Program 3 (TCP Analysis)

Reza Naeemi

Documentation

This assignment looks at behavior of the TCP protocol through a number of experiments. The Professor provided hw3 executable software is utilized as a basis for drawing a TCP state transition diagram as well as a corresponding timing chart. A custom version of the network traffic generator program "ttcp" is also used to determine how throughput is affected by setting a number of different parameters while in use, including message size, number of messages transferred, socket buffer size, and use of the Nagle's algorithm. Throughout testing, the Linux commands tcpdump, netstat, and strace are used to observe how TCP segments are transmitted.

TCP Analysis

Determine the TCP network interactions of the Professor provided hw3 executable software using tcpdump. To perform this test, three xterm sessions were opened on two separate Linux lab computers where data was captured using a network sniffer running on the machine acting as a server.

Methodology:

- On server: tcpdump -vtt host clientHostName and port serverPort and tcp >& tcpdump.test1.out
- On server: ./hw3 serverPort
- On client: ./hw3 serverPort serverHostName

Data captured from server tcpdump output to file from a single client execution.

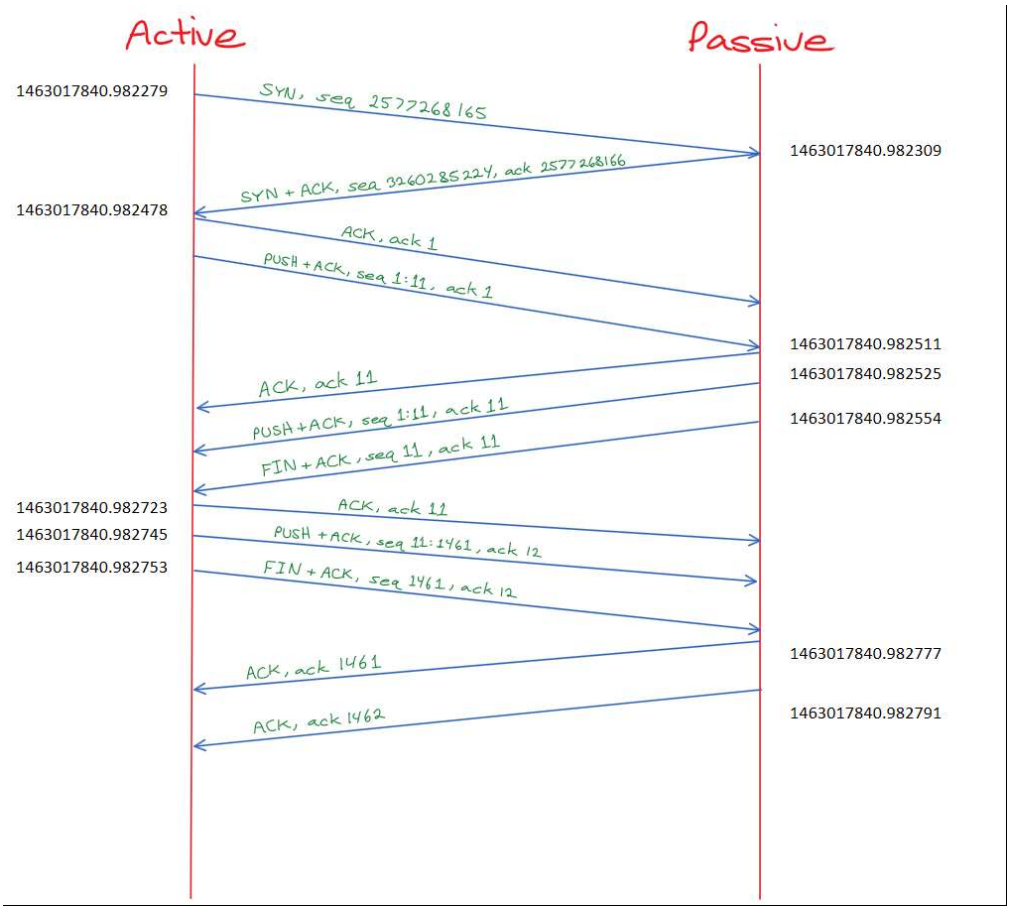


Figure 1: Timing Chart for Professor's hw3 executable

The timing chart in Figure 1 is derived from the tcpdump output and shows a connection initiated by the client to a server. The client sends 10 bytes, which the server returns. The server initiates a disconnection, but reads in 1450 bytes before the connection terminates. The raw tcpdump output is included in the attached file, tcpdump.test1.out.

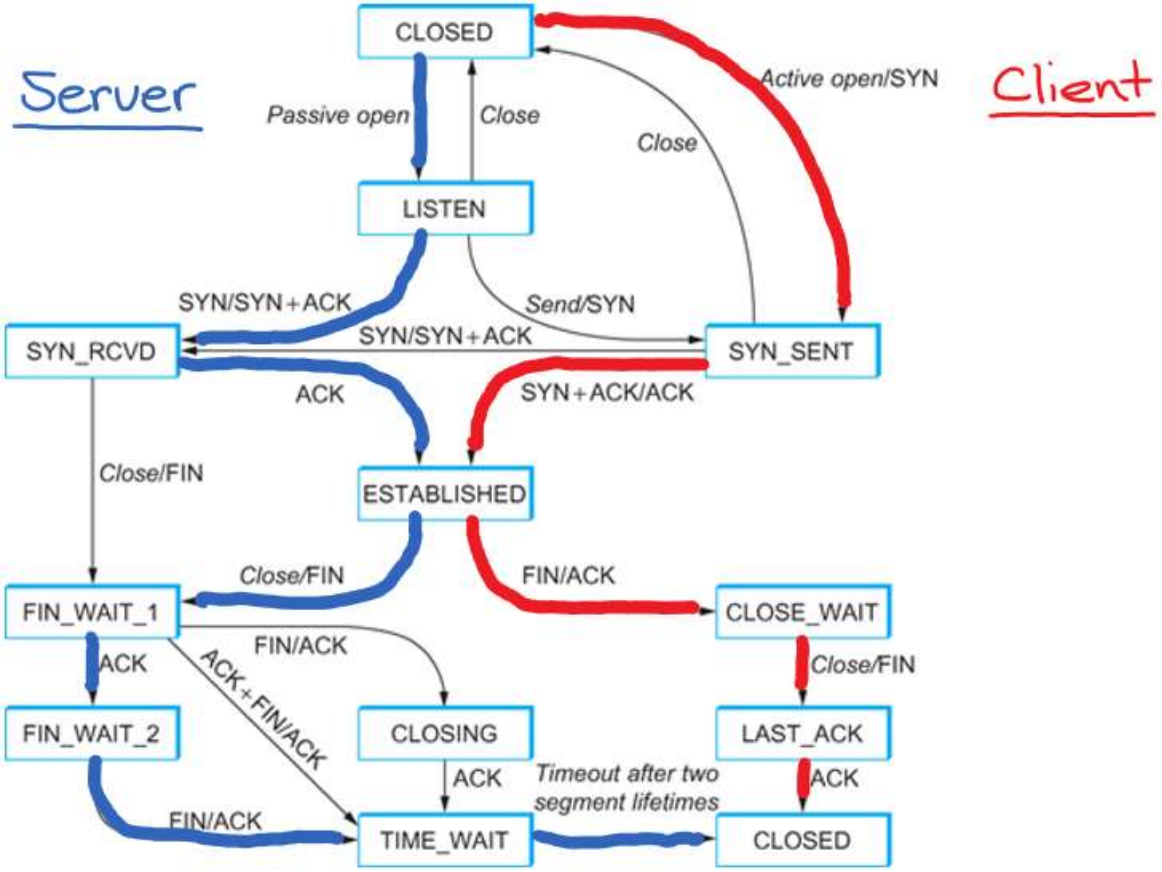


Figure 2: State Transition Diagram for Professor's hw3 executable

The state transition diagram in Figure 2 maps the actions found on the timing chart. Note that the server initiated shutdown and enters wait states. Red lines represent state changes by the client (sender) and blue lines represent state changes by the server (receiver).

Performance Results

A series of network tests were performed using the Professor provided copy of ttcp, a utility for testing network throughput. For all tests, the total data transmitted was 67108864 bytes (64MB) on the UWB's Linux lab 1Gbps LAN. Tests were performed on a variety of terminals over the course of several days. Attempts were made to perform network tests at off-hour times utilizing similar network conditions. To eliminate variance in network traffic and to accurately represent differences in throughput with different parameters set on ttcp, most tests are performed with 10 trials with the average of those trials graphed and reported. In total, 202 network tests were performed using bash scripts to gather data for the four testing regimens below.

In several testing regimens, the throughput test is performed twice: once without the -D flag, and once with the -D flag. On the ttcp program, the -D flag disables buffer TCP writes setting the TCP_NODELAY socket option which disables Nagle's algorithm. This algorithm interacts badly with TCP delayed acknowledgments, a feature introduced into TCP at roughly the same time in the early 1980s, but by a different group. With both algorithms enabled, applications that do two successive writes to a TCP connection, followed by a read that will not be fulfilled until after the data from the second

write has reached the destination, experience a constant delay of up to 500 milliseconds, the "ACK delay" (source: Wikipedia, Nagle's Algorithm). For each of these testing cases, a performance comparison is shown for 'without -D' and 'with -D' on the same graph to indicate how the algorithm affects performance under similar conditions. To minimize impact of changing network conditions, tests were scripted to collect data in quick succession using averages where possible.

Test 2 Data and Analysis

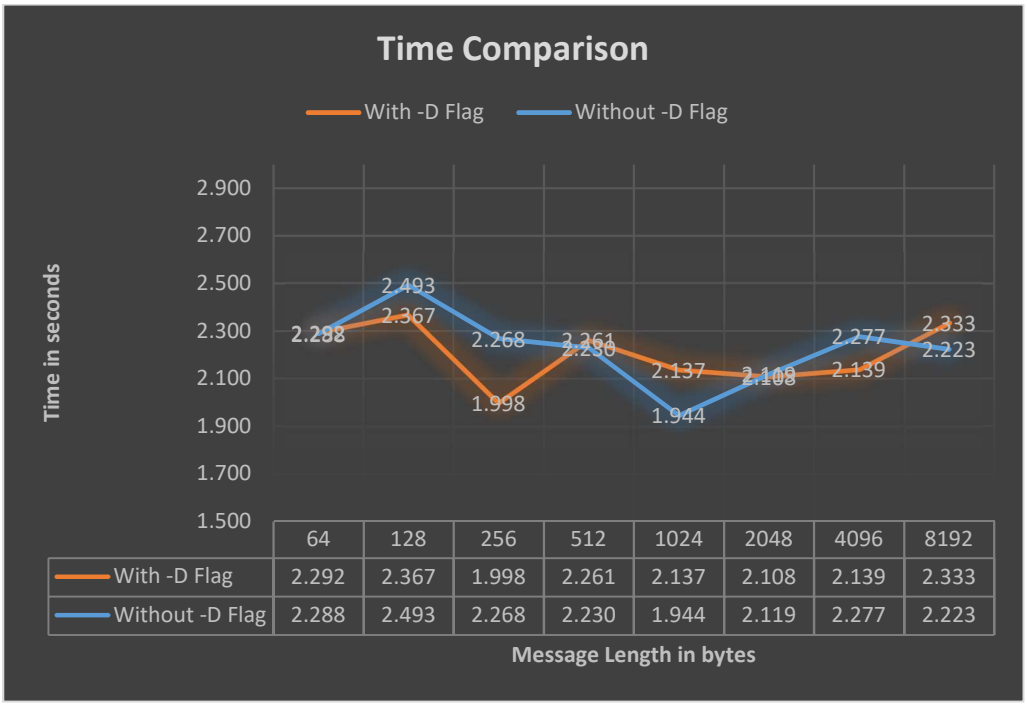
Run ttcp on any two of UW1-320's machines, (i.e., uw1-320-00 ~ uw1-320-15) in the following test cases without and with -D option.

Methodology:

- On server: ./ttcp -r -pServerPort
- On client: ./ttcp -t -lLength -nMessages serverHostName -pServerPort
  - where length was 64, 128, 256, 512, 1024, 2048, 4096, 8192, and
  - where messages was 1048576, 524288, 262144, 131072, 65536, 32768, 16384, 8192

Data captured from client output to typescript file with 10 client executions per length/messages pair. Averages of each length/messages pair are reported for received bytes.

Both server and client were simultaneously run with and without the -D flag to show the difference of disabling buffering of TCP writes.



	64	128	256	512	1024	2048	4096	8192
With -D Flag	2.292	2.367	1.998	2.261	2.137	2.108	2.139	2.333
Without -D Flag	2.288	2.493	2.268	2.230	1.944	2.119	2.277	2.223

Figure 3: Test 2 Speed Comparison (higher is better)

This test looks at three variables: buffer length (in bytes), number of messages sent, and Nagle's algorithm. In all cases, length * messages = 64MB of data transmitted. There is a clear trend showing that setting TCP_NODELAY (disabling Nagle's algorithm by setting the 'with -D' option) improves network performance slightly in all cases except for when the buffer length was 4096. Using TCP_NODELAY had the best results when the buffer size was small or large, but network throughput was the lowest at a 1024 byte message size. Results were also similar to 512 bytes, which is a close multiple of 1024. Since the MTU on the network is likely 1500 and other tests showed that the maximum segment size (MSS) was being set as 1460 bytes, it might be that this performance drop-off was due to inefficiency caused by IP fragmentation and overhead management. A combination of the current algorithm for TCP delay acknowledgements or the gigabit network is working in the favor of disabling Nagle's algorithm.

Enabling write buffering (Nagle's algorithm, using 'without -D') shows maximum throughput where message sizes were 128 bytes and 4096 bytes. While these results are consistent due to the nature of repeated tests, they may not be consistent in all environments and instead unique to this network and these current conditions. Throughout the range of message lengths, throughput was inconsistent showing that tuning is necessary to get good speeds. A variety of different conditions beyond the Linux lab would have to be introduced to show that using Nagle's algorithm is the better option, but my results do not show this to be true on this LAN.

Test 3 Data and Analysis

Run ttcp in the following particular test case without -D while running tcpdump on another xterm. Check from the tcpdump output if TCP maximum segment size (MSS) is 1460 bytes or not.

Methodology:

- On server: tcpdump -vtt host clientHostName and port ServerPort and tcp >& tcpdump.test3.out
- On server: ./ttcp -r -pServerPort
- On client: ./ttcp -t -l32768 -n2048 serverHostName -pServerPort

Data captured from server tcpdump output to file from a single client execution.

- Filter: cat tcpdump.test3.out | grep "serverHostName.port >"

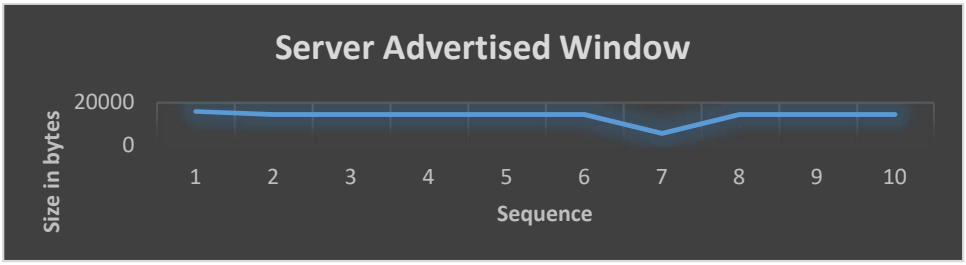


Figure 4: Test 3 Server Advertised Window Analysis (larger and steady is better)

The maximum segment size (MSS) was reported to be 1460 bytes in the tcpdump output.

From the tcpdump output, it can be seen that the server's TCP advertised window is mostly steady at 14480 bytes. Interestingly, it is initially advertised at 15924 bytes and drops to its lowest point of 2896 bytes before stabilizing. Throughout the sequence, it does drop to 5792 bytes sporadically before returning immediately to 14480 bytes. In almost all cases, advertised size is extremely close to scalar multiples of 1448 bytes even though the maximum segment size is never reported as such. The most frequent hits are the steady state size of 14480 bytes (10 x 1448) and the reduced size 5792 bytes (4 x 1448). Other values are off by a few bytes in either direction, but more closely match multiples of 1448 and not 1460.

As observed, the advertised window never matches the additive increment or slow start algorithm. It can be concluded that this version of ttcp is using its own algorithm.

Test 4 Data and Analysis

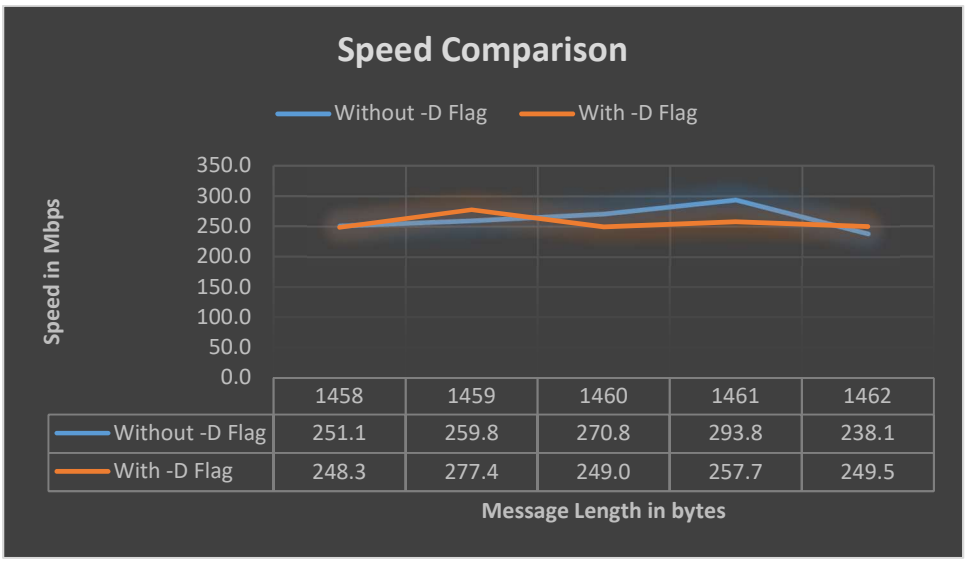
Run ttcp in the following 5 test cases where -l option is from 1458 to 1462 with and without -D option.

Methodology:

- On server: ./ttcp -r -pServerPort
- On client: ./ttcp -t -lLength -nMessages serverHostName -pServerPort
  - where length was 1458, 1459, 1460, 1461, 1462, and
  - where messages was 46028, 45996, 45965, 45934, 45902

Data captured from client output to typescript file with 10 client executions per length/messages pair. Averages of each length/messages pair are reported for received bytes.

Both server and client were simultaneously run with and without the -D flag to show the difference of disabling buffering of TCP writes.



	1458	1459	1460	1461	1462
Without -D Flag	251.1	259.8	270.8	293.8	238.1
With -D Flag	248.3	277.4	249.0	257.7	249.5

Figure 5: Test 4 Speed Comparison (higher is better)

In contrast to the wide range of message lengths analyzed in test 2, this test looks at a very limited range that matches the maximum segment size (MSS) utilized by ttcp. Another contrast is the reversal of performance characteristics of TCP buffering for these particular message lengths. Where results 'with -D' were better between message sizes 1024 bytes to 2048 bytes in test 2, when operating at or near the MSS, results 'without -D' are marginally better in performance. The best performance for test 4 was recorded when message length was equal to the maximum segment size of 1460 bytes, where throughput was 228.3 +/- 34.7 Mbps (Without -D) and 220.1 +/- 20.8 Mbps (With -D) respectively, which closely matched the best performance recorded in test 2. While there wasn't a large difference between using Nagle's algorithm in this case, the network is most efficient when message size is equivalent to segment size, however exceeding MSS in both cases results in a heavy (and immediate) performance drop.

Test 5 Data and Analysis

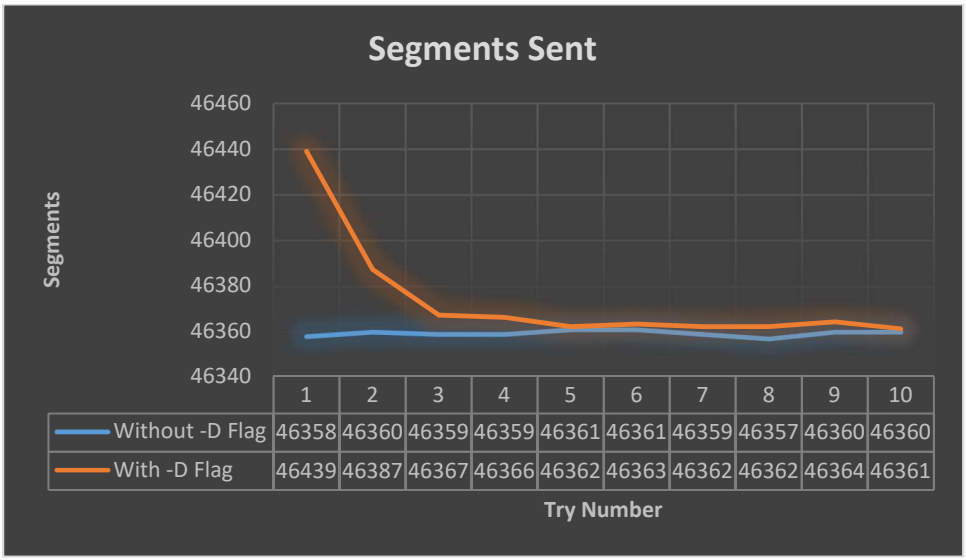
Part a) Run ttcp in the following particular test case with and without -D option. Run netstat right before and after each execution of ttcp to count the tcp packets sent, received, and retransmitted.

Methodology:

- On server: ./ttcp -r
- On client: netstat -st | grep segments
- On client: ./ttcp -t -l64 -n1048576 serverHostName -pServerPort
- On client: netstat –st | grep segments

Data captured from client output to typescript file with 10 client executions on the fixed length/messages pair. Data is represented as the difference of segments recorded before and after each ttcp execution. No averaging or smoothing is applied.

Both server and client were simultaneously run with and without the -D flag to show the difference of disabling buffering of TCP writes.



	1	2	3	4	5	6	7	8	9	10
Without -D Flag	46358	46360	46359	46359	46361	46361	46359	46357	46360	46360
With -D Flag	46439	46387	46367	46366	46362	46363	46362	46362	46364	46361

Figure 6: Segments Transmitted During ttcp Executions

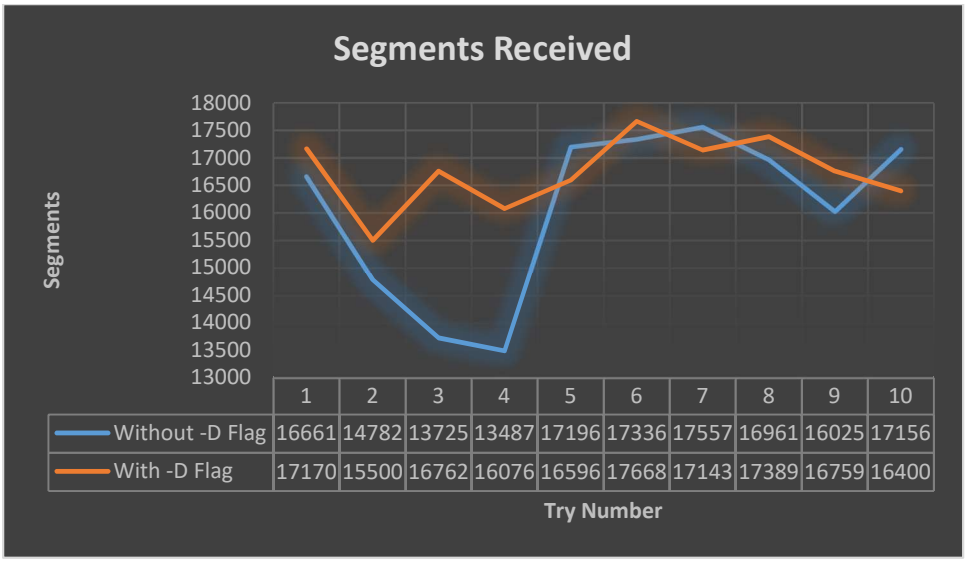


Figure 7: Segments Received During ttcp Executions

There were no retransmissions captured during the 10 client executions.

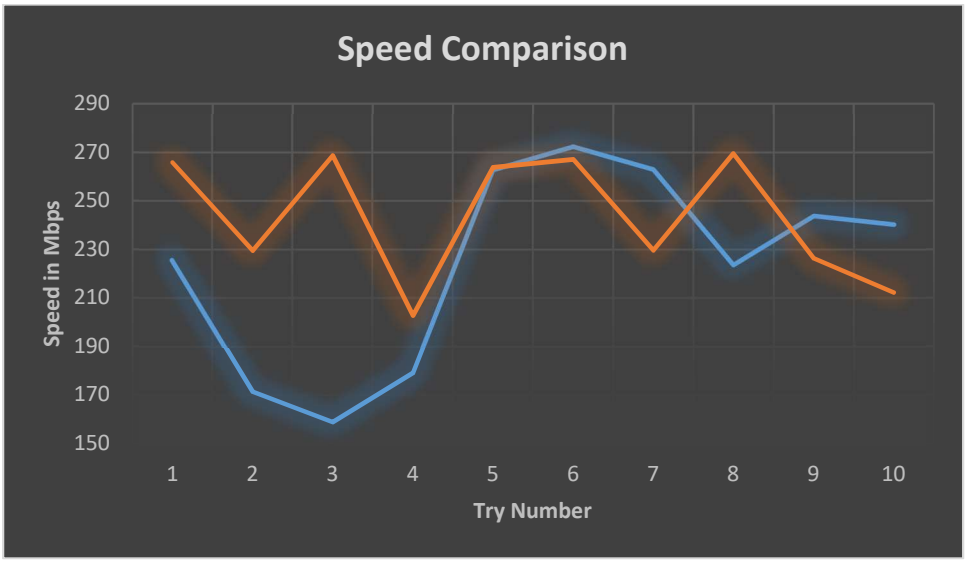


Figure 8: Throughput Comparison Associated with Segments Transmitted and Received (higher is better)

Unlike previous tests, this test shows a single message size and raw results of individual test without averaging. Figures 6, 7, and 8 also show the relationships between segments transmitted and received, and the relationship to throughput. It can be seen in Figure 6 that while number of segments sent stabilize over time from the perspective of the client, the number of segments received varies even in the case where there were no retransmissions. Further, this appears to have a relationship on the overall throughput where the shape of the graph for received segments closely mirrors the graph for speed comparison for runs without -D and with -D. It should also be noted that it is not a direct relationship, as the 'without -D' test shows a decrease in segments received in the 3, 4 interval where throughput speed increases in the same interval.

Part b) Run "strace -ttT ttcp" in the following particular test case without and with -D option.

Methodology:

- On server: ./ttcp -r -pServerPort
- On client: strace -ttT ./ttcp -t -l64 -n1048576 serverHostName -pServerPort >& strace.test5.out

Data captured from client strace dump output to file from a single client execution.

Both server and client were simultaneously run with and without the -D flag to show the difference of disabling buffering of TCP writes.

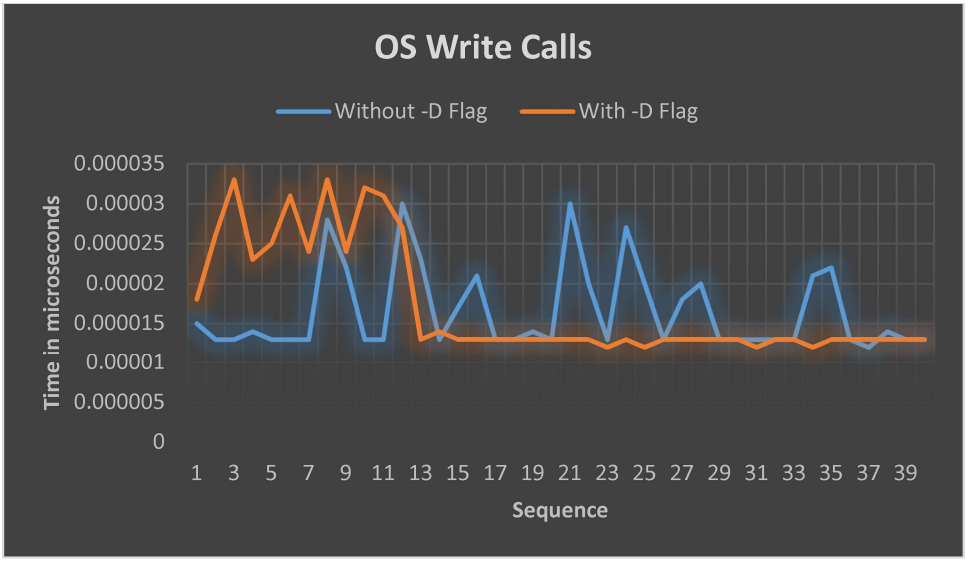


Figure 9: Time Reported Through strace for First 40 OS Write Calls on First Trial (lower is better)

The first 40 segments of the first client execution was monitored to record the duration of system write calls using strace. This depicts overhead of the system with I/O interactions using ttcp 'without -D' and 'with -D'. These results show there is a lot of additional wait states associated using TCP_NODELAY (with -D) when the connection is first established but does not take long to become stable with a 13 microsecond latency. On the other hand, while running ttcp without -D (using Nagle's algorithm) also focused around the 13 microsecond latency mark for write calls, there was repeated spikes where write operation times would more than double. In test 2 where message length was also 64 bytes, performance 'with -D' was superior to 'without -D,' showing that this increased average latency 'without -D' does have an adverse impact over time.

Conclusions

On this particular gigabit LAN, the difference between message lengths with and without TCP buffering was very small, never exceeding a difference of 20 Mbps of throughput. These results may not be typical to real world use where latency and network stability is much more varied. However, even on a stable LAN, the same testing conditions would reveal wildly different results for each client execution. It was apparent early on that it would be necessary to use large sample sizes to get results that weren't random and meaningless. Test 5 in particular shows how network throughput can change from one client execution to the next, ranging from as low as 158.7 Mbps to as high as 262.9 Mbps. For these tests, the application of TCP_NODELAY did yield positive results in nearly every test case. However, the relationship between segments received and throughput, especially in the case of no retransmissions, shows that actions happening at the lower network layers have dramatic effect on overall speed and transmission times, and network tuning is not just limited to changing message length at the TCP level.