BCIT

COMP 7005 Final Project

December 2, 2014

Christopher Eng

A00842832

Project Introduction

The submitted COMP 7005 Final Project is a half-duplex TCP Reno-implemented Send-And-Wait protocol simulator. Using a C implementation, it is comprised of three components:

- Transmitter
- Receiver
- Network Emulator

A file transfer is performed from the transmitter to the receiver through the network emulator. The network emulator relays the data between the two end-points, while delaying and dropping packets to simulate an unreliable network.

Environment Setup

The program was developed for use on a Linux-based operating system. Each component contains the main source file, 3-4 header files, and a configuration file. Due to the use of timers, compilation requires linking with the real-time library.

Configuration File

The configuration file, <connections.cfg>, must be at least 7 lines long. The first line of the file is a descriptor on what the file contains. The second line onwards stores configurations on a line-by-line basis. Each line stores a key and its value, separated by whitespace. Given the nature of the project, the source files expect 6 keys:

- transmitter_ip
- transmitter_port
- network_ip
- network_port
- receiver_ip
- receiver_port

Transmitter

Source files: <tcp_clnt.c>, <config.h>, <connections.cfg>, <log.h>, <packet.h>, and <timer.h>.

Compilation using GCC: ‘*gcc tcp_clnt.c –o tcp_clnt -lrt*’

Runtime: ‘*./tcp_clnt <filename>*’

Parameters:
<filename> - The name of the file to send.

Receiver

Source files: <tcp_svr.c>, <config.h>, <connections.cfg>, <log.h>, <packet.h>, and <timer.h>.

Compilation using GCC: ‘*gcc tcp_svr.c –o tcp_svr -lrt*’

Runtime: ‘*./tcp_svr <optional_window_size>*’

Parameters:

<optional_window_size> - An optional argument indicating how many packets the receiver can store in its buffer. Defaulted to 128 MSS.

Network Emulator

Source files: <tcp_network.c>, <config.h>, <connections.cfg>, <packet.h>, and <timer.h>.

Compilation using GCC: ‘*gcc tcp_network.c –o tcp_network -lrt*’

Runtime: ‘*./tcp_network <bit_error_rate> <delay_usec>*’

Parameters:
<bit_error_rate> - A floating-point value from 0 to 1 indicating what percentage of packets received to discard. 0 will operate the program in ideal conditions.
<delay_usec> - An integer indicating how many microseconds each packet should be held for before relaying it to its proper destination.

Pseudo-Code

Transmitter

```
Read Filename parameter

Read Configuration File

Error if Configuration File is missing Network IP & Network Port

Error if file doesn’t exist

Open connection to Network

Generate random initial SeqNum

Send SYN Packet to network

Start timer
```

Case	**Timer expires**	**Receive SYN-ACK with expected SeqNum**
Action	Resend SYN packet to network	Stop timer
Action	Start timer	Send ACK to network

Send filename to network

Start timer

Receive responses until a FIN packet response is read

Case	Timer expires
Action	Resend first packet in window to network
Action	Set SSThresh to half WindowSize, WindowSize to 1, Timer to twice its duration
Action	Start timer

Case	Receive packet with last ACKed SeqNum
Action	Increment duplicate ACK count
Condition	**If duplicate ACK count = 3**
Action	Send first packet in window to network
Action	Set SSthresh to half WindowSize, WindowSize to half WindowSize + 3, duplicate ACK count to 0

Case	Receive packet with an ACK greater than last ACKed SeqNum
Action	Stop timer
Action	Set timer duration based on RTT
Action	Remove ACKed packets from window
Action	Double WindowSize if slow start (SSThresh > WindowSize) Increment WindowSize if congestion avoidance (not slow start)
Action	Read from file and load packet in window until window is full
Action	Load 1-byte packets if receiver advertises an empty window
Action	If entire file has been read and window has room, add FIN packet to window
Action	Send all unsent packets in window to network
Action	Start timer

Send ACK packet for received FIN packet to network

Start timer

Case	Timer expires	Receive FIN packet
Action	Close connection	Stop timer
Action		Resend ACK packet to network
Action		Start timer

Receiver

Read Optional Window Size parameter

Read Configuration File

Error if Configuration File is missing Receiver Port

Listen for connections

Open connection

Receive SYN packet

Generate random initial SeqNum

Send SYN-ACK packet

Start timer

Case	**Timer expires**	**Receive ACK with expected SeqNum or Data Packet with next expected SeqNum**
Action	Resend SYN-ACK packet	Stop timer
Action	Start timer	If ACK, wait to receive Data Packet
Action		Set filename from Data Packet

Receive packets until a FIN packet has been received and ACKed

Case	**Receive packet with expected SeqNum**
Action	Write data to file
Action	Check buffer for any sequential packets stored and write data to file, Remove packets from buffer
Action	If timer is stopped, start timer If timer is started, stop it and send ACK packet

Case	**Timer expires**
Action	Send ACK packet

Case	**Receive packet with SeqNum greater than expected SeqNum**
Action	Stop timer
Action	Resend last ACKed packet
Action	Add packet to buffer

Case	**Receive packet with SeqNum less than expected SeqNum**
Action	Resend last ACKed packet

Save file to Filename

Send FIN packet to network

Start timer

Case	Timer expires	Receive ACK packet with expected SeqNum
Action	Resend FIN packet	Stop timer
Action		Close connection

Network Emulator

Read Bit Error Rate and Delay parameters

Read Configuration File

Error if Configuration File is missing Network Port, Receiver IP, & Receiver Port

Listen for connections

Open connection to Transmitter

Open connection to Receiver

Loop until a connection closes

Case	Receive packet
Action	Generate random number percentage
Condition	**If percentage is greater than Bit Error Rate**
Action	Add packet to end of list
Action	Set packet timer value to remaining delay required before sending, minus all packet timer values closer to front in list
Action	If list only has 1 packet, start timer

Case	Timer expires
Action	Send packet to other end-point
Action	Remove packet from beginning of list
Action	Start timer with stored delay

Implementation

Packets

Packets are implemented in the packet.h header file. The PacketType is an integer code that represents the different types of packets found in a TCP transmission. This is used for the 3-way handshake, data transmission, and closing the connection.

The SeqNum (Sequence Number) starts as a random number and increments by the number of bytes sent. A packet is identified by the first byte it is sending. To verify a packet has been sent, the receiver will send a packet containing an AckNum representing the next expected Sequence Number. This is calculated by adding the number of bytes in the received packet with its Sequence Number. The number of bytes can be found in the DataLength attribute. For the sake of simplicity, packets were designed to store 512 bytes of data storage even if this amount was not used. This is why the DataLength attribute was added. The data attribute is a character array that holds the actual packet payload.

Finally, the WindowSize is the size of the window for the sender. Since bytes are a set size in the emulator, this is represented by number of MSS packets (512 bytes). The transmitter must look at the WindowSize to ensure it is not overloading the receiver with more packets than can be stored. Since WindowSize is not in bytes, we cannot send smaller packets of data due to lack of data. However, when the receiver window is full, the transmitter sends 1-byte packets to ensure data is still sent, so the window will eventually clear up.

Window

The window acts as a linked list of packets, to represent a storage buffer for packets in a specific order. This is also implemented in the packet.h header file. WindowSegments hold a pointer to the next WindowSegment in the list. In the transmitter, we maintain pointers to the beginning and end of the linked list. This makes it easy to add packets to the end of the window and send packets from the beginning. The receiver buffer window is different, in that the window stores out-of-order packets that have larger sequence numbers than expected. Due to this, the receiver must maintain an ordered list. With each out-of-order packet received, it must iterate through the list and find the correct position to store the packet.

There is also a stored delay value and connection identifier, used only by the network emulator to know how long this packet has until it should be sent and which connection to send it through. The user sets a time period of delay per packet. Each packet that arrives should wait this amount of time before being sent to the other end-point. By utilizing a timer, we are able to set this timer and send the packet when a timeout occurs. For each packet that is received while the timer is running, we calculate how much time is left on the timer. The delay value for the packet is how much delay time is required to wait for the correct delay. For every sequential packet received, we subtract the delay time of the packets in front of it. When the first timer finishes, we load the timer with the next packet's delay time value and send it on timeout.

Timer

Timer calculations can be found in the timer.h header file. Global variables for estimatedRTT and devRTT are declared and used in this header file for the transmitter timer calculations. These variables allow the timer values to be weighted based on the actual round trip time for sending and receiving a packet. Initially, the estimated round trip time and deviation round trip time are both set to

250 msec. The initial timer value is 1 second based on the formula: *estimatedRTT = estimatedRTT + 4*devRTT*. The transmitter calculates the new estimated round trip time and deviation round trip time using the TCP calculations with the formulas: *estimatedRTT = (1-a)*estimatedRTT + a*sampleRTT*, where *a* = 0.125, and *devRTT = (1-b)*devRTT + b*|sampleRTT – estimatedRTT|*, where *b* = 0.25. *sampleRTT* is the new round trip time for the last sent packet. On a timeout, the transmitter doubles the timer value temporarily, to ensure the next packet sent makes it back.

Logger

The logger is utilized in both the transmitter and receiver modules. They are used to document error events that occur and the information of packets that are sent or received. The logger is setup to save information to the file <output.log>. Logger functions are on a line-by-line basis and prepend the timestamp in front of each line.

3-way Handshake

One unclear issue that arose during implementation was what happens when the receiver does not receive an ACK to its SYN-ACK packet, since a DATA packet can be sent by the transmitter immediately after without waiting for a response. Light research online was able to explain that if the receiver receives the transmitter's DATA packet, this implies that the 3-way handshake ACK was sent but not received. DATA packets have associated timers, which allow for retransmission if a response does not arrive. An identical issue was found when closing the connection.

Closing Connection

At the end of the program, the transmitter sends a FIN packet to signify it is finished sending and wants to close the connection. The receiver responds with an ACK to the FIN and sends its own FIN when it is finished ACKing any buffered packets in its window, signifying it is also ready to close its connection. The transmitter then ACKs this FIN packet and waits for a period of time before closing the connection, in case more packets arrive. In TCP, this is generally from 30 seconds to 2 minutes. For the purpose of simulation, this timer was implemented to utilize the standard timer value. If a FIN packet is received by the transmitter before the timer runs out, this indicates the ACK sent did not arrive and it is resent.

One unclear issue that arose during implementation was what happens when the transmitter does not receive an ACK to its FIN packet, since a FIN packet can be sent by the receiver immediately after without waiting for a response. Light research online was able to explain that if the transmitter receives the receiver's FIN packet, this implies that the ACK was sent but not received. If the receiver's FIN packet is not received, a timer on the receiving end allows for retransmission.

Receiver

The receiver acts as depicted by the receiver pseudo-code. One TCP feature implemented is the use of a timer. The timer value is set to 500 microseconds and represents the receiver waiting for additional sequential packets to come before ACKing one. In TCP, the receiver will wait 500 microseconds for another incoming packet to ACK both at the same time. This will only occur once, so the ACK is sent if either the timeout occurs or if a new packet arrives while the timer is running. At this point, the new packet is also ACKed and the timer is stopped so that the ACK is not delayed beyond approving 2 packets.

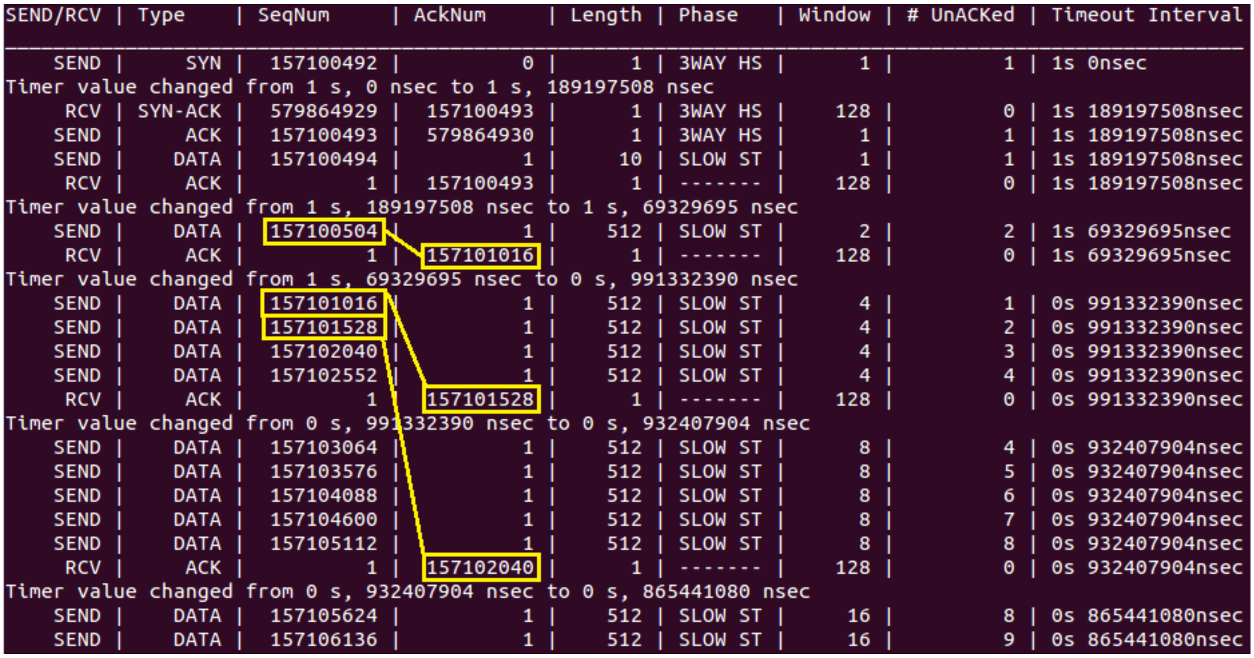
Test Document

The following general program features will not be documented, as they were tested during development and are unrelated to the TCP protocols:

- Valid arguments
- File Existing
- Logging
- Configuration file
- Socket networking (Connecting, sending, receiving, polling)
- Timer (Initializing, handling timeout, setting, getting)
- Bit Error Rate
- Network Delay

Transmitter Sliding Window

The following table depicts the screenshot data by each round trip. Each row describes the sliding window by displaying which packets were sent and ACKed, how many packets remain unACKed in the window, how large the window is, and how many packets forward the window slid this round trip. The data is during TCP Slow Start, so the WindowSize is increasing while the window slides. Sequence numbers are depicted by the last 4 digits. This screenshot was taken of the transmitter’s console log for an ideal TCP transmission, without any dropped packets or network delay.



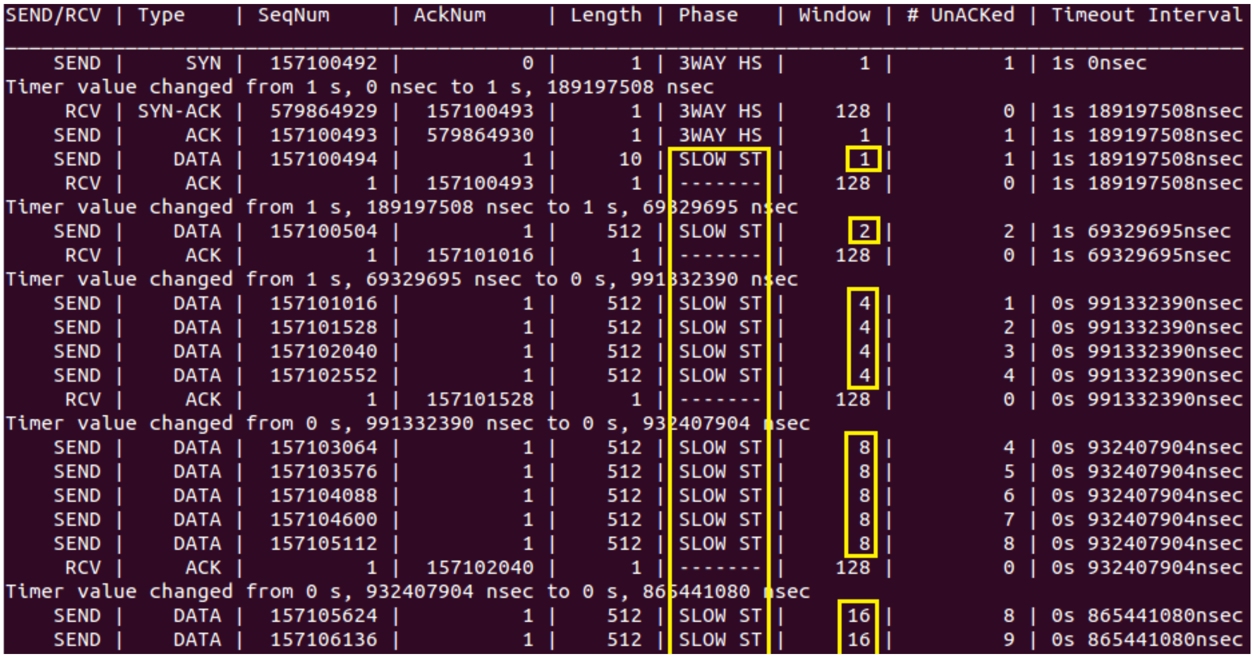
```
SEND/RCV | Type     | SeqNum      | AckNum      | Length | Phase    | Window | # UnACKed | Timeout Interval
-----------------------------------------------------------------------------------------------------------
    SEND |      SYN |  157100492 |           0 |      1 | 3WAY HS |      1 |         1 | 1s 0nsec
Timer value changed from 1 s, 0 nsec to 1 s, 189197508 nsec
     RCV | SYN-ACK |  579864929 |  157100493 |      1 | 3WAY HS |    128 |         0 | 1s 189197508nsec
    SEND |      ACK |  157100493 |  579864930 |      1 | 3WAY HS |      1 |         1 | 1s 189197508nsec
    SEND |     DATA |  157100494 |           1 |     10 | SLOW ST |      1 |         1 | 1s 189197508nsec
     RCV |      ACK |          1 |  157100493 |      1 | ------- |    128 |         0 | 1s 189197508nsec
Timer value changed from 1 s, 189197508 nsec to 1 s, 69329695 nsec
    SEND |     DATA |  157100504 |           1 |    512 | SLOW ST |      2 |         2 | 1s 69329695nsec
     RCV |      ACK |          1 |  157101016 |      1 | ------- |    128 |         0 | 1s 69329695nsec
Timer value changed from 1 s, 69329695 nsec to 0 s, 991332390 nsec
    SEND |     DATA |  157101016 |           1 |    512 | SLOW ST |      4 |         1 | 0s 991332390nsec
    SEND |     DATA |  157101528 |           1 |    512 | SLOW ST |      4 |         2 | 0s 991332390nsec
    SEND |     DATA |  157102040 |           1 |    512 | SLOW ST |      4 |         3 | 0s 991332390nsec
    SEND |     DATA |  157102552 |           1 |    512 | SLOW ST |      4 |         4 | 0s 991332390nsec
     RCV |      ACK |          1 |  157101528 |      1 | ------- |    128 |         0 | 0s 991332390nsec
Timer value changed from 0 s, 991332390 nsec to 0 s, 932407904 nsec
    SEND |     DATA |  157103064 |           1 |    512 | SLOW ST |      8 |         4 | 0s 932407904nsec
    SEND |     DATA |  157103576 |           1 |    512 | SLOW ST |      8 |         5 | 0s 932407904nsec
    SEND |     DATA |  157104088 |           1 |    512 | SLOW ST |      8 |         6 | 0s 932407904nsec
    SEND |     DATA |  157104600 |           1 |    512 | SLOW ST |      8 |         7 | 0s 932407904nsec
    SEND |     DATA |  157105112 |           1 |    512 | SLOW ST |      8 |         8 | 0s 932407904nsec
     RCV |      ACK |          1 |  157102040 |      1 | ------- |    128 |         0 | 0s 932407904nsec
Timer value changed from 0 s, 932407904 nsec to 0 s, 865441080 nsec
    SEND |     DATA |  157105624 |           1 |    512 | SLOW ST |     16 |         8 | 0s 865441080nsec
    SEND |     DATA |  157106136 |           1 |    512 | SLOW ST |     16 |         9 | 0s 865441080nsec
```

SeqNums Sent	SeqNums ACKed	# UnACKed Pkts	WindowSize	Window Slide Amt
0494, 0504	0494, 0504	0	2	2
1016, 1528, 2040, 2552	1016	3	4	1
3064, 3576, 4088, 4600, 5112	1528	7	8	1

We notice the sequence numbers added each RTT fill up the open spaces in the window.

Slow Start Phase

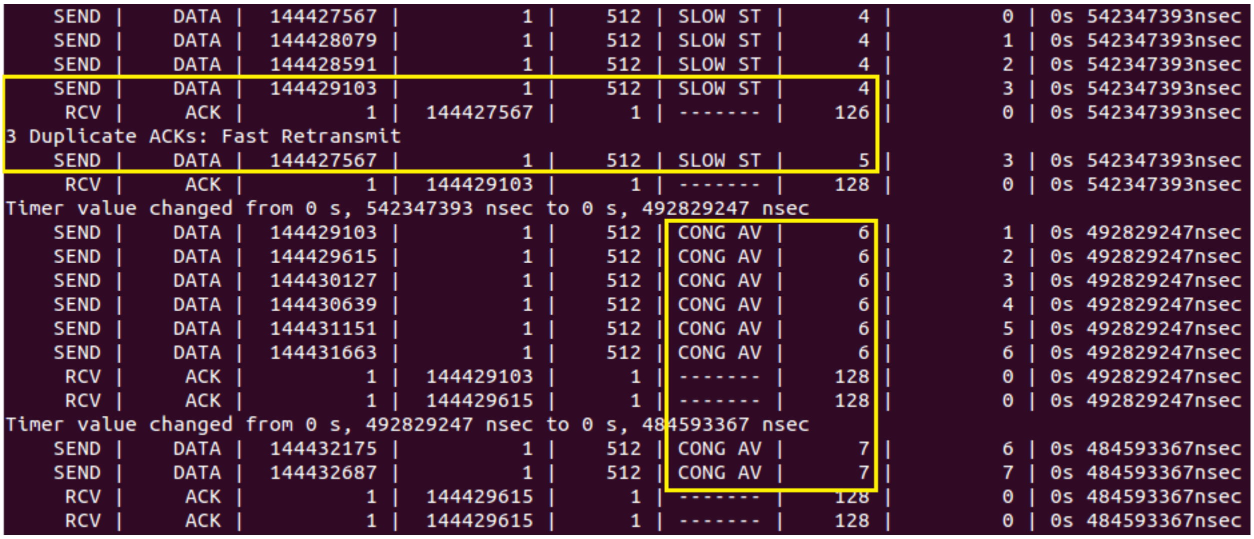
TCP Slow Start can be tested by watching the WindowSize double each round trip. The phase column in the console log shows "SLOW ST" when this phase is active. This screenshot was taken of the transmitter's console log for an ideal TCP transmission, without any dropped packets or network delay.



```
SEND/RCV | Type     | SeqNum       | AckNum       | Length | Phase    | Window | # UnACKed | Timeout Interval
-----------------------------------------------------------------------------------------------------------
    SEND |      SYN |  157100492 |            0 |        1 | 3WAY HS |       1 |          1 | 1s 0nsec
Timer value changed from 1 s, 0 nsec to 1 s, 189197508 nsec
     RCV |  SYN-ACK |  579864929 |  157100493 |        1 | 3WAY HS |     128 |          0 | 1s 189197508nsec
    SEND |      ACK |  157100493 |  579864930 |        1 | 3WAY HS |       1 |          1 | 1s 189197508nsec
    SEND |     DATA |  157100494 |            1 |       10 | SLOW ST |       1 |          1 | 1s 189197508nsec
     RCV |      ACK |          1 |  157100493 |        1 | ------- |     128 |          0 | 1s 189197508nsec
Timer value changed from 1 s, 189197508 nsec to 1 s, 69329695 nsec
    SEND |     DATA |  157100504 |            1 |      512 | SLOW ST |       2 |          2 | 1s 69329695nsec
     RCV |      ACK |          1 |  157101016 |        1 | ------- |     128 |          0 | 1s 69329695nsec
Timer value changed from 1 s, 69329695 nsec to 0 s, 991332390 nsec
    SEND |     DATA |  157101016 |            1 |      512 | SLOW ST |       4 |          1 | 0s 991332390nsec
    SEND |     DATA |  157101528 |            1 |      512 | SLOW ST |       4 |          2 | 0s 991332390nsec
    SEND |     DATA |  157102040 |            1 |      512 | SLOW ST |       4 |          3 | 0s 991332390nsec
    SEND |     DATA |  157102552 |            1 |      512 | SLOW ST |       4 |          4 | 0s 991332390nsec
     RCV |      ACK |          1 |  157101528 |        1 | ------- |     128 |          0 | 0s 991332390nsec
Timer value changed from 0 s, 991332390 nsec to 0 s, 932407904 nsec
    SEND |     DATA |  157103064 |            1 |      512 | SLOW ST |       8 |          4 | 0s 932407904nsec
    SEND |     DATA |  157103576 |            1 |      512 | SLOW ST |       8 |          5 | 0s 932407904nsec
    SEND |     DATA |  157104088 |            1 |      512 | SLOW ST |       8 |          6 | 0s 932407904nsec
    SEND |     DATA |  157104600 |            1 |      512 | SLOW ST |       8 |          7 | 0s 932407904nsec
    SEND |     DATA |  157105112 |            1 |      512 | SLOW ST |       8 |          8 | 0s 932407904nsec
     RCV |      ACK |          1 |  157102040 |        1 | ------- |     128 |          0 | 0s 932407904nsec
Timer value changed from 0 s, 932407904 nsec to 0 s, 865441080 nsec
    SEND |     DATA |  157105624 |            1 |      512 | SLOW ST |      16 |          8 | 0s 865441080nsec
    SEND |     DATA |  157106136 |            1 |      512 | SLOW ST |      16 |          9 | 0s 865441080nsec
```

Congestion Avoidance Phase

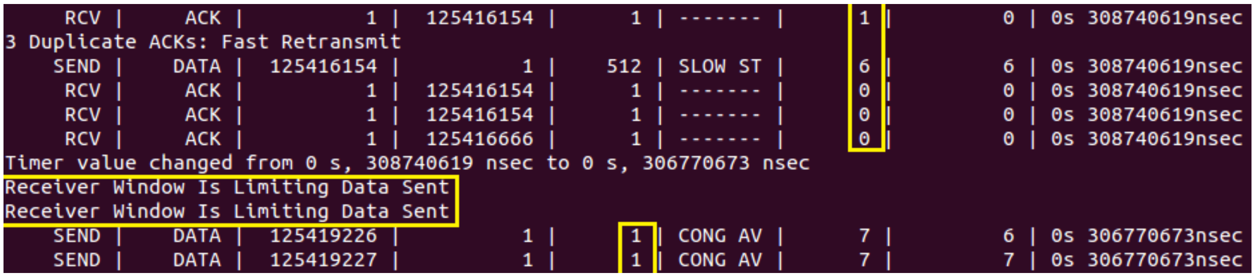
TCP Congestion Avoidance phase begins when the window size grows larger than the slow start threshold. The slow start threshold is set on a packet loss. In the screenshot, fast retransmit causes the ssthresh to become half of the WindowSize 4 to become 2. The new WindowSize 5 is larger than 2, so the next send begins in congestion avoidance. The phase is set to “CONG AV” when congestion avoidance is active. We notice the WindowSize is incrementing with each round trip time. This screenshot was taken of the transmitter’s console log for a TCP transmission with a 5% rate of dropped packets and no network delay.



```
    SEND |     DATA |  144427567 |            1 |      512 | SLOW ST |       4 |          0 | 0s 542347393nsec
    SEND |     DATA |  144428079 |            1 |      512 | SLOW ST |       4 |          1 | 0s 542347393nsec
    SEND |     DATA |  144428591 |            1 |      512 | SLOW ST |       4 |          2 | 0s 542347393nsec
    SEND |     DATA |  144429103 |            1 |      512 | SLOW ST |       4 |          3 | 0s 542347393nsec
     RCV |      ACK |          1 |  144427567 |        1 | ------- |     126 |          0 | 0s 542347393nsec
3 Duplicate ACKs: Fast Retransmit
    SEND |     DATA |  144427567 |            1 |      512 | SLOW ST |       5 |          3 | 0s 542347393nsec
     RCV |      ACK |          1 |  144429103 |        1 | ------- |     128 |          0 | 0s 542347393nsec
Timer value changed from 0 s, 542347393 nsec to 0 s, 492829247 nsec
    SEND |     DATA |  144429103 |            1 |      512 | CONG AV |       6 |          1 | 0s 492829247nsec
    SEND |     DATA |  144429615 |            1 |      512 | CONG AV |       6 |          2 | 0s 492829247nsec
    SEND |     DATA |  144430127 |            1 |      512 | CONG AV |       6 |          3 | 0s 492829247nsec
    SEND |     DATA |  144430639 |            1 |      512 | CONG AV |       6 |          4 | 0s 492829247nsec
    SEND |     DATA |  144431151 |            1 |      512 | CONG AV |       6 |          5 | 0s 492829247nsec
    SEND |     DATA |  144431663 |            1 |      512 | CONG AV |       6 |          6 | 0s 492829247nsec
     RCV |      ACK |          1 |  144429103 |        1 | ------- |     128 |          0 | 0s 492829247nsec
     RCV |      ACK |          1 |  144429615 |        1 | ------- |     128 |          0 | 0s 492829247nsec
Timer value changed from 0 s, 492829247 nsec to 0 s, 484593367 nsec
    SEND |     DATA |  144432175 |            1 |      512 | CONG AV |       7 |          6 | 0s 484593367nsec
    SEND |     DATA |  144432687 |            1 |      512 | CONG AV |       7 |          7 | 0s 484593367nsec
     RCV |      ACK |          1 |  144429615 |        1 | ------- |     128 |          0 | 0s 484593367nsec
     RCV |      ACK |          1 |  144429615 |        1 | ------- |     128 |          0 | 0s 484593367nsec
```

Receiver Window

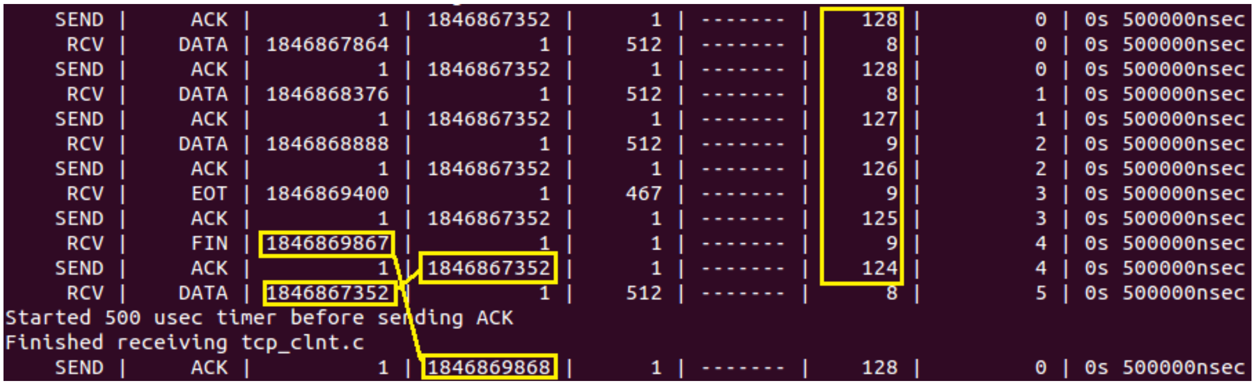
The receiver window can limit the transmitter's rate of sending by running out of space. When the receiver buffers packets to fill the entire receiver window, it advertises a WindowSize 0 to the transmitter. The transmitter has no choice but to send packets with a DataLength of 1 to keep the data sending so ACKs continue to flow, but not overload the receiver. This screenshot was taken of the receiver's console log for a TCP transmission, with a 10% rate of dropped packets and 100 usec network delay. The receiver window was initialized to 1 MSS large.



```
     RCV |      ACK |           1 |  125416154 |       1 | ------- |       1 |          0 | 0s 308740619nsec
3 Duplicate ACKs: Fast Retransmit
    SEND |     DATA |  125416154 |           1 |     512 | SLOW ST |       6 |          6 | 0s 308740619nsec
     RCV |      ACK |           1 |  125416154 |       1 | ------- |       0 |          0 | 0s 308740619nsec
     RCV |      ACK |           1 |  125416154 |       1 | ------- |       0 |          0 | 0s 308740619nsec
     RCV |      ACK |           1 |  125416666 |       1 | ------- |       0 |          0 | 0s 308740619nsec
Timer value changed from 0 s, 308740619 nsec to 0 s, 306770673 nsec
Receiver Window Is Limiting Data Sent
Receiver Window Is Limiting Data Sent
    SEND |     DATA |  125419226 |           1 |       1 | CONG AV |       7 |          6 | 0s 306770673nsec
    SEND |     DATA |  125419227 |           1 |       1 | CONG AV |       7 |          7 | 0s 306770673nsec
```

Receiver Out-of-order Packets

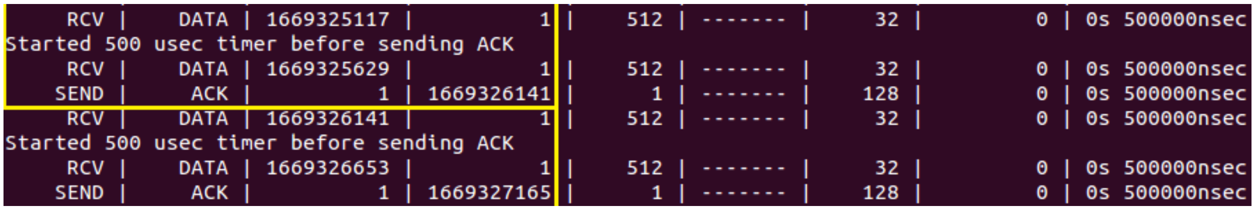
Proof that the receiver can handle out-of-order packets is done by showing that the packets are buffered. In the screenshot, we notice that even though the receiver is receiving packets, the sequence numbers are greater than what it is expecting. These packets are being buffered in the receiver window, shown by a decrementing WindowSize from 128. When the transmitter finally sends the missing packet, the receiver can ACK all buffered packets. This screenshot was taken of the receiver's console log for a TCP transmission, with a 10% rate of dropped packets and 100 usec network delay.



```
    SEND |      ACK |          1 | 1846867352 |       1 | ------- |     128 |          0 | 0s 500000nsec
     RCV |     DATA | 1846867864 |          1 |     512 | ------- |       8 |          0 | 0s 500000nsec
    SEND |      ACK |          1 | 1846867352 |       1 | ------- |     128 |          0 | 0s 500000nsec
     RCV |     DATA | 1846868376 |          1 |     512 | ------- |       8 |          1 | 0s 500000nsec
    SEND |      ACK |          1 | 1846867352 |       1 | ------- |     127 |          1 | 0s 500000nsec
     RCV |     DATA | 1846868888 |          1 |     512 | ------- |       9 |          2 | 0s 500000nsec
    SEND |      ACK |          1 | 1846867352 |       1 | ------- |     126 |          2 | 0s 500000nsec
     RCV |      EOT | 1846869400 |          1 |     467 | ------- |       9 |          3 | 0s 500000nsec
    SEND |      ACK |          1 | 1846867352 |       1 | ------- |     125 |          3 | 0s 500000nsec
     RCV |      FIN | 1846869867 |          1 |       1 | ------- |       9 |          4 | 0s 500000nsec
    SEND |      ACK |          1 | 1846867352 |       1 | ------- |     124 |          4 | 0s 500000nsec
     RCV |     DATA | 1846867352 |          1 |     512 | ------- |       8 |          5 | 0s 500000nsec
Started 500 usec timer before sending ACK
Finished receiving tcp_clnt.c
    SEND |      ACK |          1 | 1846869868 |       1 | ------- |     128 |          0 | 0s 500000nsec
```

Receiver Waiting 500 Microseconds for Packets

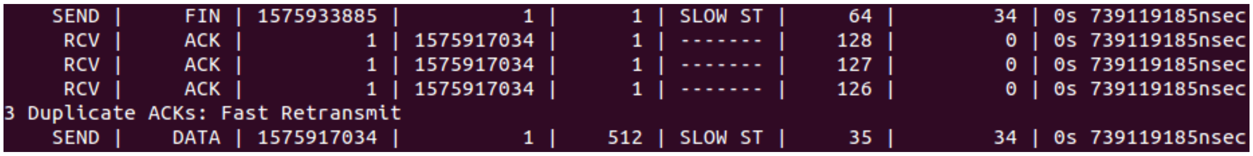
Upon receiving DATA, the receiver should start a 500 usec timer to possibly receive a sequential DATA packet. The receiver ACKs the initial packet along with the new packet. If another packet is received, it should be ACKed immediately. If no packet is received, after the 500 usec is up, the initial packet is ACKed. This is tested by noticing the 500 usec timer starting in the receiver's console log. We notice that the ACK applies to both DATA packets received. This screenshot was taken of the receiver's console log for an ideal TCP transmission, without any dropped packets or network delay.



```
     RCV |     DATA | 1669325117 |           1|     512 | ------- |      32 |           0 | 0s 500000nsec
Started 500 usec timer before sending ACK
     RCV |     DATA | 1669325629 |           1|     512 | ------- |      32 |           0 | 0s 500000nsec
    SEND |      ACK |          1 | 1669326141|       1 | ------- |     128 |           0 | 0s 500000nsec
     RCV |     DATA | 1669326141 |           1|     512 | ------- |      32 |           0 | 0s 500000nsec
Started 500 usec timer before sending ACK
     RCV |     DATA | 1669326653 |           1|     512 | ------- |      32 |           0 | 0s 500000nsec
    SEND |      ACK |          1 | 1669327165|       1 | ------- |     128 |           0 | 0s 500000nsec
```

Fast Retransmit

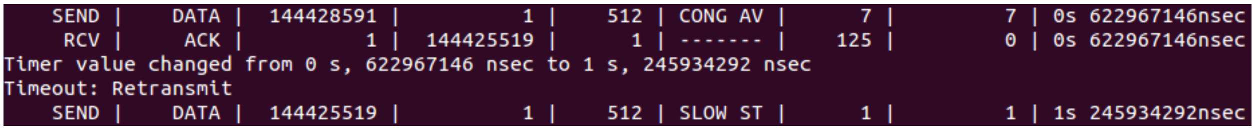
Fast Retransmission is performed when the transmitter received 3 duplicate ACKs for the same sequence number. At this point, the transmitter will send the expected sequence number immediately and half the WindowSize + 3. The screenshot shows this happening, as well as a message printing to the console indicating a triple duplicate ACK event. This screenshot was taken of the transmitter's console log for a TCP transmission with a 5% rate of dropped packets and no network delay.



```
    SEND |      FIN | 1575933885 |           1 |       1 | SLOW ST |      64 |          34 | 0s 739119185nsec
     RCV |      ACK |          1 | 1575917034 |       1 | ------- |     128 |           0 | 0s 739119185nsec
     RCV |      ACK |          1 | 1575917034 |       1 | ------- |     127 |           0 | 0s 739119185nsec
     RCV |      ACK |          1 | 1575917034 |       1 | ------- |     126 |           0 | 0s 739119185nsec
3 Duplicate ACKs: Fast Retransmit
    SEND |     DATA | 1575917034 |           1 |     512 | SLOW ST |      35 |          34 | 0s 739119185nsec
```

Transmitter Timeout

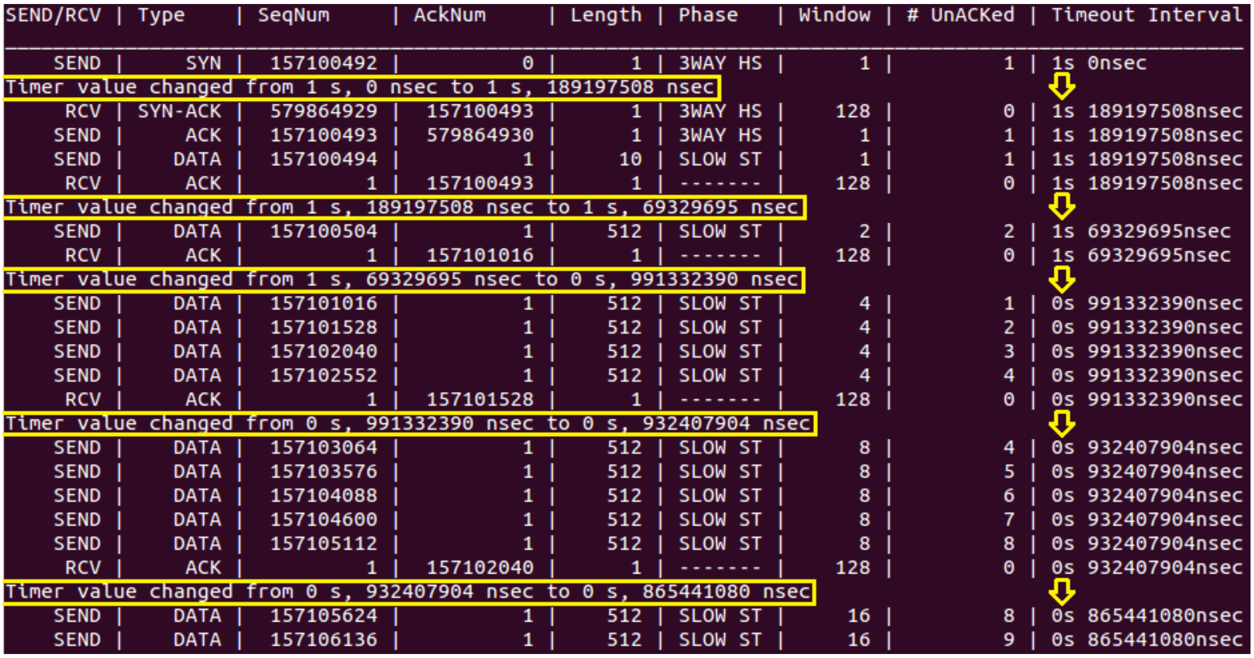
Transmitter Timeout occurs when none of the packets sent are ACKed in a round trip time. When this happens, the transmission's timer will expire and display a "Timeout: Retransmit". At this point, the first unACKed packet is resent and the WindowSize is reset to 1. This screenshot was taken of the transmitter's console log for a TCP transmission with a 5% rate of dropped packets and 100 usec of network delay.



```
    SEND |     DATA |  144428591 |           1 |     512 | CONG AV |       7 |           7 | 0s 622967146nsec
     RCV |      ACK |          1 |  144425519 |       1 | ------- |     125 |           0 | 0s 622967146nsec
Timer value changed from 0 s, 622967146 nsec to 1 s, 245934292 nsec
Timeout: Retransmit
    SEND |     DATA |  144425519 |           1 |     512 | SLOW ST |       1 |           1 | 1s 245934292nsec
```

Transmitter Timer Calculations

Transmitter Timer Calculations were verified to be correct during development through unit testing. These calculations are performed during runtime when the transmitter completes a round trip time cycle. This means that a packet is sent and a new ACK is returned without a timeout occurring. This event is not logged in output.log, but is printed to the console. Some of these messages are highlighted in the following screenshot. This screenshot was taken of the transmitter's console log for an ideal TCP transmission, without any dropped packets or network delay.

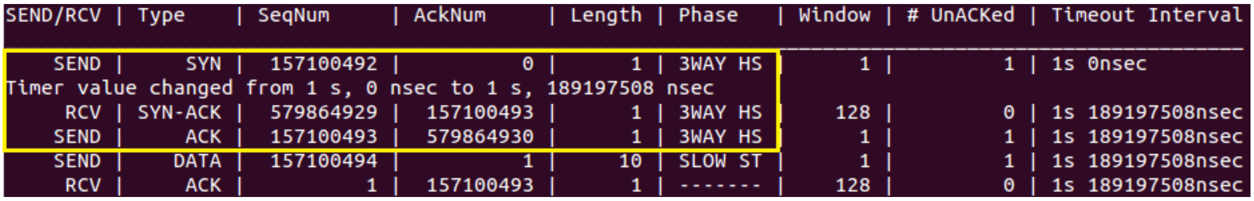


```
SEND/RCV | Type    | SeqNum     | AckNum     | Length | Phase    | Window | # UnACKed | Timeout Interval
-------------------------------------------------------------------------------------------------------
    SEND |     SYN |  157100492 |          0 |      1 | 3WAY HS |      1 |         1 | 1s 0nsec
Timer value changed from 1 s, 0 nsec to 1 s, 189197508 nsec
     RCV | SYN-ACK |  579864929 |  157100493 |      1 | 3WAY HS |    128 |         0 | 1s 189197508nsec
    SEND |     ACK |  157100493 |  579864930 |      1 | 3WAY HS |      1 |         1 | 1s 189197508nsec
    SEND |    DATA |  157100494 |          1 |     10 | SLOW ST |      1 |         1 | 1s 189197508nsec
     RCV |     ACK |          1 |  157100493 |      1 | ------- |    128 |         0 | 1s 189197508nsec
Timer value changed from 1 s, 189197508 nsec to 1 s, 69329695 nsec
    SEND |    DATA |  157100504 |          1 |    512 | SLOW ST |      2 |         2 | 1s 69329695nsec
     RCV |     ACK |          1 |  157101016 |      1 | ------- |    128 |         0 | 1s 69329695nsec
Timer value changed from 1 s, 69329695 nsec to 0 s, 991332390 nsec
    SEND |    DATA |  157101016 |          1 |    512 | SLOW ST |      4 |         1 | 0s 991332390nsec
    SEND |    DATA |  157101528 |          1 |    512 | SLOW ST |      4 |         2 | 0s 991332390nsec
    SEND |    DATA |  157102040 |          1 |    512 | SLOW ST |      4 |         3 | 0s 991332390nsec
    SEND |    DATA |  157102552 |          1 |    512 | SLOW ST |      4 |         4 | 0s 991332390nsec
     RCV |     ACK |          1 |  157101528 |      1 | ------- |    128 |         0 | 0s 991332390nsec
Timer value changed from 0 s, 991332390 nsec to 0 s, 932407904 nsec
    SEND |    DATA |  157103064 |          1 |    512 | SLOW ST |      8 |         4 | 0s 932407904nsec
    SEND |    DATA |  157103576 |          1 |    512 | SLOW ST |      8 |         5 | 0s 932407904nsec
    SEND |    DATA |  157104088 |          1 |    512 | SLOW ST |      8 |         6 | 0s 932407904nsec
    SEND |    DATA |  157104600 |          1 |    512 | SLOW ST |      8 |         7 | 0s 932407904nsec
    SEND |    DATA |  157105112 |          1 |    512 | SLOW ST |      8 |         8 | 0s 932407904nsec
     RCV |     ACK |          1 |  157102040 |      1 | ------- |    128 |         0 | 0s 932407904nsec
Timer value changed from 0 s, 932407904 nsec to 0 s, 865441080 nsec
    SEND |    DATA |  157105624 |          1 |    512 | SLOW ST |     16 |         8 | 0s 865441080nsec
    SEND |    DATA |  157106136 |          1 |    512 | SLOW ST |     16 |         9 | 0s 865441080nsec
```

3-Way Handshake

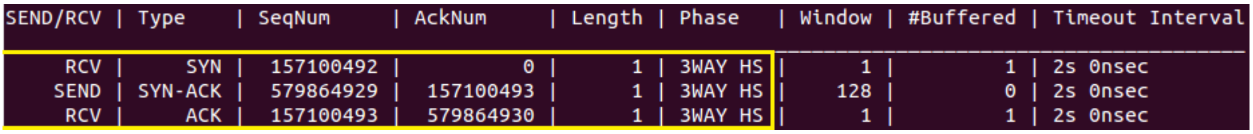
TCP utilizes the 3-way handshake to initialize connections between a client and server. The 3-way handshake is implemented to send and receive the following sequence of PacketTypes: SYN, SYN-ACK, and ACK. This can be found in program screenshots with the defined phase "3WAY HS". These screenshot were taken of the console logs for an ideal TCP transmission, without any dropped packets or network delay.

Transmitter:



```
SEND/RCV | Type    | SeqNum     | AckNum     | Length | Phase    | Window | # UnACKed | Timeout Interval
-------------------------------------------------------------------------------------------------------
    SEND |     SYN |  157100492 |          0 |      1 | 3WAY HS |      1 |         1 | 1s 0nsec
Timer value changed from 1 s, 0 nsec to 1 s, 189197508 nsec
     RCV | SYN-ACK |  579864929 |  157100493 |      1 | 3WAY HS |    128 |         0 | 1s 189197508nsec
    SEND |     ACK |  157100493 |  579864930 |      1 | 3WAY HS |      1 |         1 | 1s 189197508nsec
    SEND |    DATA |  157100494 |          1 |     10 | SLOW ST |      1 |         1 | 1s 189197508nsec
     RCV |     ACK |          1 |  157100493 |      1 | ------- |    128 |         0 | 1s 189197508nsec
```

Receiver:

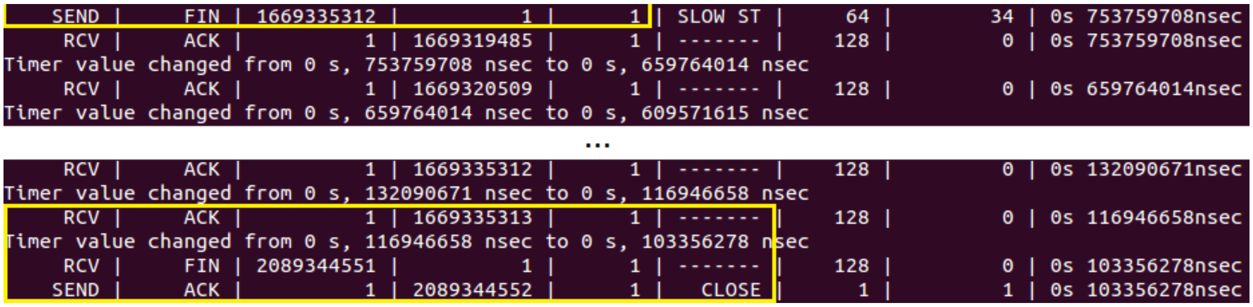


```
SEND/RCV | Type    | SeqNum     | AckNum     | Length | Phase    | Window | #Buffered | Timeout Interval
-------------------------------------------------------------------------------------------------------
     RCV |     SYN |  157100492 |          0 |      1 | 3WAY HS |      1 |         1 | 2s 0nsec
    SEND | SYN-ACK |  579864929 |  157100493 |      1 | 3WAY HS |    128 |         0 | 2s 0nsec
     RCV |     ACK |  157100493 |  579864930 |      1 | 3WAY HS |      1 |         1 | 2s 0nsec
```

Closing Connection

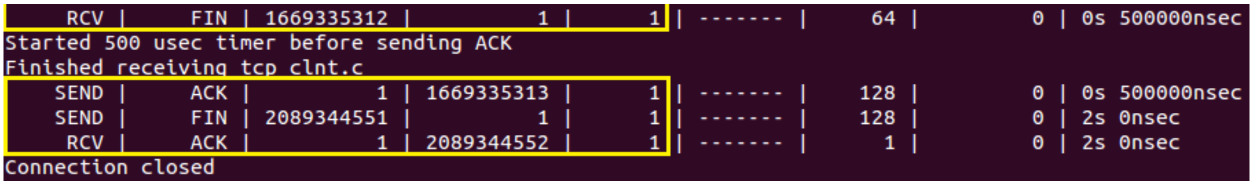
TCP closing connection protocol is completed using the FIN PacketType. The transmitter sends a FIN when it has finished sending all it needs to. Once the receiver responds to the FIN with an ACK, the receiver sends a FIN of its own meaning it also has finished sending all it needs to. This can be seen in the following screenshots through the PacketType combination, followed by the connection closing. These screenshot were taken of the console logs for an ideal TCP transmission, without any dropped packets or network delay.

Transmitter:



```
    SEND |      FIN | 1669335312 |          1 |       1 | SLOW ST |     64 |           34 | 0s 753759708nsec
     RCV |      ACK |          1 | 1669319485 |       1 | ------- |    128 |            0 | 0s 753759708nsec
Timer value changed from 0 s, 753759708 nsec to 0 s, 659764014 nsec
     RCV |      ACK |          1 | 1669320509 |       1 | ------- |    128 |            0 | 0s 659764014nsec
Timer value changed from 0 s, 659764014 nsec to 0 s, 609571615 nsec
                                          ...
     RCV |      ACK |          1 | 1669335312 |       1 | ------- |    128 |            0 | 0s 132090671nsec
Timer value changed from 0 s, 132090671 nsec to 0 s, 116946658 nsec
     RCV |      ACK |          1 | 1669335313 |       1 | ------- |    128 |            0 | 0s 116946658nsec
Timer value changed from 0 s, 116946658 nsec to 0 s, 103356278 nsec
     RCV |      FIN | 2089344551 |          1 |       1 | ------- |    128 |            0 | 0s 103356278nsec
    SEND |      ACK |          1 | 2089344552 |       1 |   CLOSE |      1 |            1 | 0s 103356278nsec
```

Receiver:



```
     RCV |      FIN | 1669335312 |          1 |       1 | ------- |     64 |            0 | 0s 500000nsec
Started 500 usec timer before sending ACK
Finished receiving tcp_clnt.c
    SEND |      ACK |          1 | 1669335313 |       1 | ------- |    128 |            0 | 0s 500000nsec
    SEND |      FIN | 2089344551 |          1 |       1 | ------- |    128 |            0 | 2s 0nsec
     RCV |      ACK |          1 | 2089344552 |       1 | ------- |      1 |            0 | 2s 0nsec
Connection closed
```

<u>Known Bugs/Issues</u>

1. A rare occurrence occurs with the transmitter where the program shuts down due to Segmentation Fault. There was not enough time to debug this issue, but it is known to appear with sequential timeouts in a row.
2. The EOT PacketType is not utilized as described in the project specifications. This does not seem to fit the TCP implementation, since the receiver does not need to know when a transmission has completed.
3. The network module delay argument was supposed to act as an average delay. Packets would be delayed by a dynamic amount based on standard deviation from the average, but instead the implemented delay is constant to the inputted parameter.
4. Timer timeout can cause duplicate printing to console. This is due to the asynchronous nature of the signal function that is run when a timeout occurs.