**Sender**

    def get\_checksum(data):

       Calculate the checksum for outgoing data

param data: one and only one character

return: the ASCII code of the character, for example ASCII('A') = 65        """

checksum = ord(data)

the checksum variable represents the value of the ascii code of the data character input to the function using the ord method and then the function returns the checksum variable that represents the ascii value

        return checksum

def is\_corrupted(reply):

        Check if the received reply from receiver is corrupted or not

        param reply: a python dictionary represent a reply sent by the receiver

        return: True -> if the reply is corrupted | False ->  if the reply is NOT corrupted

here we check that the reply checksum value of the reply’s acknowledgement is = to the ascii value of the acknowledgement of the reply to make sure that the acknowledgement is not corrupted

        if (ord(reply['ack']) == reply['checksum']):

            return False

        else:

            return True

        pass

def rdt\_send(self, process\_buffer):

for data in process\_buffer:

print(f'Sender Expecting: seq:{self.sequence}')

checksum = RDTSender.get\_checksum(data)

pkt = RDTSender.make\_pkt(self.sequence, data, checksum)

clone =RDTSender.clone\_packet(pkt)

print(f'Sender sent: msg={pkt}')

reply = self.net\_srv.udt\_send(pkt)

while(RDTSender.is\_corrupted(reply) or RDTSender.is\_expected\_seq(reply, self.sequence) == False):

pkt = clone

clone = RDTSender.clone\_packet(pkt)

print(f'Sender out: msg={pkt}')

reply = self.net\_srv.udt\_send(pkt)

if(self.sequence=='0'):

self.sequence='1'

else:

self.sequence='0'

print("Sender Done!")

return

In our implementation to the function (rdt\_send), we carried out a “for loop” that checks on each character that we need to send (as a sender). We start the loop with the condition that there are ‘data’ in our buffer that are to be sent, our first edit to perform our implementation was to add a print statement to give detail (sequence number) about the character that is currently in the buffer and is in the process of being sent. Our following set of steps were initializing the checksum, creating the packet itself, creating a clone to the packet, displaying the packet, sending it to the network. Within the “for loop”, we nested a “While loop”. The objective of the while loop is to make sure that the packet sent was a. Not corrupted. b. Had the correct sequence number. If any of those two conditions was not met, we would enter the loop to make sure that we correct the fault that occurred (that of points a or b). Upon entering the loop, we wrote a few statements. To start off, we made sure that the packet that would be resent(pkt) would have the value and information of the clone we created beforehand so that our initial packet would not get corrupted. We then clone the packet (pkt) to make sure we still have the original packet unharmed. Afterwards, we print the packet we are sending then actually send it to the network. After each packet is sent successfully, we flip the sequence so that we can keep track of any further corruption or messed up sequence number in following packets. Finally, we have sent the whole message and announce that the sender is done.

    def is\_expected\_seq(reply, exp\_seq):

        Check if the received reply from receiver has the expected sequence number

        param reply: a python dictionary represent a reply sent by the receiver

        param exp\_seq: the sender expected sequence number '0' or '1' represented as a character

        :return: True -> if ack in the reply match the   expected sequence number otherwise False

if (reply['ack']==exp\_seq):

            return True

        else:

            return False

        pass

Here we check that the reply’s acknowledgement is = to the expected sequence number

**Receiver**

  def is\_corrupted(packet):

        Check if the received packet from sender is corrupted or not

            param packet: a python dictionary represent a packet received from the sender

            return: True -> if the reply is corrupted | False ->  if the reply is NOT corrupted

        if (ord(packet['data']) == packet['checksum']):

            return False

        else:

            return True

        pass

Here we are checking that the ascii value of the data using the ord function of the packet input is = to checksum value of the sent packet. If they are = then the packet is not corrupted other wise means that it is corrupted

    def is\_expected\_seq(rcv\_pkt, exp\_seq):

        Check if the received reply from receiver has the expected sequence number

         param rcv\_pkt: a python dictionary represent a packet received by the receiver

         param exp\_seq: the receiver expected sequence number '0' or '1' represented as a character

         return: True -> if ack in the reply match the   expected sequence number otherwise False

        if (rcv\_pkt['sequence\_number']==exp\_seq):

            return True

        else:

            return False

        pass

Here we are checking that received packet sequence number is = to expected sequence number

    def rdt\_rcv(self, rcv\_pkt):

       Implement the RDT v2.2 for the receiver

        param rcv\_pkt: a packet delivered by the network layer 'udt\_send()' to the receiver

        return: the reply packet

        print(f'Reciever Expecting: seq:{self.sequence}') here printing the expected sequence number

        if(RDTReceiver.is\_corrupted(rcv\_pkt)==True or RDTReceiver.is\_expected\_seq(rcv\_pkt,self.sequence)==False ):

checking if the message is corrupted or has a different sequence number than the expected

            print(f'Corruption Occured: msg={rcv\_pkt}') saying that the message is corrupted and listing the message

            if(self.sequence == '0'):

                reply\_pkt = RDTReceiver.make\_reply\_pkt('1',ord('1'))

            else:

                reply\_pkt = RDTReceiver.make\_reply\_pkt('0',ord('0'))

  else:

               print(f'Recieved: msg={rcv\_pkt}')

        # deliver the data to the process in the application layer

               ReceiverProcess.deliver\_data(rcv\_pkt['data'])

               reply\_pkt = RDTReceiver.make\_reply\_pkt(self.sequence, ord(self.sequence))

               if(self.sequence=='0'):

                   self.sequence='1'

                   #return reply\_pkt

               else:

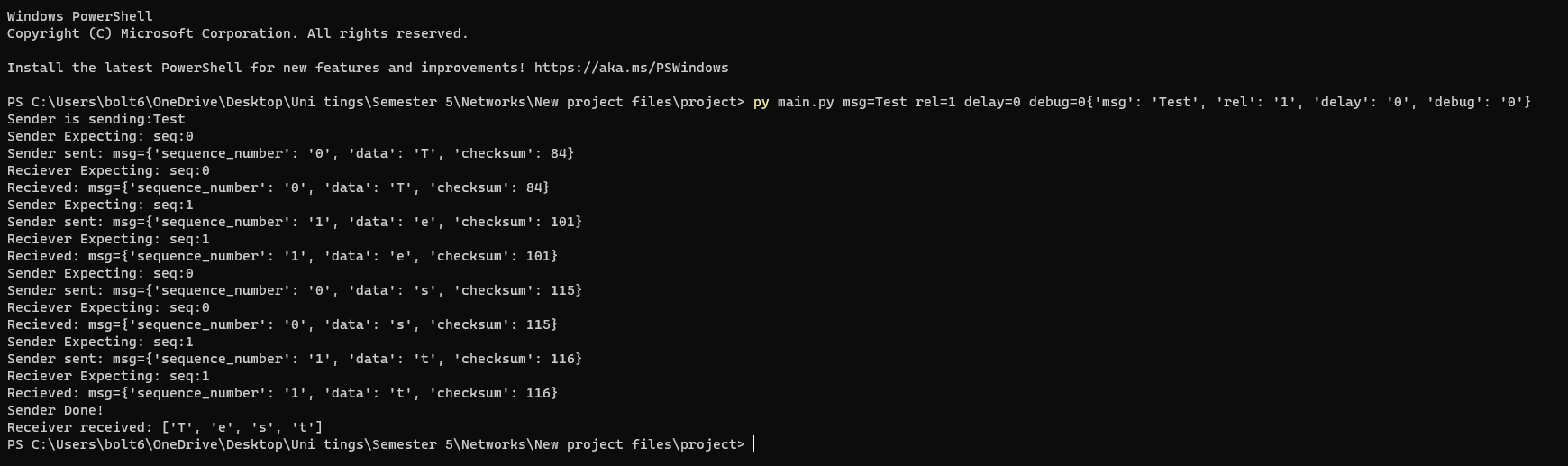
                   self.sequence='0'

                   #return reply\_pkt

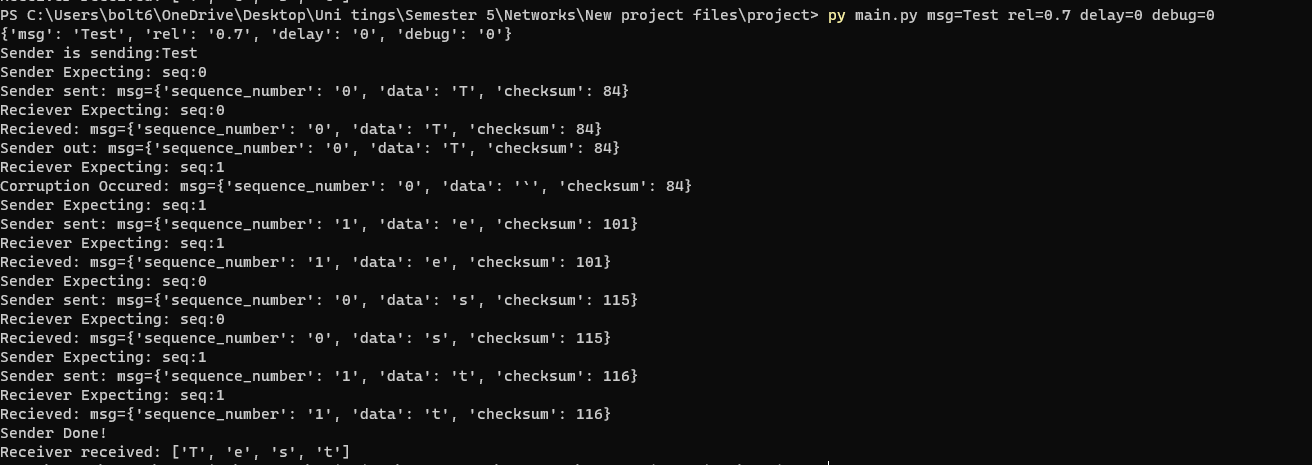
        return reply\_pkt

Here checking that if the expected sequence number =0 we reply and send the reply with acknowledgement of 1 and ascii value of it as checksum and the opposite other wise at the end of the method if the message is not corrupted and sequence number as expected then we print the message and deliver the data inside the packet to the application layer and replying with the sequence number as acknowledgment and it’s ascii value as checksum and then changing the sequence number of the self if it’s one then 0 and the opposite and return the reply to the sender.

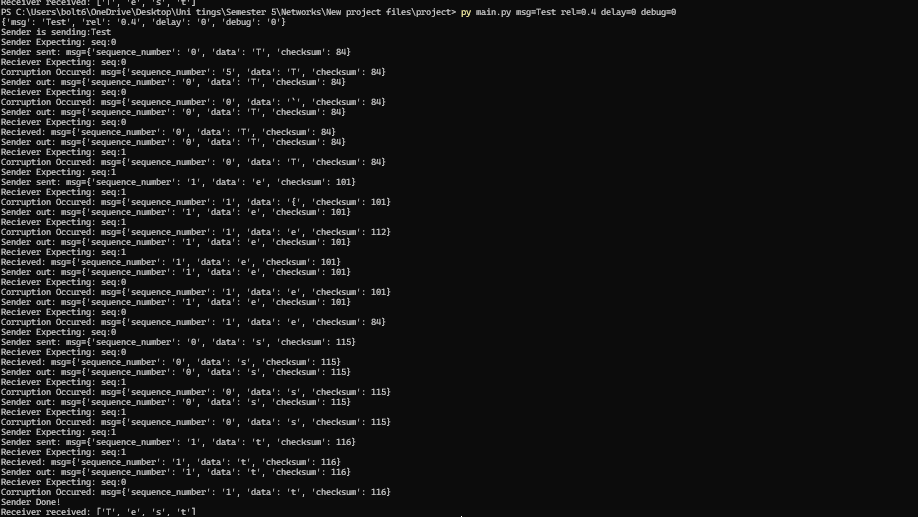
Terminal when reliability = 1.0



Terminal when reliability = 0.7

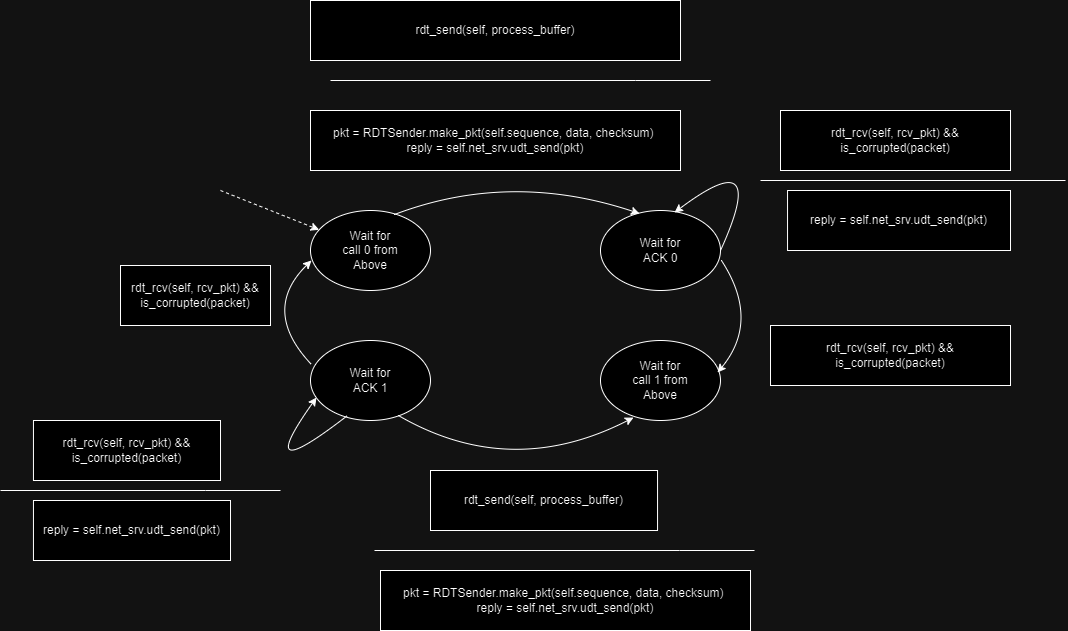


Terminal when reliability = 0.4



**FSM**

Sender



Receiver

