Introduction to communication networks project - Principles of Reliable Data Transfer

Submitted By:

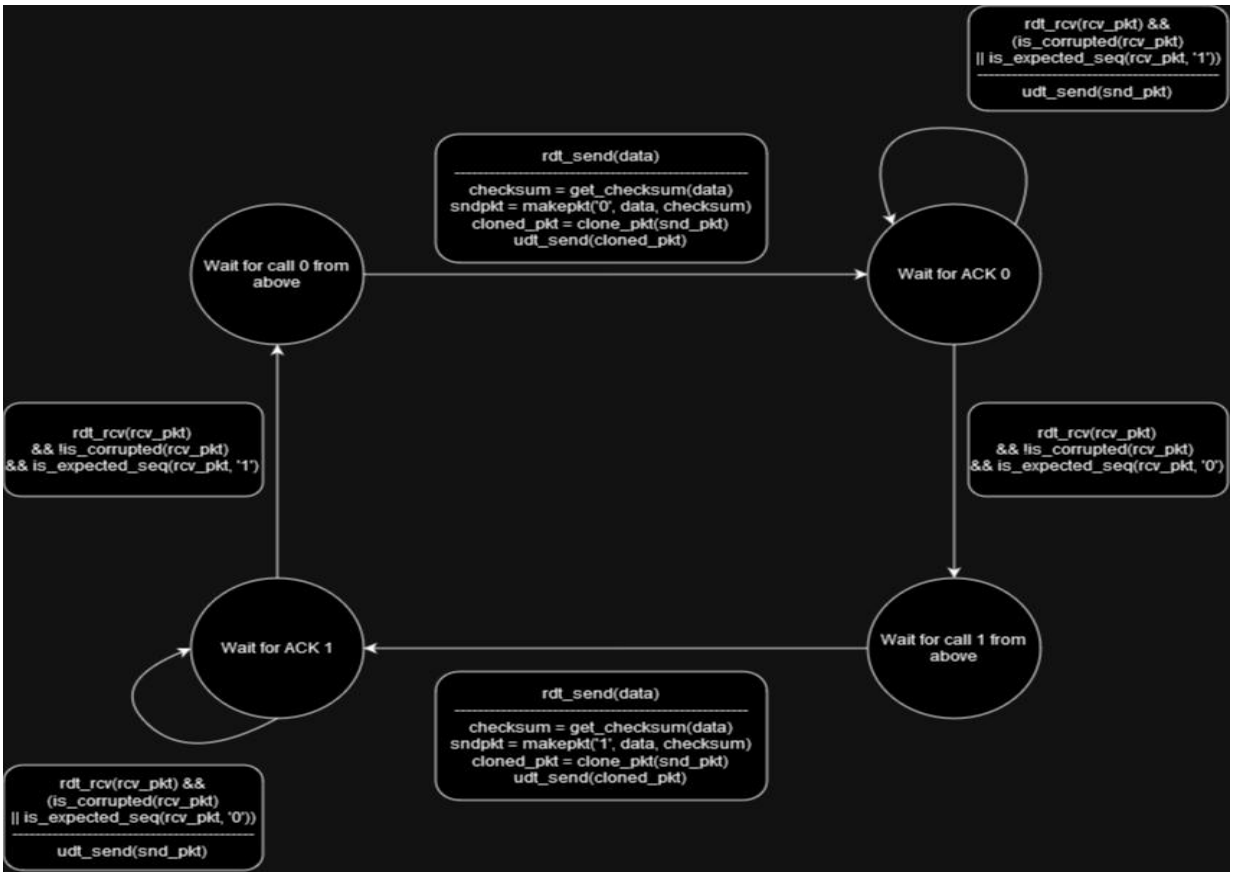
SeifEldin Khaled ,55-25218, T-11, Seifeldin.abbas@student.guc.edu.eg

Yousef Yasser ,55-3437, T-11, yousef.mostafa@student.guc.edu.eg

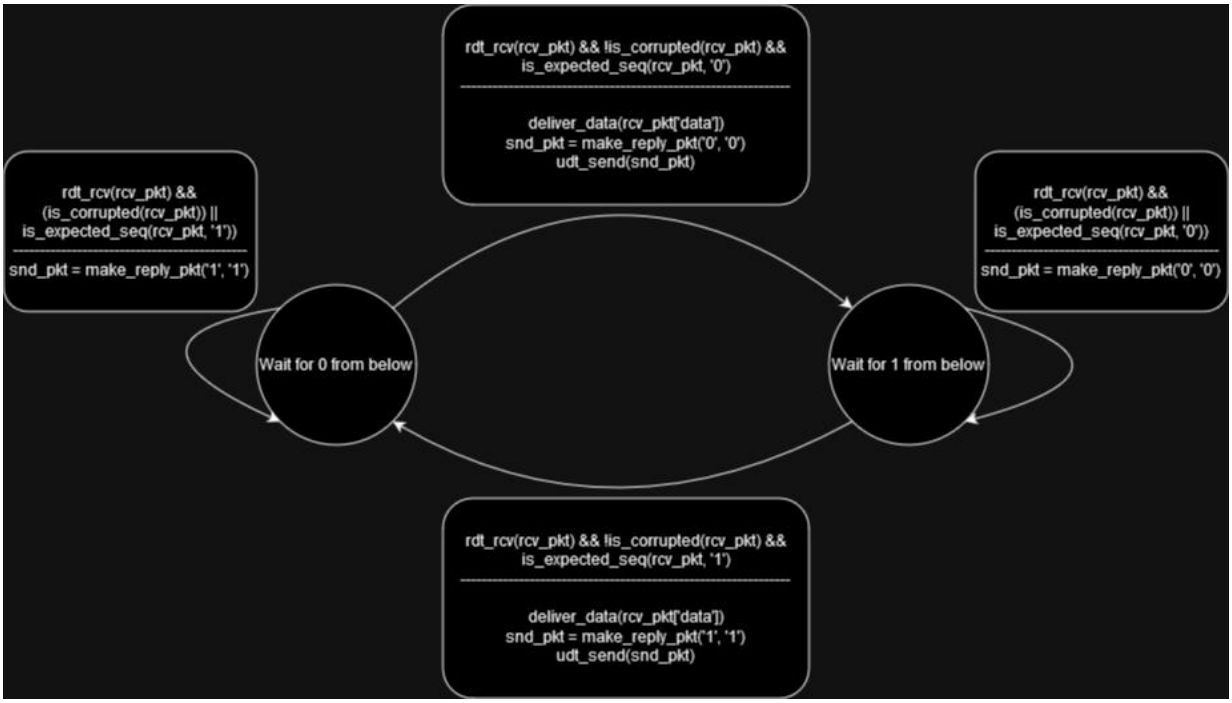
Contributions:

Seif worked on the sender side while Yousef worked on the receiver side, we both worked on the debugging together at the end

Sender FSM diagram:



Receiver FSM diagram:



Pseudocode for the RDT sender:

LOOP over each character in the sender's buffer

    Calculate checksum

    Construct packet

    REPEAT

        Clone the packet

        Send the cloned packet via the network layer

    UNTIL reply is not corrupted

Pseudocode for the RDT receiver:

IF packet is corrupted

    Flip the acknowledgement to notify the sender

ELSE

    Deliver data to application layer into receiver's buffer

    Flip the sequence number to receive next data

Return the reply packet containing the acknowledgement and its checksum

Necessary changes made to the skeleton code:

1- sender.py:

```
@staticmethod
def get_checksum(data):
    """ Calculate the checksum for outgoing data
    :param data: one and only one character, for example data = 'A'
    :return: the ASCII code of the character, for example ASCII('A') = 65
    """
    # TODO provide your own implementation
    checksum = ord(data)
    return checksum
```

Calculated the checksum from the given data (character) by converting it to its corresponding ASCII value to be sent with the data for data validation at receiver side (ensure packet was not corrupted)

```
checksum = RDTSender.get_checksum(data)
pkt = RDTSender.make_pkt(self.sequence, data, checksum)

print(f'Sender expecting seq_num: {self.sequence}')
print(f'Sender sending: {pkt}')

cloned_pkt = RDTSender.clone_packet(pkt)
reply = self.net_srv.udt_send(cloned_pkt)
print(f'Sender received: {reply}')

while(RDTSender.is_corrupted(reply) or not RDTSender.is_expected_seq(reply, self.sequence)):
    print(f'Sender expecting seq_num: {self.sequence}')
    print(f'Sender sending: {pkt}')
    cloned_pkt = RDTSender.clone_packet(pkt)
    reply = self.net_srv.udt_send(cloned_pkt)
    print(f'Sender received: {reply}')

self.sequence = '1' if self.sequence == '0' else '0'
```

Sender keeps on sending the cloned packet until the acknowledgement sent by receiver has been received successfully by the sender with no corruption, print statements was added for visualizing the sending and receiving process as well as corruption errors, the sender had to send a cloned packet and not the original packet as when corruption occurs the original packet is unchanged and can be sent again, finally the sender's sequence number flips whenever a packet is sent successfully to send the following data in the buffer.

2- network.py:

```
if s_test and self.pkt_corrupt:
    self.__corrupt_packet()
    print(f"Network layer: corruption occurred {self.packet}")
```

Print statement alerts the user whenever the packet is corrupted

```
if r_test and self.ack_corrupt:
    self.__corrupt_reply()
    print(f"Network layer: corruption occurred {self.reply}")
```

Print statement alerts the user whenever the acknowledgement is corrupted

3- receiver.py:

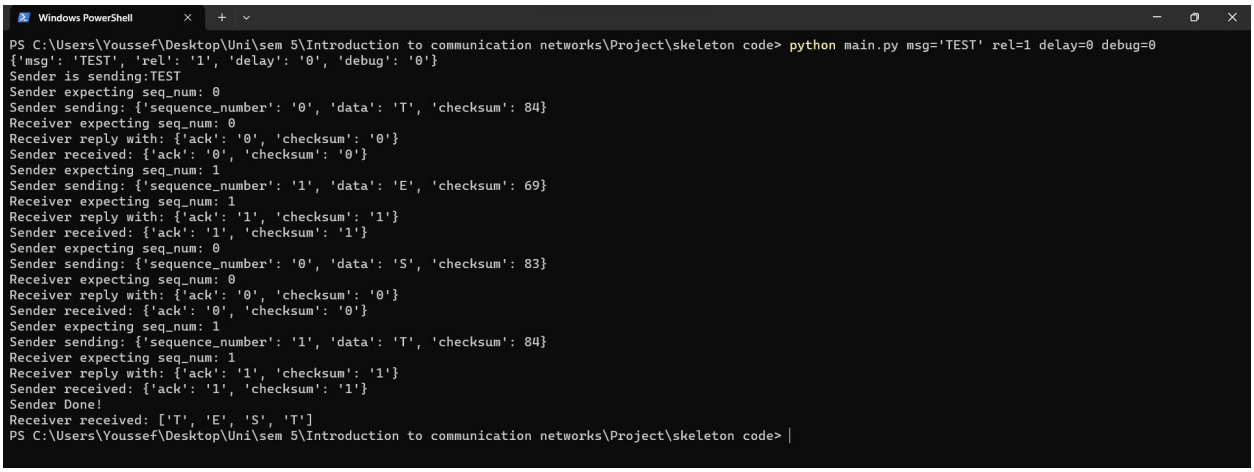
```python
@staticmethod
def is_corrupted(packet):
    """ Check if the received packet from sender is corrupted or not
        :param packet: a python dictionary represent a packet received from the sender
        :return: True -> if the reply is corrupted | False ->  if the reply is NOT corrupted
    """
    # TODO provide your own implementation
    return packet['checksum'] != ord(packet['data'])

@staticmethod
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
    """
    # TODO provide your own implementation
    return rcv_pkt['sequence_number'] == exp_seq
```

A check is made to compare the checksum sent by the sender to the actual data (converted to its ASCII value), this validates that no corruption happened to the data during transmission.

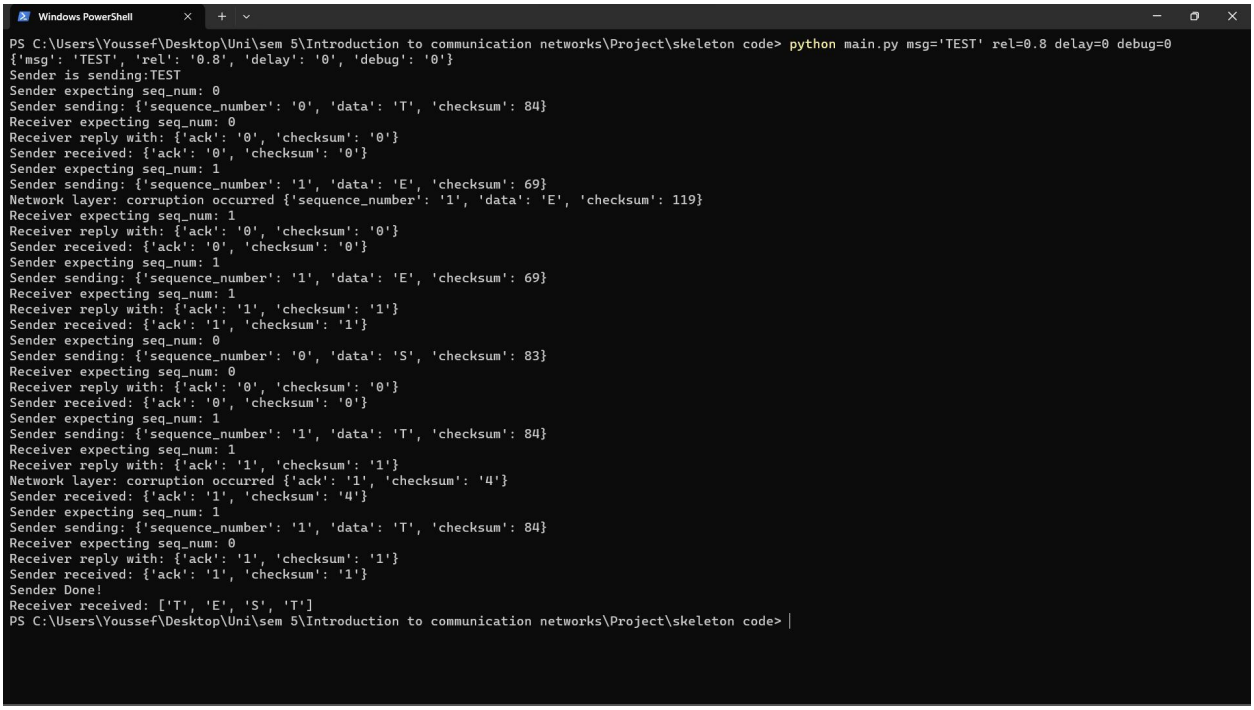
Another check is made to ensure that the senders's sequence number was not corrupted by comparing it to the receiver's sequence number.

Test Cases:

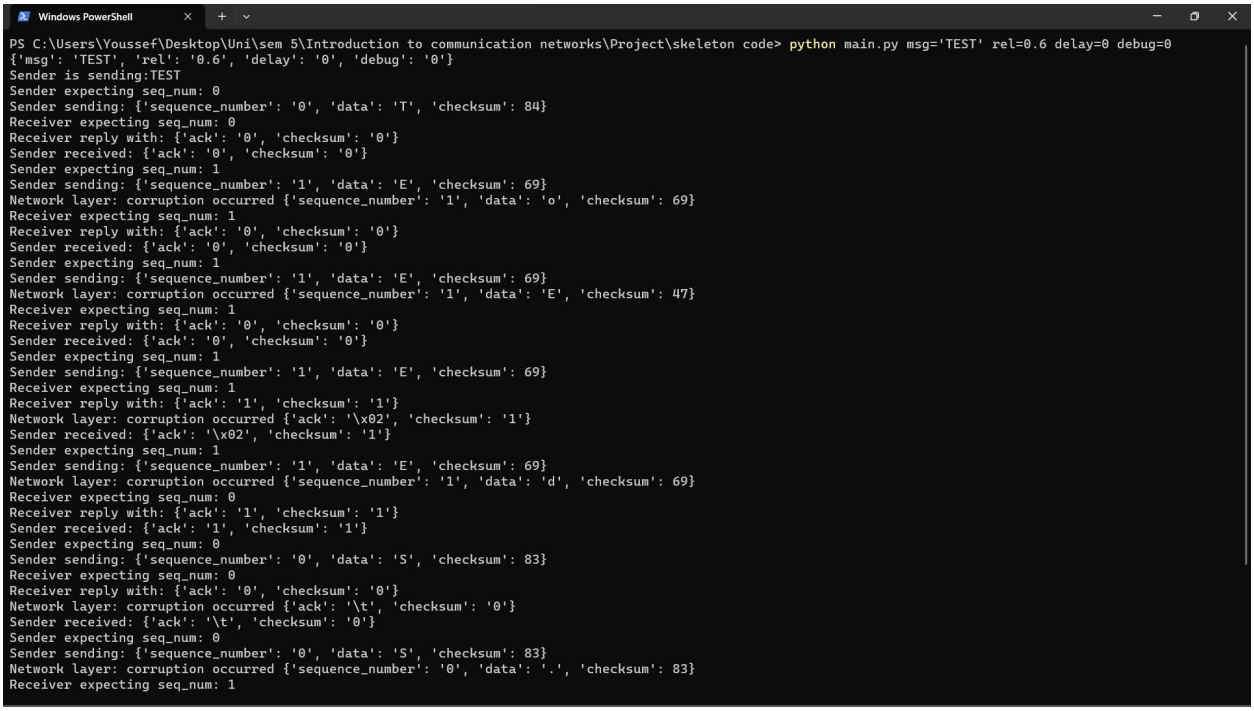
1- Reliability = 1 (No Corruption)



2- Reliability = 0.8 (20% Corruption probability for packet or acknowledgement)



3- Reliability = 0.6



```
Receiver reply with: {'ack': '0', 'checksum': '0'}
Sender received: {'ack': '0', 'checksum': '0'}
Sender expecting seq_num: 1
Sender sending: {'sequence_number': '1', 'data': 'T', 'checksum': 84}
Receiver expecting seq_num: 1
Receiver reply with: {'ack': '1', 'checksum': '1'}
Sender received: {'ack': '1', 'checksum': '1'}
Sender Done!
Receiver received: ['T', 'E', 'S', 'T']
PS C:\Users\Youssef\Desktop\Uni\sem 5\Introduction to communication networks\Project\skeleton code>
```