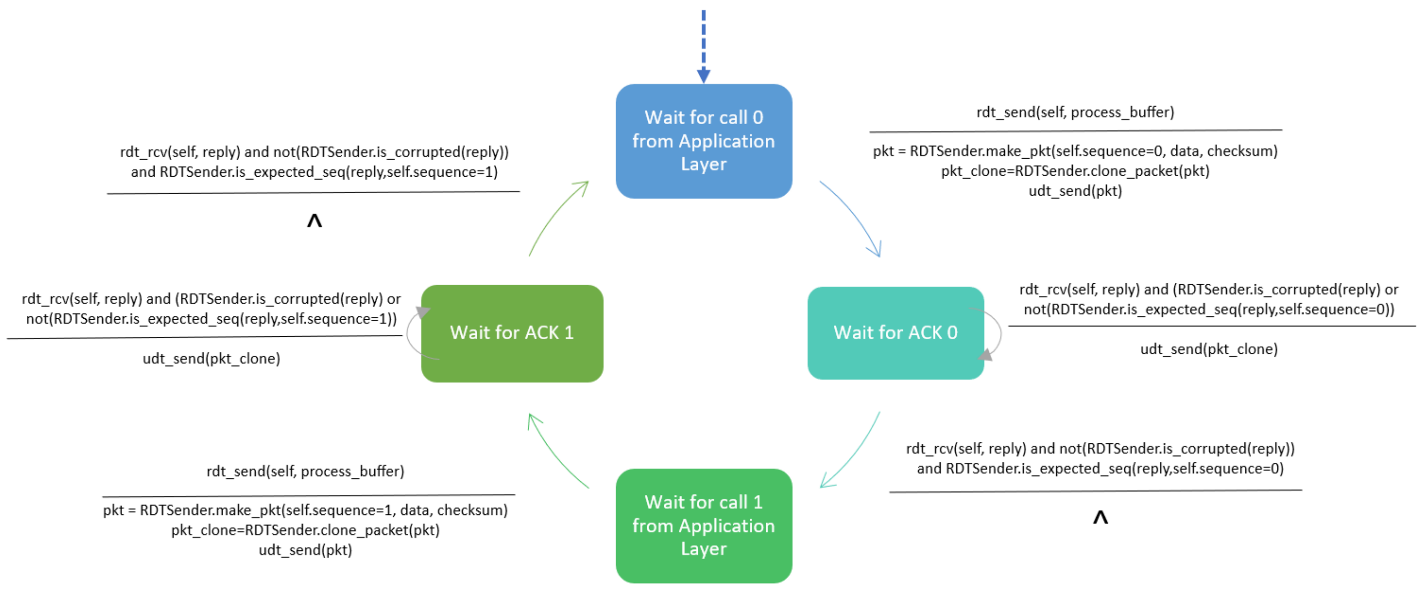
Networks Report
RDT V2.2 IMPLEMENTATION

Section 1
Team Members:
Hana Mohamed Seif | 55-25257 | T-21 | hana.seif@student.guc.edu.eg
Yasmine Elsadat | 55- 3780 | T-21 | yasmine.elsadat@student.guc.edu.eg
Malak hesham | 55-6656 | T-19 | malak.montasser@student.guc.edu.eg

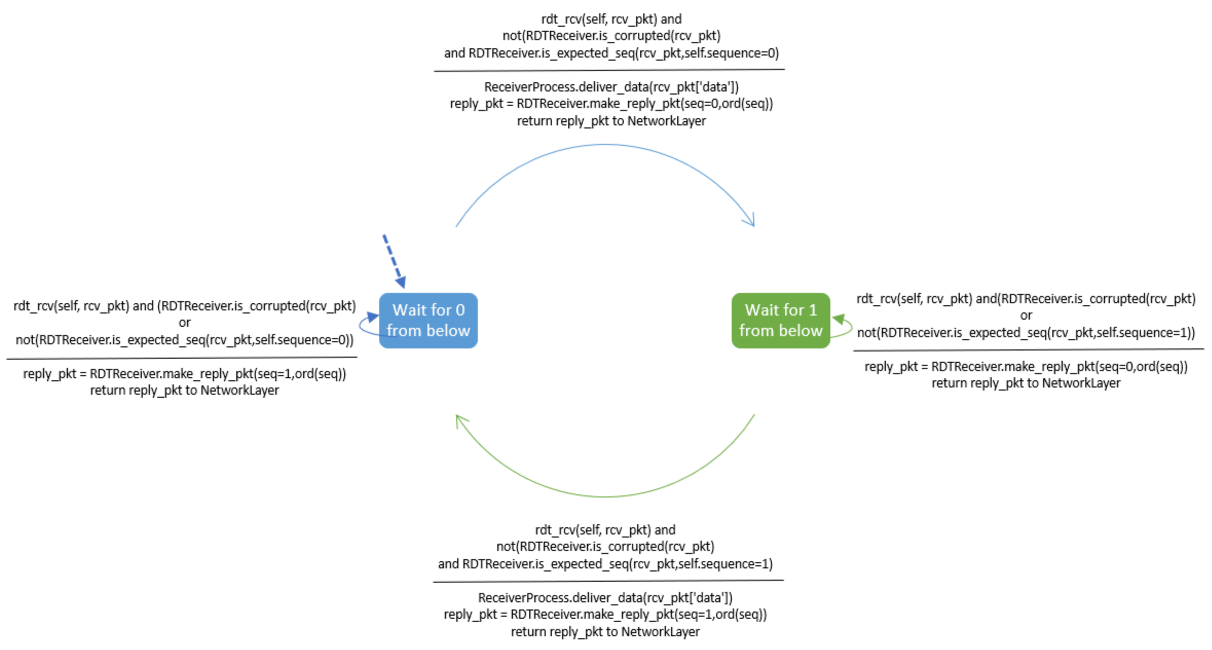
Contributions: We worked on the project as a team during several zoom meetings. We started by implementing the methods in the sender and receiver class and testing our code to make sure it mimics the RDT's behaviour with its required functionalities. We then formatted the print statements and made a color code to show the difference between the receiver and sender sides as well as corrupted packets.

Section 2

Sender FSM:


Wait for call 0 from Application Layer
rdt_send(self, process_buffer)
pkt = RDTSender.make_pkt(self.sequence=0, data, checksum)
pkt_clone=RDTSender.clone_packet(pkt)
udt_send(pkt)
Wait for ACK 0
rdt_rcv(self, reply) and (RDTSender.is_corrupted(reply) or not(RDTSender.is_expected_seq(reply,self.sequence=0))
udt_send(pkt_clone)
rdt_rcv(self, reply) and not(RDTSender.is_corrupted(reply)) and RDTSender.is_expected_seq(reply,self.sequence=0)
Λ
Wait for call 1 from Application Layer
rdt_send(self, process_buffer)
pkt = RDTSender.make_pkt(self.sequence=1, data, checksum)
pkt_clone=RDTSender.clone_packet(pkt)
udt_send(pkt)
Wait for ACK 1
rdt_rcv(self, reply) and (RDTSender.is_corrupted(reply) or not(RDTSender.is_expected_seq(reply,self.sequence=1))
udt_send(pkt_clone)
rdt_rcv(self, reply) and not(RDTSender.is_corrupted(reply)) and RDTSender.is_expected_seq(reply,self.sequence=1)
Λ

Receiver FSM:


rdt_rcv(self, rcv_pkt) and not(RDTReceiver.is_corrupted(rcv_pkt) and RDTReceiver.is_expected_seq(rcv_pkt,self.sequence=0)
ReceiverProcess.deliver_data(rcv_pkt['data'])
reply_pkt = RDTReceiver.make_reply_pkt(seq=0,ord(seq))
return reply_pkt to NetworkLayer
Wait for 0 from below
rdt_rcv(self, rcv_pkt) and (RDTReceiver.is_corrupted(rcv_pkt) or not(RDTReceiver.is_expected_seq(rcv_pkt,self.sequence=0))
reply_pkt = RDTReceiver.make_reply_pkt(seq=1,ord(seq))
return reply_pkt to NetworkLayer
Wait for 1 from below
rdt_rcv(self, rcv_pkt) and(RDTReceiver.is_corrupted(rcv_pkt) or not(RDTReceiver.is_expected_seq(rcv_pkt,self.sequence=1))
reply_pkt = RDTReceiver.make_reply_pkt(seq=0,ord(seq))
return reply_pkt to NetworkLayer
rdt_rcv(self, rcv_pkt) and not(RDTReceiver.is_corrupted(rcv_pkt) and RDTReceiver.is_expected_seq(rcv_pkt,self.sequence=1)
ReceiverProcess.deliver_data(rcv_pkt['data'])
reply_pkt = RDTReceiver.make_reply_pkt(seq=1,ord(seq))
return reply_pkt to NetworkLayer

Section 3: Pseudocode

Sender Class Pseudocode

```
FUNCTION get_checksum(data)
        checksum ⟵ ASCII_CODE_OF(data)
        RETURN checksum
END FUNCTION
-------------------------------------------------------------------------------------------

 FUNCTION is_corrupted(reply)
         IF checksum of reply = ASCII_CODE_OF(acknowledgement)   THEN
                          RETURN  FALSE
               ELSE
                         RETURN TRUE
         END IF
END FUNCTION
-------------------------------------------------------------------------------------------

FUNCTION is_expected_seq(reply, exp_seq)
        IF the sequence number of the reply packet(reply) = the expected sequence
number(exp_seq) THEN
                       RETURN TRUE
      ELSE
                       RETURN FALSE
       END IF
END FUNCTION
-------------------------------------------------------------------------------------------
FUNCTION rdt_send(process_buffer)
      FOR data in the process_buffer
            DECLARE variable checksum ⟵ ASCII_CODE_OF(data)
            DECLARE variable pkt ⟵ make packet consisting of current sequence number,
data, and checksum
           DECLARE variable pkt_clone ⟵ make a clone of the pkt arguments
           DECLARE variable reply ⟵ send packet and get reply packet from the Network
Layer
           WHILE reply is corrupted or doesn't have the expected sequence number of the
sender
                       reply⟵ new reply after retransmission of a copy of the clone packet

           ENDWHILE
           Toggle the sequence number of sender

      ENDFOR
 PRINT 'Sender Done!'
END FUNCTION
```

Receiver Class PseudoCode

```
FUNCTION is_corrupted(packet)
  IF checksum of packet =ASCII_CODE_OF(packet_data)   THEN
          RETURN FALSE
          ELSE return TRUE
  END IF
END FUNCTION
-------------------------------------------------------------------------------------------
FUNCTION is_expected_seq(rcv_pkt, exp_seq)
IF the sequence number of the received packet(rcv_pkt) = the expected sequence
number(exp_seq) THEN
   RETURN TRUE
ELSE
   RETURN FALSE
END IF
END FUNCTION
-------------------------------------------------------------------------------------------

FUNCTION rdt_rcv(rcv_pkt):
          DECLARE variable seq to be the sequence number of the packet to be sent as the
        reply

          IF received packet is not corrupted AND received packet has expected sequence
        number THEN
                    Deliver data of received packet to the application layer
                    seq⟵ receiver sequence number
                    Toggle the sequence number of receiver
          ELSE
                    seq⟵ toggle the value of receiver sequence number
          ENDIF
        RETURN the reply packet consisting of seq and ASCII_CODE_OF(seq)
END FUNCTION
```

SECTION 4: Changes in Skeleton Code

In the Sender Class:

1) We implemented the get_checksum method in the sender class that calculates the ASCII value of the data which is a single character using the ord(char) in python , This is done to be compared later to detect whether the data has been corrupted or not

2) We edited the is_corrupted method implementing it such that it returns true indicating that corruption happened when the checksum we get from the reply packet is not equal to the checksum we calculate using our implemented get_checksum method of the acknowledgement that we get from the reply packet. Otherwise it returns False indicating that no corruption happened.

3) We implemented the is_expected_seq method that compares the expected sequence number at the sender side with the sequence number in the reply packet (which is the acknowledgement coming back from the receiver that can only possibly be 0 or 1)
This is an indication to the fact that receiver received what he is waiting for correctly and handles duplicate packet issues.

4) We completed the implementation of the rdt_send including the code that handles the data corruption that might occur. This was done by first calculating checksum of data, making a packet using the data and getting a clone for each packet we are processing from the process_buffer using the clone_packet in case the packet gets corrupted to have a copy of it. It then proceeds to get the reply packet from the receiver and keeps retransmitting the same packet if the reply packet was corrupted or the receiver received corrupted data until the reply has the expected sequence number and the receiver has the correct data.

In the Receiver Class:

1)We implemented is_corrupted method which is similar to that of the sender class that checks if the checksum in the incoming packet is the same as the calculated checksum of the data using the ord function. This is done to check whether the data coming from the sender has been corrupted in the network layer or not.

2)We implemented the is_expected_seq method similar to the way we implemented it in the sender class where it makes sure that the sequence number we get from the received packet is equal to the sequence number the receiver is expecting.

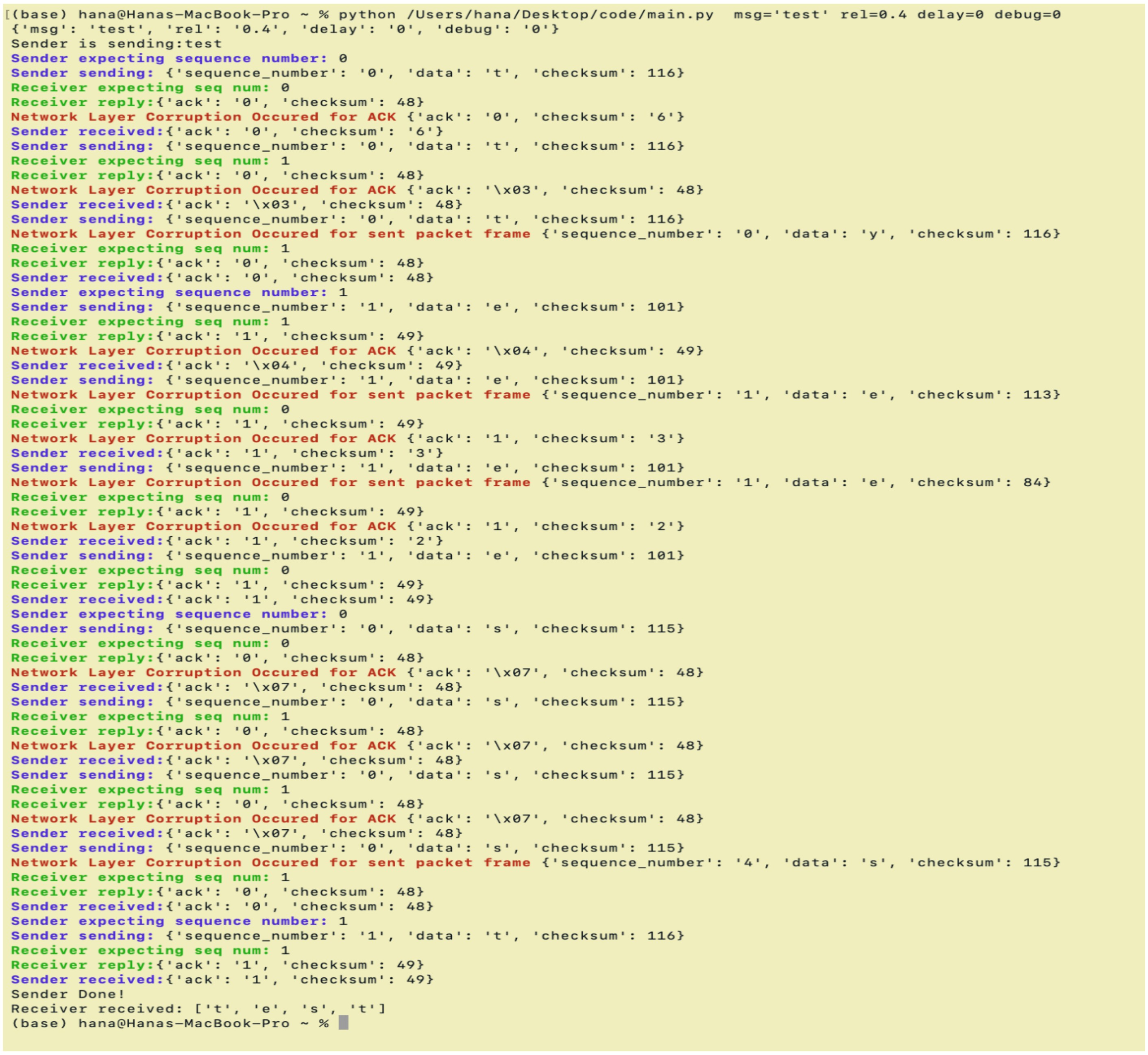
3)We implemented the rdt_rcv method where we check the received packet If it is not corrupted, then we deliver the data to the application layer,and the receiver replies with a positive acknowledgement, so that the sender will know on receiving this message that its packet has been successfully received and that it can start sending the next data in the buffer.
Otherwise if the packet is corrupted, the receiver replies with the opposite sequence number than what he/she/it is expecting, indicating a negative acknowledgement to the sender so that the sender will retransmit the packet.

We also added print statements in the Sender, Receiver, and Network layers to keep track of the messages' timeline (including what the sender is expecting as a seq num , what the sender is sending , what the receiver replied with and what the sender received,etc.) with the following color coding:

Green-> sender
Blue -> receiver
Red -> network layer corruption

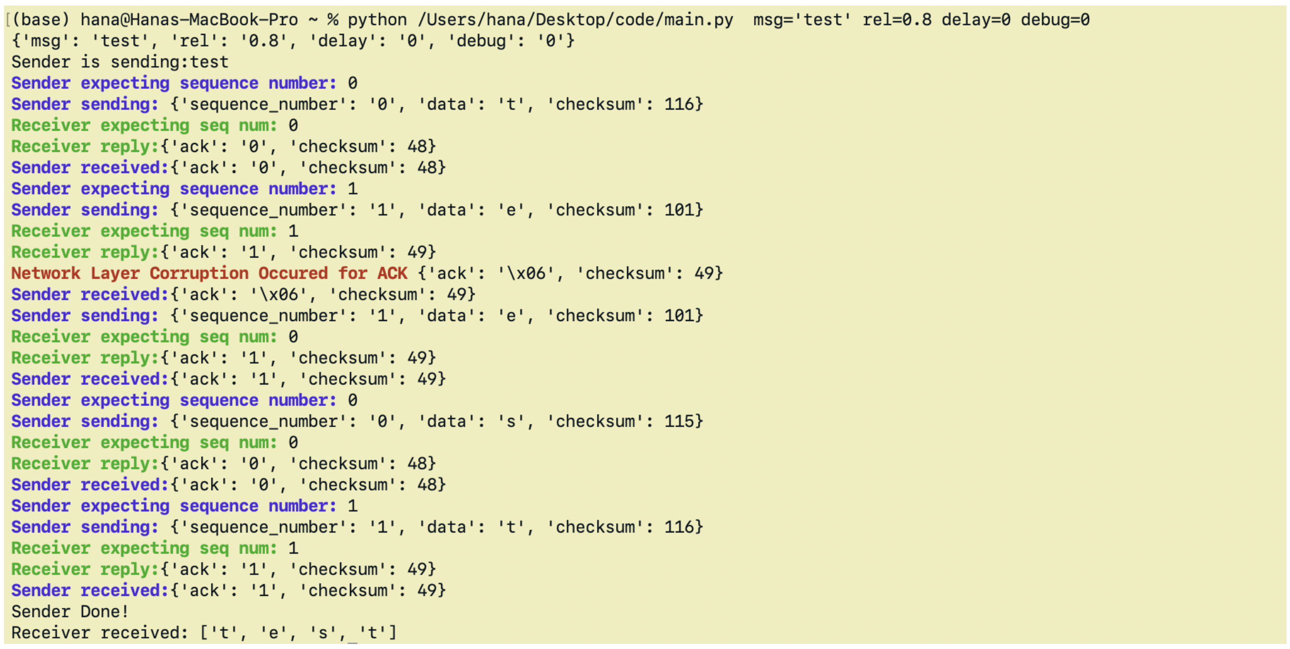
Section 5: Test Cases

1) REL=0.4



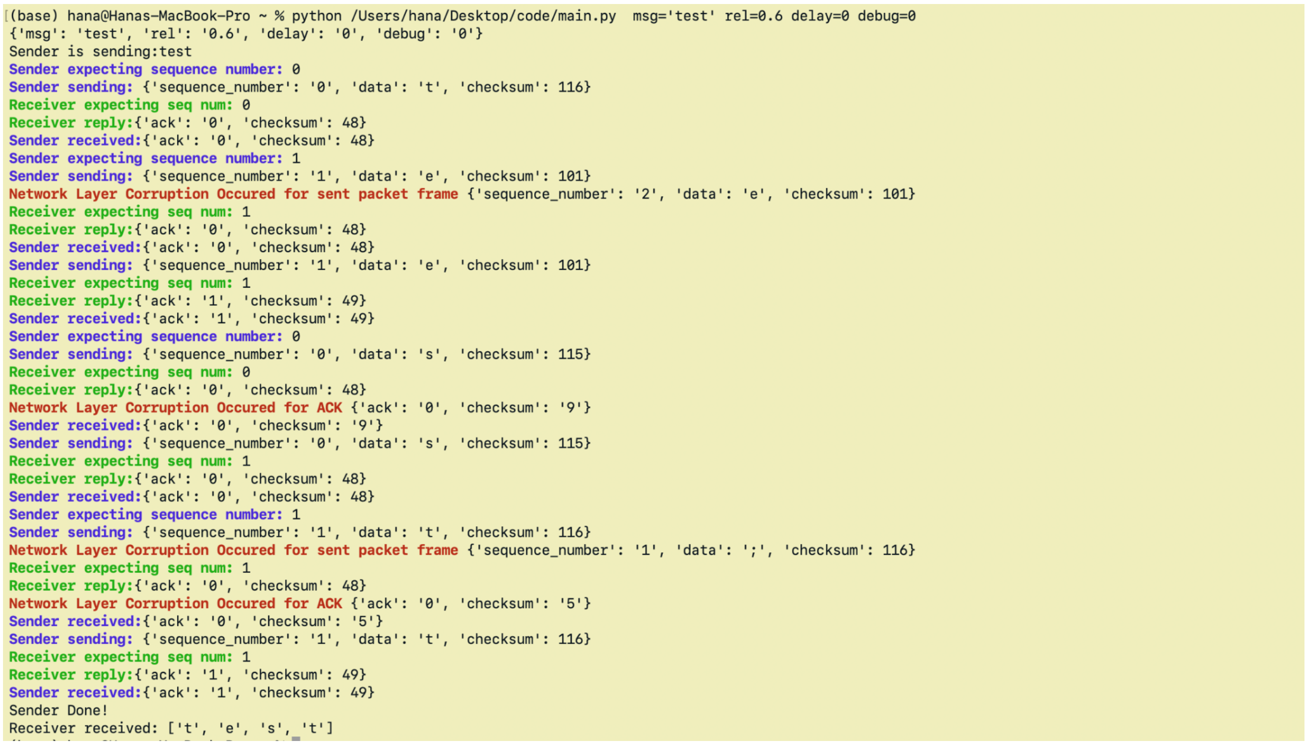
```
[(base) hana@Hanas-MacBook-Pro ~ % python /Users/hana/Desktop/code/main.py  msg='test' rel=0.4 delay=0 debug=0
{'msg': 'test', 'rel': '0.4', 'delay': '0', 'debug': '0'}
Sender is sending:test
Sender expecting sequence number: 0
Sender sending: {'sequence_number': '0', 'data': 't', 'checksum': 116}
Receiver expecting seq num: 0
Receiver reply:{'ack': '0', 'checksum': 48}
Network Layer Corruption Occured for ACK {'ack': '0', 'checksum': '6'}
Sender received:{'ack': '0', 'checksum': '6'}
Sender sending: {'sequence_number': '0', 'data': 't', 'checksum': 116}
Receiver expecting seq num: 1
Receiver reply:{'ack': '0', 'checksum': 48}
Network Layer Corruption Occured for ACK {'ack': '\x03', 'checksum': 48}
Sender received:{'ack': '\x03', 'checksum': 48}
Sender sending: {'sequence_number': '0', 'data': 't', 'checksum': 116}
Network Layer Corruption Occured for sent packet frame {'sequence_number': '0', 'data': 'y', 'checksum': 116}
Receiver expecting seq num: 1
Receiver reply:{'ack': '0', 'checksum': 48}
Sender received:{'ack': '0', 'checksum': 48}
Sender expecting sequence number: 1
Sender sending: {'sequence_number': '1', 'data': 'e', 'checksum': 101}
Receiver expecting seq num: 1
Receiver reply:{'ack': '1', 'checksum': 49}
Network Layer Corruption Occured for ACK {'ack': '\x04', 'checksum': 49}
Sender received:{'ack': '\x04', 'checksum': 49}
Sender sending: {'sequence_number': '1', 'data': 'e', 'checksum': 101}
Network Layer Corruption Occured for sent packet frame {'sequence_number': '1', 'data': 'e', 'checksum': 113}
Receiver expecting seq num: 0
Receiver reply:{'ack': '1', 'checksum': 49}
Network Layer Corruption Occured for ACK {'ack': '1', 'checksum': '3'}
Sender received:{'ack': '1', 'checksum': '3'}
Sender sending: {'sequence_number': '1', 'data': 'e', 'checksum': 101}
Network Layer Corruption Occured for sent packet frame {'sequence_number': '1', 'data': 'e', 'checksum': 84}
Receiver expecting seq num: 0
Receiver reply:{'ack': '1', 'checksum': 49}
Network Layer Corruption Occured for ACK {'ack': '1', 'checksum': '2'}
Sender received:{'ack': '1', 'checksum': '2'}
Sender sending: {'sequence_number': '1', 'data': 'e', 'checksum': 101}
Receiver expecting seq num: 0
Receiver reply:{'ack': '1', 'checksum': 49}
Sender received:{'ack': '1', 'checksum': 49}
Sender expecting sequence number: 0
Sender sending: {'sequence_number': '0', 'data': 's', 'checksum': 115}
Receiver expecting seq num: 0
Receiver reply:{'ack': '0', 'checksum': 48}
Network Layer Corruption Occured for ACK {'ack': '\x07', 'checksum': 48}
Sender received:{'ack': '\x07', 'checksum': 48}
Sender sending: {'sequence_number': '0', 'data': 's', 'checksum': 115}
Receiver expecting seq num: 1
Receiver reply:{'ack': '0', 'checksum': 48}
Network Layer Corruption Occured for ACK {'ack': '\x07', 'checksum': 48}
Sender received:{'ack': '\x07', 'checksum': 48}
Sender sending: {'sequence_number': '0', 'data': 's', 'checksum': 115}
Receiver expecting seq num: 1
Receiver reply:{'ack': '0', 'checksum': 48}
Network Layer Corruption Occured for ACK {'ack': '\x07', 'checksum': 48}
Sender received:{'ack': '\x07', 'checksum': 48}
Sender sending: {'sequence_number': '0', 'data': 's', 'checksum': 115}
Network Layer Corruption Occured for sent packet frame {'sequence_number': '4', 'data': 's', 'checksum': 115}
Receiver expecting seq num: 1
Receiver reply:{'ack': '0', 'checksum': 48}
Sender received:{'ack': '0', 'checksum': 48}
Sender expecting sequence number: 1
Sender sending: {'sequence_number': '1', 'data': 't', 'checksum': 116}
Receiver expecting seq num: 1
Receiver reply:{'ack': '1', 'checksum': 49}
Sender received:{'ack': '1', 'checksum': 49}
Sender Done!
Receiver received: ['t', 'e', 's', 't']
(base) hana@Hanas-MacBook-Pro ~ %
```

2) REL=1



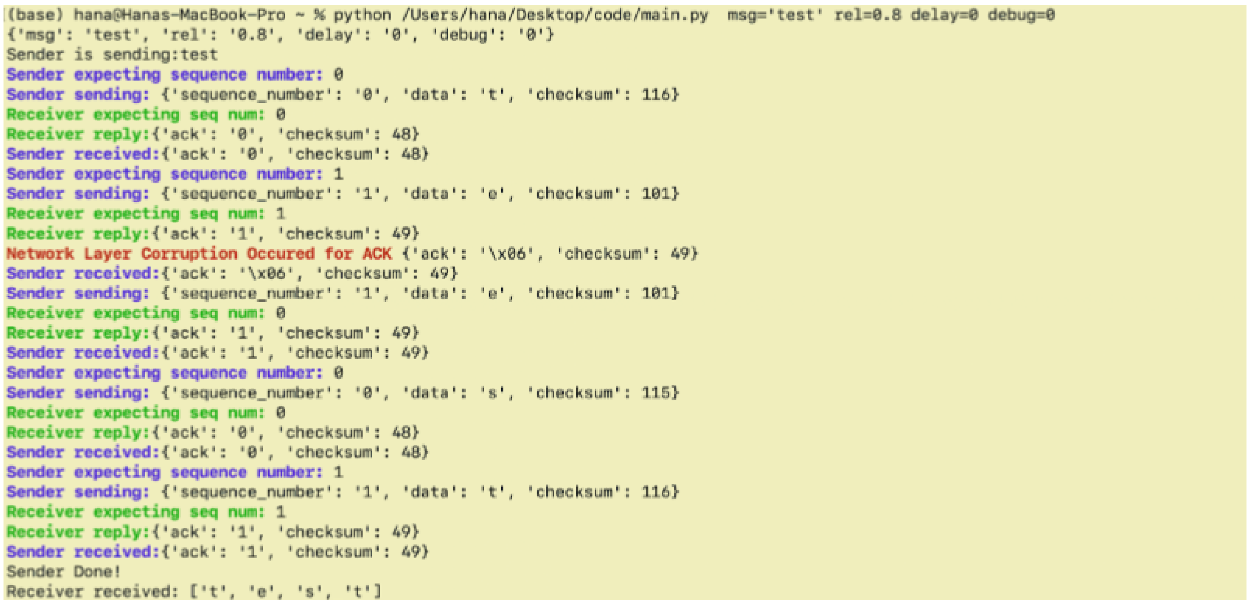
```
(base) hana@Hanas-MacBook-Pro ~ % python /Users/hana/Desktop/code/main.py  msg='test' rel=0.8 delay=0 debug=0
{'msg': 'test', 'rel': '0.8', 'delay': '0', 'debug': '0'}
Sender is sending:test
Sender expecting sequence number: 0
Sender sending: {'sequence_number': '0', 'data': 't', 'checksum': 116}
Receiver expecting seq num: 0
Receiver reply:{'ack': '0', 'checksum': 48}
Sender received:{'ack': '0', 'checksum': 48}
Sender expecting sequence number: 1
Sender sending: {'sequence_number': '1', 'data': 'e', 'checksum': 101}
Receiver expecting seq num: 1
Receiver reply:{'ack': '1', 'checksum': 49}
Network Layer Corruption Occured for ACK {'ack': '\x06', 'checksum': 49}
Sender received:{'ack': '\x06', 'checksum': 49}
Sender sending: {'sequence_number': '1', 'data': 'e', 'checksum': 101}
Receiver expecting seq num: 0
Receiver reply:{'ack': '1', 'checksum': 49}
Sender received:{'ack': '1', 'checksum': 49}
Sender expecting sequence number: 0
Sender sending: {'sequence_number': '0', 'data': 's', 'checksum': 115}
Receiver expecting seq num: 0
Receiver reply:{'ack': '0', 'checksum': 48}
Sender received:{'ack': '0', 'checksum': 48}
Sender expecting sequence number: 1
Sender sending: {'sequence_number': '1', 'data': 't', 'checksum': 116}
Receiver expecting seq num: 1
Receiver reply:{'ack': '1', 'checksum': 49}
Sender received:{'ack': '1', 'checksum': 49}
Sender Done!
Receiver received: ['t', 'e', 's', 't']
```

3) REL= 0.6



```
(base) hana@Hanas-MacBook-Pro ~ % python /Users/hana/Desktop/code/main.py  msg='test' rel=0.6 delay=0 debug=0
{'msg': 'test', 'rel': '0.6', 'delay': '0', 'debug': '0'}
Sender is sending:test
Sender expecting sequence number: 0
Sender sending: {'sequence_number': '0', 'data': 't', 'checksum': 116}
Receiver expecting seq num: 0
Receiver reply:{'ack': '0', 'checksum': 48}
Sender received:{'ack': '0', 'checksum': 48}
Sender expecting sequence number: 1
Sender sending: {'sequence_number': '1', 'data': 'e', 'checksum': 101}
Network Layer Corruption Occured for sent packet frame {'sequence_number': '2', 'data': 'e', 'checksum': 101}
Receiver expecting seq num: 1
Receiver reply:{'ack': '0', 'checksum': 48}
Sender received:{'ack': '0', 'checksum': 48}
Sender sending: {'sequence_number': '1', 'data': 'e', 'checksum': 101}
Receiver expecting seq num: 1
Receiver reply:{'ack': '1', 'checksum': 49}
Sender received:{'ack': '1', 'checksum': 49}
Sender expecting sequence number: 0
Sender sending: {'sequence_number': '0', 'data': 's', 'checksum': 115}
Receiver expecting seq num: 0
Receiver reply:{'ack': '0', 'checksum': 48}
Network Layer Corruption Occured for ACK {'ack': '0', 'checksum': '9'}
Sender received:{'ack': '0', 'checksum': '9'}
Sender sending: {'sequence_number': '0', 'data': 's', 'checksum': 115}
Receiver expecting seq num: 1
Receiver reply:{'ack': '0', 'checksum': 48}
Sender received:{'ack': '0', 'checksum': 48}
Sender expecting sequence number: 1
Sender sending: {'sequence_number': '1', 'data': 't', 'checksum': 116}
Network Layer Corruption Occured for sent packet frame {'sequence_number': '1', 'data': ';', 'checksum': 116}
Receiver expecting seq num: 1
Receiver reply:{'ack': '0', 'checksum': 48}
Network Layer Corruption Occured for ACK {'ack': '0', 'checksum': '5'}
Sender received:{'ack': '0', 'checksum': '5'}
Sender sending: {'sequence_number': '1', 'data': 't', 'checksum': 116}
Receiver expecting seq num: 1
Receiver reply:{'ack': '1', 'checksum': 49}
Sender received:{'ack': '1', 'checksum': 49}
Sender Done!
Receiver received: ['t', 'e', 's', 't']
```

4) REL=0.8



```
(base) hana@Hanas-MacBook-Pro ~ % python /Users/hana/Desktop/code/main.py  msg='test' rel=0.8 delay=0 debug=0
{'msg': 'test', 'rel': '0.8', 'delay': '0', 'debug': '0'}
Sender is sending:test
Sender expecting sequence number: 0
Sender sending: {'sequence_number': '0', 'data': 't', 'checksum': 116}
Receiver expecting seq num: 0
Receiver reply:{'ack': '0', 'checksum': 48}
Sender received:{'ack': '0', 'checksum': 48}
Sender expecting sequence number: 1
Sender sending: {'sequence_number': '1', 'data': 'e', 'checksum': 101}
Receiver expecting seq num: 1
Receiver reply:{'ack': '1', 'checksum': 49}
Network Layer Corruption Occured for ACK {'ack': '\x06', 'checksum': 49}
Sender received:{'ack': '\x06', 'checksum': 49}
Sender sending: {'sequence_number': '1', 'data': 'e', 'checksum': 101}
Receiver expecting seq num: 0
Receiver reply:{'ack': '1', 'checksum': 49}
Sender received:{'ack': '1', 'checksum': 49}
Sender expecting sequence number: 0
Sender sending: {'sequence_number': '0', 'data': 's', 'checksum': 115}
Receiver expecting seq num: 0
Receiver reply:{'ack': '0', 'checksum': 48}
Sender received:{'ack': '0', 'checksum': 48}
Sender expecting sequence number: 1
Sender sending: {'sequence_number': '1', 'data': 't', 'checksum': 116}
Receiver expecting seq num: 1
Receiver reply:{'ack': '1', 'checksum': 49}
Sender received:{'ack': '1', 'checksum': 49}
Sender Done!
Receiver received: ['t', 'e', 's', 't']
```