Joe Montroy

Peter Dordal

COMP 343 - 002

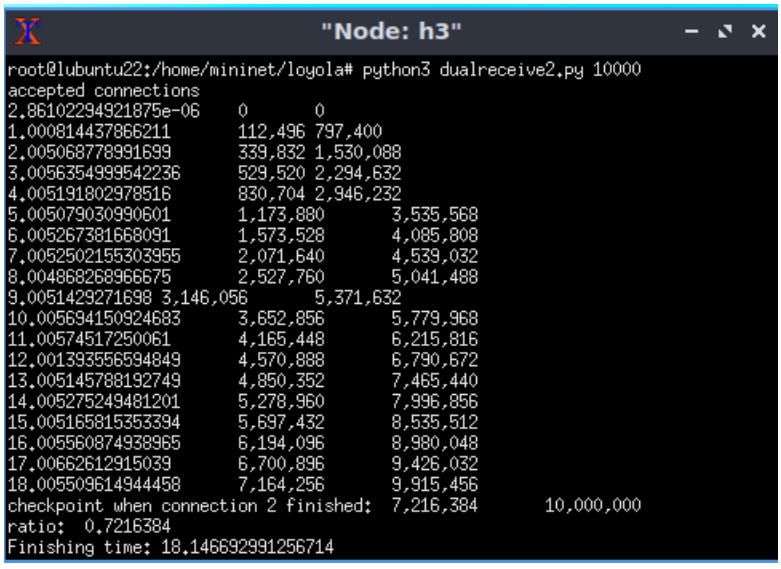
5 December 2025

Mininet Project

1. See attached pcapng file.
2. TCP Reno vs. TCP Cubic

I began by running python3 dualrecieve2.py on h3 and python3 sender.py 10000 10.0.3.10 5430 reno & python3 sender.py 10000 10.0.3.10 5431 cubic on H1. Reno is being sent to port 5430, and Cubic is being sent to 5431. I ran this, changing the delay between r —- h3 between 50ms, 100ms, and 200ms. For each case, I ran it 3+ times because nothing can be proven if it is only run once.

For a delay of 50ms, here are the runs:



Reno completed 72% of the blocks and had the fraction of 7,216,804/10,000,000 or .7216. Cubic has a large lead the whole way.

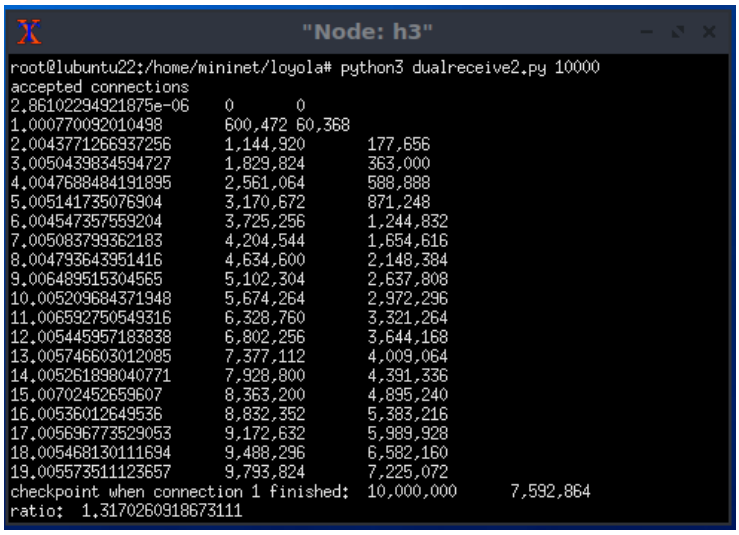
```
root@lubuntu22:/home/mininet/loyola# python3 dualreceive2.py 10000
accepted connections
2.1457672119140625e-06  0       0
1.000708341598507       99,464  820,568
2.0083813667297363      316,664 1,566,288
3.0076074600219727      704,728 2,131,008
4.005077600479126       1,223,112       2,566,856
5.000635385513306       1,658,960       3,070,760
6.004839181900024       2,171,552       3,518,192
7.007539510726929       2,653,736       3,971,416
8.007882595062256       3,192,392       4,402,920
9.008636713027954       3,735,392       4,764,920
10.0082688331604        4,274,048       5,232,624
11.0077064037323        4,673,696       5,782,864
12.005748987197876      5,129,816       6,280,976
13.003790855407715      5,674,264       6,680,624
14.005111932754517      6,155,000       7,154,120
15.00862741470337       6,632,840       7,637,752
16.007442474365234      7,007,872       8,215,504
17.005460262298584      7,282,992       8,896,064
18.006515502929688      7,445,168       9,677,984
checkpoint when connection 2 finished:  7,546,528       10,000,000
ratio:  0.7546528
```

The ratio was similar, with Reno only completing 75% of the block transfer in the time it took Reno to do 100%. It's a fraction of 7,546,528/10,000,000 or 0.7546. Cubic again took a large lead in the beginning, began to lose it a little bit, then took it back

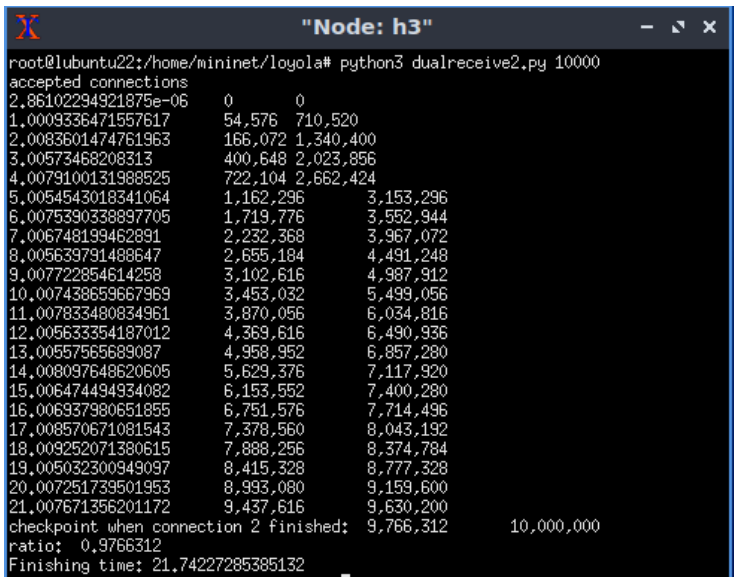
```
root@lubuntu22:/home/mininet/loyola# python3 dualreceive2.py 10000
accepted connections
2.384185791015625e-06   0       0
1.0006580352783203      145,800 795,952
2.007521629333496       383,272 1,518,504
3.006736993789673       633,776 2,220,784
4.007447242736816       1,008,808       2,798,536
5.005985260009766       1,354,880       3,405,248
6.005052804946899       1,679,232       3,988,792
7.0071635246276855      2,071,640       4,552,064
8.005016803741455       2,548,032       5,013,976
9.0074157939148         2,975,192       5,543,944
10.008383512496948      3,440,000       6,026,128
11.007526636123657      3,922,184       6,482,248
12.008017539978027      4,411,608       6,957,192
13.007063388824463      4,918,408       7,406,072
14.004930973052979      5,326,744       7,949,072
15.008707046508789      5,635,168       8,602,120
16.0070903301239        6,105,768       9,082,856
17.008206844329834      6,485,144       9,657,712
checkpoint when connection 2 finished:  6,695,104       10,000,000
ratio:  0.6695104
Finishing time: 17.58425998687744
root@lubuntu22:/home/mininet/loyola#
```

The third trial was similar to the other two. Reno completed 66.9% of the transfer in the time it took Cubic. The fraction was 6,695,104/10,000,000 or 0.6695. Cubic took a large lead and increased it as the round went on. It is safe to say that Cubic is the faster TCP protocol with a bandwidth of 50ms. For Reno, the average throughput was roughly 390,000 bytes/390 blocks per second, where Cubics was 531,000 bytes/ 531 blocks per second. I calculated this by adding the sum of all of the second-to-last values and dividing it by the sum of seconds that value was recorded at.

Next is 100ms, I ran 4 tests because the results were more varied:

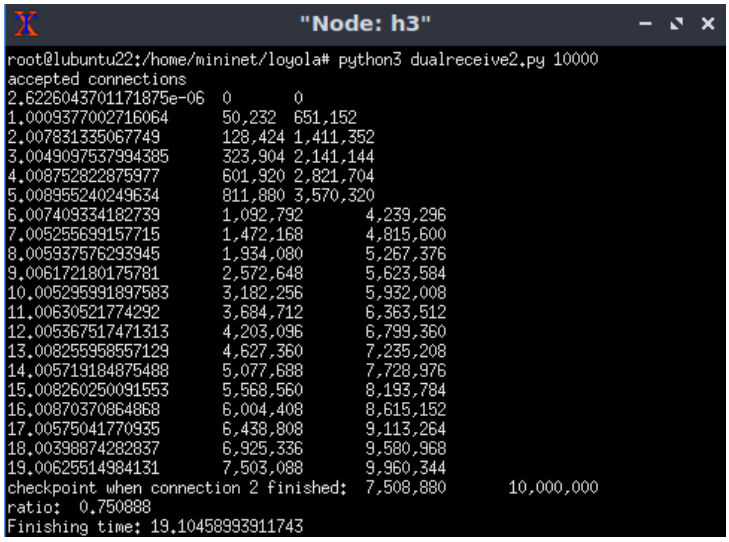


Reno jumps out to a large lead and holds onto it for the rest of the round. Cubic only completed 76% of its transfer by the time Reno finished, and the fraction was 10,000,000/7,592,864 or 1.317.



In this one, Cubic jumps out to a massive lead in the first few seconds, but then slowly loses it as Reno slowly crawls its way back

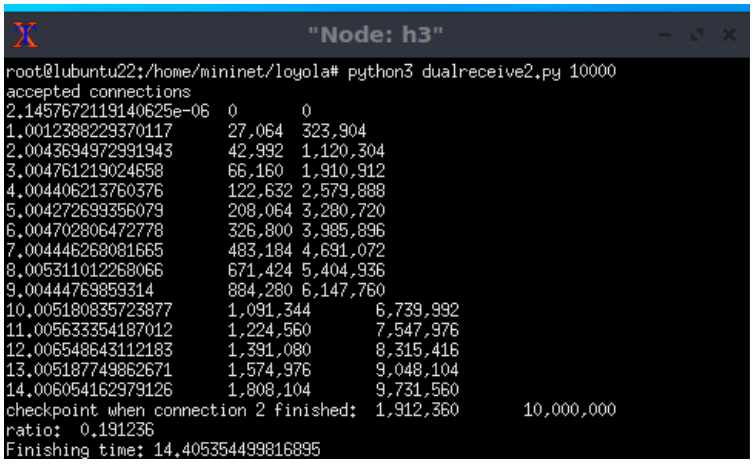
over the course of the test. Cubic barely held on in the end as Reno completed 97% of its transfer. If they were transferring 15000 blocks, Reno would have taken the cake.

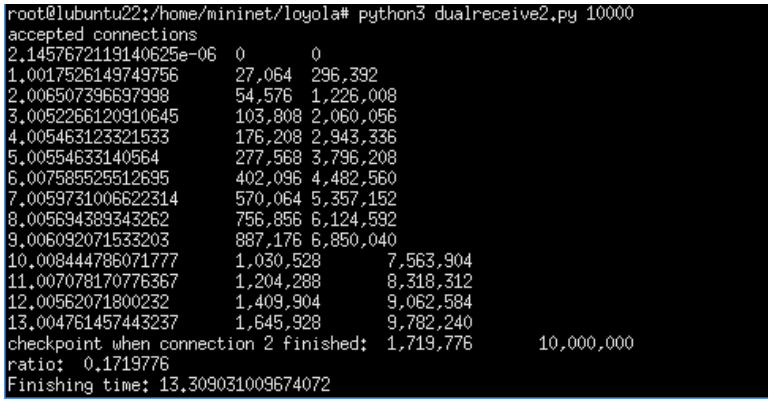


Similarly to test #2, Cubic creates a large gap, but it is slowly closed by Reno, not in time to beat Cubic, but it completed 75% of its transfer in the time it took Cubic to complete.

The conclusion I came to for a 100ms delay is that there is no clear winner. For 50ms, it seemed like Cubic won most of the time, but for 100ms, it flip-flopped back and forth. I ran probably 10 tests total, and Reno would win one and Cubic the next. This is due to the phase effects that change the throughput of the transfer, essentially randomizing it every time. It's also reflected in the throughput data; the average throughput of the tests that I measured was Reno 453,000 bytes/ 453 blocks per second and Cubic 454,000 bytes/ 454 blocks per second. Which is nearly identical to one another.

RTT 200ms:


"Node: h3"
root@lubuntu22:/home/mininet/loyola# python3 dualreceive2.py 10000
accepted connections
2.1457672119140625e-06 0 0
1.0012388229370117 27,064 323,904
2.0043694972991943 42,992 1,120,304
3.004761219024658 66,160 1,910,912
4.004406213760376 122,632 2,579,888
5.004272699356079 208,064 3,280,720
6.004702806472778 326,800 3,985,896
7.004446268081665 483,184 4,691,072
8.005311012268066 671,424 5,404,936
9.00444769859314 884,280 6,147,760
10.005180835723877 1,091,344 6,739,992
11.005633354187012 1,224,560 7,547,976
12.006548643112183 1,391,080 8,315,416
13.005187749862671 1,574,976 9,048,104
14.006054162979126 1,808,104 9,731,560
checkpoint when connection 2 finished: 1,912,360 10,000,000
ratio: 0.191236
Finishing time: 14.405354499816895


root@lubuntu22:/home/mininet/loyola# python3 dualreceive2.py 10000
accepted connections
2.1457672119140625e-06 0 0
1.0017526149749756 27,064 296,392
2.006507396697998 54,576 1,226,008
3.0052266120910645 103,808 2,060,056
4.005463123321533 176,208 2,943,336
5.00554633140564 277,568 3,796,208
6.007585525512695 402,096 4,482,560
7.0059731006622314 570,064 5,357,152
8.005694389343262 756,856 6,124,592
9.006092071533203 887,176 6,850,040
10.008444786071777 1,030,528 7,563,904
11.00708170776367 1,204,288 8,318,312
12.00562071800232 1,409,904 9,062,584
13.004761457443237 1,645,928 9,782,240
checkpoint when connection 2 finished: 1,719,776 10,000,000
ratio: 0.1719776
Finishing time: 13.309031009674072


root@lubuntu22:/home/mininet/loyola# python3 dualreceive2.py 10000
accepted connections
2.384185791015625e-06 0 0
1.0008759498596191 34,304 300,736
2.005389928817749 51,680 1,155,056
3.006990671157837 76,296 2,120,872
4.0066845417022705 124,080 3,020,080
5.006129741668701 197,928 3,900,464
6.005078554153442 300,736 4,748,992
7.0048744678497314 436,848 5,501,952
8.00531530380249 583,096 6,301,248
9.008179664611816 701,832 7,012,216
10.007819175720215 822,016 7,753,592
11.005654096603394 987,088 8,523,928
12.007576942443848 1,170,984 9,295,712
checkpoint when connection 2 finished: 1,375,152 10,000,000
ratio: 0.1375152
Finishing time: 12.956084966659546

I’m going to make this short and sweet. Cubic dominated the entire 200ms competition, usually winning by a factor of 9 or 10. The throughput for Reno was 116,000 bytes/ 116 blocks per second, and Cubic was 739,000 bytes/ 739 blocks per second, which was the highest out of all of the delay times. It is clear that with a 200ms delay, Cubic is the way to go.