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.

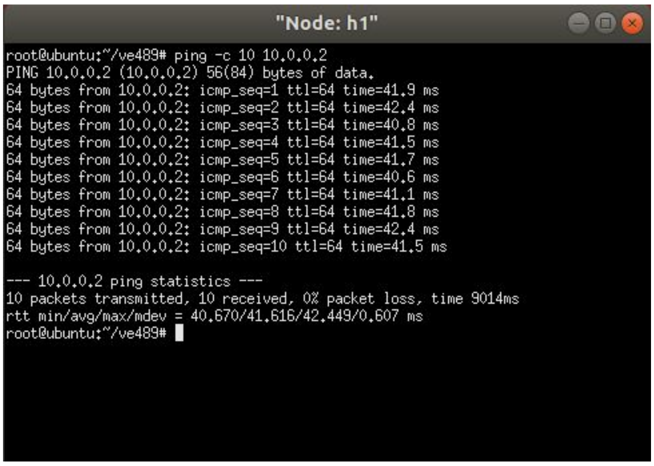


Figure 1: Screenshot of pinging 10 times from h_1 to h_2.

   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.

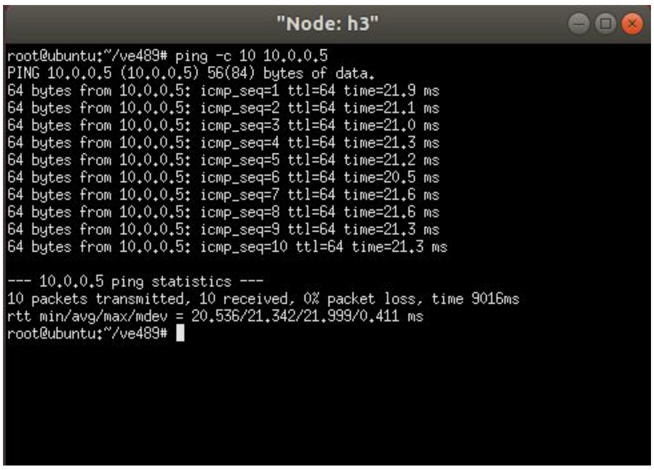


Figure 2: Screenshot of pinging 10 times from h_3 to h_5.

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.

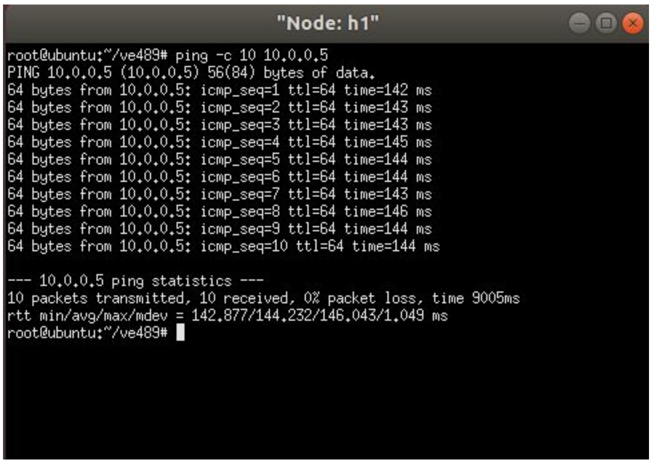


Figure 3: Screenshot of pinging 10 times from h_1 to h_5.

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.

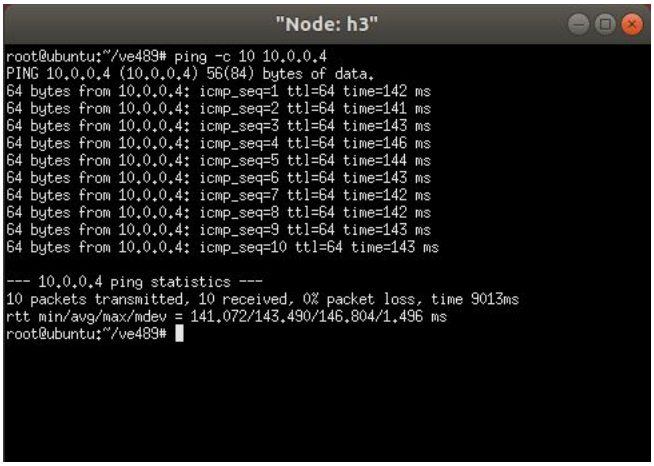


Figure 4: Screenshot of pinging 10 times from h_3 to h_4.

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.

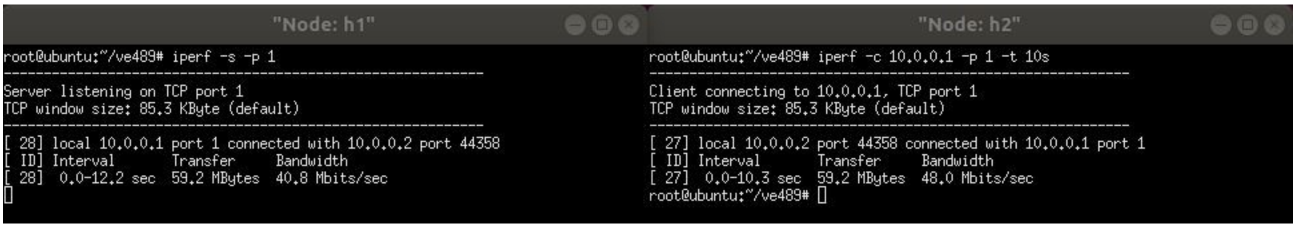


Figure 5: Screenshot of link bandwidth between h_1 and h_2.

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

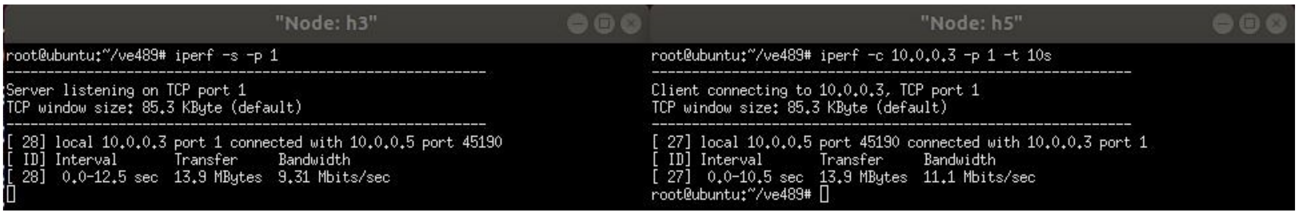


Figure 6: Screenshot of link bandwidth between h_3 and h_5.

4. **Path throughput using iperf.**

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

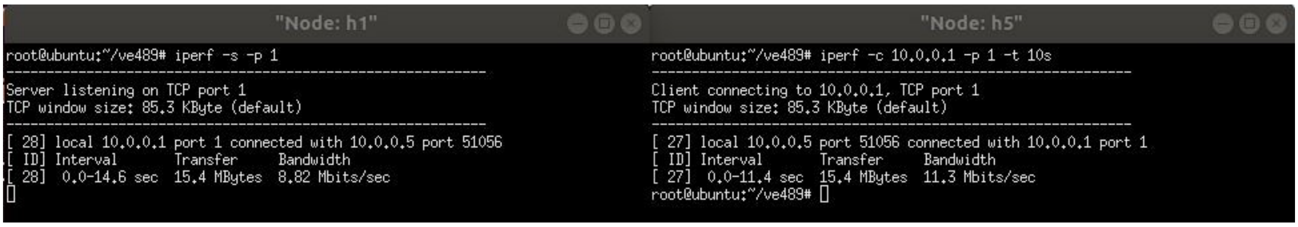


Figure 7: Screenshot of throughput between h_1 and h_5.

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

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.

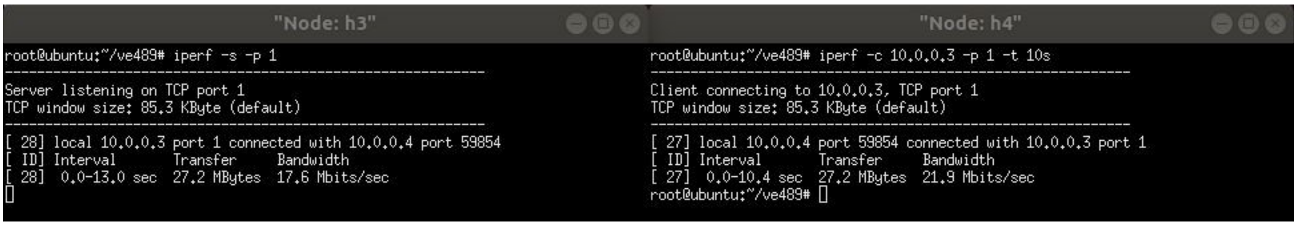


Figure 8: Screenshot of throughput between h_3 and h_4.

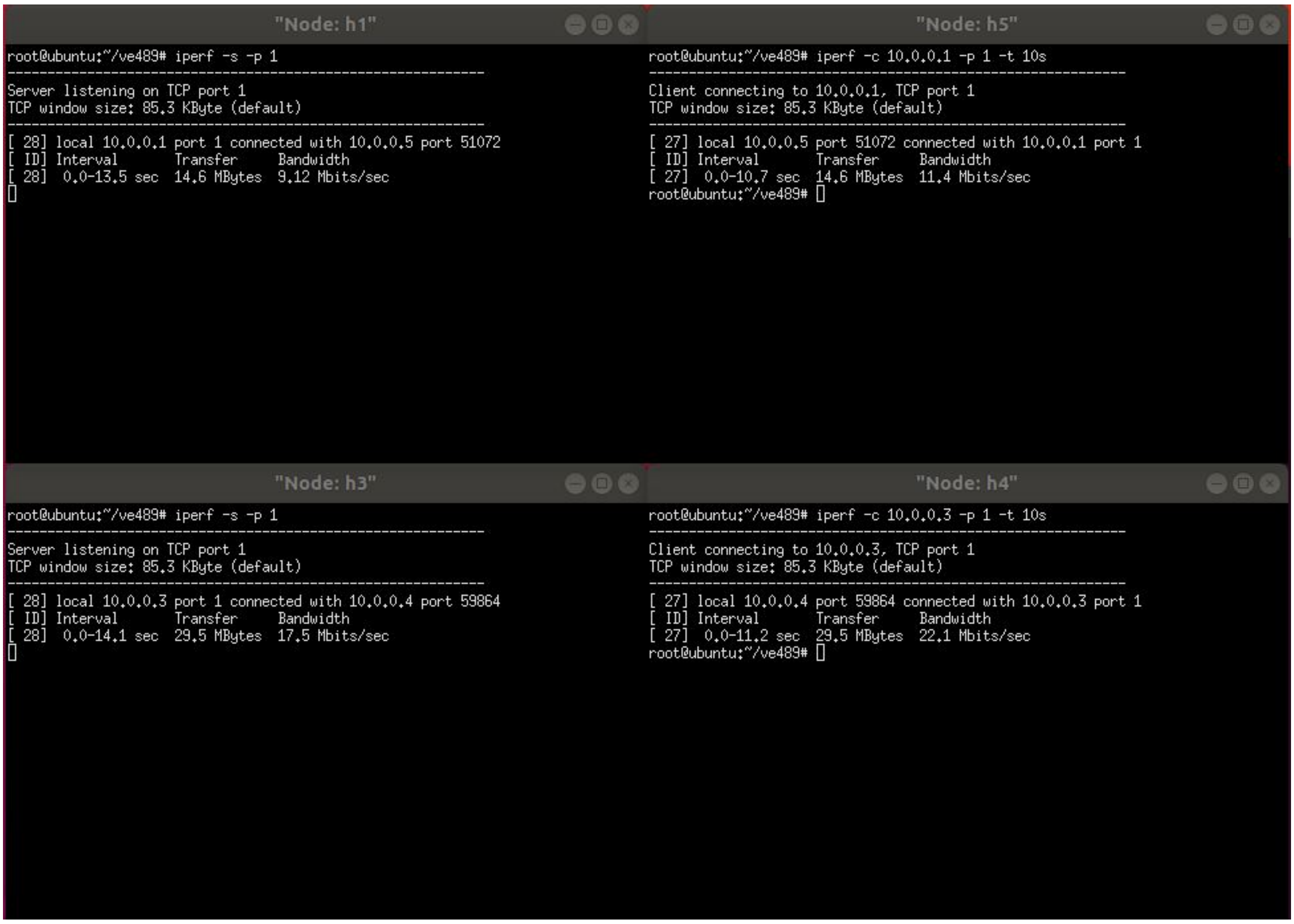
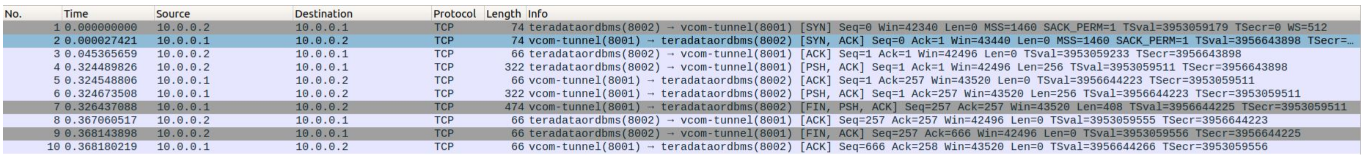


Figure 9: Screenshot of (h_1, h_5) and (h_3, h_4) sharing the same link.



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	10.0.0.2	10.0.0.1	TCP	74	teradataordbms(8002) → vcom-tunnel(8001) [SYN] 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) → teradataordbms(8002) [SYN, ACK] Seq=0 Ack=1 Win=43440 Len=0 MSS=1460 SACK_PERM=1 TSval=3956643898 TSecr=…
3	0.045365659	10.0.0.2	10.0.0.1	TCP	66	teradataordbms(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	teradataordbms(8002) → vcom-tunnel(8001) [PSH, ACK] Seq=1 Ack=1 Win=42496 Len=256 TSval=3953059511 TSecr=3956643898
5	0.324548806	10.0.0.1	10.0.0.2	TCP	66	vcom-tunnel(8001) → teradataordbms(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) → teradataordbms(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) → teradataordbms(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	teradataordbms(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	teradataordbms(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) → teradataordbms(8002) [ACK] Seq=666 Ack=258 Win=43520 Len=0 TSval=3956644266 TSecr=3953059556

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

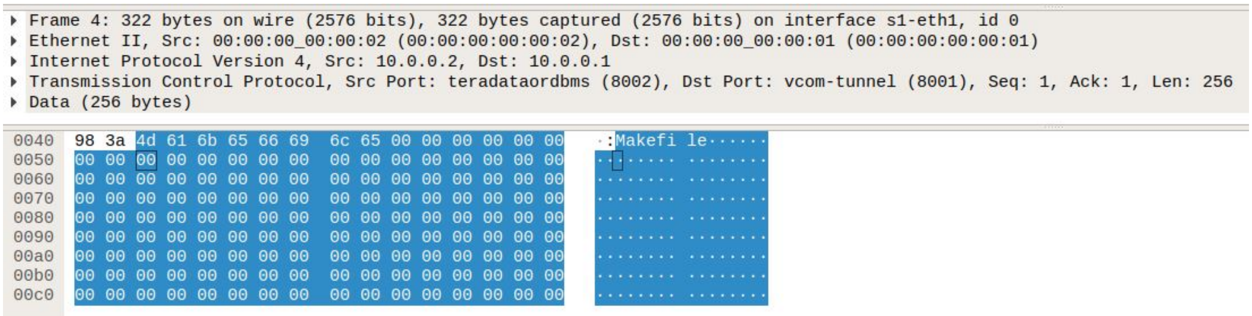


Figure 11: Screenshot of the packet that contains the requested filename.

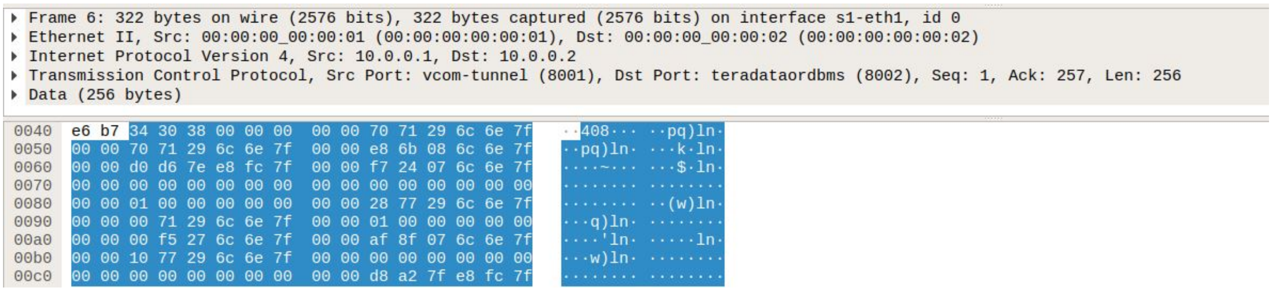


Figure 12: Screenshot of the packet that contains the file size.

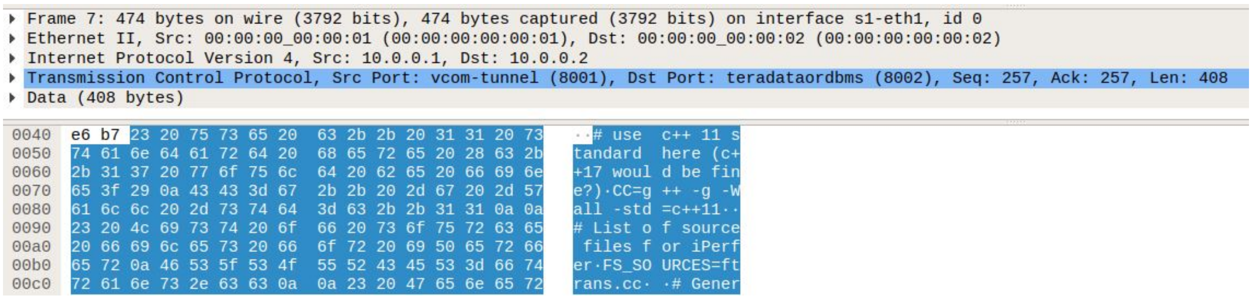


Figure 13: Screenshot of the packet that contains the file data, 408 bytes in total.

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 slided
...
```

```
# 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.

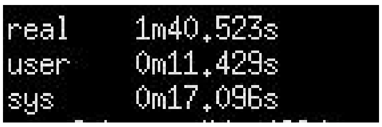


Figure 14: Screenshot of the time the sender use to send a 4.4MB file using simple SR.

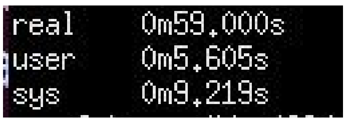


Figure 15: Screenshot of the time the sender use to send a 4.4MB file using duplicate ACK.

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.

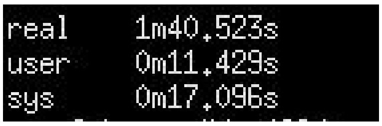


Figure 16: Screenshot of the time the sender use to send a 4.4MB file using simple SR.

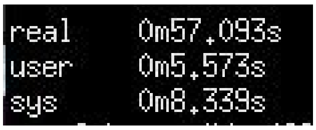


Figure 17: Screenshot of the time the sender use to send a 4.4MB file using NACK.

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.

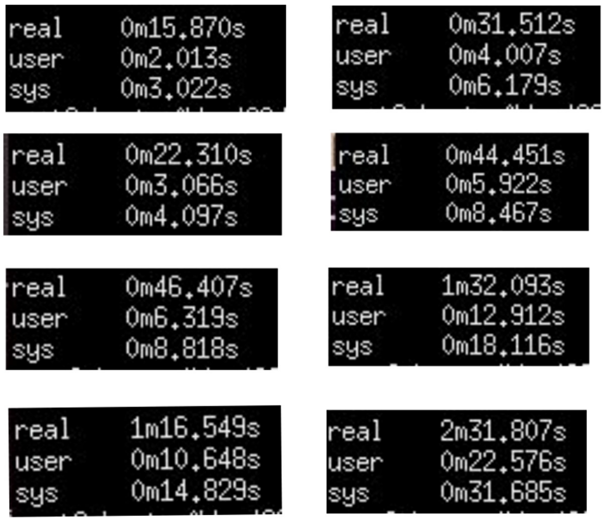


Figure 18: Screenshots of the time to transmit a 4.4MB file with different delays and window sizes.