

CS5313: Computer Network

Fall 2020

Instructor: Dr. Deepak Tosh

## **Assignment 2**

**Erik Macik, M.S.**

**Aaron Espinosa, M.S**

## Introduction

It is important to understand how reliable data transfer works. Since reliable data transfer in a general context is considered a problem, as it does not only occur at the transport layer, it also occurs at the link and application layers. The general problem is thus of central importance to networking. [1]

Within the reliable data transfer protocol, a simple alternating bit protocol can be designed. This protocol is also known as a stop-and-wait protocol: after sending each packet the sender stops and waits for feedback from the receiver indicating that the packet has been received. [2]

The stop-and-wait protocol offers good control over the flow of errors, but it can cause significant performance problems, as the sender always waits for confirmation, even if they have the next packet ready to send. The sliding window protocol handles this efficiency problem by sending more than one packet at a time with larger sequence number. Practically it is implemented in two protocols namely: Go back N (GBN) and Selective Repeat (SR) [3]

In this report, we will explain how we implement both versions of the reliable data transfer protocol: Stop-and-Wait protocol and Go-Back-N (GBN) protocol using python programming language. Figure 1.

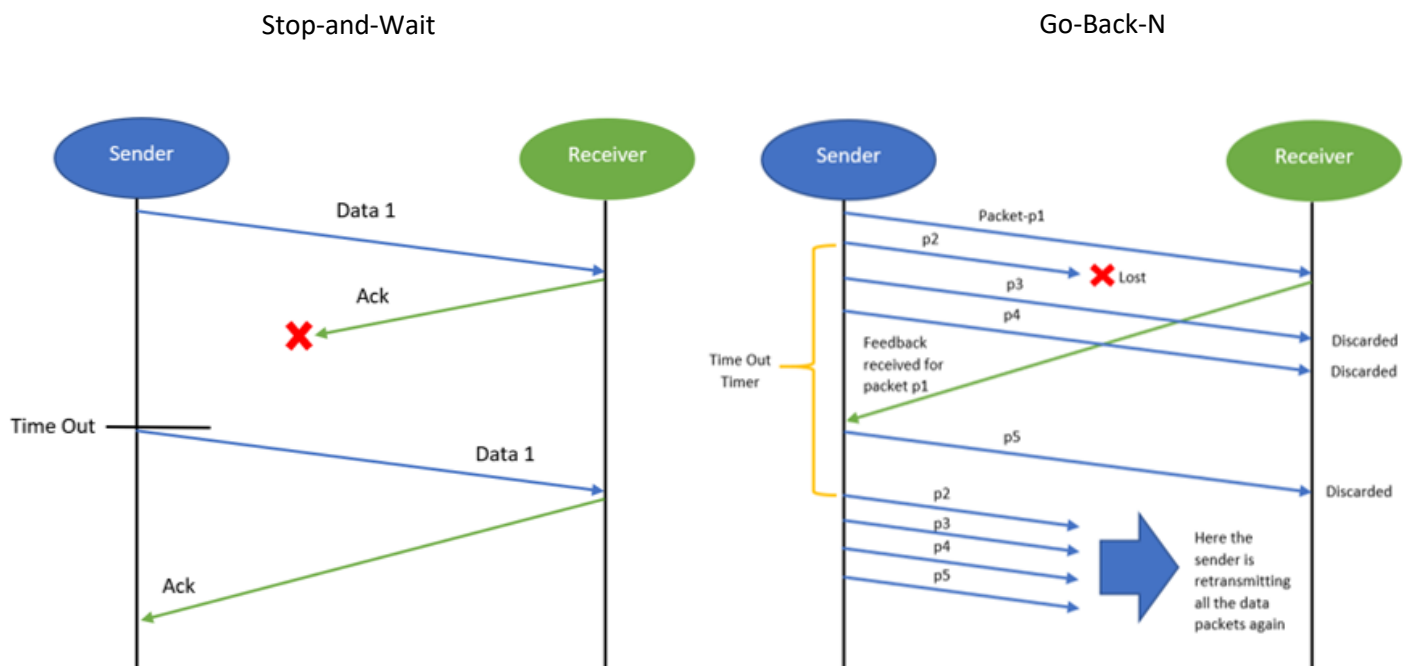
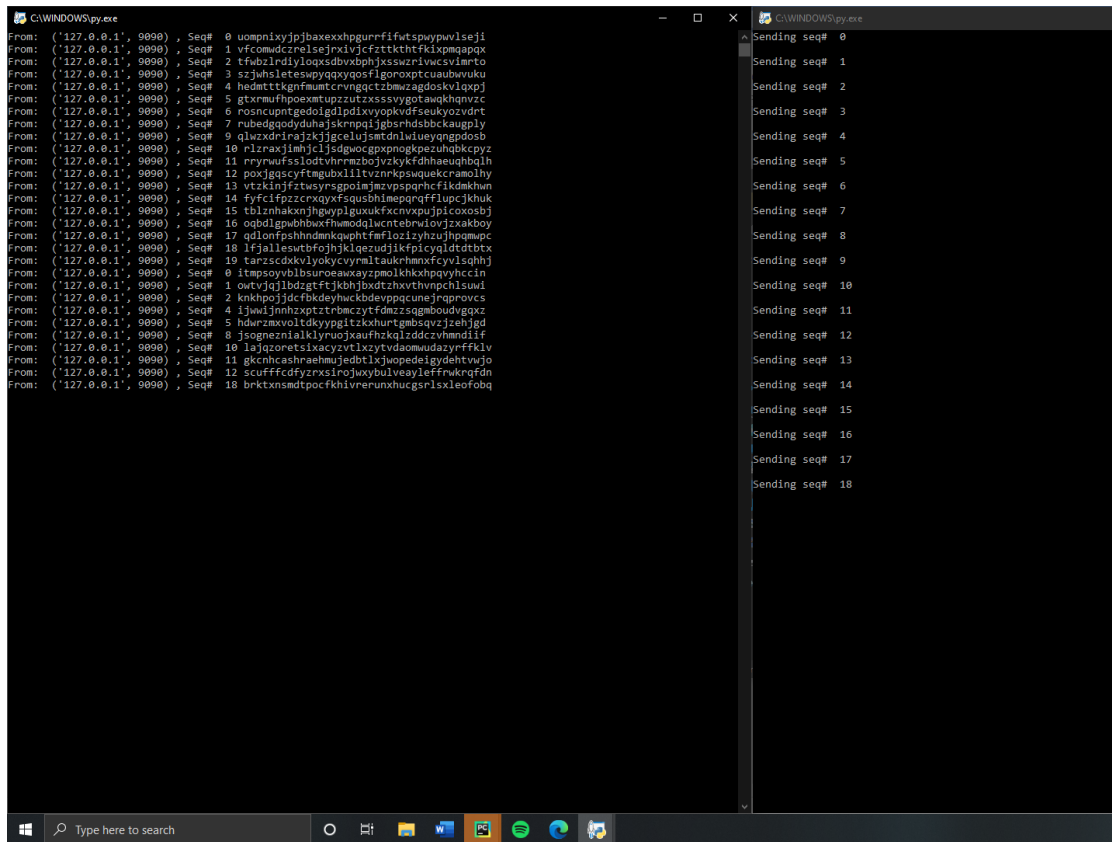


Figure 1 Stop-and-Wait and Go-Back-N protocols.

## Method

- First the receiver and sender files were run on two different consoles to see how it worked. Figure 2.

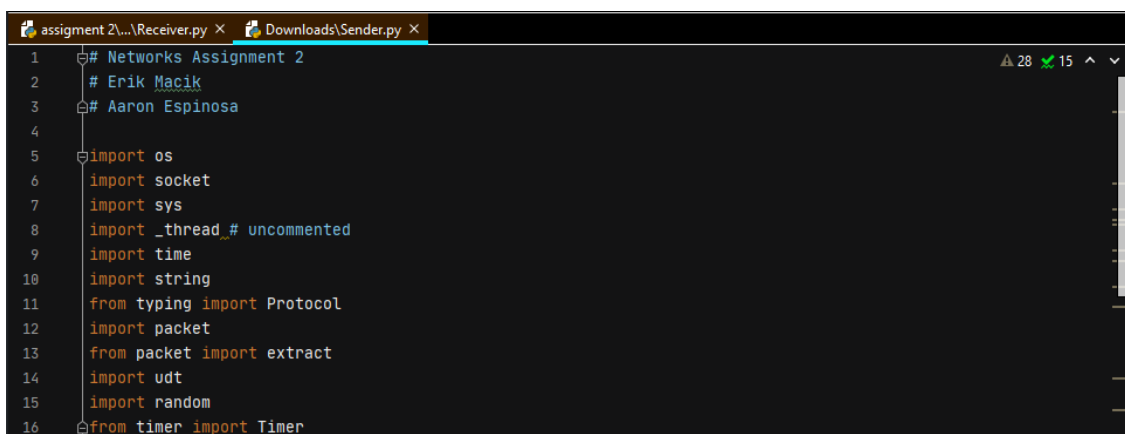


```
C:\WINDOWS\py.exe
From: ('127.0.0.1', 9090), Seq# 0 uompnixjpbaxexhpgunrrfifutspwypwlvseji
From: ('127.0.0.1', 9090), Seq# 1 vfcmdczrelsejrxivjcfztktktftkixpmqapqx
From: ('127.0.0.1', 9090), Seq# 2 tfubzldndyloqsdvxbphjxsswzrivwcvimto
From: ('127.0.0.1', 9090), Seq# 3 szjhsiletespyqayqosflgoroptcuaubawku
From: ('127.0.0.1', 9090), Seq# 4 hedatttgnfaumtcrvngqctzbmzagsdoskvlxqpj
From: ('127.0.0.1', 9090), Seq# 5 gtxrmufhpoeuxtupzutzxsssvygotawqkhqnvzc
From: ('127.0.0.1', 9090), Seq# 6 roscupntgedolgdipdixvyopkvdfseukyzvdrf
From: ('127.0.0.1', 9090), Seq# 7 rubedggoodyduhajskrnpqjgsrhdshbckauggly
From: ('127.0.0.1', 9090), Seq# 9 qlwzdrirajzjkjgcelujstndlwueyqngpdsob
From: ('127.0.0.1', 9090), Seq# 10 rlraxjlmhjcljsguocgpxpnogkpezuhqbkcpyz
From: ('127.0.0.1', 9090), Seq# 11 rryrwufsslodtvhrmzbojvzykykfdhaeqhblh
From: ('127.0.0.1', 9090), Seq# 12 powjgascyftmgubullityznrkpsuquekrcramolhy
From: ('127.0.0.1', 9090), Seq# 13 vtzkinjftatsyngspolmjanvzpsqnhcfikdkhwn
From: ('127.0.0.1', 9090), Seq# 14 fyfcifpzczcxayxfsqushimepgqgfflupcjkhuk
From: ('127.0.0.1', 9090), Seq# 15 tblznhakxnjhgwpilguxukfxcnvxpupjpicoxosbj
From: ('127.0.0.1', 9090), Seq# 16 oqbdlgpwbhbwxfhmodqlwntebvuiolvjzakkboy
From: ('127.0.0.1', 9090), Seq# 17 qdlonfshhndmkwplthmfzozizhyzujpmpuc
From: ('127.0.0.1', 9090), Seq# 18 ifjalliesutbrofjhklqezudjikfpicyqltdtdbtx
From: ('127.0.0.1', 9090), Seq# 19 tarzscdxxlvkyokcyvrmmtaukrhmxfcyvisqhhj
From: ('127.0.0.1', 9090), Seq# 0 itmpsoyvblbsuroeawxayzpmolkhkhxhpqvyhccin
From: ('127.0.0.1', 9090), Seq# 1 owtvjqlbdzgtftfjkbhjbodtzhovthvnpchisudl
From: ('127.0.0.1', 9090), Seq# 2 knkhpjddcfbkdeykwckbdevppacunejgpprovcz
From: ('127.0.0.1', 9090), Seq# 4 ijwajnnhzxptztrbmczytfdmzsqgmboudvgxz
From: ('127.0.0.1', 9090), Seq# 5 hdwrmxvoldtkypgitzkxhuntsqbsqvzjehjd
From: ('127.0.0.1', 9090), Seq# 8 jsgoneznialklyruojkaufhizkqlddczvhandiif
From: ('127.0.0.1', 9090), Seq# 10 lajqoretslkacyzytlxzytdomduazyrrfkiv
From: ('127.0.0.1', 9090), Seq# 11 gkcnhcashaehmujedbtlxjwopedisgydehtwjo
From: ('127.0.0.1', 9090), Seq# 12 scufffcdfyzrsifojxybulveayleffwkrqgfdn
From: ('127.0.0.1', 9090), Seq# 18 brktxnsmdtpocfkhiwrenunxhucsrslxleofobq

C:\WINDOWS\py.exe
Sending seq# 0
Sending seq# 1
Sending seq# 2
Sending seq# 3
Sending seq# 4
Sending seq# 5
Sending seq# 6
Sending seq# 7
Sending seq# 8
Sending seq# 9
Sending seq# 10
Sending seq# 11
Sending seq# 12
Sending seq# 13
Sending seq# 14
Sending seq# 15
Sending seq# 16
Sending seq# 17
Sending seq# 18
```

Figure 2 Receiver and Sender work.

After reviewing the code, we focus on the sender. We add the libraries that would be needed and uncomment `_thread`. Figure 3.



```
1  # Networks Assignment 2
2  # Erik Macik
3  # Aaron Espinosa
4
5  import os
6  import socket
7  import sys
8  import _thread # uncommented
9  import time
10 import string
11 from typing import Protocol
12 import packet
13 from packet import extract
14 import udt
15 import random
16 from timer import Timer
```

Figure 3 Import modules.

In this part the lines of code that are seen in the following image (figure 4) were uncommented and a new variable declared as *end*.

```
18 # Some already defined parameters
19 PACKET_SIZE = 512
20 RECEIVER_ADDR = ('localhost', 8080)
21 SENDER_ADDR = ('localhost', 9090)
22 SLEEP_INTERVAL = 0.05 # (In seconds)
23 TIMEOUT_INTERVAL = 0.5
24 WINDOW_SIZE = 4
25
26 # You can use some shared resources over the two threads
27 base = 0 # uncommented
28 end = 0
29 mutex = _thread.allocate_lock() # uncommented
30 timer = Timer(TIMEOUT_INTERVAL) # uncommented
31
```

Figure 4 Adding and uncomment variables.

Then, we specify the number of bytes of our file data to send assigning to our function payload the file and the size of the package. Later, all packets are getting from a file along with an ending packet. Finally, a list of packets is created of size of *packet\_size* from the file. (Figure 5)

```
34 # Generate random payload of any length
35 def generate_payload(length=10):
36     letters = string.ascii_lowercase
37     result_str = ''.join(random.choice(letters) for i in range(length))
38
39     return result_str
40
41 # Get specified number of bytes of data from file to send
42 def generate_payload_from_file(opened_file, length=PACKET_SIZE):
43     return opened_file.read(length)
44
45 # Get all packets from a file along with an ending packet
46 def get_packets_from_file(filename, packet_size=PACKET_SIZE):
47     packets = []
48     file = open(filename, "r")
49     file_size = os.path.getsize(filename)
50     seq = 0
51
52     # Create a list of packet of size packet_size from file
53     while file_size > 0:
54         data = generate_payload_from_file(file, packet_size).encode()
55         file_size -= packet_size
56         pkt = packet.make(seq, data)
57         packets.append(pkt)
58         print("Created packet seq#", seq)
59         seq += 1
60     end = packet.make(len(packets), "END".encode())
61     packets.append(end)
62     return packets
63
```

Figure 5 Creating and assigning the package size.

And the stop-and-wait protocol code was left the same. Figure 6.

```

65 # Send packets from file using Stop_n_wait protocol
66 def send_snw(sock, filename, packet_size=PACKET_SIZE):
67     global base
68
69     # Get all packets from file
70     packets = get_packets_from_file(filename)
71
72     # Start thread to listen for acks from receiver
73     _thread.start_new_thread(receive_snw, (sock, ))
74
75     # Send each packet
76     while base < len(packets):
77         # base is only incremented by the thread listening for acks
78         pkt = packets[base]
79         print("Sending seq# ", base, "\n")
80         udt.send(pkt, sock, RECEIVER_ADDR)
81         timer.start()
82
83         # loop while the timer is running
84         while timer.running():
85             # if it timed out, resend the packet
86             if timer.timeout():
87                 timer.stop()

```

Figure 6 Stop-and-Wait protocol.

Once having the sender ready we move on to the receiver code. Here, we incorporate the missing part of the Stop-and-Wait code adding the file name and the variables that we used. Then, we added a while loop to continually receive the packets until we get the last one. Inside of the same loop, we used a condition statements if to tell the program if we already have the packet send a repeat ack, and write the data to a file if it does not end then send an ack of that packet to the sender. Figure 7.

```

72 # Receive packets from the sender w/ Stop-n-wait protocol
73 def receive_snw(sock, filename):
74     # File to write to
75     file = open(filename, "w")
76     ack_number = 0
77     endStr = ''
78
79     # Continually receive packets until we get the last one
80     while endStr != 'END':
81         pkt, senderaddr = udt.recv(sock)
82         seq, data = packet.extract(pkt)
83
84         # If we already have this packet, just send back a repeat ack
85         if seq < ack_number:
86             ack_packet = packet.make(seq, ''.encode())
87             print("Sending Repeat ack#", seq)
88             udt.send(ack_packet, sock, SENDER_ADDR)
89             continue
90
91         # Write the data to a file if it's not the end
92         endStr = data.decode()
93         if endStr != 'END':
94             file.write(endStr)
95
96         # Send ack of this packet to sender
97         print("Received #", seq, "\n")
98         ack_packet = packet.make(ack_number, ''.encode())
99         print("Sending ack#", ack_number)
100         udt.send(ack_packet, sock, SENDER_ADDR)
101
102         # Keep track of sequence numbers received
103         ack_number += 1

```

Figure 7 Stop-and-Wait Receiver code.

Once having the sender ready we move on to the receive code. Here, we incorporate the missing part of the GBN code adding the file name and the variables that we used. Then, we added a while loop to continually receive the packets until we get the last one. Figure 7.

```

20 # Continually receive packets until we get the last one
21 while endStr != 'END':
22     pkt, senderaddr = udt.recv(sock)
23     seq, data = packet.extract(pkt)
24
25     if seq == ack_num:
26         ack_packet = packet.make(ack_num, ''.encode())
27         print("sending ack for packet#", ack_num)
28         udt.send(ack_packet, sock, SENDER_ADDR)
29         ack_num += 1
30         # Write the data to a file if it's not the end
31         endStr = data.decode()
32         if endStr != 'END':
33             file.write(endStr)

```

Figure 8 Receive packets from the sender Stop-and-Wait.

Finally, for the GBN protocol we incorporate the missing part code adding the file name and the variables that we used as well. Then, we added a while loop to continually receive the packets until we get the last one. And again, we used condition statements if to write the data to a file if it is not the end. Figure 9.

```

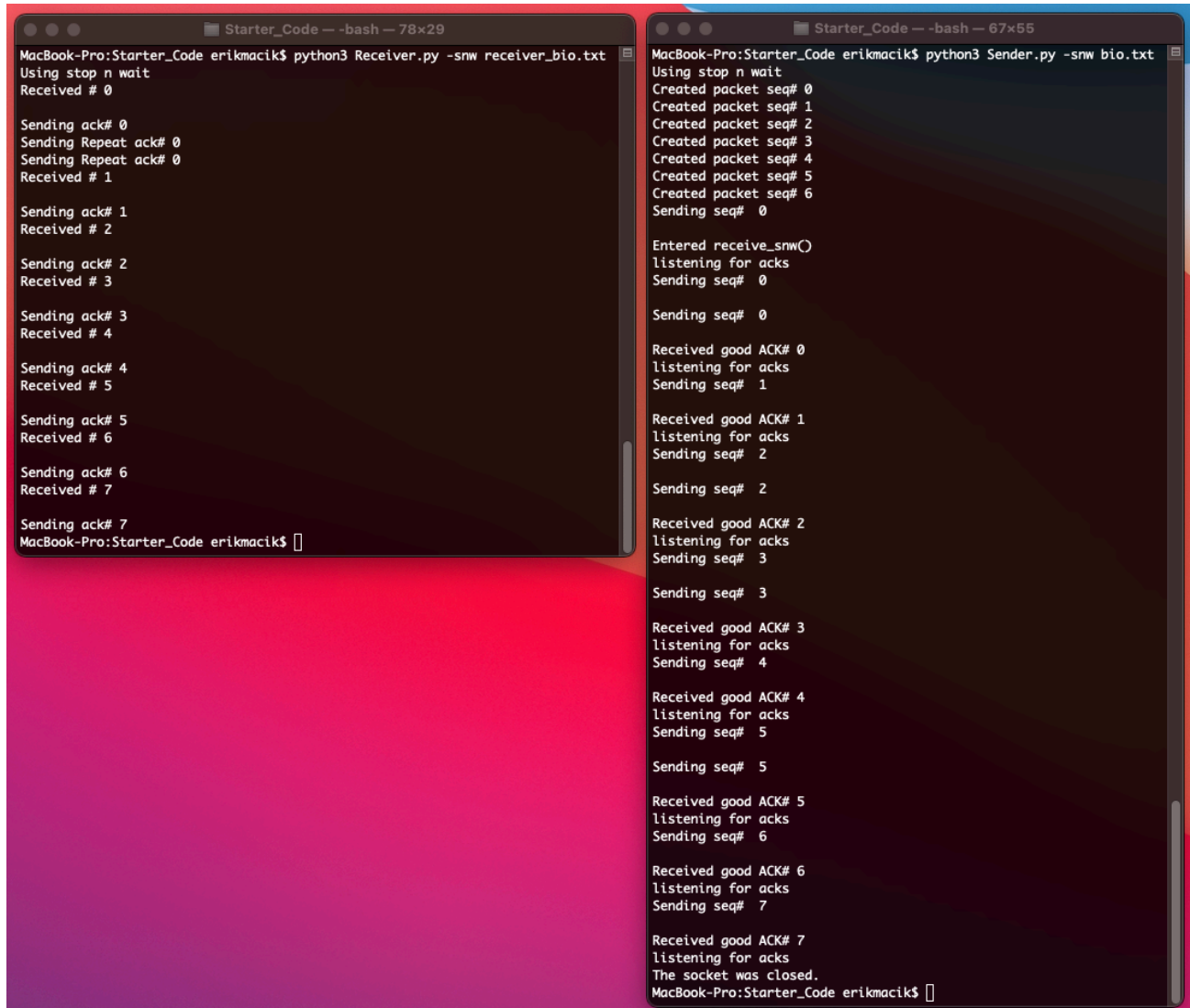
14 # Receive packets from the sender w/ GBN protocol
15 def receive_gbn(sock, filename):
16     file = open(filename, "w")
17     ack_num = 0
18     endStr = ''
19
20     # Continually receive packets until we get the last one
21     while endStr != 'END':
22         pkt, senderaddr = udt.recv(sock)
23         seq, data = packet.extract(pkt)
24
25         if seq == ack_num:
26             ack_packet = packet.make(ack_num, ''.encode())
27             print("sending ack for packet#", ack_num)
28             udt.send(ack_packet, sock, SENDER_ADDR)
29             ack_num += 1
30             # Write the data to a file if it's not the end
31             endStr = data.decode()
32             if endStr != 'END':
33                 file.write(endStr)

```

Figure 9 Receive packets from the sender GBN.

## Conclusions

### Stop-and-Wait Protocol



```
MacBook-Pro:Starter_Code erikmacik$ python3 Receiver.py -snw receiver_bio.txt
Using stop n wait
Received # 0

Sending ack# 0
Sending Repeat ack# 0
Sending Repeat ack# 0
Received # 1

Sending ack# 1
Received # 2

Sending ack# 2
Received # 3

Sending ack# 3
Received # 4

Sending ack# 4
Received # 5

Sending ack# 5
Received # 6

Sending ack# 6
Received # 7

Sending ack# 7
MacBook-Pro:Starter_Code erikmacik$
```

```
MacBook-Pro:Starter_Code erikmacik$ python3 Sender.py -snw bio.txt
Using stop n wait
Created packet seq# 0
Created packet seq# 1
Created packet seq# 2
Created packet seq# 3
Created packet seq# 4
Created packet seq# 5
Created packet seq# 6
Sending seq# 0

Entered receive_snw()
listening for acks
Sending seq# 0

Sending seq# 0

Received good ACK# 0
listening for acks
Sending seq# 1

Received good ACK# 1
listening for acks
Sending seq# 2

Sending seq# 2

Received good ACK# 2
listening for acks
Sending seq# 3

Sending seq# 3

Received good ACK# 3
listening for acks
Sending seq# 4

Received good ACK# 4
listening for acks
Sending seq# 5

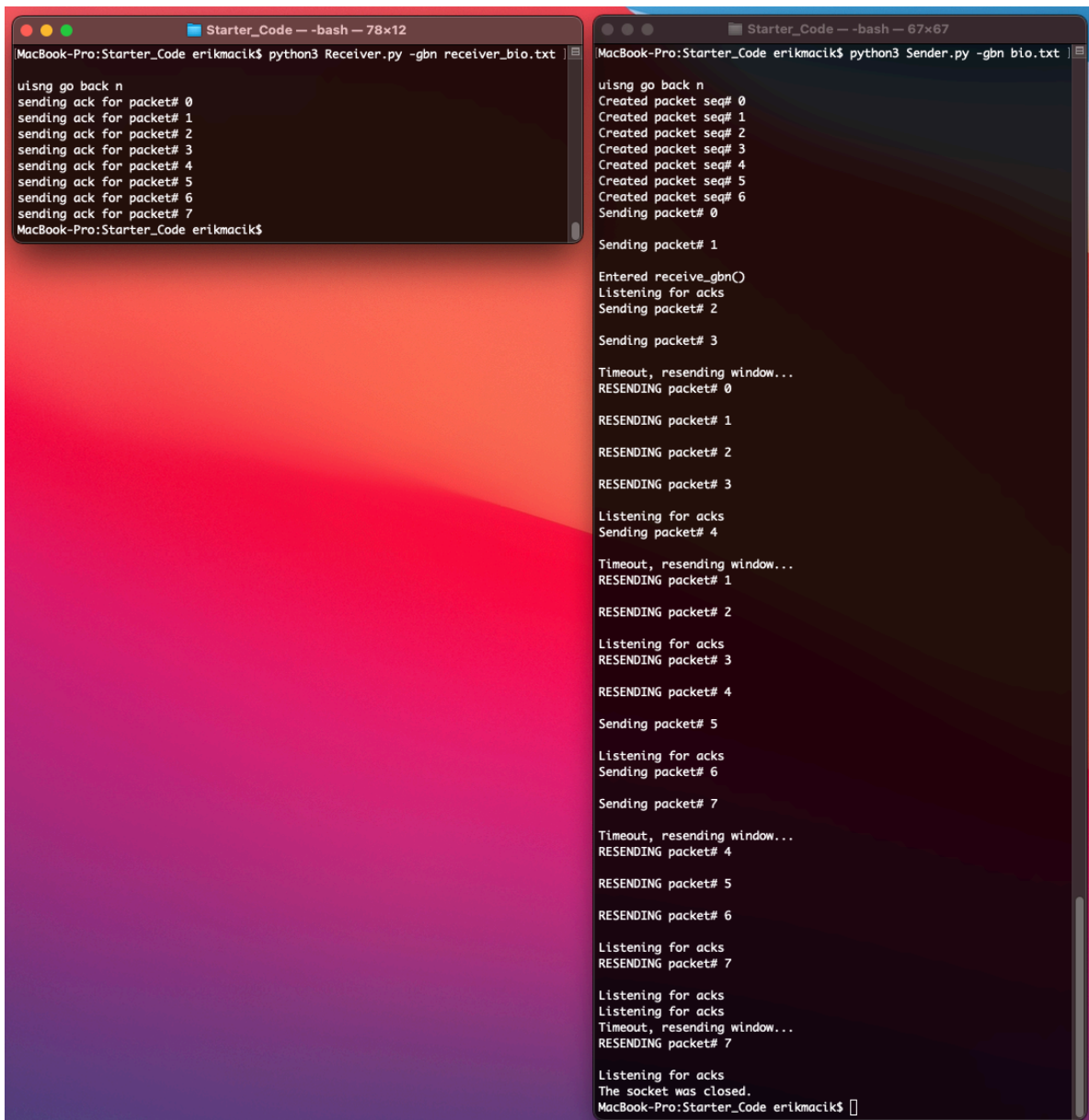
Sending seq# 5

Received good ACK# 5
listening for acks
Sending seq# 6

Received good ACK# 6
listening for acks
Sending seq# 7

Received good ACK# 7
listening for acks
The socket was closed.
MacBook-Pro:Starter_Code erikmacik$
```

## Go-Back-N Protocol



```
MacBook-Pro:Starter_Code erikmacik$ python3 Receiver.py -gbn receiver_bio.txt
using go back n
sending ack for packet# 0
sending ack for packet# 1
sending ack for packet# 2
sending ack for packet# 3
sending ack for packet# 4
sending ack for packet# 5
sending ack for packet# 6
sending ack for packet# 7
MacBook-Pro:Starter_Code erikmacik$

MacBook-Pro:Starter_Code erikmacik$ python3 Sender.py -gbn bio.txt
using go back n
Created packet seq# 0
Created packet seq# 1
Created packet seq# 2
Created packet seq# 3
Created packet seq# 4
Created packet seq# 5
Created packet seq# 6
Sending packet# 0
Sending packet# 1
Entered receive_gbn()
Listening for acks
Sending packet# 2
Sending packet# 3
Timeout, resending window...
RESENDING packet# 0
RESENDING packet# 1
RESENDING packet# 2
RESENDING packet# 3
Listening for acks
Sending packet# 4
Timeout, resending window...
RESENDING packet# 1
RESENDING packet# 2
Listening for acks
RESENDING packet# 3
RESENDING packet# 4
Sending packet# 5
Listening for acks
Sending packet# 6
Sending packet# 7
Timeout, resending window...
RESENDING packet# 4
RESENDING packet# 5
RESENDING packet# 6
Listening for acks
RESENDING packet# 7
Listening for acks
Listening for acks
Timeout, resending window...
RESENDING packet# 7
Listening for acks
The socket was closed.
MacBook-Pro:Starter_Code erikmacik$
```



## Students contribution

Task list	Erik Macik	Aaron Espinosa
Code	✓	✓
Report	✓	✓

## References

- [1] James F. Kurose, Keith W. Ross, «3.4 Principles of Reliable Data Transfer,» de *Computer Networking A Top-Down Approach*, New Jersey, Pearson, 2017.
- [2] G. Shute, «Reliable Data Transfer,» University Of Minnesota Duluth, [En línea]. Available: <https://www.d.umn.edu/~gshute/net/reliable-data-transfer.xhtml#:~:text=For%20connection%2Doriented%20service%20provided,designed%20using%20some%20basic%20tools..> [Último acceso: 5 Octubre 2020].
- [3] James F. Kurose, Keith W. Ross, «Computer Networking A Top-Down Approach,» de *3.4.3 Go-Back-N (GBN)*, New Jersey, Pearson, 2017.