# Project Part 1: Stop-n'-Wait & Go-Back-N Investigation Report

Arian Djahed

November 8, 2024

## 1 Introduction

In this report, I will detail my findings from investigating the Stop-n'-Wait and Go-back-N protocols as per the project instructions. I will first go over each protocol along with the code I created to simulate them and how I went about covering every criteria laid out in the project instructions. Then, I will discuss my results upon running my code and compare/contrast the Stop-n'-Wait and Go-back-N results.

## 2 My Code

I began the investigation by familiarizing myself with the given `BaseUDPClient.py` and `BaseUDPServer.py` files along with the socket library invoked therein. The rest boiled down to devising a way to represent each protocol programmatically.

As I was doing so, I found that each of the files I needed to make had certain commonalities that could be placed into their own python file and then imported into the other files in order to avoid redundancy and having to unnecessarily change values in multiple files when it could be changed in one and then have that change apply to multiple files. These commonalities include the server details, the function to simulate an error (the errors in question being either a packet loss or a bit error in this case), and the actual probability values for said errors. Thanks to this last one, I could set the conditions for each trial by only changing the probabilities in this file instead of having to change them individually in every file. The code for this file is as follows:

### 2.1 UDP_Constants.py

```
import random

# Set server details
addr = '127.0.0.1'
port = 50000

# Probability of bit error and packet loss
bit_error_prob = 0.1
packet_loss_prob = 0.1

def simulate_error(probability: float) -> bool:
    # Returns True if an event with the given probability should occur
    return random.random() < probability
```

### 2.2 Stop-n'-Wait

In this protocol, packets are sent one at a time, and the sender "stops and waits" for an acknowledgment from the receiver end before sending the next one

Using the given `BaseUDPClient.py` and `BaseUDPServer.py` files as a foundation, I created two scripts—aptly named `Stop_n'_Wait_Client.py` and `Stop_n'_Wait_Server.py`—in order to simulate the UDP Stop-n'-Wait protocol on both ends. When I actually ran the tests, I first ran the latter script in one terminal shell tab to prime the virtual server to receive packets. Then, I ran the former script in another terminal shell tab to start sending packets. I did this in this particular order so that the total time elapsed did not include time spent sending packets before the server end was set up. By setting up the server end first, I circumvented this issue. The code for each can be found below:

### 2.2.1 Stop_n'_Wait_Client.py

```
import socket, time
from UDP_Constants import addr as rem_addr, port as rem_port, bit_error_prob,
    packet_loss_prob, simulate_error

# Send 1000 packets, each of 1 KB
num_packets = 1000
packet_size = 1024  # 1 KB

def stop_and_wait_sender() -> None:
    # Counts for each scenario
    bit_errors = 0
    packet_losses = 0
    timeouts = 0
    packets_sent = 0

    # Create a UDP socket
    udpsoc = socket.socket(type=socket.SOCK_DGRAM)
    udpsoc.settimeout(1)  # Set timeout for ACK reception

    packet_data = b'X' * packet_size
    start_time = time.time()

    for packet_num in range(num_packets):
        while True:
            # Simulate packet loss
            if not simulate_error(packet_loss_prob):
                # Send the packet
                udpsoc.sendto(packet_data, (rem_addr, rem_port))
                packets_sent += 1
                print(f"Sent packet {packet_num}")
            else:
                packet_losses += 1
                print(f"Packet {packet_num} lost")

            try:
                ack, _ = udpsoc.recvfrom(1024)

                # Simulate ACK bit error
                if simulate_error(bit_error_prob):
                    bit_errors += 1
                    print(f"ACK for packet {packet_num} corrupted")
                    continue

                print(f"Received ACK for packet {packet_num}")
                break  # Exit loop on successful ACK

            # Catch timeout
            except socket.timeout:
                timeouts += 1
                print(f"Timeout, resending packet {packet_num}")

    total_time = time.time() - start_time
    print(f"Total transfer time: {total_time} seconds")
    print(f"{bit_errors} bit errors, {packet_losses} packet losses, {timeouts} timeouts")
    print(f"{packets_sent} total packets sent (including retransmissions)")
    print(f"{packets_sent * packet_size} total bytes of data sent")
    print(f"Utilization rate: {total_time / (packets_sent * packet_size)}")

if __name__ == '__main__':
    stop_and_wait_sender()
```

### 2.2.2 Stop_n'_Wait_Server.py

```
import socket, time
from UDP_Constants import addr as loc_addr, port as loc_port, bit_error_prob,
    packet_loss_prob, simulate_error

def stop_and_wait_receiver() -> None:
    # Create a UDP socket
    udpsoc = socket.socket(type=socket.SOCK_DGRAM)
    udpsoc.bind((loc_addr, loc_port))

    while True:
        recv_data, recv_addr = udpsoc.recvfrom(1024)

        # Simulate packet loss
        if simulate_error(packet_loss_prob):
            print("Packet loss simulated for data packet")
            continue  # Skip ACK to simulate loss

        # Simulate bit error
        if simulate_error(bit_error_prob):
            print("Data packet received but marked as corrupted")
            continue  # Skip ACK to simulate error

        print("Packet received successfully")

        # Simulate RTT
        time.sleep(0.05)

        # Send ACK if no error or loss
        if not simulate_error(packet_loss_prob):
            ack_message = b'ACK'
            udpsoc.sendto(ack_message, recv_addr)
            print("ACK sent")
        else:
            print("ACK lost")
```

```
if __name__ == '__main__':
    stop_and_wait_receiver()
```

## 2.3 Go-back-N

In this protocol, packets are sent in groups; the amount of packets that can be sent at a time depends on a “sliding window” that “slides” forward every time an acknowledgment for a packet is received. This way, the sender can keep sending packets whilst also waiting for acknowledgments without having to “stop.”

For the Go-back-N protocol, I essentially followed the same procedure that I did before, down to the order in which I ran my python files. The only difference here, of course, is the aim to simulate Go-back-N instead of Stop-n’-Wait. The code for my files for this can be found below:

### 2.3.1 Go_Back_N_Client.py

```
import socket, time
from threading import Timer
from UDP_Constants import addr as rem_addr, port as rem_port, bit_error_prob,
    packet_loss_prob, simulate_error

# Configuration for Go-back-N
window_size = 5
num_packets = 1000
packet_size = 1024  # 1 KB
timeout_interval = 1  # Timeout for retransmission

# State variables
base = 0  # First unacknowledged packet
next_seq_num = 0  # Next sequence number to be sent

# Timeout handler for retransmission
def retransmit_packets(udpsoc, packet_data) -> None:
    global base, next_seq_num
    for seq in range(base, next_seq_num):
        if not simulate_error(packet_loss_prob):
            udpsoc.sendto(packet_data[seq], (rem_addr, rem_port))
            print(f"Retransmitting packet {seq}")
        else:
            print(f"Packet {seq} lost during retransmission")

def go_back_n_sender() -> None:
    # Counts for each scenario
    bit_errors = 0
    packet_losses = 0
    timeouts = 0
    packets_sent = 0

    # Create a UDP socket
    global base, next_seq_num
    udpsoc = socket.socket(type=socket.SOCK_DGRAM)
    udpsoc.settimeout(timeout_interval)

    packet_data = [b'X' * packet_size for _ in range(num_packets)]
    timer = None
    start_time = time.time()
```

```
   while base < num_packets:
        # Send packets within the window
        while next_seq_num < base + window_size and next_seq_num < num_packets:
            if not simulate_error(packet_loss_prob):  # Simulate packet loss
                udpsoc.sendto(packet_data[next_seq_num], (rem_addr, rem_port))
                packets_sent += 1
                print(f"Sent packet {next_seq_num}")
            else:
                packet_losses += 1
                print(f"Packet {next_seq_num} lost")
            next_seq_num += 1

        # Start timer for the base packet if not already started
        if base < next_seq_num and timer is None:
            timer = Timer(timeout_interval, retransmit_packets, [udpsoc, packet_data])
            timer.start()

        try:
            ack, _ = udpsoc.recvfrom(1024)

            # Simulate ACK bit error
            if simulate_error(bit_error_prob):
                bit_errors += 1
                print(f"Corrupted ACK received, ignoring")
                continue

            ack_num = int(ack.decode())
            print(f"Received ACK for packet {ack_num}")

            # Slide the window
            if ack_num >= base:
                base = ack_num + 1
                if base == next_seq_num:
                    timer.cancel()  # All packets in the window are acknowledged
                    timer = None
                else:
                    timer = Timer(timeout_interval, retransmit_packets, [udpsoc, packet_data])
                    timer.start()

        except socket.timeout:
            timeouts += 1
            print("Timeout, retransmitting unacknowledged packets")
            retransmit_packets(udpsoc, packet_data)

    total_time = time.time() - start_time
    print(f"Total transfer time: {total_time} seconds")
    print(f"{bit_errors} bit errors, {packet_losses} packet losses, {timeouts} timeouts")
    print(f"{packets_sent} total packets sent (including retransmissions)")
    print(f"{packets_sent * packet_size} total bytes of data sent")
    print(f"Utilization rate: {total_time / (packets_sent * packet_size)}")

if __name__ == '__main__':
    go_back_n_sender()
```

#### 2.3.2 Go_Back_N_Server.py

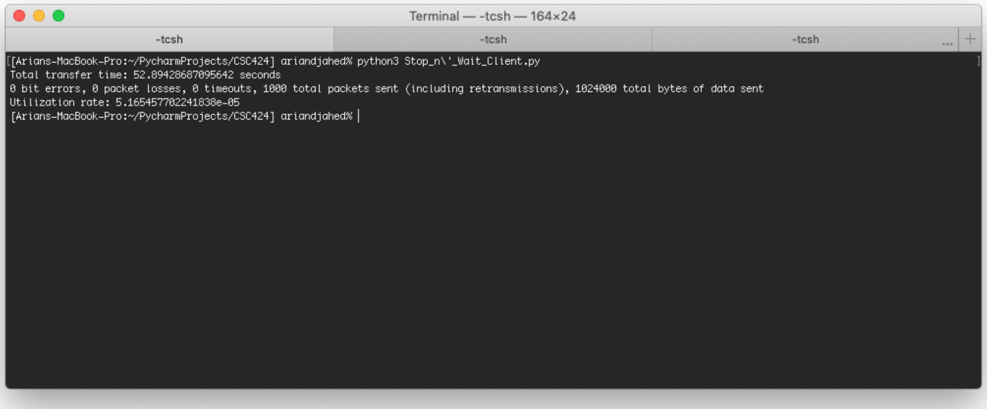
```
import socket, time
from UDP_Constants import addr as loc_addr, port as loc_port, bit_error_prob,
    packet_loss_prob, simulate_error

def go_back_n_receiver() -> None:
    # Create a UDP socket
    udpsoc = socket.socket(type=socket.SOCK_DGRAM)
    udpsoc.bind((loc_addr, loc_port))

    expected_packet = 0

    while True:
        recv_data, recv_addr = udpsoc.recvfrom(1024)

        # Simulate packet loss
        if simulate_error(packet_loss_prob):
            print("Packet loss simulated for data packet")
            continue  # Skip ACK to simulate loss

        # Simulate bit error
        if simulate_error(bit_error_prob):
            print("Data packet received but marked as corrupted")
            continue  # Skip ACK to simulate error

        print(f"Received packet {expected_packet}")

        # Simulate RTT
        time.sleep(0.05)

        # Send ACK for the expected packet
        if recv_data.decode() is not None:
            if not simulate_error(packet_loss_prob):
                ack_message = str(expected_packet).encode()
                udpsoc.sendto(ack_message, recv_addr)
                print(f"ACK {expected_packet} sent")
            else:
                print("ACK lost")
            expected_packet += 1

if __name__ == '__main__':
    go_back_n_receiver()
```

## 3 Results

### 3.1 Code Output

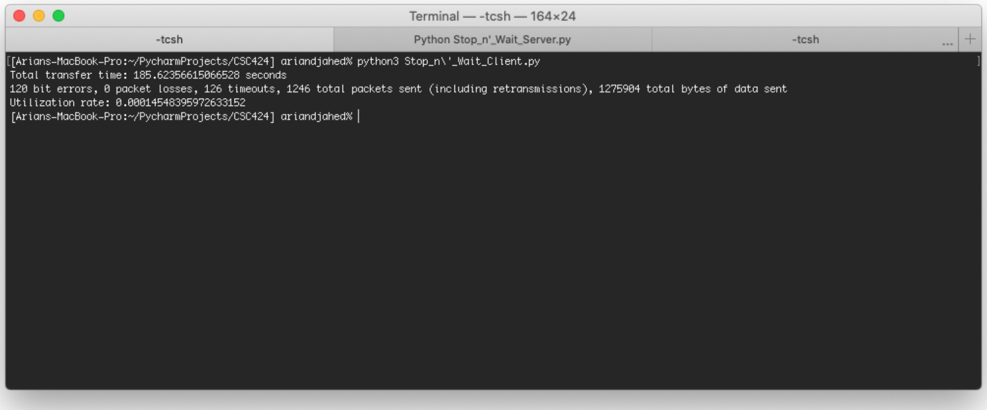
What follows are the screenshots of my test runs of each trial of the investigation (due to their size, they start on the next page). In order to achieve the desired results, I adjusted the `bit_error_prob` and `packet_loss_prob` variables in `UDP_Constants.py` accordingly before running each trial. In order to make sure that the runtime measurements were as accurate as possible, I also commented out every print statements except for the final ones that execute after all the packets are sent.

### 3.1.1 Stop-n'-Wait (bit_error_prob=0 & packet_loss_prob=0)



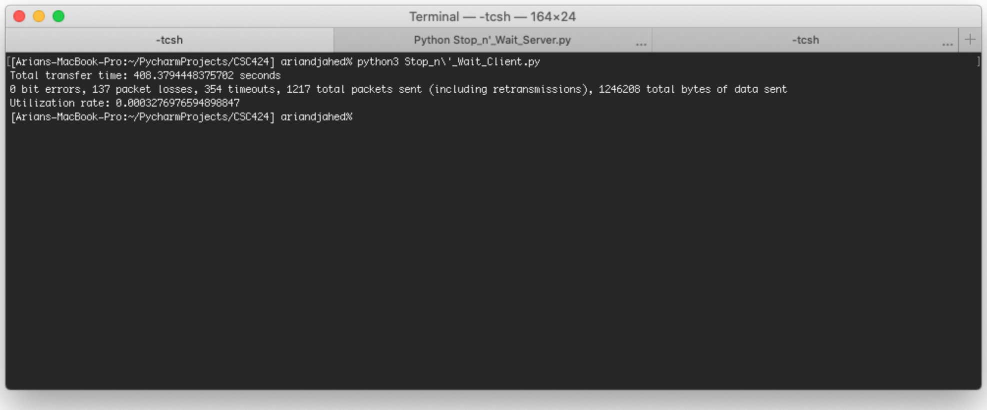
```
[Arians-MacBook-Pro:~/PycharmProjects/CSC424] ariandjahed% python3 Stop_n\'_Wait_Client.py
Total transfer time: 52.89428687095642 seconds
0 bit errors, 0 packet losses, 0 timeouts, 1000 total packets sent (including retransmissions), 1024000 total bytes of data sent
Utilization rate: 5.165457702241838e-05
[Arians-MacBook-Pro:~/PycharmProjects/CSC424] ariandjahed%
```

### 3.1.2 Stop-n'-Wait (bit_error_prob=0.1 & packet_loss_prob=0)



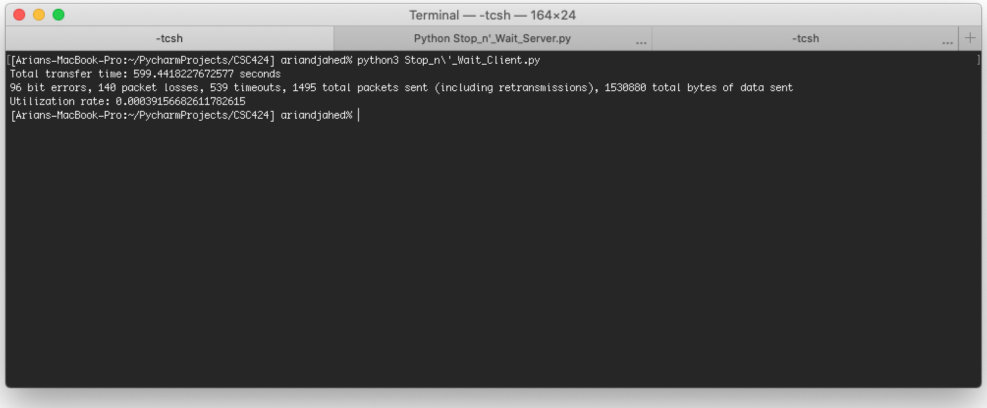
```
[Arians-MacBook-Pro:~/PycharmProjects/CSC424] ariandjahed% python3 Stop_n\'_Wait_Client.py
Total transfer time: 185.62356615066528 seconds
120 bit errors, 0 packet losses, 126 timeouts, 1246 total packets sent (including retransmissions), 1275904 total bytes of data sent
Utilization rate: 0.00014548395972633152
[Arians-MacBook-Pro:~/PycharmProjects/CSC424] ariandjahed%
```

### 3.1.3 Stop-n'-Wait (bit_error_prob=0 & packet_loss_prob=0.1)



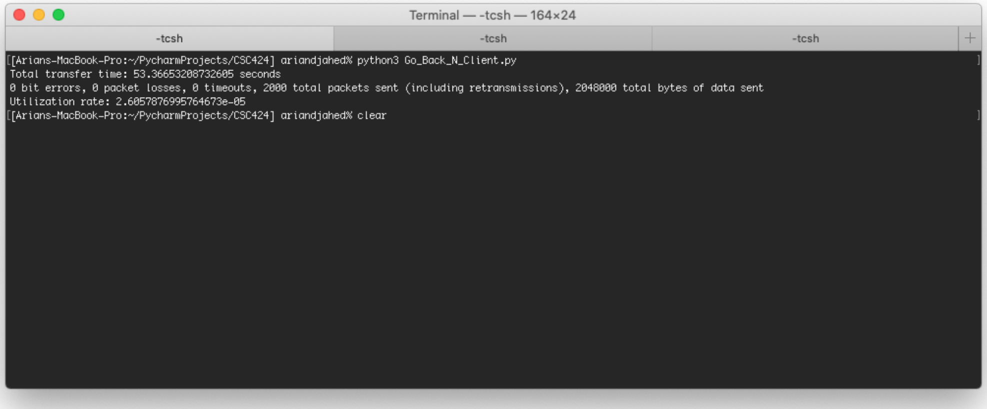
```
[Arians-MacBook-Pro:~/PycharmProjects/CSC424] ariandjahed% python3 Stop_n\'_Wait_Client.py
Total transfer time: 408.3794448375702 seconds
0 bit errors, 137 packet losses, 354 timeouts, 1217 total packets sent (including retransmissions), 1246208 total bytes of data sent
Utilization rate: 0.0003276976594898847
[Arians-MacBook-Pro:~/PycharmProjects/CSC424] ariandjahed%
```

### 3.1.4 Stop-n'-Wait (bit_error_prob=0.1 & packet_loss_prob=0.1)



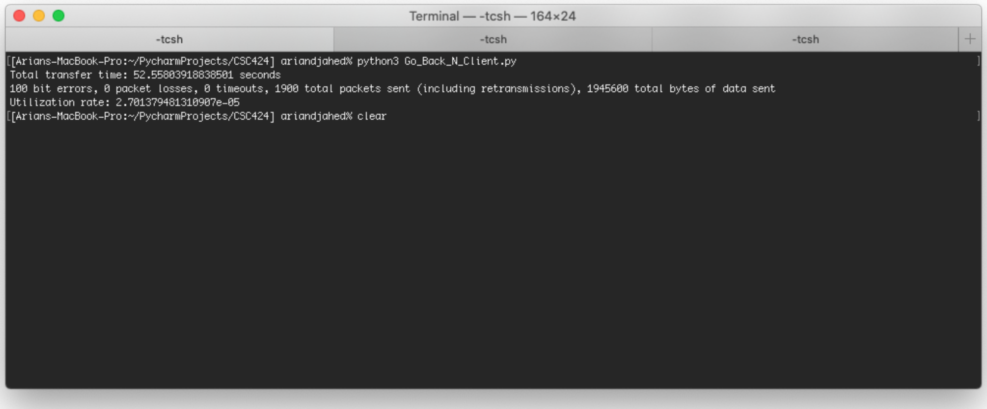
```
[[Arians-MacBook-Pro:~/PycharmProjects/CSC424] ariandjahed% python3 Stop_n\'_Wait_Client.py
 Total transfer time: 599.4418227672577 seconds
 96 bit errors, 140 packet losses, 539 timeouts, 1495 total packets sent (including retransmissions), 1530880 total bytes of data sent
 Utilization rate: 0.00039156682611782615
 [Arians-MacBook-Pro:~/PycharmProjects/CSC424] ariandjahed% |
```

### 3.1.5 Go-back-N (bit_error_prob=0 & packet_loss_prob=0)



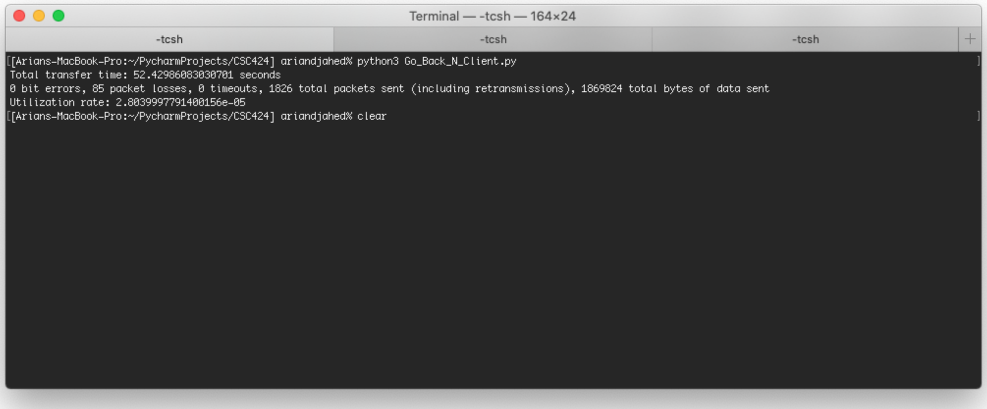
```
[[Arians-MacBook-Pro:~/PycharmProjects/CSC424] ariandjahed% python3 Go_Back_N_Client.py
 Total transfer time: 53.36653208732605 seconds
 0 bit errors, 0 packet losses, 0 timeouts, 2000 total packets sent (including retransmissions), 2048000 total bytes of data sent
 Utilization rate: 2.6057876995764673e-05
[[Arians-MacBook-Pro:~/PycharmProjects/CSC424] ariandjahed% clear
```

### 3.1.6 Go-back-N (bit_error_prob=0.1 & packet_loss_prob=0)



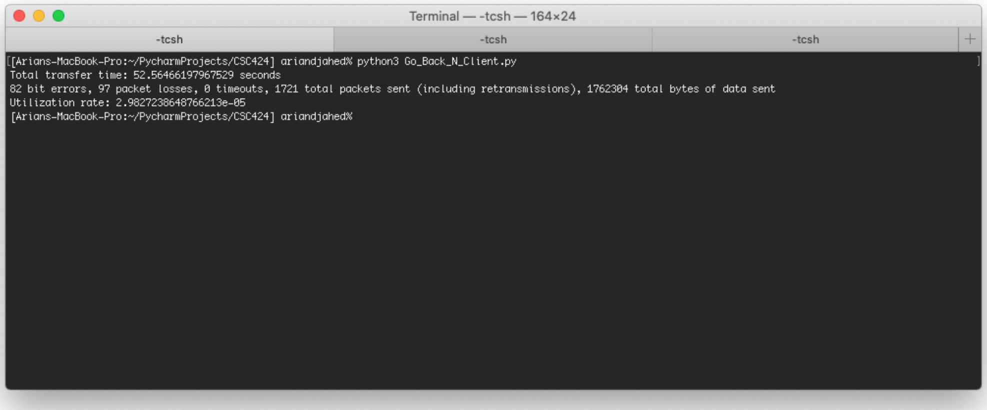
```
[[Arians-MacBook-Pro:~/PycharmProjects/CSC424] ariandjahed% python3 Go_Back_N_Client.py
 Total transfer time: 52.55803918838501 seconds
 100 bit errors, 0 packet losses, 0 timeouts, 1900 total packets sent (including retransmissions), 1945600 total bytes of data sent
 Utilization rate: 2.701379481310907e-05
[[Arians-MacBook-Pro:~/PycharmProjects/CSC424] ariandjahed% clear
```

### 3.1.7 Go-back-N (bit_error_prob=0 & packet_loss_prob=0.1)



```
[[Arians-MacBook-Pro:~/PycharmProjects/CSC424] ariandjahed% python3 Go_Back_N_Client.py
 Total transfer time: 52.42986083030701 seconds
 0 bit errors, 85 packet losses, 0 timeouts, 1826 total packets sent (including retransmissions), 1869824 total bytes of data sent
 Utilization rate: 2.8039997791400156e-05
[[Arians-MacBook-Pro:~/PycharmProjects/CSC424] ariandjahed% clear
```

### 3.1.8 Go-back-N (bit_error_prob=0.1 & packet_loss_prob=0.1)



```
[[Arians-MacBook-Pro:~/PycharmProjects/CSC424] ariandjahed% python3 Go_Back_N_Client.py
 Total transfer time: 52.56466197967529 seconds
 82 bit errors, 97 packet losses, 0 timeouts, 1721 total packets sent (including retransmissions), 1762304 total bytes of data sent
 Utilization rate: 2.9827238648766213e-05
 [Arians-MacBook-Pro:~/PycharmProjects/CSC424] ariandjahed%
```

## 3.2 Data Table

All of the relevant quantitative information that I gathered is laid out in the table below (note: total packets sent includes retransmissions and the second timeout row excludes timeouts casued by packet losses):

| Protocol | Stop-and-Wait | | | | Go-back-N | | | |
|---|---|---|---|---|---|---|---|---|
| bit_error_prob value | 0 | 0.1 | 0 | 0.1 | 0 | 0.1 | 0 | 0.1 |
| packet_loss_prob value | 0 | 0 | 0.1 | 0.1 | 0 | 0 | 0.1 | 0.1 |
| Runtime | 52.89s | 185.62s | 408.38s | 599.44s | 53.37s | 52.56s | 52.43s | 52.56s |
| Bit errors | 0 | 120 | 0 | 96 | 0 | 100 | 0 | 82 |
| Packet losses | 0 | 0 | 137 | 140 | 0 | 0 | 85 | 97 |
| Timeouts | 0 | 126 | 354 | 539 | 0 | 0 | 0 | 0 |
| Timeouts sans PL | 0 | 126 | 217 | 399 | 0 | 0 | 0 | 0 |
| Total packets sent | 1000 | 1246 | 1217 | 1495 | 2000 | 1900 | 1826 | 1721 |
| Total data sent (B) | 1024000 | 1275904 | 1246208 | 1530880 | 2048000 | 1945600 | 1869824 | 1762304 |
| Utilization rate (s/B) | 0.000052 | 0.000145 | 0.000328 | 0.000392 | 0.000026 | 0.000027 | 0.000028 | 0.000029 |

## 3.3 Discussion

For the Stop-n'-Wait cases, it's interesting to note that the packet losses contributed far more to the overall time delay than the bit errors did. This can be seen by the fact that bit errors alone resulted in 132.73 extra seconds of runtime relative to the control case, as opposed to packet losses alone resulting in 350.49 extra seconds of runtime relative to the control case. In addition, every time there was a packet loss, there was also a timeout; however, there was no correlation between bit errors and timeouts. (This is why I made the distinction between total timeouts and timeouts not associated with packet losses in the table above.) I also found it surprising that, in the absence of bit errors and packet losses, no timeouts occurred.

For the Go-back-N cases, the most interesting finding was that—as opposed to the Stop-n'-Wait cases—the runtime remained virtually unchanged throughout, even in the presence of errors. In addition, the total number of packets sent also continually decreased when more errors were introduced. The fact that this amount was 2000 in the absence of errors implies that the client had to send each packet an average of 2 times in order for all of them to go through, and this is most likely a consequence of Go-back-N's "sliding window" feature. It's interesting to note that, in spite of this, the utilization rate still increases as more errors are introduced. However, in spite of this fact, the utilization rates for every Go-Back-N trial are far smaller than the utilization rates for any Stop-n'-Wait trial. Furthermore, there were—surprisingly—never any timeouts in the Go-back-N trials.

## 3.4 Conclusion

In accordance with what is known about the Stop-n'-Wait and Go-back-N protocols, the results of this investigation aptly show that, even though Stop-n'-Wait is more straightforward, Go-back-N excels when it comes to actually providing both fast and consistent performance in the presence of errors. This is because Go-back-N's "sliding window" feature means that the connection established between the sender and the receiver is continuously being used as acknowledgments are being sent. In the case of Stop-n'-Wait, you are quite literally "stopping and waiting" for acknowledgments, meaning that the connection isn't being used at all while the sender awaits an acknowledgment. Even though this means that there are technically more total package transmissions in the case of Go-back-N, it ultimately ends up being a small price to pay for the time saved.