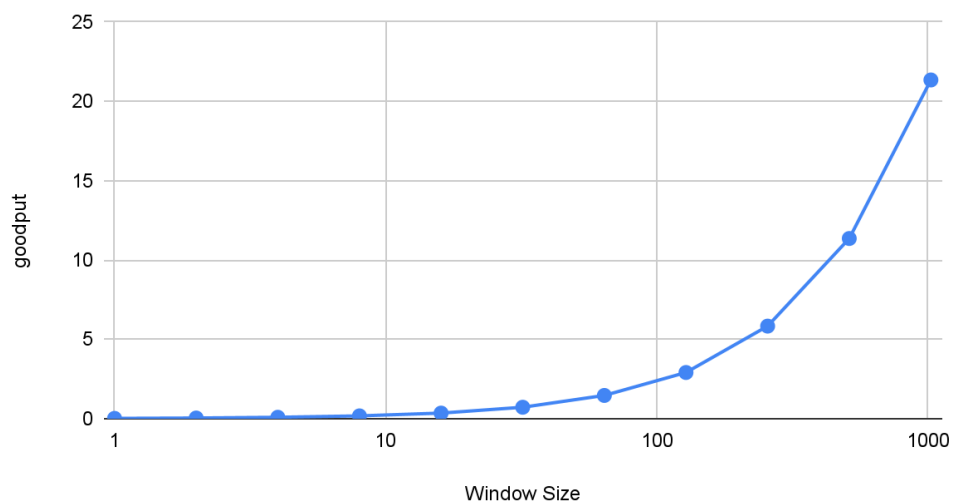


Joshua Clapp

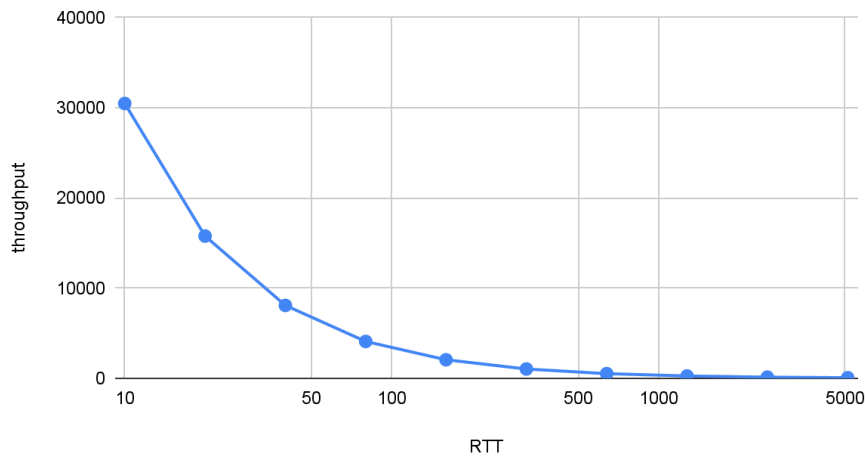
1. Set packet loss p to zero in both directions, the RTT to 0.5 seconds, and bottleneck link speed to $S = 1$ Gbps. Examine how your goodput scales with window size W . This should be done by plotting the steady-state rate $r(W)$ for $W = 1, 2, 4, 8, \dots, 210$ and keeping the x axis on a log-scale. 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.

Window Size v Goodput



- a.
 - b. Yes, this does match the theory discussed in class as there is no packet loss, and with a relatively high RTT, the chances for congestion at the router given these bounds is low. This means that the goodput will scale with the window size's exponential growth.
2. Expanding on the previous question, fix the window size at $W = 30$ packets and vary the RTT = 10, 20, 40, ..., 5120 ms. Plot stable rate $r(RTT)$, again placing the x-axis on a logscale. 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 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.

r(RTT) transfer kps vs. X



- a.
 - b. This matches the theory because when the overall estimation of delay increases, the throughput should decrease drastically. This effectively emulates a busy receiver that has things like long propagation, computing, and queueing delays.
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)?

```
Main: sender W = 8000, RTT 0.010 sec, loss 0 / 0, link 1000 Mbps
Main: initializing DWORD array with 2^25 elements... done in 256 ms
Main: connected to 127.0.0.1 in 0.002 sec, pkt size 1472 bytes
[ 2] B 64442 ( 94.3 MB) N 64442 T 0 F 0 W 8000 S 377.372 Mbps RTT 0.000
[3.463] <-- FIN-ACK 91679 window FC6FB7CB
Main: transfer finished in 2.814 sec, 381571.37 Kbps, checksum FC6FB7CB
Main: estRTT 0.001, ideal rate 98722184.640 Kbps
```

a.

i. CPU configuration

Device specifications

Aspire R5-471T

| | |
|---------------|---|
| Device name | DESKTOP-JVECLPU |
| Processor | Intel(R) Core(TM) i7-6500U CPU @ 2.50GHz 2.60 GHz |
| Installed RAM | 8.00 GB (7.88 GB usable) |
| Device ID | 60A01AEF-12EA-46FA-8EB8-1D309E130BCF |
| Product ID | 00325-97104-82212-AAOEM |
| System type | 64-bit operating system, x64-based processor |

ii. Pen and touch Touch support with 10 touch points


- iii. I was not able to exceed 1 gigabit, likely due to my 8 year old computer struggling to keep up with the load.

```
Main: sender W = 8000, RTT 0.010 sec, loss 0 / 0, link 10000 Mbps
Main: initializing DWORD array with 2^25 elements... done in 190 ms
Main: connected to 127.0.0.1 in 0.003 sec, pkt size 9000 bytes
[ 2] B 14243 ( 128.1 MB) N 14243 T 0 F 0 W 8000 S 83.407 Mbps RTT 0.000
[5.901] <-- FIN-ACK 14927 window FC6FB7CB
Main: transfer finished in 5.282 sec, 203283.19 Kbps, checksum FC6FB7CB
Main: estRTT 0.001, ideal rate 739866151.408 Kbps
```

b.

4. Use buffer size $2^{(23)}$ 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. 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 71 ms
Main: connected to s3.irl.cs.tamu.edu in 0.201 sec, pkt size 1472 bytes
[ 2] B 256 ( 0.4 MB) N 512 T 0 F 0 W 300 S 1.499 Mbps RTT 0.273
[ 4] B 1933 ( 2.8 MB) N 2233 T 0 F 0 W 300 S 5.657 Mbps RTT 0.353
[ 6] B 3631 ( 5.3 MB) N 3931 T 0 F 0 W 300 S 7.086 Mbps RTT 0.353
[ 8] B 5329 ( 7.8 MB) N 5629 T 0 F 0 W 300 S 7.800 Mbps RTT 0.353
[10] B 7027 (10.3 MB) N 7327 T 0 F 0 W 300 S 8.228 Mbps RTT 0.353
[12] B 8726 (12.8 MB) N 9026 T 0 F 0 W 300 S 8.515 Mbps RTT 0.353
[14] B 10424 (15.3 MB) N 10724 T 0 F 0 W 300 S 8.720 Mbps RTT 0.353
[16] B 12121 (17.7 MB) N 12421 T 0 F 0 W 300 S 8.872 Mbps RTT 0.353
[18] B 13822 (20.2 MB) N 14122 T 0 F 0 W 300 S 8.989 Mbps RTT 0.353
[20] B 15521 (22.7 MB) N 15821 T 0 F 0 W 300 S 9.089 Mbps RTT 0.353
[22] B 17221 (25.2 MB) N 17521 T 0 F 0 W 300 S 9.165 Mbps RTT 0.353
[24] B 18920 (27.7 MB) N 19220 T 0 F 0 W 300 S 9.229 Mbps RTT 0.353
[26] B 20619 (30.2 MB) N 20919 T 0 F 0 W 300 S 9.288 Mbps RTT 0.353
[28] B 22319 (32.7 MB) N 22619 T 0 F 0 W 300 S 9.334 Mbps RTT 0.353
[29.287] <-- FIN-ACK 22920 window D70096AB
Main: transfer finished in 28.707 sec, 9350.87 Kbps, checksum D70096AB
Main: estRTT 0.353, ideal rate 10009.555 Kbps
```

i.

- b. Zeros in Both Directions

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 71 ms
Main: connected to s3.irl.cs.tamu.edu in 0.202 sec, pkt size 1472 bytes
[ 2] B 256 ( 0.4 MB) N 512 T 0 F 0 W 300 S 1.499 Mbps RTT 0.273
[ 4] B 1935 ( 2.8 MB) N 2235 T 0 F 0 W 300 S 5.663 Mbps RTT 0.353
[ 6] B 3937 ( 5.8 MB) N 4237 T 0 F 0 W 300 S 7.683 Mbps RTT 0.353
[ 8] B 5634 ( 8.2 MB) N 5934 T 0 F 0 W 300 S 8.245 Mbps RTT 0.353
[10] B 7333 (10.7 MB) N 7633 T 0 F 0 W 300 S 8.585 Mbps RTT 0.353
[12] B 9031 (13.2 MB) N 9331 T 0 F 0 W 300 S 8.813 Mbps RTT 0.353
[14] B 10729 (15.7 MB) N 11029 T 0 F 0 W 300 S 8.971 Mbps RTT 0.353
[16] B 12428 (18.2 MB) N 12728 T 0 F 0 W 300 S 9.094 Mbps RTT 0.353
[18] B 14128 (20.7 MB) N 14427 T 0 F 0 W 300 S 9.188 Mbps RTT 0.353
[20] B 15827 (23.2 MB) N 16127 T 0 F 0 W 300 S 9.264 Mbps RTT 0.353
[22] B 17526 (25.7 MB) N 17826 T 0 F 0 W 300 S 9.329 Mbps RTT 0.353
[24] B 19226 (28.1 MB) N 19526 T 0 F 0 W 300 S 9.381 Mbps RTT 0.353
[26] B 20925 (30.6 MB) N 21225 T 0 F 0 W 300 S 9.422 Mbps RTT 0.353
[28] B 22624 (33.1 MB) N 22920 T 0 F 0 W 300 S 9.463 Mbps RTT 0.353
[29.288] <-- FIN-ACK 22920 window D70096AB
Main: transfer finished in 28.705 sec, 9351.52 Kbps, checksum D70096AB
Main: estRTT 0.353, ideal rate 10007.833 Kbps

```

- i.
 - c. The protocols both have almost the exact same rate 9350.87 Kbps versus 9351.52 Kbps. This as a result of the 10% chance of losing a ACK, and this does not really affect the throughput. Because the next largest ack will update the sender base, so as long as multiple in order acks are not lost, there is no effect on the throughput.
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.
- a. The receiver appears to permanently be in slow start, as it updates its window to be whatever the most recently acked packet was. If the acked sequence number was 24313, the window would become 24313. This is capped at 80,000 where the window becomes static.