a. Header Section
■ Shruti Jayavardhanan
■ 3/19/2024
■ Coding language - Python

b. Compilation Section
-Make sure Python is installed in the system.
-Navigate to the project directory using the command line or terminal
-Since I'm using Python, explicit compilation is not necessary, so can run the files directly without compilation.

c. Execution Section
- Enter "python mainSNW.py" in the command prompt or terminal.

d. Description Section
- The Stop-and-Wait protocol is a simple flow control mechanism used in data communication. In this implementation, the protocol is used to transfer data from a sender to a receiver in a computer network.
- Sender:
Sender (S_sender class):

The sender waits for messages from the application layer (layer 5) in the "WAIT_LAYER5" state. When a message is received from layer 5 (S_output function), the sender creates a packet with the current sequence number and sends it to the receiver via layer 3.After sending the packet, the sender transitions to the "WAIT_ACK" state and starts a timer for packet timeout. Upon receiving an ACK from the receiver (S_input function), the sender verifies the acknowledgment number and transitions back to the "WAIT_LAYER5" state.
If the timer expires before receiving an ACK (S_handle_timer function), the sender retransmits the packet and restarts the timer.

-Receiver:
The receiver waits for packets from layer 3.Upon receiving a packet from layer 3 (R_input function), the receiver verifies the checksum and the sequence number to ensure the packet's integrity and correctness.If the received packet is valid, the receiver passes the payload to layer 5 and sends an ACK to the sender with the acknowledgment number.
The receiver then updates its expected sequence number.If the received packet is corrupted, it is discarded, and appropriate counters are updated.

- Functions Implemented:
S_output(message): Handles the message received from layer 5. It constructs a packet with the message and sends it to layer 3.
S_input(received_packet): Processes the packet received from layer 3. It verifies the ACK number and handles it accordingly.

S_handle_timer(): Manages the expiration of the sender's timer. It retransmits the packet if no ACK is received within the timeout period.

R_input(received_packet): Deals with the packet received from layer 3 at the receiver's end. It verifies the packet's integrity, sends ACK if the packet is correct, and updates the sequence number.

Initialization functions: Initializes relevant class variables and states for both sender and receiver.

-Additional Data Structures/Fields/Methods:

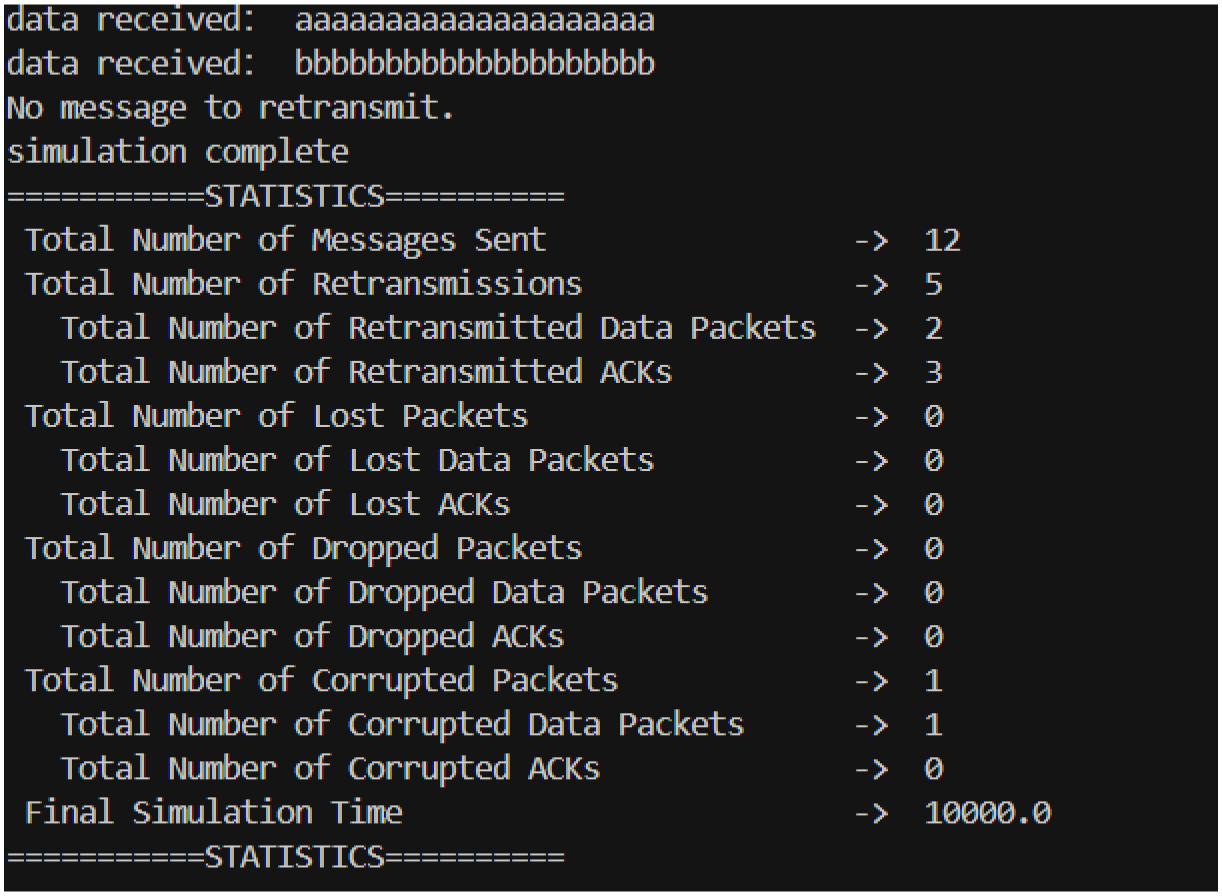
Message Buffer (self.message): Stores the message being sent by the sender, allowing for retransmission if needed.

e. Evaluation Section

- My project doesn't exactly produce the correct outputs.

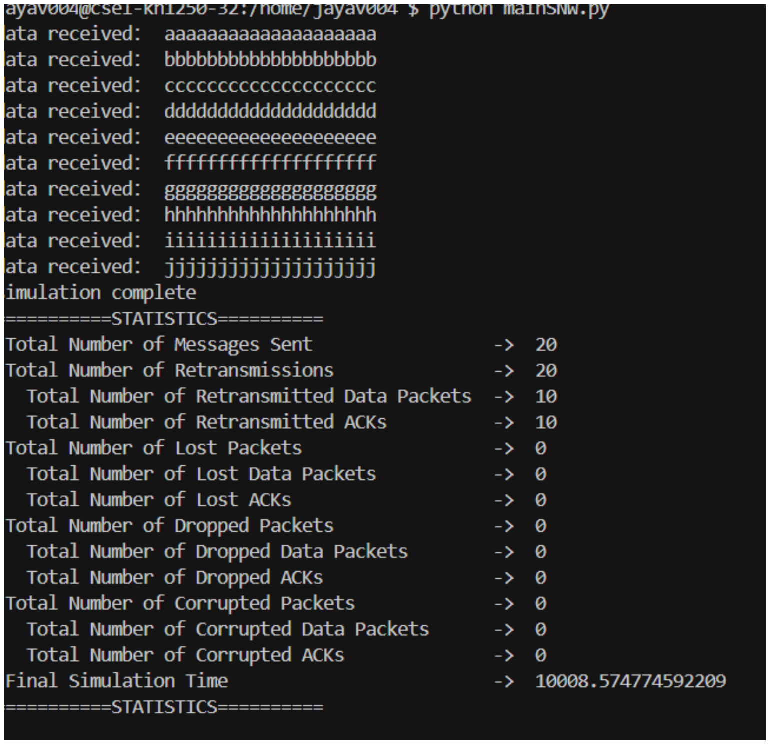
Here are the types of outputs I got on the lab machine.

1. LOSS = 0.1 CORRUPT = 0.2



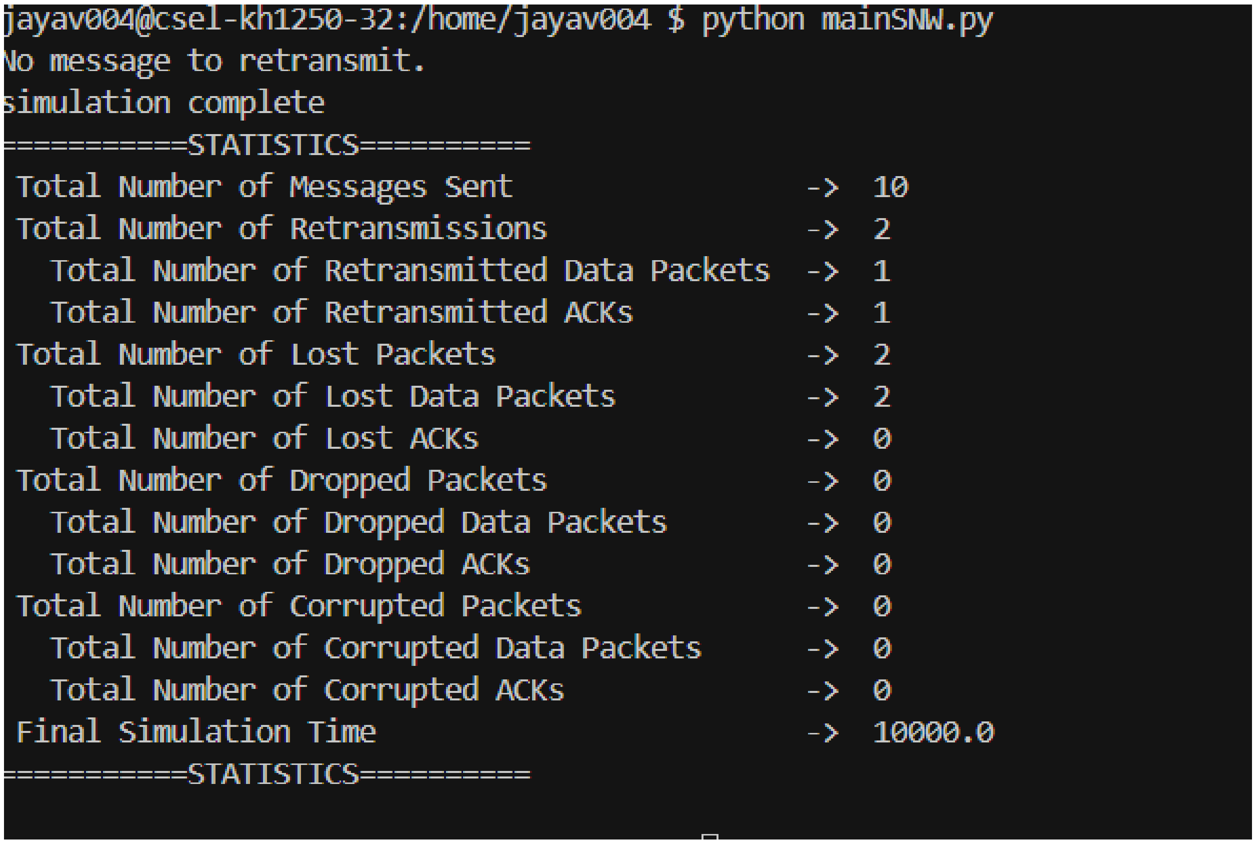
```
data received:  aaaaaaaaaaaaaaaaaaaa
data received:  bbbbbbbbbbbbbbbbbbbb
No message to retransmit.
simulation complete
===========STATISTICS==========
 Total Number of Messages Sent                  ->  12
 Total Number of Retransmissions                ->  5
   Total Number of Retransmitted Data Packets   ->  2
   Total Number of Retransmitted ACKs           ->  3
 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              ->  1
   Total Number of Corrupted Data Packets       ->  1
   Total Number of Corrupted ACKs               ->  0
 Final Simulation Time                          ->  10000.0
===========STATISTICS==========
```

2. 0.0 and 0.0



```
ayav004@csel-kh1250-32:/home/jayav004 $ python mainSNW.py
ata received:  aaaaaaaaaaaaaaaaaaaa
ata received:  bbbbbbbbbbbbbbbbbbbb
ata received:  cccccccccccccccccccc
ata received:  dddddddddddddddddddd
ata received:  eeeeeeeeeeeeeeeeeeee
ata received:  ffffffffffffffffffff
ata received:  gggggggggggggggggggg
ata received:  hhhhhhhhhhhhhhhhhhhh
ata received:  iiiiiiiiiiiiiiiiiiii
ata received:  jjjjjjjjjjjjjjjjjjjj
imulation complete
==========STATISTICS==========
Total Number of Messages Sent              ->  20
Total Number of Retransmissions            ->  20
  Total Number of Retransmitted Data Packets  ->  10
  Total Number of Retransmitted ACKs       ->  10
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                      ->  10008.574774592209
==========STATISTICS==========
```

3. 3 and 4



```
jayav004@csel-kh1250-32:/home/jayav004 $ python mainSNW.py
No message to retransmit.
simulation complete
===========STATISTICS==========
 Total Number of Messages Sent                  ->  10
 Total Number of Retransmissions                ->  2
   Total Number of Retransmitted Data Packets  ->  1
   Total Number of Retransmitted ACKs          ->  1
 Total Number of Lost Packets                   ->  2
   Total Number of Lost Data Packets           ->  2
   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                          ->  10000.0
===========STATISTICS==========
```

4. 0.9 and 0.9

```
No message to retransmit.
waiting for ack, new message is dropped: <msg.msg object at 0x7fe4f839bd30>
waiting for ack, new message is dropped: <msg.msg object at 0x7fe4f839bdc0>
waiting for ack, new message is dropped: <msg.msg object at 0x7fe4f839bd30>
waiting for ack, new message is dropped: <msg.msg object at 0x7fe4f839bdc0>
waiting for ack, new message is dropped: <msg.msg object at 0x7fe4f839bd30>
waiting for ack, new message is dropped: <msg.msg object at 0x7fe4f839bdc0>
waiting for ack, new message is dropped: <msg.msg object at 0x7fe4f839bd30>
waiting for ack, new message is dropped: <msg.msg object at 0x7fe4f839bdc0>
waiting for ack, new message is dropped: <msg.msg object at 0x7fe4f839bd30>
waiting for ack, new message is dropped: <msg.msg object at 0x7fe4f839bdc0>
waiting for ack, new message is dropped: <msg.msg object at 0x7fe4f839bd30>
waiting for ack, new message is dropped: <msg.msg object at 0x7fe4f839bdc0>
waiting for ack, new message is dropped: <msg.msg object at 0x7fe4f839bd30>
waiting for ack, new message is dropped: <msg.msg object at 0x7fe4f839bdc0>
waiting for ack, new message is dropped: <msg.msg object at 0x7fe4f839bd30>
waiting for ack, new message is dropped: <msg.msg object at 0x7fe4f839bdc0>
waiting for ack, new message is dropped: <msg.msg object at 0x7fe4f839bd30>
waiting for ack, new message is dropped: <msg.msg object at 0x7fe4f839bdc0>
waiting for ack, new message is dropped: <msg.msg object at 0x7fe4f839bd30>
waiting for ack, new message is dropped: <msg.msg object at 0x7fe4f839bdc0>
waiting for ack, new message is dropped: <msg.msg object at 0x7fe4f839bd30>
waiting for ack, new message is dropped: <msg.msg object at 0x7fe4f839bdc0>
waiting for ack, new message is dropped: <msg.msg object at 0x7fe4f839bd30>
waiting for ack, new message is dropped: <msg.msg object at 0x7fe4f839bdc0>
```