Header :

1. Name : Devajya Khanna
2. Submission Date : 3/18/2024
3. Information : Language used is python, file to run is mainSNW.py.
4. Assumption : I have only made a single assumption which is that the message sent to layer three, should not be counted in the counters. I make this assumption since the purpose of the counters seems to be that we are using them to compare loss, corruption and sender-receiver interaction. Adding counters reduces symmetry, making it harder and less useful to analyze the values reported by the counter.

Compilation :

1. If on a linux environment navigate to the folder with the files.
2. Run command "python3 mainSNW.py" to execute the file for Stop and wait or single bit protocol and “python3 mainGBN.py” for the go back n or sliding window protocol.

Execution :

1. Open the file simulator.py
2. Modify the variables present, to change the probabilities of loss, corruption, number of packets to send, rate of sending etc.
3. Follow the instructions under the compilation section.
4. On execution, the information is dumped onto the terminal

Description :

- Overview :
    - Phase 1 of this project concerns 4 major files, and their dependencies. SNW_Sender and SNW_Receiver emulate the sender and receiver sides of Stop and Wait packet transmission that is simulated by the file simulator.py. The execution of this program can be done through running mainSNW.py. Additionally there are files that will be mentioned later as relevant that provide useful utilities for enabling the required functionality of the system
    - Phase 2 of this project also concerns 4 files and their dependencies. GBN_Sender and GBN_Receiver simulate the sending and reception of packets through simulator.py. The execution of this program can be done through running mainGBN.py. Additionally there are files that will be mentioned later as relevant that provide useful utilities for enabling the required functionality of the system

- File Descriptions - Phase 1:
    - SNW_Sender.py : This file implements the functionality required for enacting the "sending packets" behavior of the simulation. It initializes the sender with some global variables and has functions to handle sending a message from an upper layer to a lower layer, receiving an ack from a lower layer and handling time-outs.
        - __init__() : Responsible for initializing global variables for the Sender.
            - State - Holds current state, which is either WAIT_LAYER5 or WAIT_ACK. The former state indicates that the sender is waiting for a message from the application layer, and the latter that it is waiting to receive an acknowledgement for a previously sent packet.
            - Seq - current sequence number (0 or 1),
            - Estimated_rtt - hypothesized round trip time after which the timer should elapse,
            - Entity - a demarcator of calling entity for the simulator's purpose (the string "S").
            - **lastTransmit - Contains the last transmitted message, useful for retransmission.**
        - S_output(message) : This function encapsulates and sends the data received as a packet to the layer below itself. The function works as follows :
            - Check if the sender is waiting for a message. If it is, the call to output is valid and so create a packet with the data passed through the message parameter. Note, the packet.py file contains useful abstractions for packets. Update counters to keep track of what happens to the packets and then send the packet (using the provided function to_layer_three()) and start a timer to keep track of the corresponding expected ack.
            - If the sender is not in a state to receive data from the application, print a message indicating that it is waiting for an ack and the current message has been ignored/dropped.
        - S_input(received_packet) : This function is called when the sender receives an acknowledgement. This is how the function works
            - Check if the sender is waiting for an ack. If it isn't, drop the ack and return.
            - If the sender is waiting for an ack, check that the packet received hasn't been corrupted, using checksum, and that it has the correct sequence number (corresponding to the last sent packet's seq number). If either of these things are true, update the necessary counters, and return
            - Otherwise, remove the timer, since the ack has been received, change the state of the sender, and invert the sequence number of the packet to send.
        - S_handle_timer() : This function handles the elapsing of timers set for receiving acks to sent packets. Its workings are pretty simple. It checks if the sender is waiting for an ack. If it is, and the function has been triggered then the timer has elapsed. Then, it uses the utilities in event_list.py and other files to retransmit the last sent packet, update counters, and restart the timer.

- SNW_Receiver.py :
  - __init__() : Responsible for initializing useful global variables for the receiver instance
    - Seqnum - The sequence number the receiver is expecting for the next packet.
    - Entity - A string demarcating the calling entity for the simulator (“R”)
  - R_input(received_packet) : This function handles the reception of a packet of data from the sender. The following steps detail its workings
    - If the packet has an unexpected sequence number resend the ack for the previous packet (1-seqnum) and update the relevant counters, or if the packet is corrupted (as determined by checksum) simply update counters and return.
    - Otherwise, pass the payload data to layer5 using the provided function (to_layer_five), send an acknowledgement to the sender, update the sequence number to anticipate the next packet and update the counters. Then return
- The other two files involved in Phase 1 are mainSNW.py and simulation.py which agave not been modified

- File Description - Phase 2
  - GBN_Sender.py : This file implements the functionality required for enacting the “sending packets” behavior of the simulation when using the GBN protocol. It initializes the sender with some global variables and has functions to handle sending a message from an upper layer to a lower layer, receiving an ack from a lower layer and handling time-outs.
    - __init__() : Responsible for initializing global variables for the Sender.
      - Seq - the sequence number for the next packet to be sent.
      - Estimated_rtt - hypothesized round trip time after which the timer should elapse.
      - Entity - a demarcator of calling entity for the simulator's purpose (the string “S”).
      - C_b - the circular buffer used to implement the packet buffer for the simulation.
      - **Window_size - The number of un-ack’d messages allowed to exist in the buffer at any one given time.**
      - **Send_base - the base pointer or position of oldest unacknowledged message.**
    - S_output(message) : This function encapsulates and sends the data received as a packet to the layer below itself. The function works as follows :
      - Check if the buffer is full. If it is, then simply drop the received message. Otherwise, check if the packet attempting to be sent, is within the permissible range of base + window size. Note, the packet.py file contains useful abstractions for packets. If it is, allow the packet to be sent and update the sequence num variable accordingly. Update counters to keep track of what happens to the packets and then send the packet

(using the provided function to_layer_three()) and start a timer to keep track of the corresponding expected ack if it is the base packet. Otherwise no timer is needed, as the oldest unacknowledged packet already has a timer on it.
        - If the sequence number is not within range, drop the message.
    - S_input(received_packet) : This function is called when the sender receives an acknowledgement. This is how the function works
        - Check if the packet is corrupted and simply return if it is. Otherwise,
        - If the acknowledgement number of the received_packet is greater than or equal to the base pointer, delete all packets from before that acknowledgement number from the buffer. This is because the buffer would only send an ACK with a higher number if all previous packets had been received. Also update the base pointer to point to the next packet that hasn't been ACK'd.
        - Note if the ack received was from the base pointer, then remove it's timer. Otherwise simply restart the timer. This is so we can keep a timer on the oldest un-ACK'd message sent.
    - S_handle_timer() : This function handles the elapsing of timers set for receiving acks to sent packets. Its workings are pretty simple. When called, it starters a timer, to ensure it can be called again if needed, and resends all of the packets in the buffer(since they are all un-ack'd)
- GBN_Receiver.py :
    - __init__() : Responsible for initializing useful global variables for the receiver instance
        - Seqnum - The sequence number the receiver is expecting for the next packet.
        - Entity - A string demarcating the calling entity for the simulator ("R")
        - **seqOfLastCorrectlyReceived -** This variable keeps track of the sequence number of the last received message. This is so that if a packet fails to pass through the if statements, an ack with an older acknowledgement can be sent.
    - R_input(received_packet) : This function handles the reception of a packet of data from the sender. The following steps detail its workings
        - If the packet has an unexpected sequence number resend the ack for the previous packet (seOfLastCorrectlyReceived) and update the relevant counters, or if the packet is corrupted (as determined by checksum) simply update counters and return.
        - Otherwise, pass the payload data to layer5 using the provided function (to_layer_five), send an acknowledgement to the sender, update the sequence number to anticipate the next packet and update the counters. Then return
-

- Evaluation Phase - 1:
  - Test Case 1 - nsimmax = 30, lossprob = 0, corruptprob = 0, Lambda = 1000 :

```
data received:  aaaaaaaaaaaaaaaaaaaa
data received:  bbbbbbbbbbbbbbbbbbbb
data received:  cccccccccccccccccccc
data received:  dddddddddddddddddddd
data received:  eeeeeeeeeeeeeeeeeeee
data received:  ffffffffffffffffffff
data received:  gggggggggggggggggggg
data received:  hhhhhhhhhhhhhhhhhhhh
data received:  iiiiiiiiiiiiiiiiiiii
data received:  jjjjjjjjjjjjjjjjjjjj
data received:  kkkkkkkkkkkkkkkkkkkk
data received:  llllllllllllllllllll
data received:  mmmmmmmmmmmmmmmmmmmm
data received:  nnnnnnnnnnnnnnnnnnnn
data received:  oooooooooooooooooooo
data received:  pppppppppppppppppppp
data received:  qqqqqqqqqqqqqqqqqqqq
data received:  rrrrrrrrrrrrrrrrrrrr
data received:  ssssssssssssssssssss
data received:  tttttttttttttttttttt
data received:  uuuuuuuuuuuuuuuuuuuu
data received:  vvvvvvvvvvvvvvvvvvvv
data received:  wwwwwwwwwwwwwwwwwwww
data received:  xxxxxxxxxxxxxxxxxxxx
data received:  yyyyyyyyyyyyyyyyyyyy
data received:  zzzzzzzzzzzzzzzzzzzz
data received:  aaaaaaaaaaaaaaaaaaaa
data received:  bbbbbbbbbbbbbbbbbbbb
data received:  cccccccccccccccccccc
data received:  dddddddddddddddddddd
simulation complete
==========STATISTICS==========
 Total Number of Messages Sent                -> 60
 Total Number of Retransmissions              -> 0
   Total Number of Retransmitted Data Packets -> 0
   Total Number of Retransmitted ACKs         -> 0
 Total Number of Lost Packets                 -> 0
   Total Number of Lost Data Packets          -> 0
   Total Number of Lost ACKs                  -> 0
 Total Number of Dropped Packets              -> 0
   Total Number of Dropped Data Packets       -> 0
   Total Number of Dropped ACKs               -> 0
 Total Number of Corrupted Packets            -> 0
   Total Number of Corrupted Data Packets     -> 0
   Total Number of Corrupted ACKs             -> 0
 Final Simulation Time                        -> 30006.030682399498
==========STATISTICS==========
```

    - Justification : This outcome is relatively simple to explain. Since there is no loss or corruption, no packets are required to be retransmitted so 60 total messages are sent, 30 “messages” and 30 acks. The simulation time can be taken as a baseline for future cases.

- Test Case 2 - nsimmax = 30, lossprob = 0, corruptprob = 0, Lambda = 100 :

```
data received:  aaaaaaaaaaaaaaaaaaaa
data received:  bbbbbbbbbbbbbbbbbbbb
data received:  cccccccccccccccccccc
data received:  dddddddddddddddddddd
data received:  eeeeeeeeeeeeeeeeeeee
data received:  ffffffffffffffffffff
data received:  gggggggggggggggggggg
data received:  hhhhhhhhhhhhhhhhhhhh
data received:  iiiiiiiiiiiiiiiiiiii
data received:  jjjjjjjjjjjjjjjjjjjj
data received:  kkkkkkkkkkkkkkkkkkkk
data received:  llllllllllllllllllll
data received:  mmmmmmmmmmmmmmmmmmmm
data received:  nnnnnnnnnnnnnnnnnnnn
data received:  oooooooooooooooooooo
data received:  pppppppppppppppppppp
data received:  qqqqqqqqqqqqqqqqqqqq
data received:  rrrrrrrrrrrrrrrrrrrr
data received:  ssssssssssssssssssss
data received:  tttttttttttttttttttt
data received:  uuuuuuuuuuuuuuuuuuuu
data received:  vvvvvvvvvvvvvvvvvvvv
data received:  wwwwwwwwwwwwwwwwwwww
data received:  xxxxxxxxxxxxxxxxxxxx
data received:  yyyyyyyyyyyyyyyyyyyy
data received:  zzzzzzzzzzzzzzzzzzzz
data received:  aaaaaaaaaaaaaaaaaaaa
data received:  bbbbbbbbbbbbbbbbbbbb
data received:  cccccccccccccccccccc
data received:  dddddddddddddddddddd
simulation complete
==========STATISTICS==========
 Total Number of Messages Sent                 ->  60
 Total Number of Retransmissions               ->  0
   Total Number of Retransmitted Data Packets  ->  0
   Total Number of Retransmitted ACKs          ->  0
 Total Number of Lost Packets                  ->  0
   Total Number of Lost Data Packets           ->  0
   Total Number of Lost ACKs                   ->  0
 Total Number of Dropped Packets               ->  0
   Total Number of Dropped Data Packets        ->  0
   Total Number of Dropped ACKs                ->  0
 Total Number of Corrupted Packets             ->  0
   Total Number of Corrupted Data Packets      ->  0
   Total Number of Corrupted ACKs              ->  0
 Final Simulation Time                         ->  3008.057050966907
==========STATISTICS==========
```

  - Justification : Everything here remains the same except for lambda which is decreased by a factor of 10. This is reflected in the final simulation time being decreased by a factor of 10 which makes sense. The time between transmitting messages has been reduced and so the total time has been reduced by the same factor.

- Test Case 3 - nsimmax = 30, lossprob = 0.2, corruptprob = 0, Lambda = 1000 :

```
data received:  aaaaaaaaaaaaaaaaaaaa
data received:  bbbbbbbbbbbbbbbbbbbb
data received:  cccccccccccccccccccc
data received:  dddddddddddddddddddd
data received:  eeeeeeeeeeeeeeeeeeee
data received:  ffffffffffffffffffff
data received:  gggggggggggggggggggg
data received:  hhhhhhhhhhhhhhhhhhhh
data received:  iiiiiiiiiiiiiiiiiiii
data received:  jjjjjjjjjjjjjjjjjjjj
data received:  kkkkkkkkkkkkkkkkkkkk
data received:  llllllllllllllllllll
data received:  mmmmmmmmmmmmmmmmmmmm
data received:  nnnnnnnnnnnnnnnnnnnn
data received:  oooooooooooooooooooo
data received:  pppppppppppppppppppp
data received:  qqqqqqqqqqqqqqqqqqqq
data received:  rrrrrrrrrrrrrrrrrrrr
data received:  ssssssssssssssssssss
data received:  tttttttttttttttttttt
data received:  uuuuuuuuuuuuuuuuuuuu
data received:  vvvvvvvvvvvvvvvvvvvv
data received:  wwwwwwwwwwwwwwwwwwww
data received:  xxxxxxxxxxxxxxxxxxxx
data received:  yyyyyyyyyyyyyyyyyyyy
data received:  zzzzzzzzzzzzzzzzzzzz
data received:  aaaaaaaaaaaaaaaaaaaa
data received:  bbbbbbbbbbbbbbbbbbbb
data received:  cccccccccccccccccccc
data received:  dddddddddddddddddddd
simulation complete
==========STATISTICS==========
 Total Number of Messages Sent                 ->  83
 Total Number of Retransmissions               ->  23
   Total Number of Retransmitted Data Packets  ->  15
   Total Number of Retransmitted ACKs          ->  8
 Total Number of Lost Packets                  ->  15
   Total Number of Lost Data Packets           ->  7
   Total Number of Lost ACKs                   ->  8
 Total Number of Dropped Packets               ->  8
   Total Number of Dropped Data Packets        ->  8
   Total Number of Dropped ACKs                ->  0
 Total Number of Corrupted Packets             ->  0
   Total Number of Corrupted Data Packets      ->  0
   Total Number of Corrupted ACKs              ->  0
 Final Simulation Time                         ->  30014.364788564646
==========STATISTICS==========
```

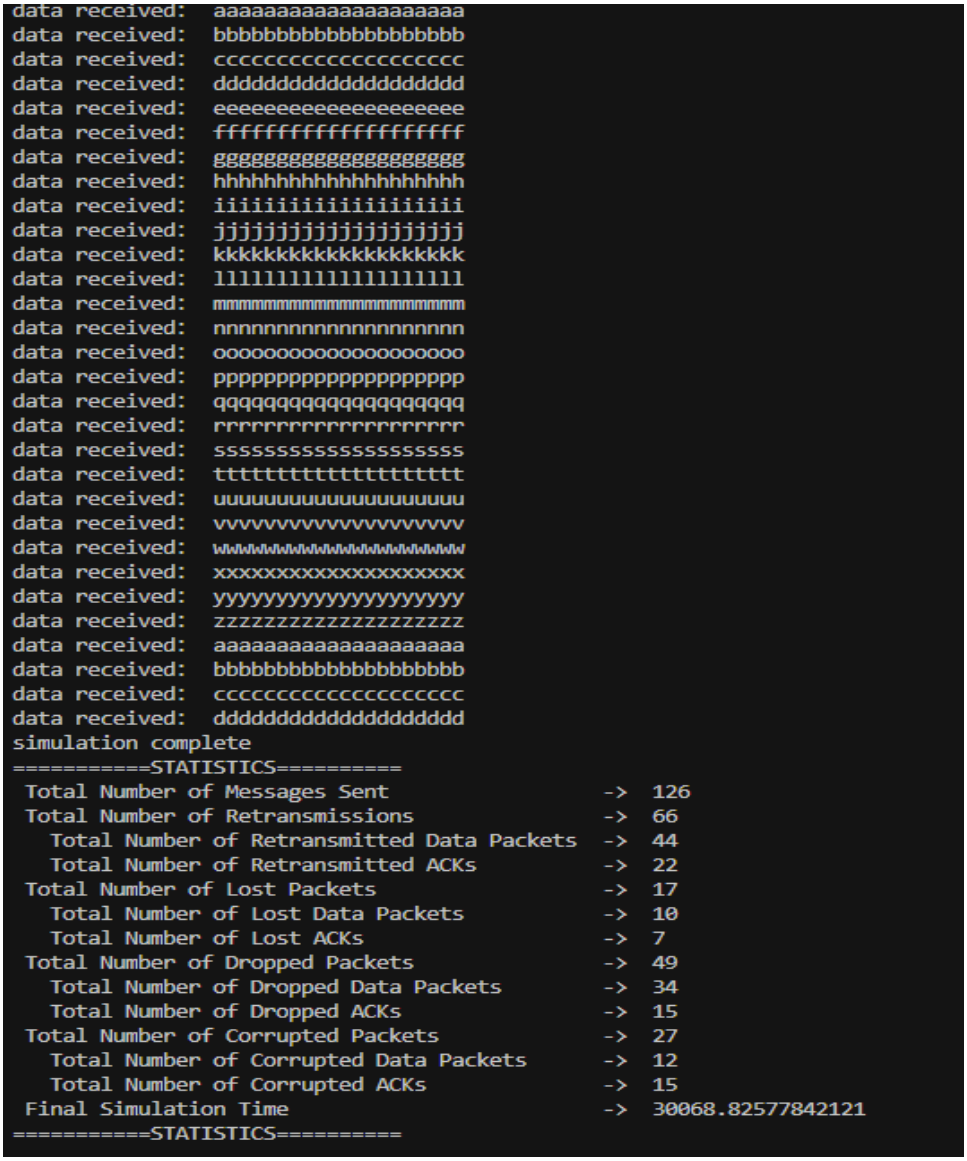
  - Justification : This is the first case where we start to see loss of packets. The first thing to notice is that the number of packets lost is 15 which is close to 12, the expected number (20% of 60). Since there is now retransmission of data, we also start to see data being dropped for being received out of sequence, late acks etc. Also notice that 83-23 = 60 which is the expected number of packets without retransmission.

- Test Case 4 - nsimmax = 30, lossprob = 0, corruptprob = 0.3, Lambda = 1000 :

```
data received:  aaaaaaaaaaaaaaaaaaaa
data received:  bbbbbbbbbbbbbbbbbbbb
data received:  cccccccccccccccccccc
data received:  dddddddddddddddddddd
data received:  eeeeeeeeeeeeeeeeeeee
data received:  ffffffffffffffffffff
data received:  gggggggggggggggggggg
data received:  hhhhhhhhhhhhhhhhhhhh
data received:  iiiiiiiiiiiiiiiiiiii
data received:  jjjjjjjjjjjjjjjjjjjj
data received:  kkkkkkkkkkkkkkkkkkkk
data received:  llllllllllllllllllll
data received:  mmmmmmmmmmmmmmmmmmmm
data received:  nnnnnnnnnnnnnnnnnnnn
data received:  oooooooooooooooooooo
data received:  pppppppppppppppppppp
data received:  qqqqqqqqqqqqqqqqqqqq
data received:  rrrrrrrrrrrrrrrrrrrr
data received:  ssssssssssssssssssss
data received:  tttttttttttttttttttt
data received:  uuuuuuuuuuuuuuuuuuuu
data received:  vvvvvvvvvvvvvvvvvvvv
data received:  wwwwwwwwwwwwwwwwwwww
data received:  xxxxxxxxxxxxxxxxxxxx
data received:  yyyyyyyyyyyyyyyyyyyy
data received:  zzzzzzzzzzzzzzzzzzzz
data received:  aaaaaaaaaaaaaaaaaaaa
data received:  bbbbbbbbbbbbbbbbbbbb
data received:  cccccccccccccccccccc
data received:  dddddddddddddddddddd
simulation complete
==========STATISTICS==========
 Total Number of Messages Sent                 ->  87
 Total Number of Retransmissions               ->  27
   Total Number of Retransmitted Data Packets  ->  18
   Total Number of Retransmitted ACKs          ->  9
 Total Number of Lost Packets                  ->  0
   Total Number of Lost Data Packets           ->  0
   Total Number of Lost ACKs                   ->  0
 Total Number of Dropped Packets               ->  27
   Total Number of Dropped Data Packets        ->  18
   Total Number of Dropped ACKs                ->  9
 Total Number of Corrupted Packets             ->  18
   Total Number of Corrupted Data Packets      ->  9
   Total Number of Corrupted ACKs              ->  9
 Final Simulation Time                         ->  30011.872619951377
==========STATISTICS==========
```

  - Justification : We now have a similar situation as test case 3, with 0 loss probability but a 0.3 corruption probability. We see that every corrupted packet is counted as dropped, as per the announcement. Additionally, for each corrupted packet we have a corrupted ack, both of which are dropped and counted as such. Also, 87-27=60 which is expected.

- Test Case 5 - nsimmax = 30, lossprob = 0.2, corruptprob = 0.3, Lambda = 1000 :



  - Justification : The first thing to notice here is that the total number of packets sent has shot up drastically. This can be explained by the introduction of both corruption and loss probabilities (126-66=60, which still checks out). Notice that the number of corrupted data packets, and the number of lost data packets remains close to the required probabilities. The total number of retransmissions is the addition of corrupted plus lost, which are higher numbers since they account for lost/corrupted acks as well. The time seems to remain roughly the same, but I expect this will change as the number of packets to send scales upward.

- Test Case 6 - nsimmax = 30, lossprob = 0.8, corruptprob = 0.8, Lambda = 1000 :

```
waiting for ack, new message dropped: bbbbbbbbbbbbbbbbbbb
data received:  aaaaaaaaaaaaaaaaaaa
waiting for ack, new message dropped: ccccccccccccccccccc
waiting for ack, new message dropped: ddddddddddddddddddddd
waiting for ack, new message dropped: eeeeeeeeeeeeeeeeeeee
waiting for ack, new message dropped: fffffffffffffffffff
waiting for ack, new message dropped: ggggggggggggggggggg
waiting for ack, new message dropped: hhhhhhhhhhhhhhhhhhh
waiting for ack, new message dropped: iiiiiiiiiiiiiiiiiii
waiting for ack, new message dropped: jjjjjjjjjjjjjjjjjjj
waiting for ack, new message dropped: kkkkkkkkkkkkkkkkkkk
data received:  lllllllllllllllllll
waiting for ack, new message dropped: mmmmmmmmmmmmmmmmmmm
waiting for ack, new message dropped: nnnnnnnnnnnnnnnnnnn
data received:  ooooooooooooooooooo
waiting for ack, new message dropped: ppppppppppppppppppp
waiting for ack, new message dropped: rrrrrrrrrrrrrrrrrrr
waiting for ack, new message dropped: sssssssssssssssssss
data received:  qqqqqqqqqqqqqqqqqqq
waiting for ack, new message dropped: ttttttttttttttttttt
waiting for ack, new message dropped: uuuuuuuuuuuuuuuuuuu
data received:  vvvvvvvvvvvvvvvvvvv
waiting for ack, new message dropped: wwwwwwwwwwwwwwwwwww
waiting for ack, new message dropped: xxxxxxxxxxxxxxxxxxx
waiting for ack, new message dropped: yyyyyyyyyyyyyyyyyyy
waiting for ack, new message dropped: zzzzzzzzzzzzzzzzzzz
waiting for ack, new message dropped: aaaaaaaaaaaaaaaaaaa
waiting for ack, new message dropped: bbbbbbbbbbbbbbbbbbb
waiting for ack, new message dropped: ccccccccccccccccccc
waiting for ack, new message dropped: ddddddddddddddddddddd
simulation complete
==========STATISTICS==========
 Total Number of Messages Sent                 ->  1196
 Total Number of Retransmissions               ->  1186
   Total Number of Retransmitted Data Packets  ->  1029
   Total Number of Retransmitted ACKs          ->  157
 Total Number of Lost Packets                  ->  968
   Total Number of Lost Data Packets           ->  839
   Total Number of Lost ACKs                   ->  129
 Total Number of Dropped Packets               ->  243
   Total Number of Dropped Data Packets        ->  215
   Total Number of Dropped ACKs                ->  28
 Total Number of Corrupted Packets             ->  61
   Total Number of Corrupted Data Packets      ->  33
   Total Number of Corrupted ACKs              ->  28
 Final Simulation Time                         ->  34615.4308402123
==========STATISTICS==========
```

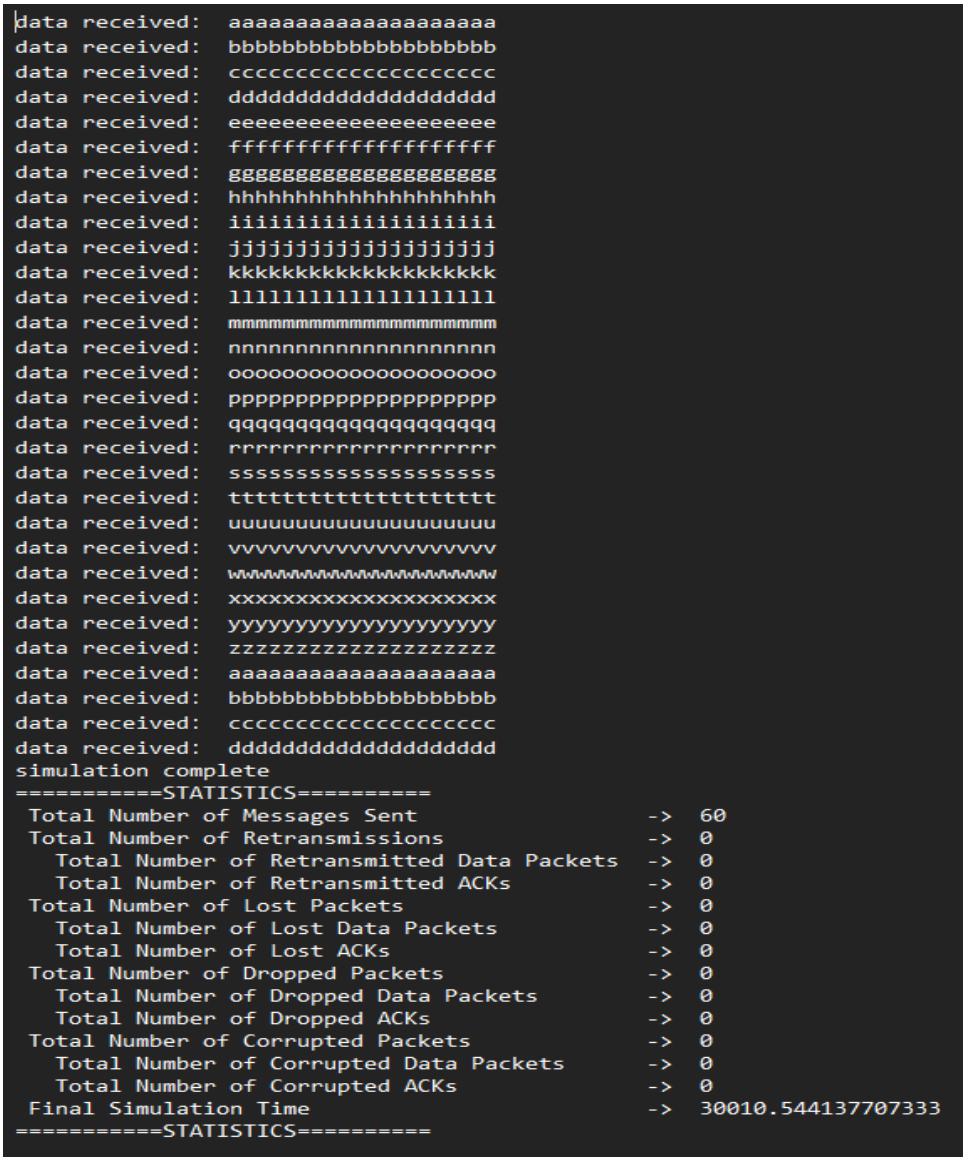
  - Justification : We now start to see some interesting things. For one, we seem to be dropping packets, out of order as can be seen in the output. This can be explained by the high loss/corruption probabilities. It is likely that the ack for the initially sent packet containing ‘a’ was lost en route, and so a retransmit started. In the meantime, the sender received b but since the sender is not in the appropriate state, it was forced to drop the packet. Similar scenarios occur throughout the simulation. This also contributes to the total number of messages sent, and the high retransmission rate. Some quick math shows that. One more thing to note here is that 1196-1186 = 10, which tracks since only 5 packets(and so 5 acks) were received, and the rest were dropped.

- Test Case 7 - nsimmax = 30, lossprob = 0.8, corruptprob = 0.8, Lambda = 100000 :

```
data received:  aaaaaaaaaaaaaaaaaaaa
data received:  bbbbbbbbbbbbbbbbbbbb
data received:  cccccccccccccccccccc
data received:  dddddddddddddddddddd
data received:  eeeeeeeeeeeeeeeeeeee
data received:  ffffffffffffffffffff
data received:  gggggggggggggggggggg
data received:  hhhhhhhhhhhhhhhhhhhh
data received:  iiiiiiiiiiiiiiiiiiii
data received:  jjjjjjjjjjjjjjjjjjjj
data received:  kkkkkkkkkkkkkkkkkkkk
data received:  llllllllllllllllllll
data received:  mmmmmmmmmmmmmmmmmmmm
data received:  nnnnnnnnnnnnnnnnnnnn
data received:  oooooooooooooooooooo
data received:  pppppppppppppppppppp
data received:  qqqqqqqqqqqqqqqqqqqq
data received:  rrrrrrrrrrrrrrrrrrrr
data received:  ssssssssssssssssssss
data received:  tttttttttttttttttttt
data received:  uuuuuuuuuuuuuuuuuuuu
data received:  vvvvvvvvvvvvvvvvvvvv
data received:  wwwwwwwwwwwwwwwwwwww
data received:  xxxxxxxxxxxxxxxxxxxx
data received:  yyyyyyyyyyyyyyyyyyyy
data received:  zzzzzzzzzzzzzzzzzzzz
data received:  aaaaaaaaaaaaaaaaaaaa
data received:  bbbbbbbbbbbbbbbbbbbb
data received:  cccccccccccccccccccc
data received:  dddddddddddddddddddd
simulation complete
==========STATISTICS==========
 Total Number of Messages Sent                 ->  5170
 Total Number of Retransmissions               ->  5110
   Total Number of Retransmitted Data Packets  ->  4381
   Total Number of Retransmitted ACKs          ->  729
 Total Number of Lost Packets                  ->  4122
   Total Number of Lost Data Packets           ->  3531
   Total Number of Lost ACKs                   ->  591
 Total Number of Dropped Packets               ->  988
   Total Number of Dropped Data Packets        ->  850
   Total Number of Dropped ACKs                ->  138
 Total Number of Corrupted Packets             ->  259
   Total Number of Corrupted Data Packets      ->  121
   Total Number of Corrupted ACKs              ->  138
 Final Simulation Time                         ->  3006699.4074638668
==========STATISTICS==========
```

  - Justification : This test overcomes the packet loss from the previous case, at the cost of larger run time. The primary issue with the last case was that the time between getting packets from the application was low, with large loss rates, causing packets to be lost. By increasing lambda, we ensure that each packet has time to receive an ack, and that the sender can be reset to its “WAIT_LAYER5” state, so all packets go through to the receiver, instead of being dropped at the sender. This however, does result in a larger runtime by a factor 100, which is what the lambda is increased by.

- Evaluation Phase - 2:
  - Note: The output wasn't fitting on my screen so I catted it to a separate file and am reporting it from there using python3 mainGBN.py >> temp.txt
  - Test Case 1 - nsimmax = 30, lossprob = 0, corruptprob = 0, Lambda = 1000 :



```
data received:  aaaaaaaaaaaaaaaaaaaa
data received:  bbbbbbbbbbbbbbbbbbbb
data received:  cccccccccccccccccccc
data received:  dddddddddddddddddddd
data received:  eeeeeeeeeeeeeeeeeeee
data received:  ffffffffffffffffffff
data received:  gggggggggggggggggggg
data received:  hhhhhhhhhhhhhhhhhhhh
data received:  iiiiiiiiiiiiiiiiiiii
data received:  jjjjjjjjjjjjjjjjjjjj
data received:  kkkkkkkkkkkkkkkkkkkk
data received:  llllllllllllllllllll
data received:  mmmmmmmmmmmmmmmmmmmm
data received:  nnnnnnnnnnnnnnnnnnnn
data received:  oooooooooooooooooooo
data received:  pppppppppppppppppppp
data received:  qqqqqqqqqqqqqqqqqqqq
data received:  rrrrrrrrrrrrrrrrrrrr
data received:  ssssssssssssssssssss
data received:  tttttttttttttttttttt
data received:  uuuuuuuuuuuuuuuuuuuu
data received:  vvvvvvvvvvvvvvvvvvvv
data received:  wwwwwwwwwwwwwwwwwwww
data received:  xxxxxxxxxxxxxxxxxxxx
data received:  yyyyyyyyyyyyyyyyyyyy
data received:  zzzzzzzzzzzzzzzzzzzz
data received:  aaaaaaaaaaaaaaaaaaaa
data received:  bbbbbbbbbbbbbbbbbbbb
data received:  cccccccccccccccccccc
data received:  dddddddddddddddddddd
simulation complete
==========STATISTICS==========
 Total Number of Messages Sent                 ->  60
 Total Number of Retransmissions               ->  0
   Total Number of Retransmitted Data Packets  ->  0
   Total Number of Retransmitted ACKs          ->  0
 Total Number of Lost Packets                  ->  0
   Total Number of Lost Data Packets           ->  0
   Total Number of Lost ACKs                   ->  0
 Total Number of Dropped Packets               ->  0
   Total Number of Dropped Data Packets        ->  0
   Total Number of Dropped ACKs                ->  0
 Total Number of Corrupted Packets             ->  0
   Total Number of Corrupted Data Packets      ->  0
   Total Number of Corrupted ACKs              ->  0
 Final Simulation Time                         ->  30010.544137707333
==========STATISTICS==========
```

    - Justification : Again, similar to the SNW example, this outcome is relatively simple to explain. Since there is no loss or corruption, no packets are required to be retransmitted so 60 total messages are sent, 30 "messages" and 30 acks. The simulation time can be taken as a baseline for future cases.

- Test Case 2 - nsimmax = 30, lossprob = 0, corruptprob = 0, Lambda = 100 :

```
data received:  aaaaaaaaaaaaaaaaaaaa
data received:  bbbbbbbbbbbbbbbbbbbb
data received:  cccccccccccccccccccc
data received:  dddddddddddddddddddd
data received:  eeeeeeeeeeeeeeeeeeee
data received:  ffffffffffffffffffff
data received:  gggggggggggggggggggg
data received:  hhhhhhhhhhhhhhhhhhhh
data received:  iiiiiiiiiiiiiiiiiiii
data received:  jjjjjjjjjjjjjjjjjjjj
data received:  kkkkkkkkkkkkkkkkkkkk
data received:  llllllllllllllllllll
data received:  mmmmmmmmmmmmmmmmmmmm
data received:  nnnnnnnnnnnnnnnnnnnn
data received:  oooooooooooooooooooo
data received:  pppppppppppppppppppp
data received:  qqqqqqqqqqqqqqqqqqqq
data received:  rrrrrrrrrrrrrrrrrrrr
data received:  ssssssssssssssssssss
data received:  tttttttttttttttttttt
data received:  uuuuuuuuuuuuuuuuuuuu
data received:  vvvvvvvvvvvvvvvvvvvv
data received:  wwwwwwwwwwwwwwwwwwww
data received:  xxxxxxxxxxxxxxxxxxxx
data received:  yyyyyyyyyyyyyyyyyyyy
data received:  zzzzzzzzzzzzzzzzzzzz
data received:  aaaaaaaaaaaaaaaaaaaa
data received:  bbbbbbbbbbbbbbbbbbbb
data received:  cccccccccccccccccccc
data received:  dddddddddddddddddddd
simulation complete
==========STATISTICS==========
 Total Number of Messages Sent                 ->  60
 Total Number of Retransmissions               ->  0
   Total Number of Retransmitted Data Packets  ->  0
   Total Number of Retransmitted ACKs          ->  0
 Total Number of Lost Packets                  ->  0
   Total Number of Lost Data Packets           ->  0
   Total Number of Lost ACKs                   ->  0
 Total Number of Dropped Packets               ->  0
   Total Number of Dropped Data Packets        ->  0
   Total Number of Dropped ACKs                ->  0
 Total Number of Corrupted Packets             ->  0
   Total Number of Corrupted Data Packets      ->  0
   Total Number of Corrupted ACKs              ->  0
 Final Simulation Time                         ->  3013.936103022875
==========STATISTICS==========
```

  - Justification : Again, similar to SNW, everything here remains the same except for lambda which is decreased by a factor of 10. This is reflected in the final simulation time being decreased by a factor of 10 which makes sense. The time between transmitting messages has been reduced and so the total time has been reduced by the same factor.

- Test Case 3 - nsimmax = 30, lossprob = 0.2, corruptprob = 0, Lambda = 1000 :

```
data received:  aaaaaaaaaaaaaaaaaaaa
data received:  bbbbbbbbbbbbbbbbbbbb
data received:  cccccccccccccccccccc
data received:  dddddddddddddddddddd
data received:  eeeeeeeeeeeeeeeeeeee
data received:  ffffffffffffffffffff
data received:  gggggggggggggggggggg
data received:  hhhhhhhhhhhhhhhhhhhh
data received:  iiiiiiiiiiiiiiiiiiii
data received:  jjjjjjjjjjjjjjjjjjjj
data received:  kkkkkkkkkkkkkkkkkkkk
data received:  llllllllllllllllllll
data received:  mmmmmmmmmmmmmmmmmmmm
data received:  nnnnnnnnnnnnnnnnnnnn
data received:  oooooooooooooooooooo
data received:  pppppppppppppppppppp
data received:  qqqqqqqqqqqqqqqqqqqq
data received:  rrrrrrrrrrrrrrrrrrrr
data received:  ssssssssssssssssssss
data received:  tttttttttttttttttttt
data received:  uuuuuuuuuuuuuuuuuuuu
data received:  vvvvvvvvvvvvvvvvvvvv
data received:  wwwwwwwwwwwwwwwwwwww
data received:  xxxxxxxxxxxxxxxxxxxx
data received:  yyyyyyyyyyyyyyyyyyyy
data received:  zzzzzzzzzzzzzzzzzzzz
data received:  aaaaaaaaaaaaaaaaaaaa
data received:  bbbbbbbbbbbbbbbbbbbb
data received:  cccccccccccccccccccc
data received:  dddddddddddddddddddd
simulation complete
===========STATISTICS==========
 Total Number of Messages Sent                 ->  75
 Total Number of Retransmissions               ->  15
   Total Number of Retransmitted Data Packets  ->  11
   Total Number of Retransmitted ACKs          ->  4
 Total Number of Lost Packets                  ->  11
   Total Number of Lost Data Packets           ->  7
   Total Number of Lost ACKs                   ->  4
 Total Number of Dropped Packets               ->  4
   Total Number of Dropped Data Packets        ->  4
   Total Number of Dropped ACKs                ->  0
 Total Number of Corrupted Packets             ->  0
   Total Number of Corrupted Data Packets      ->  0
   Total Number of Corrupted ACKs              ->  0
 Final Simulation Time                         ->  30007.97533104624
===========STATISTICS==========
```

  - Justification : This is the first case where we start to see loss of packets. The first thing to notice is that the number of packets lost is 11 which is very close to 12, the expected number (20% of 60). Since there is now retransmission of data, we also start to see data being dropped for being received out of sequence, late acks etc. Also notice that 75-15 = 60 which is the expected number of packets without retransmission.

- Test Case 4 - nsimmax = 30, lossprob = 0, corruptprob = 0.3, Lambda = 1000 :

```
data received:  aaaaaaaaaaaaaaaaaaaa
data received:  bbbbbbbbbbbbbbbbbbbb
data received:  cccccccccccccccccccc
data received:  dddddddddddddddddddd
data received:  eeeeeeeeeeeeeeeeeeee
data received:  ffffffffffffffffffff
data received:  gggggggggggggggggggg
data received:  hhhhhhhhhhhhhhhhhhhh
data received:  iiiiiiiiiiiiiiiiiiii
data received:  jjjjjjjjjjjjjjjjjjjj
data received:  kkkkkkkkkkkkkkkkkkkk
data received:  llllllllllllllllllll
data received:  mmmmmmmmmmmmmmmmmmmm
data received:  nnnnnnnnnnnnnnnnnnnn
data received:  oooooooooooooooooooo
data received:  pppppppppppppppppppp
data received:  qqqqqqqqqqqqqqqqqqqq
data received:  rrrrrrrrrrrrrrrrrrrr
data received:  ssssssssssssssssssss
data received:  tttttttttttttttttttt
data received:  uuuuuuuuuuuuuuuuuuuu
data received:  vvvvvvvvvvvvvvvvvvvv
data received:  wwwwwwwwwwwwwwwwwwww
data received:  xxxxxxxxxxxxxxxxxxxx
data received:  yyyyyyyyyyyyyyyyyyyy
data received:  zzzzzzzzzzzzzzzzzzzz
data received:  aaaaaaaaaaaaaaaaaaaa
data received:  bbbbbbbbbbbbbbbbbbbb
data received:  cccccccccccccccccccc
data received:  dddddddddddddddddddd
simulation complete
==========STATISTICS==========
 Total Number of Messages Sent                 ->  101
 Total Number of Retransmissions               ->  41
   Total Number of Retransmitted Data Packets  ->  30
   Total Number of Retransmitted ACKs          ->  11
 Total Number of Lost Packets                  ->  0
   Total Number of Lost Data Packets           ->  0
   Total Number of Lost ACKs                   ->  0
 Total Number of Dropped Packets               ->  41
   Total Number of Dropped Data Packets        ->  30
   Total Number of Dropped ACKs                ->  11
 Total Number of Corrupted Packets             ->  30
   Total Number of Corrupted Data Packets      ->  19
   Total Number of Corrupted ACKs              ->  11
 Final Simulation Time                         ->  30216.976318131176
==========STATISTICS==========
```

  - Justification : We now have a similar situation as test case 3, with 0 loss probability but a 0.3 corruption probability. We see that every corrupted packet is counted as dropped, as per the announcement for the previous assignment. Additionally, for each corrupted packet we have a corrupted ack, both of which are dropped and counted as such. Also, 101-41=60 which is expected (There is a larger number of packets and acks going back and forth, especially due to retransmission without buffering implemented so the behavior is expected as compared to SNW).

- Test Case 5 - nsimmax = 30, lossprob = 0.2, corruptprob = 0.3, Lambda = 1000 :

```
data received:  aaaaaaaaaaaaaaaaaaaa
data received:  bbbbbbbbbbbbbbbbbbbb
data received:  cccccccccccccccccccc
data received:  dddddddddddddddddddd
data received:  eeeeeeeeeeeeeeeeeeee
data received:  ffffffffffffffffffff
data received:  gggggggggggggggggggg
data received:  hhhhhhhhhhhhhhhhhhhh
data received:  iiiiiiiiiiiiiiiiiiii
data received:  jjjjjjjjjjjjjjjjjjjj
data received:  kkkkkkkkkkkkkkkkkkkk
data received:  llllllllllllllllllll
data received:  mmmmmmmmmmmmmmmmmmmm
data received:  nnnnnnnnnnnnnnnnnnnn
data received:  oooooooooooooooooooo
data received:  pppppppppppppppppppp
data received:  qqqqqqqqqqqqqqqqqqqq
data received:  rrrrrrrrrrrrrrrrrrrr
data received:  ssssssssssssssssssss
data received:  tttttttttttttttttttt
data received:  uuuuuuuuuuuuuuuuuuuu
data received:  vvvvvvvvvvvvvvvvvvvv
data received:  wwwwwwwwwwwwwwwwwwww
data received:  xxxxxxxxxxxxxxxxxxxx
data received:  yyyyyyyyyyyyyyyyyyyy
data received:  zzzzzzzzzzzzzzzzzzzz
data received:  aaaaaaaaaaaaaaaaaaaa
data received:  bbbbbbbbbbbbbbbbbbbb
data received:  cccccccccccccccccccc
data received:  dddddddddddddddddddd
simulation complete
==========STATISTICS==========
 Total Number of Messages Sent                  ->  131
 Total Number of Retransmissions                ->  71
   Total Number of Retransmitted Data Packets   ->  51
   Total Number of Retransmitted ACKs           ->  20
 Total Number of Lost Packets                   ->  25
   Total Number of Lost Data Packets            ->  15
   Total Number of Lost ACKs                    ->  10
 Total Number of Dropped Packets                ->  46
   Total Number of Dropped Data Packets         ->  36
   Total Number of Dropped ACKs                 ->  10
 Total Number of Corrupted Packets              ->  26
   Total Number of Corrupted Data Packets       ->  16
   Total Number of Corrupted ACKs               ->  10
 Final Simulation Time                          ->  30132.398394927954
==========STATISTICS==========
```

  - Justification : This is where we see our first real deviation from behavior from SNW. The first thing to note is 131-71 = 60 which is the expected number of packets+acks to be sent. We also notice that the number of corrupted data packets and lost data packets (15 and 16 respectively) are reasonably close to their boundaries of 12 and 15. Beyond that the individual trackers all count as expected, with corrupted acks being double counted as corrupted and lost.

- Test Case 6 - nsimmax = 30, lossprob = 0.8, corruptprob = 0.8, Lambda = 1000 :

```
data received:  aaaaaaaaaaaaaaaaaaa
data received:  bbbbbbbbbbbbbbbbbbb
data received:  ccccccccccccccccccc
window full, new message is dropped: ddddddddddddddddddddd
window full, new message is dropped: eeeeeeeeeeeeeeeeeeee
window full, new message is dropped: fffffffffffffffffff
window full, new message is dropped: ggggggggggggggggggg
window full, new message is dropped: hhhhhhhhhhhhhhhhhhh
window full, new message is dropped: iiiiiiiiiiiiiiiiiii
window full, new message is dropped: jjjjjjjjjjjjjjjjjjj
window full, new message is dropped: kkkkkkkkkkkkkkkkkkk
window full, new message is dropped: lllllllllllllllllll
data received:  mmmmmmmmmmmmmmmmmmmm
data received:  nnnnnnnnnnnnnnnnnnn
data received:  ooooooooooooooooooo
window full, new message is dropped: ppppppppppppppppppp
window full, new message is dropped: qqqqqqqqqqqqqqqqqqqq
window full, new message is dropped: rrrrrrrrrrrrrrrrrrr
window full, new message is dropped: sssssssssssssssssss
window full, new message is dropped: ttttttttttttttttttt
window full, new message is dropped: uuuuuuuuuuuuuuuuuuu
window full, new message is dropped: vvvvvvvvvvvvvvvvvvv
window full, new message is dropped: wwwwwwwwwwwwwwwwwww
window full, new message is dropped: xxxxxxxxxxxxxxxxxxx
window full, new message is dropped: yyyyyyyyyyyyyyyyyyy
data received:  zzzzzzzzzzzzzzzzzzz
data received:  aaaaaaaaaaaaaaaaaaa
data received:  bbbbbbbbbbbbbbbbbbb
window full, new message is dropped: ccccccccccccccccccc
window full, new message is dropped: ddddddddddddddddddddd
simulation complete
==========STATISTICS==========
 Total Number of Messages Sent                -> 3733
 Total Number of Retransmissions              -> 3715
   Total Number of Retransmitted Data Packets -> 3576
   Total Number of Retransmitted ACKs         -> 139
 Total Number of Lost Packets                 -> 2955
   Total Number of Lost Data Packets          -> 2833
   Total Number of Lost ACKs                  -> 122
 Total Number of Dropped Packets              -> 787
   Total Number of Dropped Data Packets       -> 764
   Total Number of Dropped ACKs               -> 23
 Total Number of Corrupted Packets            -> 627
   Total Number of Corrupted Data Packets     -> 604
   Total Number of Corrupted ACKs             -> 23
 Final Simulation Time                        -> 40438.179139338405
==========STATISTICS==========
```

- Justification : We now start to see some interesting things. For one, we seem to be dropping packets, meaning that the loss probabilities are so high, the buffer cannot free itself up in time, leading to lost packets from the application layer. We also see a massive increase in the total number of messages transmitted. This is probably to do with the number of retransmits for acks, since there is significant corruption and loss. It's likely that after the first batch populated the buffer,(namely a, b, c since window size is 3), multiple retransmits occurred until finally space was made for m, n and o. One more thing to note here is that 3733-3715 = 18 , which tracks since only 9 packets(and so 9 acks) were received, and the rest were dropped.

- Test Case 7 - nsimmax = 30, lossprob = 0.8, corruptprob = 0.8, Lambda = 100000 :

```
data received:  aaaaaaaaaaaaaaaaaaaa
data received:  bbbbbbbbbbbbbbbbbbbb
data received:  cccccccccccccccccccc
data received:  dddddddddddddddddddd
data received:  eeeeeeeeeeeeeeeeeeee
data received:  ffffffffffffffffffff
data received:  gggggggggggggggggggg
data received:  hhhhhhhhhhhhhhhhhhhh
data received:  iiiiiiiiiiiiiiiiiiii
data received:  jjjjjjjjjjjjjjjjjjjj
data received:  kkkkkkkkkkkkkkkkkkkk
data received:  llllllllllllllllllll
data received:  mmmmmmmmmmmmmmmmmmmm
data received:  nnnnnnnnnnnnnnnnnnnn
data received:  oooooooooooooooooooo
data received:  pppppppppppppppppppp
data received:  qqqqqqqqqqqqqqqqqqqq
data received:  rrrrrrrrrrrrrrrrrrrr
data received:  ssssssssssssssssssss
data received:  tttttttttttttttttttt
data received:  uuuuuuuuuuuuuuuuuuuu
data received:  vvvvvvvvvvvvvvvvvvvv
data received:  wwwwwwwwwwwwwwwwwwww
data received:  xxxxxxxxxxxxxxxxxxxx
data received:  yyyyyyyyyyyyyyyyyyyy
data received:  zzzzzzzzzzzzzzzzzzzz
data received:  aaaaaaaaaaaaaaaaaaaa
data received:  bbbbbbbbbbbbbbbbbbbb
data received:  cccccccccccccccccccc
data received:  dddddddddddddddddddd
simulation complete
==========STATISTICS==========
 Total Number of Messages Sent                  ->  16664
 Total Number of Retransmissions                ->  16604
   Total Number of Retransmitted Data Packets   ->  15976
   Total Number of Retransmitted ACKs           ->  628
 Total Number of Lost Packets                   ->  13367
   Total Number of Lost Data Packets            ->  12834
   Total Number of Lost ACKs                    ->  533
 Total Number of Dropped Packets                ->  3237
   Total Number of Dropped Data Packets         ->  3142
   Total Number of Dropped ACKs                 ->  95
 Total Number of Corrupted Packets              ->  2609
   Total Number of Corrupted Data Packets       ->  2514
   Total Number of Corrupted ACKs               ->  95
 Final Simulation Time                          ->  3004240.7569898223
==========STATISTICS==========
```

  - Justification : This test overcomes the packet loss from the previous case, at the cost of significantly larger run time and also number of packets sent. The primary issue with the last case was that the time between getting packets from the application was low, with large loss rates, causing packets to be lost. By increasing lambda, we ensure that each packet has time to receive an ack and so the buffer can free its slots before receiving a message it is forced to drop because its window is full. We can see that all 30 packets(and all 30 acks) are received via 16664 - 16604 = 60.