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.

Test 1 Execution

Based on your tcpdump results, draw a transition diagram as shown on textbook diagram 5.7. Trace the diagram for the hw3 client and server respectively. (Draw thick red lines along appropriate arrows for the client and think blue lines for the server.)

* hw3\_demo.cpp tcp dump

tcpdump: listening on eno1, link-type EN10MB (Ethernet), capture size 262144 bytes

1557465209.296342 IP (tos 0x0, ttl 64, id 4208, offset 0, flags [DF], proto TCP (6), length 60)

172.21.198.80.39758 > 172.21.198.81.8681: Flags [S], cksum 0xbe16 (correct), seq 3315030483, win 29200, options [mss 1460,sackOK,TS val 214321571 ecr 0,nop,wscale 7], length 0

1557465209.296353 IP (tos 0x0, ttl 64, id 0, offset 0, flags [DF], proto TCP (6), length 60)

172.21.198.81.8681 > 172.21.198.80.39758: Flags [S.], cksum 0xe4fb (incorrect -> 0x5774), seq 3804538489, ack 3315030484, win 28960, options [mss 1460,sackOK,TS val 629520573 ecr 214321571,nop,wscale 7], length 0

1557465209.296503 IP (tos 0x0, ttl 64, id 4209, offset 0, flags [DF], proto TCP (6), length 52)

172.21.198.80.39758 > 172.21.198.81.8681: Flags [.], cksum 0xf67b (correct), ack 1, win 229, options [nop,nop,TS val 214321571 ecr 629520573], length 0

1557465209.296514 IP (tos 0x0, ttl 64, id 4210, offset 0, flags [DF], proto TCP (6), length 62)

172.21.198.80.39758 > 172.21.198.81.8681: Flags [P.], cksum 0xe524 (correct), seq 1:11, ack 1, win 229, options [nop,nop,TS val 214321571 ecr 629520573], length 10

1557465209.296518 IP (tos 0x0, ttl 64, id 7879, offset 0, flags [DF], proto TCP (6), length 52)

172.21.198.81.8681 > 172.21.198.80.39758: Flags [.], cksum 0xe4f3 (incorrect -> 0xf673), ack 11, win 227, options [nop,nop,TS val 629520573 ecr 214321571], length 0

1557465209.296547 IP (tos 0x0, ttl 64, id 7880, offset 0, flags [DF], proto TCP (6), length 62)

172.21.198.81.8681 > 172.21.198.80.39758: Flags [P.], cksum 0xe4fd (incorrect -> 0xe51c), seq 1:11, ack 11, win 227, options [nop,nop,TS val 629520573 ecr 214321571], length 10

1557465209.296557 IP (tos 0x0, ttl 64, id 7881, offset 0, flags [DF], proto TCP (6), length 52)

172.21.198.81.8681 > 172.21.198.80.39758: Flags [F.], cksum 0xe4f3 (incorrect -> 0xf668), seq 11, ack 11, win 227, options [nop,nop,TS val 629520573 ecr 214321571], length 0

1557465209.296673 IP (tos 0x0, ttl 64, id 4211, offset 0, flags [DF], proto TCP (6), length 52)

172.21.198.80.39758 > 172.21.198.81.8681: Flags [.], cksum 0xf667 (correct), ack 11, win 229, options [nop,nop,TS val 214321571 ecr 629520573], length 0

1557465209.296724 IP (tos 0x0, ttl 64, id 4212, offset 0, flags [DF], proto TCP (6), length 1502)

172.21.198.80.39758 > 172.21.198.81.8681: Flags [P.], cksum 0xea9d (incorrect -> 0x7f07), seq 11:1461, ack 12, win 229, options [nop,nop,TS val 214321571 ecr 629520573], length 1450

1557465209.296728 IP (tos 0x0, ttl 64, id 4214, offset 0, flags [DF], proto TCP (6), length 52)

172.21.198.80.39758 > 172.21.198.81.8681: Flags [F.], cksum 0xf0bb (correct), seq 1461, ack 12, win 229, options [nop,nop,TS val 214321571 ecr 629520573], length 0

1557465209.296743 IP (tos 0x0, ttl 64, id 7882, offset 0, flags [DF], proto TCP (6), length 52)

172.21.198.81.8681 > 172.21.198.80.39758: Flags [.], cksum 0xe4f3 (incorrect -> 0xf0a7), ack 1461, win 249, options [nop,nop,TS val 629520574 ecr 214321571], length 0

1557465209.296750 IP (tos 0x0, ttl 64, id 7883, offset 0, flags [DF], proto TCP (6), length 52)

172.21.198.81.8681 > 172.21.198.80.39758: Flags [.], cksum 0xe4f3 (incorrect -> 0xf0a6), ack 1462, win 249, options [nop,nop,TS val 629520574 ecr 214321571], length 0

12 packets captured

12 packets received by filter

0 packets dropped by kernel

* hw3.cpp tcp dump

tcpdump: listening on eno1, link-type EN10MB (Ethernet), capture size 262144 bytes

1557465075.296282 IP (tos 0x0, ttl 64, id 24869, offset 0, flags [DF], proto TCP (6), length 60)

172.21.198.80.39728 > 172.21.198.81.8681: Flags [S], cksum 0x378b (correct), seq 2165221346, win 29200, options [mss 1460,sackOK,TS val 214288071 ecr 0,nop,wscale 7], length 0

1557465075.296297 IP (tos 0x0, ttl 64, id 0, offset 0, flags [DF], proto TCP (6), length 60)

172.21.198.81.8681 > 172.21.198.80.39728: Flags [S.], cksum 0xe4fb (incorrect -> 0xd8a8), seq 3819643055, ack 2165221347, win 28960, options [mss 1460,sackOK,TS val 629487073 ecr 214288071,nop,wscale 7], length 0

1557465075.296478 IP (tos 0x0, ttl 64, id 24870, offset 0, flags [DF], proto TCP (6), length 52)

172.21.198.80.39728 > 172.21.198.81.8681: Flags [.], cksum 0x77b0 (correct), ack 1, win 229, options [nop,nop,TS val 214288071 ecr 629487073], length 0

1557465075.296491 IP (tos 0x0, ttl 64, id 24871, offset 0, flags [DF], proto TCP (6), length 62)

172.21.198.80.39728 > 172.21.198.81.8681: Flags [P.], cksum 0x779e (correct), seq 1:11, ack 1, win 229, options [nop,nop,TS val 214288071 ecr 629487073], length 10

1557465075.296497 IP (tos 0x0, ttl 64, id 5457, offset 0, flags [DF], proto TCP (6), length 52)

172.21.198.81.8681 > 172.21.198.80.39728: Flags [.], cksum 0xe4f3 (incorrect -> 0x77a8), ack 11, win 227, options [nop,nop,TS val 629487073 ecr 214288071], length 0

1557465075.296699 IP (tos 0x0, ttl 64, id 5458, offset 0, flags [DF], proto TCP (6), length 62)

172.21.198.81.8681 > 172.21.198.80.39728: Flags [P.], cksum 0xe4fd (incorrect -> 0x7795), seq 1:11, ack 11, win 227, options [nop,nop,TS val 629487074 ecr 214288071], length 10

1557465075.296712 IP (tos 0x0, ttl 64, id 5459, offset 0, flags [DF], proto TCP (6), length 52)

172.21.198.81.8681 > 172.21.198.80.39728: Flags [F.], cksum 0xe4f3 (incorrect -> 0x779c), seq 11, ack 11, win 227, options [nop,nop,TS val 629487074 ecr 214288071], length 0

1557465075.296870 IP (tos 0x0, ttl 64, id 24872, offset 0, flags [DF], proto TCP (6), length 52)

172.21.198.80.39728 > 172.21.198.81.8681: Flags [.], cksum 0x779b (correct), ack 11, win 229, options [nop,nop,TS val 214288071 ecr 629487074], length 0

1557465075.296919 IP (tos 0x0, ttl 64, id 24873, offset 0, flags [DF], proto TCP (6), length 1502)

172.21.198.80.39728 > 172.21.198.81.8681: Flags [P.], cksum 0xea9d (incorrect -> 0x71e8), seq 11:1461, ack 12, win 229, options [nop,nop,TS val 214288071 ecr 629487074], length 1450

1557465075.296925 IP (tos 0x0, ttl 64, id 24875, offset 0, flags [DF], proto TCP (6), length 52)

172.21.198.80.39728 > 172.21.198.81.8681: Flags [F.], cksum 0x71ef (correct), seq 1461, ack 12, win 229, options [nop,nop,TS val 214288071 ecr 629487074], length 0

1557465075.296942 IP (tos 0x0, ttl 64, id 5460, offset 0, flags [DF], proto TCP (6), length 52)

172.21.198.81.8681 > 172.21.198.80.39728: Flags [.], cksum 0xe4f3 (incorrect -> 0x71dc), ack 1461, win 249, options [nop,nop,TS val 629487074 ecr 214288071], length 0

1557465075.296951 IP (tos 0x0, ttl 64, id 5461, offset 0, flags [DF], proto TCP (6), length 52)

172.21.198.81.8681 > 172.21.198.80.39728: Flags [.], cksum 0xe4f3 (incorrect -> 0x71db), ack 1462, win 249, options [nop,nop,TS val 629487074 ecr 214288071], length 0

12 packets captured

13 packets received by filter

0 packets dropped by kernel

State Transition Diagram

The state transition diagram 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).

A close up of a map

Description automatically generated

Timing Chart

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

A close up of a map

Description automatically generated

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.

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

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.

1. 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.
2. 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.

Test 3

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

1. 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.
2. The maximum segment size (MSS) is 1460 bytes.

Test 4

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

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.

Test 5 (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.

The effect of “with -D” is almost trivial to segments received, segments sent out, and segments retransmitted.

Test 5 (b)

Thereafter, run " strace -ttT ttcp " in the following particular test case without and with -D option.

“With -D” results in shorter time for OS to execute write system calls.