from 10.0.0.5: icmp_seq=8 ttl=64 time=146 ms
64 bytes from 10.0.0.5: icmp_seq=9 ttl=64 time=144 ms
64 bytes from 10.0.0.5: icmp_seq=10 ttl=64 time=144 ms

--- 10.0.0.5 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9005ms
rtt min/avg/max/mdev = 142.877/144.232/146.043/1.049 ms
root@ubuntu:~/ve489#

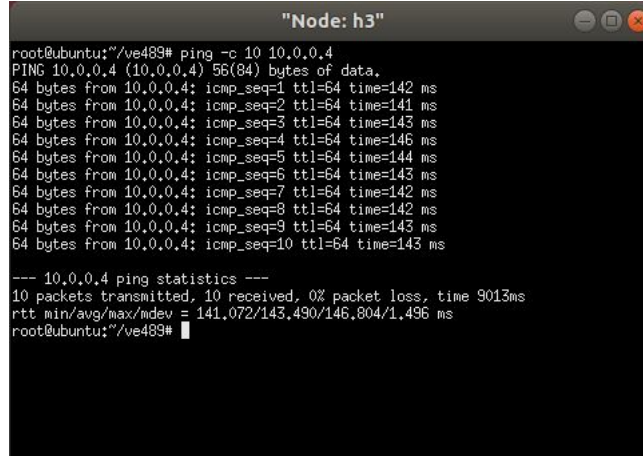
```

Figure 3: Screenshot of pinging 10 times from  $h_1$  to  $h_5$ .

The average round-trip time (RTT) between ( $h_3$  and  $h_4$ ) is

$$\frac{142 + 141 + 143 + 146 + 144 + 143 + 142 + 142 + 143 + 143}{10} = 142.9ms.$$

Link  $L_2$  and  $L_3$  are used and according to the given parameters, the theoretical delay between  $h_3$  and  $h_4$  is  $(40 + 30) \times 2 = 140ms$ . The round-trip time is close to this theoretical delay.



```

"Node: h3"
root@ubuntu:~/ve489# ping -c 10 10.0.0.4
PING 10.0.0.4 (10.0.0.4) 56(84) bytes of data:
64 bytes from 10.0.0.4: icmp_seq=1 ttl=64 time=142 ms
64 bytes from 10.0.0.4: icmp_seq=2 ttl=64 time=141 ms
64 bytes from 10.0.0.4: icmp_seq=3 ttl=64 time=143 ms
64 bytes from 10.0.0.4: icmp_seq=4 ttl=64 time=146 ms
64 bytes from 10.0.0.4: icmp_seq=5 ttl=64 time=144 ms
64 bytes from 10.0.0.4: icmp_seq=6 ttl=64 time=143 ms
64 bytes from 10.0.0.4: icmp_seq=7 ttl=64 time=142 ms
64 bytes from 10.0.0.4: icmp_seq=8 ttl=64 time=142 ms
64 bytes from 10.0.0.4: icmp_seq=9 ttl=64 time=143 ms
64 bytes from 10.0.0.4: icmp_seq=10 ttl=64 time=143 ms

--- 10.0.0.4 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9013ms
rtt min/avg/max/mdev = 141.072/143.490/146.804/1.496 ms
root@ubuntu:~/ve489#

```

Figure 4: Screenshot of pinging 10 times from  $h_3$  to  $h_4$ .

## 3. Link bandwidth using iperf.

The bandwidth result on the server  $h_1$  is  $40.8Mbps$ . The bandwidth result on the client  $h_2$  is  $48.0Mbps$ . They are both close to the link  $L_1$ 's bandwidth  $50Mbps$ . 59.2 MBytes of data are transferred and received.

```

"Node: h1"
root@ubuntu:~/ve489# iperf -s -p 1
-----
Server listening on TCP port 1
TCP window size: 85,3 KByte (default)
-----
[ 28] local 10.0.0.1 port 1 connected with 10.0.0.2 port 44358
[ ID] Interval      Transfer    Bandwidth
[ 28] 0.0-12.2 sec  59,2 MBytes 40,8 Mbits/sec
root@ubuntu:~/ve489#

"Node: h2"
root@ubuntu:~/ve489# iperf -c 10.0.0.1 -p 1 -t 10s
-----
Client connecting to 10.0.0.1, TCP port 1
TCP window size: 85,3 KByte (default)
-----
[ 27] local 10.0.0.2 port 44358 connected with 10.0.0.1 port 1
[ ID] Interval      Transfer    Bandwidth
[ 27] 0.0-10.3 sec  59,2 MBytes 48,0 Mbits/sec
root@ubuntu:~/ve489#

```

Figure 5: Screenshot of link bandwidth between  $h_1$  and  $h_2$ .

The bandwidth result on the server  $h_3$  is  $9.31Mbps$ . The bandwidth result on the client  $h_5$  is  $11.1Mbps$ . They are both close to the link  $L_4$ 's bandwidth  $10Mbps$ . 13.9 MBytes of data are transferred and received.

```

"Node: h3"
root@ubuntu:~/ve489# iperf -s -p 1
-----
Server listening on TCP port 1
TCP window size: 85,3 KByte (default)
-----
[ 28] local 10.0.0.3 port 1 connected with 10.0.0.5 port 45190
[ ID] Interval      Transfer    Bandwidth
[ 28] 0.0-12,5 sec  13,9 MBytes 9,31 Mbits/sec
root@ubuntu:~/ve489#

"Node: h5"
root@ubuntu:~/ve489# iperf -c 10.0.0.3 -p 1 -t 10s
-----
Client connecting to 10.0.0.3, TCP port 1
TCP window size: 85,3 KByte (default)
-----
[ 27] local 10.0.0.5 port 45190 connected with 10.0.0.3 port 1
[ ID] Interval      Transfer    Bandwidth
[ 27] 0.0-10,5 sec  13,9 MBytes 11,1 Mbits/sec
root@ubuntu:~/ve489#

```

Figure 6: Screenshot of link bandwidth between  $h_3$  and  $h_5$ .

#### 4. Path throughput using iperf.

The bandwidth result on the server  $h_1$  is  $8.82Mbps$ . The bandwidth result on the client  $h_5$  is  $11.3Mbps$ . Link  $L_4$  is the bottleneck link in this path and the result is close to  $L_4$ 's bandwidth  $10Mbps$ .

```

"Node: h1"
root@ubuntu:~/ve489# iperf -s -p 1
-----
Server listening on TCP port 1
TCP window size: 85,3 KByte (default)
-----
[ 28] local 10.0.0.1 port 1 connected with 10.0.0.5 port 51056
[ ID] Interval      Transfer    Bandwidth
[ 28] 0.0-14,6 sec  15,4 MBytes 8,82 Mbits/sec
root@ubuntu:~/ve489#

"Node: h5"
root@ubuntu:~/ve489# iperf -c 10.0.0.1 -p 1 -t 10s
-----
Client connecting to 10.0.0.1, TCP port 1
TCP window size: 85,3 KByte (default)
-----
[ 27] local 10.0.0.5 port 51056 connected with 10.0.0.1 port 1
[ ID] Interval      Transfer    Bandwidth
[ 27] 0.0-11,4 sec  15,4 MBytes 11,3 Mbits/sec
root@ubuntu:~/ve489#

```

Figure 7: Screenshot of throughput between  $h_1$  and  $h_5$ .

The bandwidth result on the server  $h_3$  is  $17.6Mbps$ . The bandwidth result on the client  $h_4$  is  $21.9Mbps$ . Link  $L_2$  is the bottleneck link in this path and the result is close to  $L_2$ 's bandwidth  $20Mbps$ .

#### 5. Multiplexing.

As shown in Figure 9, the latency and bandwidth is very similar to the previous results in question 2 and 4. Link  $L_2$  is shared. And since the results doesn't change much from when they are transferring alone, they are sharing the link bandwidth fairly.

## 2 Socket Programming

### 2.1 TCP File Transfer Server

As shown in Figure 10, 11, 12, 13, these are the packets captured at the client. Client is at 10.0.0.2:8002 and server is at 10.0.0.1:8001.

```

"Node: h3"
root@ubuntu:~/ve489# iperf -s -p 1
Server listening on TCP port 1
TCP window size: 85.3 KByte (default)

[ 28] local 10.0.0.3 port 1 connected with 10.0.0.4 port 59854
[ ID] Interval      Transfer    Bandwidth
[ 28] 0.0-13.0 sec  27.2 MBytes 17.6 Mbits/sec

"Node: h4"
root@ubuntu:~/ve489# iperf -c 10.0.0.3 -p 1 -t 10s
Client connecting to 10.0.0.3, TCP port 1
TCP window size: 85.3 KByte (default)

[ 27] local 10.0.0.4 port 59854 connected with 10.0.0.3 port 1
[ ID] Interval      Transfer    Bandwidth
[ 27] 0.0-10.4 sec  27.2 MBytes 21.9 Mbits/sec
root@ubuntu:~/ve489#

```

Figure 8: Screenshot of throughput between  $h_3$  and  $h_4$ .

```

"Node: h1"
root@ubuntu:~/ve489# iperf -s -p 1
Server listening on TCP port 1
TCP window size: 85.3 KByte (default)

[ 28] local 10.0.0.1 port 1 connected with 10.0.0.5 port 51072
[ ID] Interval      Transfer    Bandwidth
[ 28] 0.0-13.5 sec  14.6 MBytes 9.12 Mbits/sec

"Node: h5"
root@ubuntu:~/ve489# iperf -c 10.0.0.1 -p 1 -t 10s
Client connecting to 10.0.0.1, TCP port 1
TCP window size: 85.3 KByte (default)

[ 27] local 10.0.0.5 port 51072 connected with 10.0.0.1 port 1
[ ID] Interval      Transfer    Bandwidth
[ 27] 0.0-10.7 sec  14.6 MBytes 11.4 Mbits/sec
root@ubuntu:~/ve489#

"Node: h3"
root@ubuntu:~/ve489# iperf -s -p 1
Server listening on TCP port 1
TCP window size: 85.3 KByte (default)

[ 28] local 10.0.0.3 port 1 connected with 10.0.0.4 port 59864
[ ID] Interval      Transfer    Bandwidth
[ 28] 0.0-14.1 sec  29.5 MBytes 17.5 Mbits/sec

"Node: h4"
root@ubuntu:~/ve489# iperf -c 10.0.0.3 -p 1 -t 10s
Client connecting to 10.0.0.3, TCP port 1
TCP window size: 85.3 KByte (default)

[ 27] local 10.0.0.4 port 59864 connected with 10.0.0.3 port 1
[ ID] Interval      Transfer    Bandwidth
[ 27] 0.0-11.2 sec  29.5 MBytes 22.1 Mbits/sec
root@ubuntu:~/ve489#

```

Figure 9: Screenshot of  $(h_1, h_5)$  and  $(h_3, h_4)$  sharing the same link.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	10.0.0.2	10.0.0.1	TCP	74	teradataorlbs(8002) → vcom-tunnel(8001) [SYN, ACK] Seq=0 Win=42340 Len=0 MSS=1460 SACK_PERM=1 TSval=3953059179 TSecr=0 WS=512
2	0.000027421	10.0.0.1	10.0.0.2	TCP	74	vcom-tunnel(8001) → teradataorlbs(8002) [SYN, ACK] Seq=0 Ack=1 Win=42440 Len=0 MSS=1460 SACK_PERM=1 TSval=3956643898 TSecr=...
3	0.045305059	10.0.0.2	10.0.0.1	TCP	66	teradataorlbs(8002) → vcom-tunnel(8001) [ACK] Seq=1 Ack=1 Win=42496 Len=0 TSval=3953059233 TSecr=3956643898
4	0.324489826	10.0.0.2	10.0.0.1	TCP	322	teradataorlbs(8002) → vcom-tunnel(8001) [PSH, ACK] Seq=1 Ack=1 Win=42496 Len=256 TSval=3953059511 TSecr=3956643898
5	0.324548886	10.0.0.1	10.0.0.2	TCP	66	vcom-tunnel(8001) → teradataorlbs(8002) [ACK] Seq=1 Ack=257 Win=43520 Len=0 TSval=3956644223 TSecr=3953059511
6	0.324673508	10.0.0.1	10.0.0.2	TCP	322	vcom-tunnel(8001) → teradataorlbs(8002) [PSH, ACK] Seq=1 Ack=257 Win=43520 Len=256 TSval=3956644223 TSecr=3953059511
7	0.326437088	10.0.0.1	10.0.0.2	TCP	474	vcom-tunnel(8001) → teradataorlbs(8002) [FIN, PSH, ACK] Seq=257 Ack=257 Win=43520 Len=408 TSval=3956644225 TSecr=3953059511
8	0.367060517	10.0.0.2	10.0.0.1	TCP	66	teradataorlbs(8002) → vcom-tunnel(8001) [ACK] Seq=257 Ack=257 Win=42496 Len=0 TSval=3953059555 TSecr=3956644223
9	0.368143898	10.0.0.2	10.0.0.1	TCP	66	teradataorlbs(8002) → vcom-tunnel(8001) [FIN, ACK] Seq=257 Ack=666 Win=42496 Len=0 TSval=3953059556 TSecr=3956644225
10	0.368180219	10.0.0.1	10.0.0.2	TCP	66	vcom-tunnel(8001) → teradataorlbs(8002) [ACK] Seq=666 Ack=258 Win=43520 Len=0 TSval=3956644266 TSecr=3953059556

Figure 10: Screenshot of all packets captured by the clinet.



▶ Frame 4: 322 bytes on wire (2576 bits), 322 bytes captured (2576 bits) on interface s1-eth1, id 0		
▶ Ethernet II, Src: 00:00:00_00:00:02 (00:00:00:00:00:02), Dst: 00:00:00_00:00:01 (00:00:00:00:00:01)		
▶ Internet Protocol Version 4, Src: 10.0.0.2, Dst: 10.0.0.1		
▶ Transmission Control Protocol, Src Port: teradataordbms (8002), Dst Port: vcom-tunnel (8001), Seq: 1, Ack: 1, Len: 256		
▶ Data (256 bytes)		
0040	98 3a 4d 61 6b 65 66 69 6c 65 00 00 00 00 00 00	..:Makefile.....
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	.....
0080	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
0090	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
00a0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
00b0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
00c0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....

Figure 11: Screenshot of the packet that contains the requested filename.

▶ Frame 6: 322 bytes on wire (2576 bits), 322 bytes captured (2576 bits) on interface s1-eth1, id 0		
▶ Ethernet II, Src: 00:00:00_00:00:01 (00:00:00:00:00:01), Dst: 00:00:00_00:00:02 (00:00:00:00:00:02)		
▶ Internet Protocol Version 4, Src: 10.0.0.1, Dst: 10.0.0.2		
▶ Transmission Control Protocol, Src Port: vcom-tunnel (8001), Dst Port: teradataordbms (8002), Seq: 1, Ack: 257, Len: 256		
▶ Data (256 bytes)		
0040	e6 b7 34 30 38 00 00 00 00 00 70 71 29 6c 6e 7f	..408... ..pq)ln..
0050	00 00 70 71 29 6c 6e 7f 00 00 e8 6b 08 6c 6e 7f	..pq)ln.. ..k..ln..
0060	00 00 d0 d6 7e e8 fc 7f 00 00 f7 24 07 6c 6e 7f	..... ..\$.ln..
0070	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
0080	00 00 01 00 00 00 00 00 00 00 28 77 29 6c 6e 7f	..... ..(w)ln..
0090	00 00 00 71 29 6c 6e 7f 00 00 01 00 00 00 00 00	..q)ln.. ..
00a0	00 00 00 f5 27 6c 6e 7f 00 00 af 8f 07 6c 6e 7f	....'ln.. ..ln..
00b0	00 00 10 77 29 6c 6e 7f 00 00 00 00 00 00 00 00	..w)ln.. ..
00c0	00 00 00 00 00 00 00 00 00 00 d8 a2 7f e8 fc 7f	.....

Figure 12: Screenshot of the packet that contains the file size.

▶ Frame 7: 474 bytes on wire (3792 bits), 474 bytes captured (3792 bits) on interface s1-eth1, id 0		
▶ Ethernet II, Src: 00:00:00_00:00:01 (00:00:00:00:00:01), Dst: 00:00:00_00:00:02 (00:00:00:00:00:02)		
▶ Internet Protocol Version 4, Src: 10.0.0.1, Dst: 10.0.0.2		
▶ Transmission Control Protocol, Src Port: vcom-tunnel (8001), Dst Port: teradataordbms (8002), Seq: 257, Ack: 257, Len: 408		
▶ Data (408 bytes)		
0040	e6 b7 23 20 75 73 65 20 63 2b 2b 20 31 31 20 73	..# use c++ 11 s
0050	74 61 6e 64 61 72 64 20 68 65 72 65 20 28 63 2b	tandard here (c+
0060	2b 31 37 20 77 6f 75 6c 64 20 62 65 20 66 69 6e	+17 woul d be fin
0070	65 3f 29 0a 43 43 3d 67 2b 2b 20 2d 67 20 2d 57	e?).CC=g ++ -g -W
0080	61 6c 6c 20 2d 73 74 64 3d 63 2b 2b 31 31 0a 0a	all -std =c++11..
0090	23 20 4c 69 73 74 20 6f 66 20 73 6f 75 72 63 65	# List o f source
00a0	20 66 69 6c 65 73 20 66 6f 72 20 69 50 65 72 66	files f or iPerf
00b0	65 72 0a 46 53 5f 53 4f 55 52 43 45 53 3d 66 74	er.FS_SO URCES=ft
00c0	72 61 6e 73 2e 63 63 0a 0a 23 20 47 65 6e 65 72	rans.cc. .# Gener

Figure 13: Screenshot of the packet that contains the file data, 408 bytes in total.

## 3 Reliable Transmission

### 3.1 Implement a simple SR

A No reordering, no loss, no error.

```
# sender log
SYN 780 0 0 # initiate a connection
ACK 780 0 0 # receive ACK for the SYN (same seq as SYN)
DATA 0 1456 -1074344063 # send out 10 packets...
DATA 1 1456 -2050894268
DATA 2 1456 1460592462
DATA 3 1456 -879084444
DATA 4 1456 -838440118
DATA 5 1456 1573440664
DATA 6 1456 -822598369
DATA 7 1456 -1116097088
DATA 8 1456 279932998
DATA 9 1456 -868873766
ACK 1 0 0 # receive ACK for packet 1
DATA 10 1456 -1489524643 # window is slided, packet 10 is thus sent.
ACK 2 0 0 # receive ACK for packet 2
DATA 11 1456 443311922 # window is slided, packet 11 is thus sent.
...
FIN 780 0 0 # tear down the connection (same seq as SYN)
ACK 780 0 0 # receive ACK for the FIN
```

```
# receiver log
SYN 780 0 0 # receive a connection request
ACK 780 0 0 # ACK with the same seq
DATA 0 1456 -1074344063 # receive packet 0
ACK 1 0 0 # ACK 1 (and slide window)
DATA 1 1456 -2050894268 # receive packet 1
ACK 2 0 0 # ACK 2 (and slide window)
...
FIN 780 0 0 # receive connection close request
ACK 780 0 0 # ACK the FIN
```

B 10 percent loss, no reordering, no error.

```

# sender log
..
DATA 19 1456 725513889
ACK 10 0 0
ACK 10 0 0 # still receive ACK 10
ACK 10 0 0
ACK 10 0 0
ACK 10 0 0
ACK 10 0 0
DATA 10 1456 -1489524643 # timeout is triggered, resend whole window
DATA 11 1456 443311922
DATA 12 1456 -267121725
DATA 13 1456 1330974691
DATA 14 1456 2109819272
DATA 15 1456 965284191
DATA 16 1456 -983191108
DATA 17 1456 1539359076
DATA 18 1456 -1934933216
DATA 19 1456 725513889
ACK 19 0 0
DATA 20 1456 -299260989 # receive ACK, proceed.
DATA 21 1456 -930311144
DATA 22 1456 943122041
DATA 23 1456 -1623696116
DATA 24 1456 -1706148246
...

```

```

# receiver log
...
DATA 11 1456 443311922 # expect packet 10 but receive packet 11
ACK 10 0 0 # ACK 10
DATA 12 1456 -267121725 # expect packet 10 but receive packet 12
ACK 10 0 0 # ACK 10
DATA 13 1456 1330974691
ACK 10 0 0
DATA 14 1456 2109819272
ACK 10 0 0
DATA 15 1456 965284191
ACK 10 0 0
DATA 16 1456 -983191108
ACK 10 0 0
DATA 17 1456 1539359076
ACK 10 0 0
DATA 18 1456 -1934933216
ACK 10 0 0
DATA 10 1456 -1489524643 # finally receive packet 10
ACK 19 0 0 # ACK 19 now
DATA 12 1456 -267121725
ACK 19 0 0
DATA 13 1456 1330974691
ACK 19 0 0
...

```

C 10 percent error, no reordering, no loss.



```
# sender log
...
DATA 22 1456 943122041
ACK 14 0 0
DATA 23 1456 -1623696116
ACK 15 0 0
DATA 24 1456 -1706148246
ACK 15 0 0
ACK 15 0 0 # still receive ACK 15, packet 15 is probably corrupted
ACK 15 0 0
ACK 15 0 0
ACK 15 0 0
ACK 15 0 0
ACK 15 0 0
ACK 15 0 0
DATA 15 1456 965284191 # timeout is triggered, resend whole window, packet 15 is retransmitted
DATA 16 1456 -983191108
DATA 17 1456 1539359076
DATA 18 1456 -1934933216
DATA 19 1456 725513889
DATA 20 1456 -299260989
DATA 21 1456 -930311144
DATA 22 1456 943122041
DATA 23 1456 -1623696116
DATA 24 1456 -1706148246
ACK 17 0 0 # receive ACK, proceed
DATA 25 1456 -168693199
DATA 26 1456 1578006857
...
```

```
# receiver log
...
DATA 14 1456 2109819272
ACK 15 0 0
DATA 15 1456 965284191 # 15 received but not ACKed because its payload is corrupted
DATA 16 1456 -983191108
ACK 15 0 0 # still ACK 15
DATA 17 1456 1539359076# 17 received but not ACKed because its payload is corrupted
DATA 18 1456 -1934933216
ACK 15 0 0
DATA 19 1456 725513889
ACK 15 0 0
DATA 20 1456 -299260989
ACK 15 0 0
DATA 21 1456 -930311144
ACK 15 0 0
DATA 22 1456 943122041
ACK 15 0 0
DATA 23 1456 -1623696116
ACK 15 0 0
DATA 24 1456 -1706148246
ACK 15 0 0
DATA 15 1456 965284191 # 15 is received correctly this time.
ACK 17 0 0 # ACK 17 now
...
```

D Reordering, no error, no loss.

```
# sender log
...
SYN 402536 0 0
ACK 402536 0 0
DATA 0 1456 -1074344063
DATA 1 1456 -2050894268
DATA 2 1456 1460592462
DATA 3 1456 -879084444
DATA 4 1456 -838440118
DATA 5 1456 1573440664
DATA 6 1456 -822598369
DATA 7 1456 -1116097088
DATA 8 1456 279932998
DATA 9 1456 -868873766
ACK 0 0 0
ACK 0 0 0 # still ACK 0
ACK 0 0 0
ACK 0 0 0
ACK 0 0 0
ACK 0 0 0
ACK 0 0 0
ACK 8 0 0 # finally ACK 8, meaning packet 0-7 are all received
DATA 10 1456 -1489524643
DATA 11 1456 443311922
DATA 12 1456 -267121725
DATA 13 1456 1330974691
DATA 14 1456 2109819272
DATA 15 1456 965284191
DATA 16 1456 -983191108
DATA 17 1456 1539359076
ACK 10 0 0 # ACK 10, meaning packet 0-9 are all received.
...
```

```

# receiver log
...
SYN 402536 0 0
ACK 402536 0 0
DATA 2 1456 1460592462 # packets are reordered, 2 is received before 0
ACK 0 0 0 # ACK 0
DATA 3 1456 -879084444
ACK 0 0 0
DATA 7 1456 -1116097088
ACK 0 0 0
DATA 5 1456 1573440664
ACK 0 0 0
DATA 9 1456 -868873766
ACK 0 0 0
DATA 4 1456 -838440118
ACK 0 0 0
DATA 6 1456 -822598369
ACK 0 0 0
DATA 1 1456 -2050894268
ACK 0 0 0
DATA 0 1456 -1074344063 # packet 0 finally received
ACK 8 0 0 # ACK 8
DATA 8 1456 279932998
ACK 10 0 0
...

```

E Reordering, 5 percent loss, 5 percent error.

### 3.2 Make sender more efficient - leverage duplicate ACK on sender side

A 10

```

# sender log
...
ACK 1 0 0
DATA 10 1456 -1489524643
ACK 2 0 0
DATA 11 1456 443311922
ACK 2 0 0
ACK 2 0 0 # receive three ACK 2
DATA 2 1456 1460592462 # resend packet 2
ACK 2 0 0
ACK 2 0 0
ACK 2 0 0 # receive three ACK 2
DATA 2 1456 1460592462 # resend packet 2
ACK 2 0 0
ACK 2 0 0
ACK 2 0 0 # receive three ACK 2
DATA 2 1456 1460592462 # resend packet 2
ACK 7 0 0 # finally ACK 7
DATA 12 1456 -267121725 # window is slid
...

```

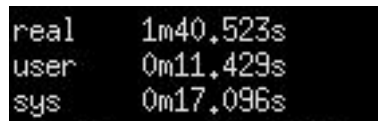
```

# receiver log
...
ACK 1 0 0
DATA 1 1456 -2050894268
ACK 2 0 0
DATA 3 1456 -879084444
ACK 2 0 0 # packet 2 is not received, still ACK 2
DATA 4 1456 -838440118
ACK 2 0 0
DATA 8 1456 279932998
ACK 2 0 0
DATA 9 1456 -868873766
ACK 2 0 0
DATA 6 1456 -822598369
ACK 2 0 0
DATA 5 1456 1573440664
ACK 2 0 0
DATA 10 1456 -1489524643
ACK 2 0 0
DATA 11 1456 443311922
ACK 2 0 0
DATA 2 1456 1460592462 # packet 2 is received
ACK 7 0 0 # ACK 7
...

```

#### B Efficiency.

As shown in Figure 14 and 15, duplicate ACK makes the sender more efficient. When transmitting a 4.4MB file, the time the sender uses is almost halved after using duplicate ACK.

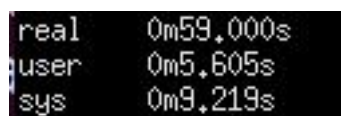


```

real    1m40.523s
user    0m11.429s
sys     0m17.096s

```

Figure 14: Screenshot of the time the sender use to send a 4.4MB file using simple SR.



```

real    0m59.000s
user    0m5.605s
sys     0m9.219s

```

Figure 15: Screenshot of the time the sender use to send a 4.4MB file using duplicate ACK.

#### C Make sure it transfers files reliably.

### 3.3 Make sender more efficient - send NACK on receiver side

A 10

```

# sender log
...
DATA 18 1456 -1934933216
ACK 10 0 0
DATA 19 1456 725513889
NACK 10 0 0 # receive NACK 10
DATA 10 1456 -1489524643 # resend packet 10
ACK 10 0 0
ACK 10 0 0
ACK 10 0 0
ACK 10 0 0
ACK 10 0 0
ACK 10 0 0
DATA 10 1456 -1489524643 # timeout is triggered, resend whole window
DATA 11 1456 443311922
DATA 12 1456 -267121725
DATA 13 1456 1330974691
DATA 14 1456 2109819272
DATA 15 1456 965284191
DATA 16 1456 -983191108
DATA 17 1456 1539359076
DATA 18 1456 -1934933216
DATA 19 1456 725513889
ACK 18 0 0 # receive ACK, proceed
DATA 20 1456 -299260989
DATA 21 1456 -930311144
...

```

```

# receiver log
...
DATA 9 1456 -868873766
ACK 10 0 0
DATA 11 1456 443311922
NACK 10 0 0 # gap in window, NACK 10
DATA 12 1456 -267121725
ACK 10 0 0
DATA 13 1456 1330974691
ACK 10 0 0
DATA 14 1456 2109819272
ACK 10 0 0
DATA 15 1456 965284191
ACK 10 0 0
DATA 16 1456 -983191108
ACK 10 0 0
DATA 17 1456 1539359076
ACK 10 0 0
DATA 10 1456 -1489524643 # receive packet 10, proceed
ACK 18 0 0
...

```

#### B Efficiency.

As shown in Figure 16 and 17, NACK makes the sender more efficient. When transmitting a 4.4MB file, the time the sender uses is almost halved after using NACK.

```
real    1m40.523s
user    0m11.429s
sys     0m17.096s
```

Figure 16: Screenshot of the time the sender use to send a 4.4MB file using simple SR.

```
real    0m57.093s
user    0m5.573s
sys     0m8.339s
```

Figure 17: Screenshot of the time the sender use to send a 4.4MB file using NACK.

C Make sure it transfers files reliably.

### 3.4 Throughput, delay and window size

The file we transmit is a 4.4MB jpg file.

When the delay is  $0.01ms$  and the window size is 10, the real time is  $15.870s$ . The throughput is about  $290721B/s$ .

When the delay is  $0.01ms$  and the window size is 5, the real time is  $31.512s$ . The throughput is about  $146412B/s$ .

When the delay is  $10ms$  and the window size is 10, the real time is  $22.310s$ . The throughput is about  $206801B/s$ .

When the delay is  $10ms$  and the window size is 5, the real time is  $44.451s$ . The throughput is about  $103794B/s$ .

When the delay is  $50ms$  and the window size is 10, the real time is  $46.407s$ . The throughput is about  $99419B/s$ .

When the delay is  $50ms$  and the window size is 5, the real time is  $1m32.093s$ . The throughput is about  $50099B/s$ .

When the delay is  $100ms$  and the window size is 10, the real time is  $1m16.549s$ . The throughput is about  $60272B/s$ .

When the delay is  $100ms$  and the window size is 5, the real time is  $2m31.807s$ . The throughput is about  $30392B/s$ .

As we can see from the measurements and calculations, throughput increases as window size increases and it decreases as delay increases.



real 0m15.870s	real 0m31.512s
user 0m2.013s	user 0m4.007s
sys 0m3.022s	sys 0m6.179s
real 0m22.310s	real 0m44.451s
user 0m3.066s	user 0m5.922s
sys 0m4.097s	sys 0m8.467s
real 0m46.407s	real 1m32.093s
user 0m6.319s	user 0m12.912s
sys 0m8.818s	sys 0m18.116s
real 1m16.549s	real 2m31.807s
user 0m10.648s	user 0m22.576s
sys 0m14.829s	sys 0m31.685s

Figure 18: Screenshots of the time to transmit a 4.4MB file with different delays and window sizes.