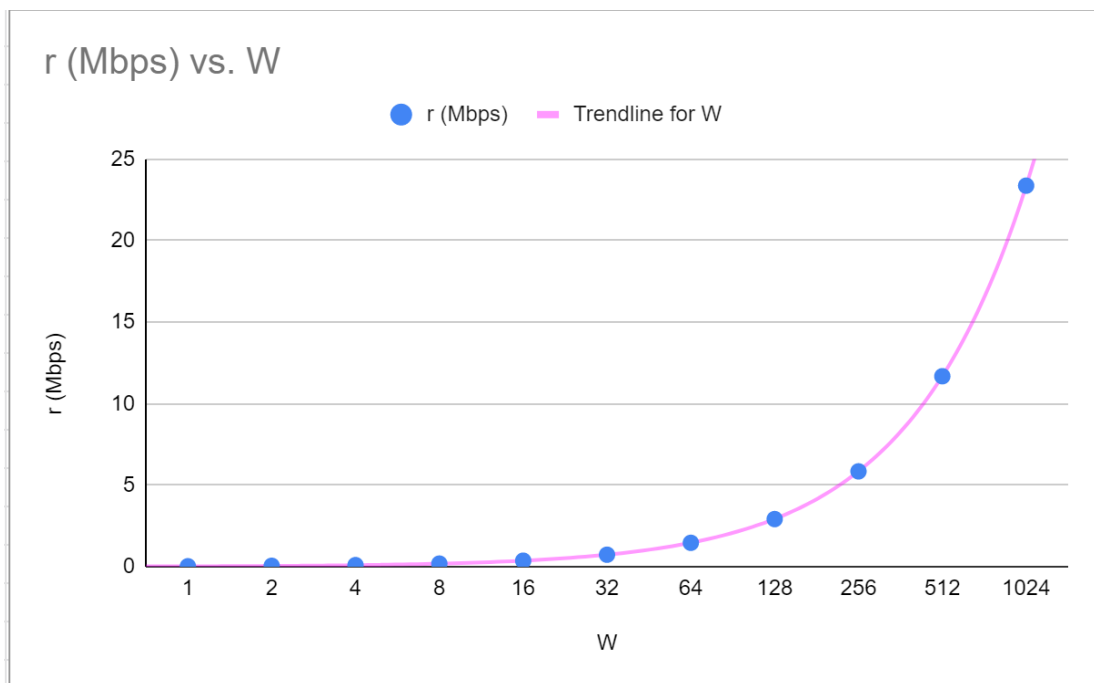


CSCE 463-500

Reliable Data Transfer over UDP - Homework 3.3

The purpose of this assignment was to write a C++ transport-layer service over UDP that can sustain non-trivial transfer rates (hundreds of megabits/sec) under low packet loss and avoid getting bogged down under heavy loss.

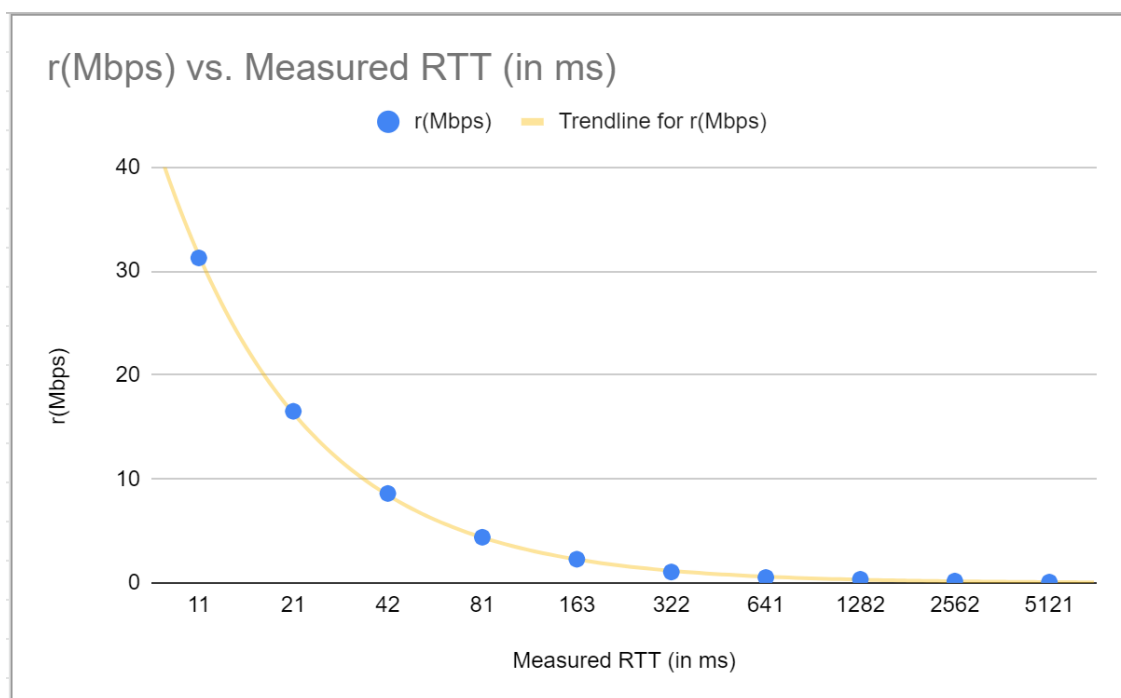
1. Set packet loss p to zero in both directions, the RTT to 0.5 seconds, and bottleneck link speed to $S = 1$ Gbps. Examine how your goodput scales with window size W . This should be done by plotting the steady-state rate $r(W)$ for $W = 1, 2, 4, 8, \dots, 210$ and keeping the x axis on a log-scale. Your peak rate will be around 24 Mbps and, depending on your home bandwidth, usage of an on-campus server might be necessary. Using curve-fitting, generate a model for $r(W)$. Discuss whether it matches the theory discussed in class.
 - a. The model for $r(W)$ obtained is included in the image below, as well as the best fit function for the curve.



- b. We can see that the generated $r(W)$ plot is linear, which matches the theory that we learned in lecture (and that is included in the class slides from 3/1/22), which says that $U_{sender} = (3L/R)/(RTT + L/R)$.

2. Expanding on the previous question, fix the window size at $W = 30$ packets and vary the $RTT = 10, 20, 40, \dots, 5120$ ms. Plot stable rate $r(RTT)$, again placing the x-axis on a log scale. Perform curve-fitting to determine a model that describes this relationship. Due to queuing/transmission delays emulated by the server and various OS kernel overhead, the 1 actual RTT may deviate from the requested RTT. Thus, use the measured average in your plots and comment on whether the resulting curve matches theory.

- a. The generated model for $r(RTT)$ obtained is included in the image below, as well as the best fit function for the curve.



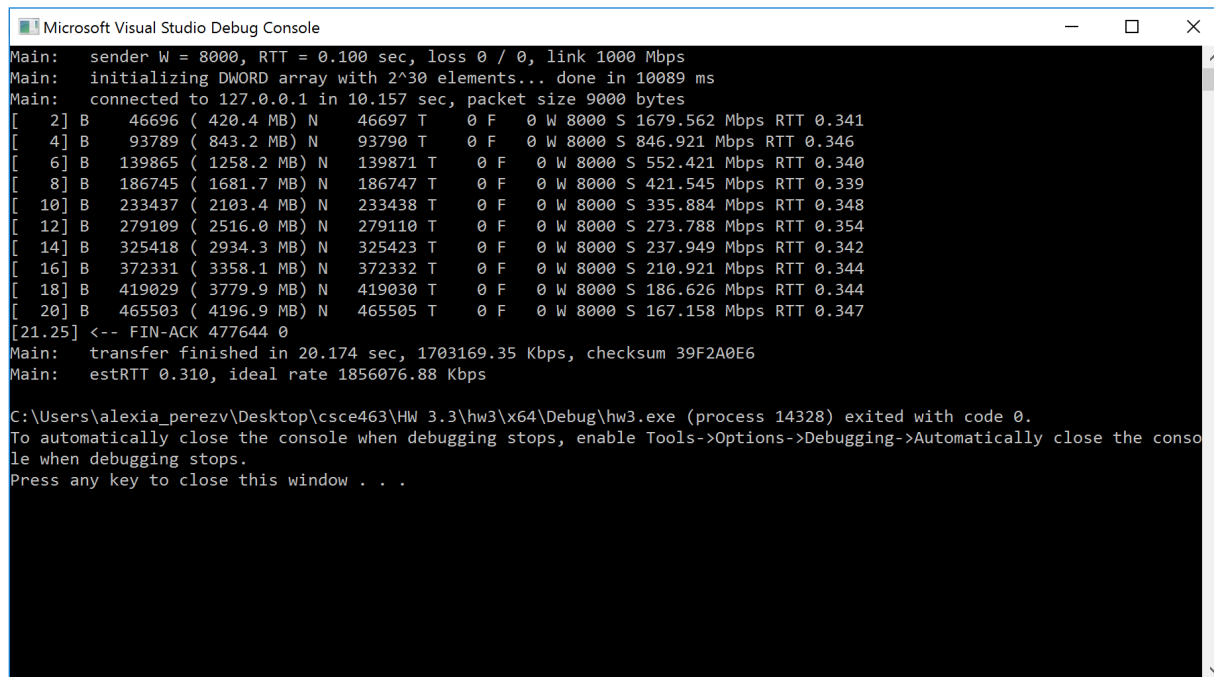
- b. We again can see that the curve is linear and it matches the theory discussed in lecture.

3. Run the dummy receiver on your localhost and produce a trace using $W = 8K$ (the other parameters do not matter as the dummy receiver ignores them, although they should still be within valid ranges). Discuss your CPU configuration and whether you managed to exceed 1 Gbps. How about 10 Gbps using 9-KB packets (see dummy-receiver discussion in Part 1)?

- a. The trace produced by running the dummy receiver on ts2.cse.tamu.edu with $W = 8K$ is included below:

```
Microsoft Visual Studio Debug Console
Main: sender W = 8000, RTT = 0.100 sec, loss 0 / 0, link 1000 Mbps
Main: initializing DWORD array with 2^30 elements... done in 10025 ms
Main: connected to localhost in 10.076 sec, packet size 1472 bytes
[ 2] B 76748 ( 113.0 MB) N 76750 T 0 F 0 W 8000 S 449.436 Mbps RTT 0.207
[ 4] B 156230 ( 229.5 MB) N 156232 T 0 F 0 W 8000 S 232.723 Mbps RTT 0.206
[ 6] B 233475 ( 342.7 MB) N 233478 T 0 F 0 W 8000 S 150.782 Mbps RTT 0.208
[ 8] B 309963 ( 454.7 MB) N 309965 T 0 F 0 W 8000 S 111.978 Mbps RTT 0.213
[10] B 389299 ( 570.9 MB) N 389307 T 0 F 0 W 8000 S 92.918 Mbps RTT 0.200
[12] B 467478 ( 685.4 MB) N 467480 T 0 F 0 W 8000 S 76.303 Mbps RTT 0.208
[14] B 545563 ( 799.8 MB) N 545565 T 0 F 0 W 8000 S 65.324 Mbps RTT 0.204
[16] B 623785 ( 914.4 MB) N 623790 T 0 F 0 W 8000 S 57.259 Mbps RTT 0.207
[18] B 701191 (1027.8 MB) N 701193 T 0 F 0 W 8000 S 50.366 Mbps RTT 0.208
[20] B 779423 (1142.4 MB) N 779425 T 0 F 0 W 8000 S 45.813 Mbps RTT 0.207
[22] B 857218 (1256.4 MB) N 857222 T 0 F 0 W 8000 S 41.415 Mbps RTT 0.213
[24] B 935019 (1370.3 MB) N 935028 T 0 F 0 W 8000 S 37.967 Mbps RTT 0.199
[26] B 1014230 (1486.3 MB) N 1014235 T 0 F 0 W 8000 S 35.682 Mbps RTT 0.204
[28] B 1093806 (1602.9 MB) N 1093815 T 0 F 0 W 8000 S 33.286 Mbps RTT 0.200
[30] B 1171170 (1716.2 MB) N 1171172 T 0 F 0 W 8000 S 30.203 Mbps RTT 0.212
[32] B 1248978 (1830.2 MB) N 1248980 T 0 F 0 W 8000 S 28.478 Mbps RTT 0.213
[34] B 1326115 (1943.2 MB) N 1326119 T 0 F 0 W 8000 S 26.571 Mbps RTT 0.207
[36] B 1402970 (2055.8 MB) N 1402971 T 0 F 0 W 8000 S 25.003 Mbps RTT 0.210
[38] B 1479751 (2168.3 MB) N 1479754 T 0 F 0 W 8000 S 23.665 Mbps RTT 0.210
[40] B 1557082 (2281.5 MB) N 1557084 T 0 F 0 W 8000 S 22.643 Mbps RTT 0.212
[42] B 1636104 (2397.3 MB) N 1636113 T 0 F 0 W 8000 S 22.036 Mbps RTT 0.196
[44] B 1713822 (2511.1 MB) N 1713824 T 0 F 0 W 8000 S 20.687 Mbps RTT 0.206
[46] B 1794247 (2629.0 MB) N 1794252 T 0 F 0 W 8000 S 20.477 Mbps RTT 0.194
[48] B 1872754 (2744.0 MB) N 1872754 T 0 F 0 W 8000 S 19.156 Mbps RTT 0.199
[50] B 1950902 (2858.4 MB) N 1950909 T 0 F 0 W 8000 S 18.305 Mbps RTT 0.203
[52] B 2031277 (2976.2 MB) N 2031281 T 0 F 0 W 8000 S 18.103 Mbps RTT 0.203
[54] B 2108479 (3089.3 MB) N 2108480 T 0 F 0 W 8000 S 16.744 Mbps RTT 0.208
[56] B 2187235 (3204.6 MB) N 2187238 T 0 F 0 W 8000 S 16.471 Mbps RTT 0.205
[58] B 2265263 (3318.9 MB) N 2265268 T 0 F 0 W 8000 S 15.756 Mbps RTT 0.201
[60] B 2341887 (3431.2 MB) N 2341888 T 0 F 0 W 8000 S 14.957 Mbps RTT 0.205
[62] B 2420345 (3546.1 MB) N 2420346 T 0 F 0 W 8000 S 14.821 Mbps RTT 0.202
[64] B 2499113 (3661.5 MB) N 2499115 T 0 F 0 W 8000 S 14.415 Mbps RTT 0.193
[66] B 2577558 (3776.4 MB) N 2577561 T 0 F 0 W 8000 S 13.920 Mbps RTT 0.212
[68] B 2656743 (3892.4 MB) N 2656749 T 0 F 0 W 8000 S 13.638 Mbps RTT 0.192
[70] B 2732675 (4003.6 MB) N 2732677 T 0 F 0 W 8000 S 12.705 Mbps RTT 0.212
[72] B 2811273 (4118.7 MB) N 2811277 T 0 F 0 W 8000 S 12.785 Mbps RTT 0.205
[74] B 2888963 (4232.5 MB) N 2888972 T 0 F 0 W 8000 S 12.296 Mbps RTT 0.205
[75.95] <-- FIN-ACK 2933721 0
Main: transfer finished in 74.895 sec, 458772.13 Kbps, checksum 39F2A0E6
Main: estRTT 0.185, ideal rate 506466.16 Kbps
C:\Users\alexia_perezv\Desktop\HW 3.3\hw3\hw3.exe (process 2796) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
```

- b. The CPU configuration of the ts2.cse.tamu.edu server is as follows:
- Processor: AMD Opteron™ 6172, 2.10 GHz (2 processors)
 - Installed RAM: 32GB
- c. I was able to reach 449.436 Mbps when using the ts2 server and running the dummy receiver, but did not reach 1 Gbps at any point in this trace.
- d. The following trace corresponds to running the dummy receiver on the ts2.cse.tamu.edu and adjusting the packet size to 9000 bytes.



```

Microsoft Visual Studio Debug Console
Main: sender W = 8000, RTT = 0.100 sec, loss 0 / 0, link 1000 Mbps
Main: initializing DWORD array with 2^30 elements... done in 10089 ms
Main: connected to 127.0.0.1 in 10.157 sec, packet size 9000 bytes
[ 2] B 46696 ( 420.4 MB) N 46697 T 0 F 0 W 8000 S 1679.562 Mbps RTT 0.341
[ 4] B 93789 ( 843.2 MB) N 93790 T 0 F 0 W 8000 S 846.921 Mbps RTT 0.346
[ 6] B 139865 ( 1258.2 MB) N 139871 T 0 F 0 W 8000 S 552.421 Mbps RTT 0.340
[ 8] B 186745 ( 1681.7 MB) N 186747 T 0 F 0 W 8000 S 421.545 Mbps RTT 0.339
[10] B 233437 ( 2103.4 MB) N 233438 T 0 F 0 W 8000 S 335.884 Mbps RTT 0.348
[12] B 279109 ( 2516.0 MB) N 279110 T 0 F 0 W 8000 S 273.788 Mbps RTT 0.354
[14] B 325418 ( 2934.3 MB) N 325423 T 0 F 0 W 8000 S 237.949 Mbps RTT 0.342
[16] B 372331 ( 3358.1 MB) N 372332 T 0 F 0 W 8000 S 210.921 Mbps RTT 0.344
[18] B 419029 ( 3779.9 MB) N 419030 T 0 F 0 W 8000 S 186.626 Mbps RTT 0.344
[20] B 465503 ( 4196.9 MB) N 465505 T 0 F 0 W 8000 S 167.158 Mbps RTT 0.347
[21.25] <-- FIN-ACK 477644 0
Main: transfer finished in 20.174 sec, 1703169.35 Kbps, checksum 39F2A0E6
Main: estRTT 0.310, ideal rate 1856076.88 Kbps

C:\Users\alexia_perezv\Desktop\csce463\HW 3.3\hw3\x64\Debug\hw3.exe (process 14328) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .

```

- e. Here, we can see that 1679.562 Mbps were reached, however, 10 Gbps were not.
- f. I would also like to point out that although I ran these tests on-campus, while connected to the TAMU Wi-Fi network (which should be faster than connecting to the server from my house), the connection took an entire 10 seconds to be established. This is something that I have not experienced throughout the development of my code for this project, and I believe it may simply be due to the Wi-Fi connection being slow rather than my code being incorrect (I have seen my connection times and speeds be much higher even when I am at home, and no changes were made to the code for it to become this much slower).

4. Use buffer size 223 DWORDs, RTT = 200 ms, window size $W = 300$ packets, link capacity $S = 10$ Mbps, and loss only in the reverse direction equal to $p = 0.1$. Show an entire trace of execution for this scenario and compare it to a similar case with no loss in either direction. Does your protocol keep the same rate in these two cases? Why or why not?

a. Loss in reverse direction:

```
Microsoft Visual Studio Debug Console
Main: sender W = 300, RTT = 0.200 sec, loss 0 / 0.1, link 10 Mbps
Main: initializing DWORD array with 2^23 elements... done in 70 ms
Main: connected to s3.irl.cs.tamu.edu in 0.403 sec, packet size 1472 bytes
[ 2] B 123 ( 0.2 MB) N 246 T 0 F 0 W 123 S 0.720 Mbps RTT 0.233
[ 4] B 1670 ( 2.2 MB) N 1970 T 0 F 0 W 300 S 4.530 Mbps RTT 0.353
[ 6] B 3377 ( 4.4 MB) N 3677 T 0 F 0 W 300 S 3.332 Mbps RTT 0.353
[ 8] B 5077 ( 6.7 MB) N 5377 T 0 F 0 W 300 S 2.489 Mbps RTT 0.353
[10] B 6789 ( 9.0 MB) N 7089 T 0 F 0 W 300 S 2.005 Mbps RTT 0.353
[12] B 8487 (11.2 MB) N 8787 T 0 F 0 W 300 S 1.657 Mbps RTT 0.353
[14] B 10185 (13.5 MB) N 10485 T 0 F 0 W 300 S 1.420 Mbps RTT 0.353
[16] B 11883 (15.8 MB) N 12183 T 0 F 0 W 300 S 1.243 Mbps RTT 0.353
[18] B 13589 (18.0 MB) N 13889 T 0 F 0 W 300 S 1.110 Mbps RTT 0.353
[20] B 15292 (20.3 MB) N 15592 T 0 F 0 W 300 S 0.997 Mbps RTT 0.353
[22] B 17001 (22.5 MB) N 17301 T 0 F 0 W 300 S 0.910 Mbps RTT 0.353
[24] B 18701 (24.7 MB) N 19001 T 0 F 0 W 300 S 0.830 Mbps RTT 0.353
[26] B 20399 (27.0 MB) N 20699 T 0 F 0 W 300 S 0.765 Mbps RTT 0.353
[28] B 22099 (29.2 MB) N 22399 T 0 F 0 W 300 S 0.711 Mbps RTT 0.353
[29.90] <-- FIN-ACK 22920 D70096AB
Main: transfer finished in 28.353 sec, 9467.62 Kbps, checksum D70096AB
Main: estRTT 0.353, ideal rate 9953.21 Kbps
C:\Users\alexia_perez\Desktop\HW 3.3\hw3\x64\Debug\hw3.exe (process 744) exited with code 0.
```

b. No loss (in either direction):

```
Microsoft Visual Studio Debug Console
Main: sender W = 300, RTT = 0.200 sec, loss 0 / 0, link 10 Mbps
Main: initializing DWORD array with 2^23 elements... done in 70 ms
Main: connected to s3.irl.cs.tamu.edu in 0.634 sec, packet size 1472 bytes
[ 2] B 64 ( 0.1 MB) N 128 T 0 F 0 W 64 S 0.375 Mbps RTT 0.525
[ 4] B 1505 ( 2.2 MB) N 1805 T 0 F 0 W 300 S 4.219 Mbps RTT 0.353
[ 6] B 3203 ( 4.7 MB) N 3503 T 0 F 0 W 300 S 3.314 Mbps RTT 0.353
[ 8] B 4903 ( 7.2 MB) N 5203 T 0 F 0 W 300 S 2.489 Mbps RTT 0.353
[10] B 6612 ( 9.7 MB) N 6912 T 0 F 0 W 300 S 2.002 Mbps RTT 0.353
[12] B 8316 (12.2 MB) N 8616 T 0 F 0 W 300 S 1.663 Mbps RTT 0.353
[14] B 10024 (14.8 MB) N 10324 T 0 F 0 W 300 S 1.429 Mbps RTT 0.354
[16] B 11722 (17.3 MB) N 12022 T 0 F 0 W 300 S 1.243 Mbps RTT 0.353
[18] B 13435 (19.8 MB) N 13735 T 0 F 0 W 300 S 1.115 Mbps RTT 0.353
[20] B 15138 (22.3 MB) N 15438 T 0 F 0 W 300 S 0.997 Mbps RTT 0.353
[22] B 16837 (24.8 MB) N 17137 T 0 F 0 W 300 S 0.904 Mbps RTT 0.353
[24] B 18546 (27.3 MB) N 18846 T 0 F 0 W 300 S 0.834 Mbps RTT 0.353
[26] B 20249 (29.8 MB) N 20549 T 0 F 0 W 300 S 0.767 Mbps RTT 0.353
[28] B 21956 (32.3 MB) N 22256 T 0 F 0 W 300 S 0.714 Mbps RTT 0.353
[30.12] <-- FIN-ACK 22920 D70096AB
Main: transfer finished in 28.353 sec, 9467.62 Kbps, checksum D70096AB
Main: estRTT 0.353, ideal rate 9940.30 Kbps
C:\Users\alexia_perez\Desktop\HW 3.3\hw3\x64\Debug\hw3.exe (process 4204) exited with code 0.
```

- c. As it can be seen, both cases do keep the same rate. It is possible that this may be caused by the fact that cumulative ACKs are used in TCP. Therefore, for the case where we have loss in the reverse direction (of 0.1), 3/300 packets will get lost (since window size is 300). Furthermore, if 3 ACKs are sent by the receiver and only the middle one is lost, then when the sender receives the 3rd ACK it automatically knows that the preceding packets must have been received, so the rate is maintained.

5. Determine the algorithm that the receiver uses to change its advertised window.

What name does this technique have in TCP? Hint: the receiver window does not grow to infinity and you need to provide its upper bound as part of the answer.

- The receiver is using TCP Flow Control in order to change its advertised window.
- Here, $\text{recvWin} = \text{recvBuf} - [\text{lastOrderedByteRecv} + \text{lastByteDelivered}]$
- The diagram below (taken from 3/10/22 lecture slides, page 21) illustrates why recvWin cannot grow to infinity (it is due to the fact that its upper bound is the recvBuf).

