David Amparan & Noah Saenz (Group 13)

Dr. Deepak Tosh

CS 4316: Computer Networks

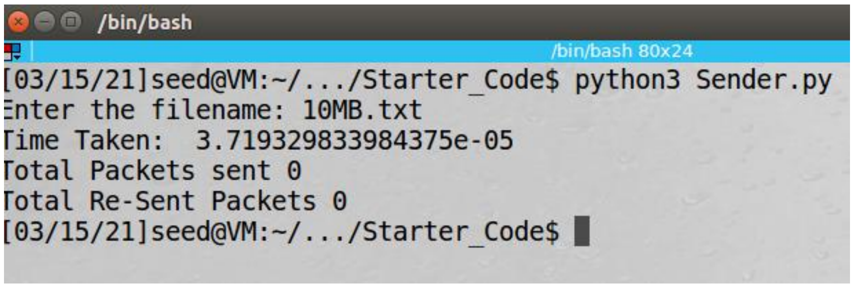
March 14, 2021

Assignment 3 [Reliable Data Transfer]

Part-1: File Transfer using Stop-and-Wait(SnW) Protocol

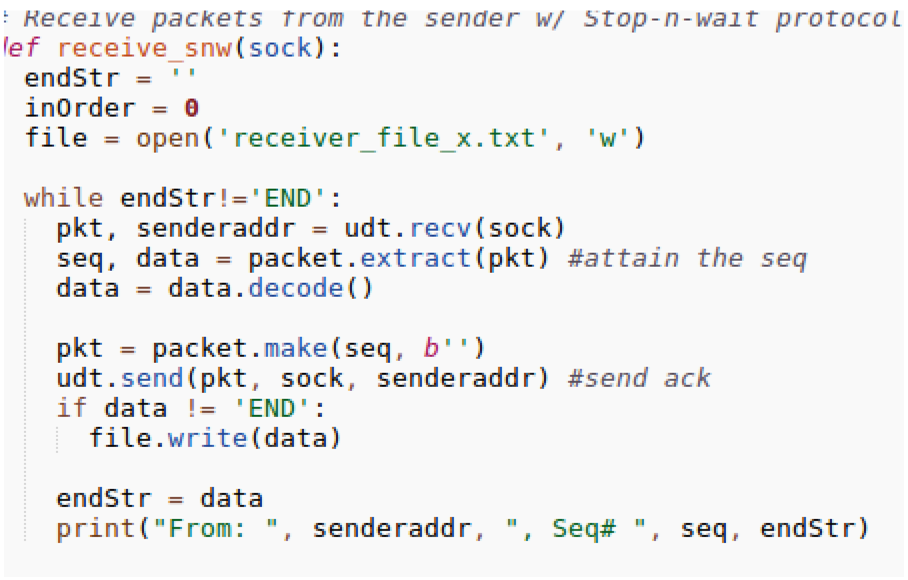
Having gone over RDT 3.0 otherwise known as Stop-and-Wait protocol, in class did not make it seem something too difficult to replicate it code. Learning about it and applying it are two entirely different things, however. Moving along, the challenges we faced with this portion was not because of the understanding of the protocol. It was the need to successfully replicate the behavior using threads. Per the instructions two threads should be used, one to send and the other to receive the acks (Sender.py). We were able to get the thread for receiving the acks but the main thread to begin the send_sw did not function correctly and thus did not execute anything as you can see below. Therefore, we decided to call send_snw directly without a thread.

```
#_thread.start_new_thread(send_snw, (sock, filename))
send_snw(sock, filename)
print('Time Taken: ', time.time() - t)
print('Total Packets sent', total_Num)
print('Total Re-Sent Packets', total_Num)
sock.close()
```



```
[03/15/21]seed@VM:~/.../Starter_Code$ python3 Sender.py
Enter the filename: 10MB.txt
Time Taken:  3.719329833984375e-05
Total Packets sent 0
Total Re-Sent Packets 0
[03/15/21]seed@VM:~/.../Starter_Code$ 
```

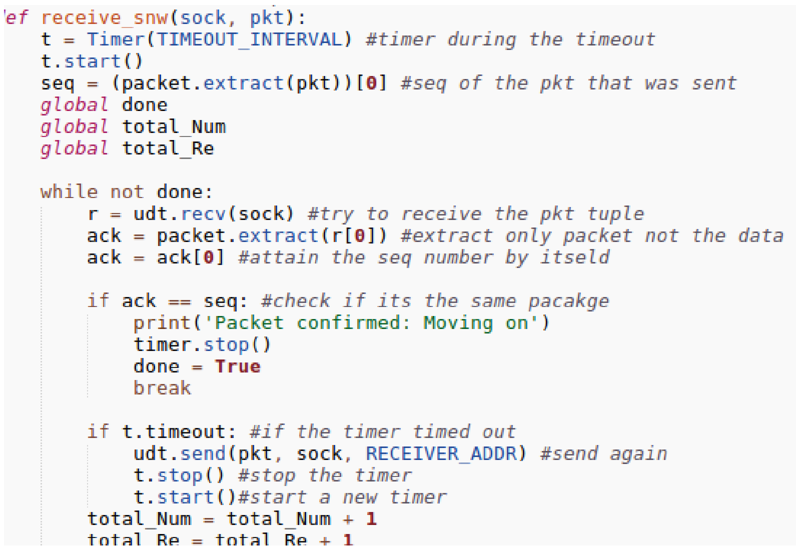
Our send_snw() was modified to take in the filename for the file you would like to send, this was mainly added for testing purposes and to make sure what file would be sent without the need to hardcode it. The actual method mimics the behavior from the original sample code given by Dr. Tosh, read data, create a packet, send packet, wait till confirmed, increment seq, and repeat. The payload was up to 512 like stated in the instructions, therefore the documents was separated into x number of packets containing 512 bytes respectively. The main portions of the method, however, lie after sending the packet. Here we call a new thread with method receive_snw(), meanwhile send remains waiting using a global var to while loop till it is confirmed.



```
Receive packets from the sender w/ Stop-n-wait protocol
ef receive_snw(sock):
  endStr = ''
  inOrder = 0
  file = open('receiver_file_x.txt', 'w')

  while endStr!='END':
    pkt, senderaddr = udt.recv(sock)
    seq, data = packet.extract(pkt) #attain the seq
    data = data.decode()

    pkt = packet.make(seq, b'')
    udt.send(pkt, sock, senderaddr) #send ack
    if data != 'END':
      file.write(data)

    endStr = data
    print("From: ", senderaddr, ", Seq# ", seq, endStr)
```

The global var done is changed within the receive method in which we handle all the cases when the packet arrives or does not arrive. Within this method we set a timer given the timer interval and will once again repeat checking for the reception of a packet, retransmission, and timeout until the done var is set to true and set to false once again when it successfully exits the method.

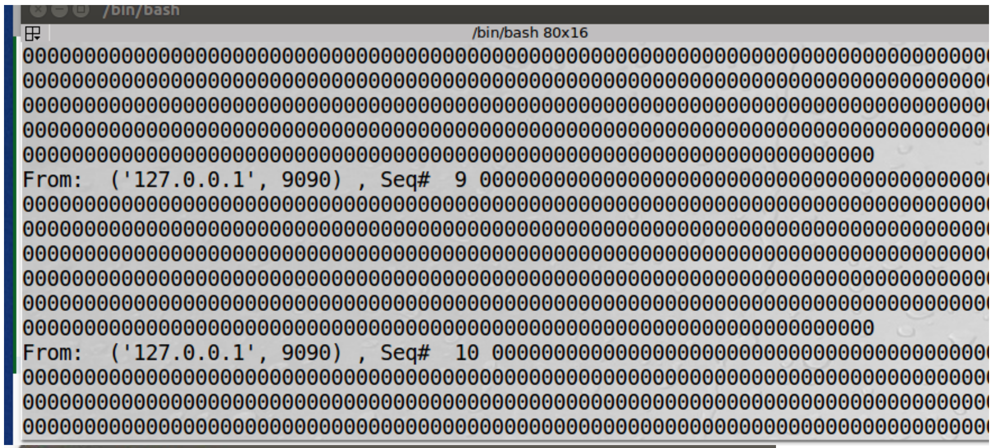
Over on Receiver.py the code implemented once again mimicked what was given to use by Dr. Tosh in the sample code. Except, now we open the file in which we will store what is transferred and is on a continues while loop until the terminator is reached. While each packet is received, we unpack it, copy the data to the text file, and send back a blank pkt with the same sequence number that was just given to us.



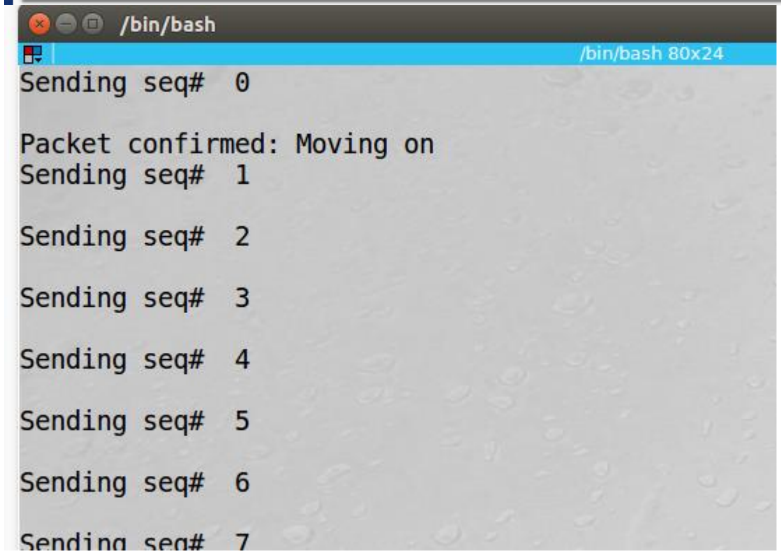
```
ef receive_snw(sock, pkt):
    t = Timer(TIMEOUT_INTERVAL) #timer during the timeout
    t.start()
    seq = (packet.extract(pkt))[0] #seq of the pkt that was sent
    global done
    global total_Num
    global total_Re

    while not done:
        r = udt.recv(sock) #try to receive the pkt tuple
        ack = packet.extract(r[0]) #extract only packet not the data
        ack = ack[0] #attain the seq number by itseld

        if ack == seq: #check if its the same pacakge
            print('Packet confirmed: Moving on')
            timer.stop()
            done = True
            break

        if t.timeout: #if the timer timed out
            udt.send(pkt, sock, RECEIVER_ADDR) #send again
            t.stop() #stop the timer
            t.start()#start a new timer
        total_Num = total_Num + 1
        total_Re = total_Re + 1
```

Results:

Our Stop and Wait protocol worked however it did have its flaws, for one sometimes the global var done can cause the program to get stuck as early as the second packet. This is really all randomized as the probability to lose the packets is random, however, all other functionalities work correctly. Unfortunately, the biggest point of the lab which was the confirmation of packets was not done as efficiently as possible. These issues can be fixed with locking the thread however, we could not figure out how to make the lock work correctly with our program.



```
0000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000
From:  ('127.0.0.1', 9090) , Seq#  9 000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000
From:  ('127.0.0.1', 9090) , Seq#  10 00000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000
```



```
Sending seq#  0

Packet confirmed: Moving on
Sending seq#  1

Sending seq#  2

Sending seq#  3

Sending seq#  4

Sending seq#  5

Sending seq#  6

Sending seq#  7
```

David	Worked on Sender.py (send_snw) and part of Receiver,py
Noah	Worked on Sender.py (receive_snw) and Part of Receiver.py
Both	Debugged and collaborated with one another to find solutions