Networks

Team members: TA Basma

1-Kamal Said 55-1387 T-18

Email:kamal.said58@yahoo.com

2-Nourhan Mohamed 55-0325 T-18
Email:Nourhan.abbas@student.guc.edu.eg

3-Omar Wael 55-26722 T-15
Email:omarkhalifa1052003@gmail.com

Contributions

Kamal and Nourhan had responsibility of finishing the TODO parts of the code

Omar Wael had responsibility of finishing the report

CHANGES

SENDER SIDE

Explanation of functions

get_checksum(data):

Get ASCII code of given data which is equivalent for us to the checksum of data.

is_corrupted(reply):

Checks if reply is corrupted. Returns true (when corrupted) if acknowledgment is any other number than 1 and 0 OR the checksum of the reply is not equivalent to the checksum of the acknowledgment by using get_checksum.

Is_expected_seq(reply,exp_seq):

Checks if the acknowledgment of the reply is as expected.

rdt_send()

We are attempting to send packets saved in the buffer of the sender side to the receiver side for each packet in the sender buffer we clone the packet so that if it got corrupted we still have the original packet.

1-We send the clone packet to receiver and receive reply message from receiver

2-We check the reply message received if corrupted using function is_corrupted(reply) and also check if the acknowledgment received is the one expected or not using function self.is_expected_seq(reply,self.sequence)

if reply corrupted or the acknowledgment received is NOT as expected print message indicating corruption and resend the clone packet again

If reply is not corrupted and acknowledgment received is as expected flip the sequence number and break the loop for this given packet so that the next packet start being sended.

RECEIVER SIDE

Explanation of functions

is_corrupted (packet):

Checks if packet is corrupted. Returns true (when corrupted) if checksum is any other number than 1 and 0 OR the checksum of the packet is not equivalent to the checksum of the data by using ord () —Returns ASCII code

Is_expected_seq (rcv_pkt, exp_seq):

Checks if the sequence number of the received packet is as expected.

rdt_rcv()

We are attempting to receive packets from sender to be saved in a buffer if the packet is not corrupted.

1-Receiver receives packet checks if packet is corrupted using is_corrupted(rcv_pkt) and also check if sequence number is the one expected or not using function self.is_expected_seq(rcv_pkt,self.sequence)

If received packet is corrupted or sequence number is NOT as expected

Print message indicating corruption and send reply to the sender with wrong acknowledgment so that sender side knows that an error occurred in past packet (i.e. send negative acknowledgment)

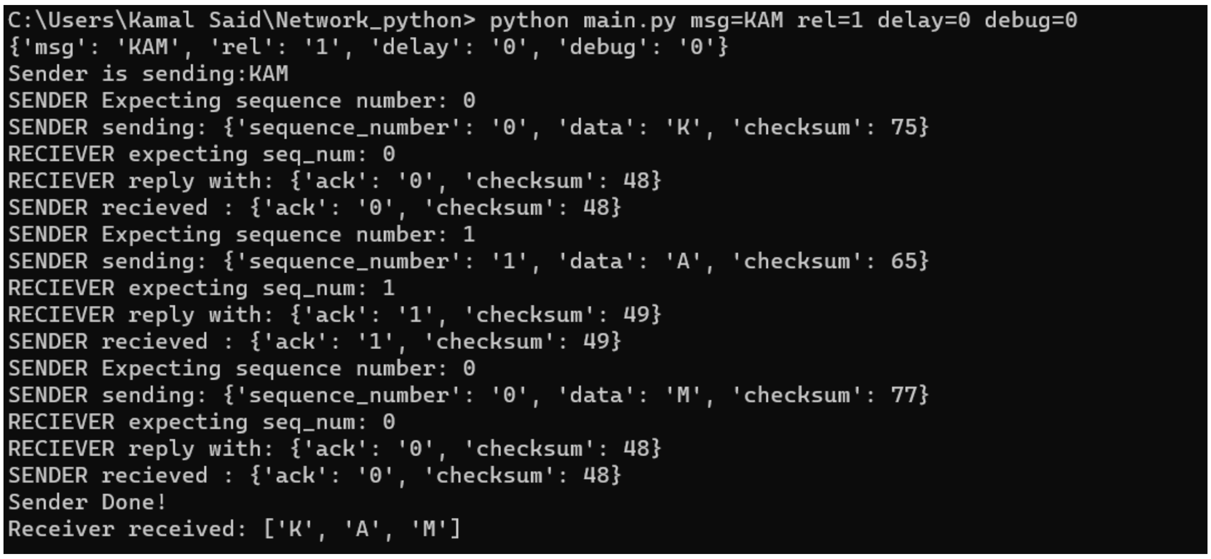
If received packet is not corrupted and the sequence number is as expected

Send reply with correct acknowledgment

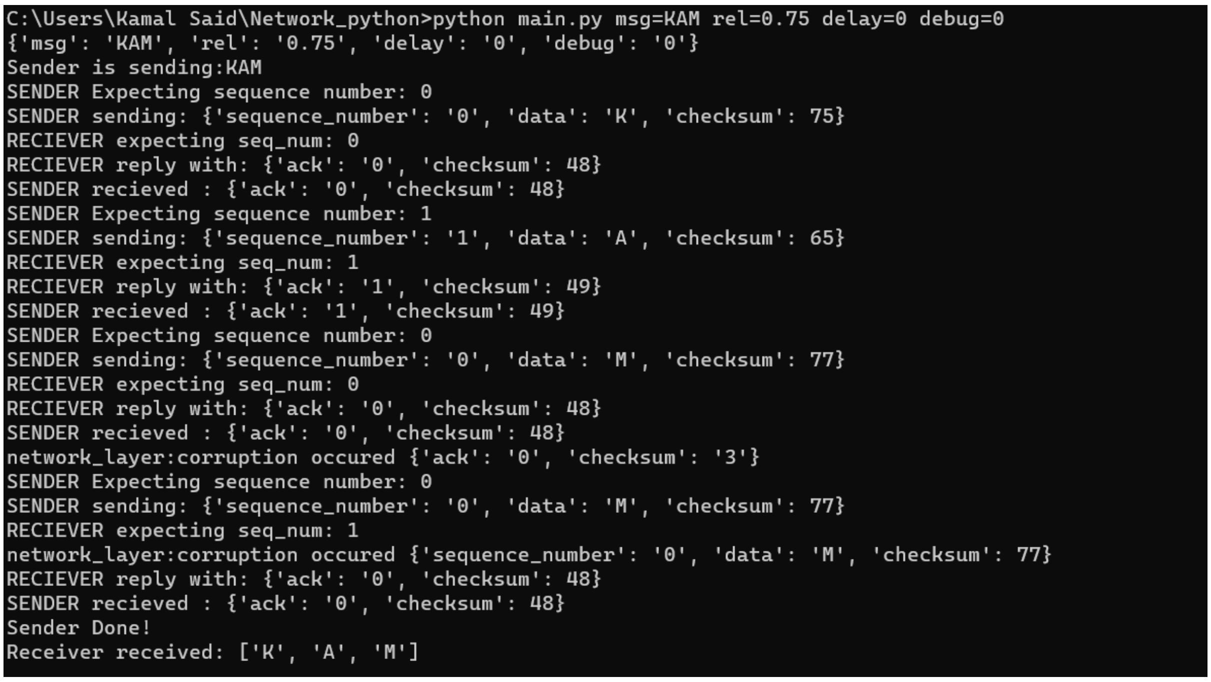
Flip the sequence number for the next packet

Save the data of received packet inside the receiver buffer

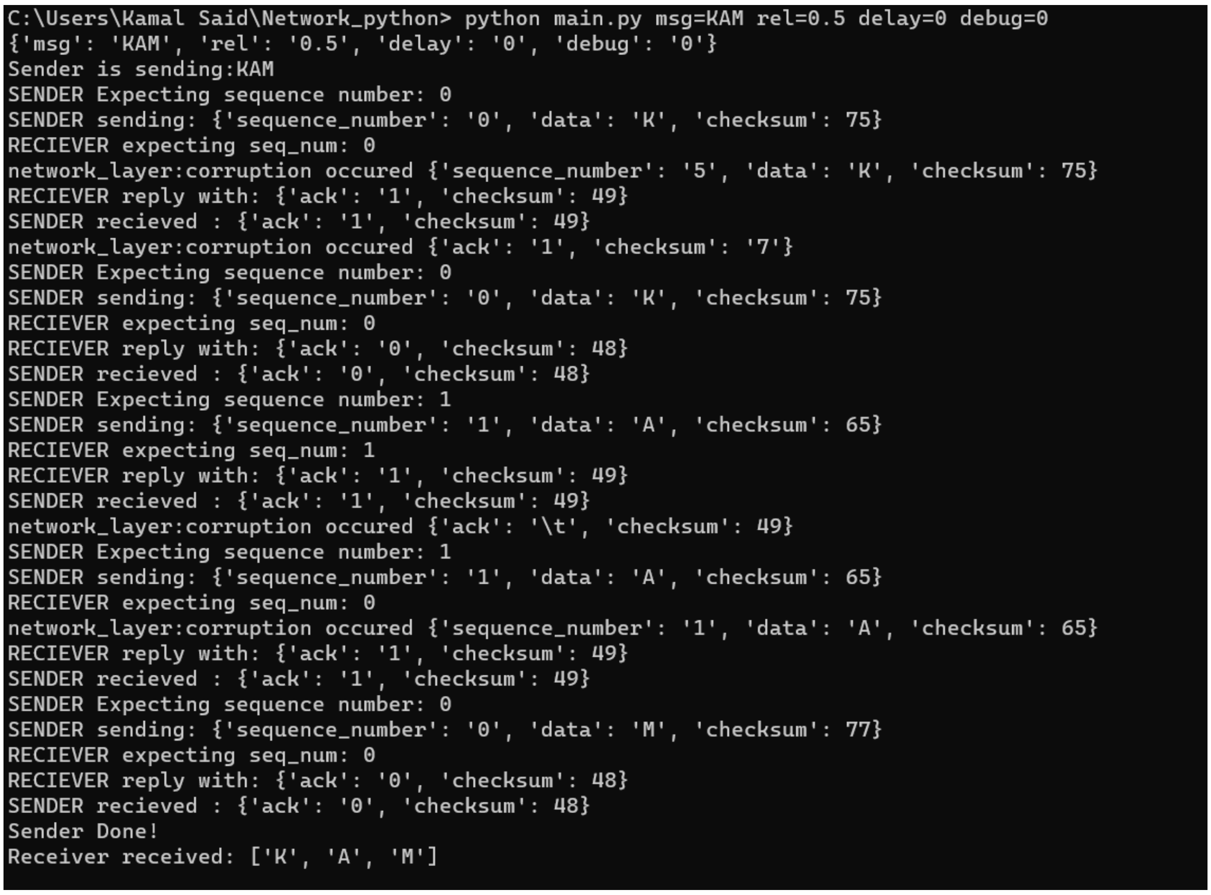
TESTS



```
C:\Users\Kamal Said\Network_python> python main.py msg=KAM rel=1 delay=0 debug=0
{'msg': 'KAM', 'rel': '1', 'delay': '0', 'debug': '0'}
Sender is sending:KAM
SENDER Expecting sequence number: 0
SENDER sending: {'sequence_number': '0', 'data': 'K', 'checksum': 75}
RECIEVER expecting seq_num: 0
RECIEVER reply with: {'ack': '0', 'checksum': 48}
SENDER recieved : {'ack': '0', 'checksum': 48}
SENDER Expecting sequence number: 1
SENDER sending: {'sequence_number': '1', 'data': 'A', 'checksum': 65}
RECIEVER expecting seq_num: 1
RECIEVER reply with: {'ack': '1', 'checksum': 49}
SENDER recieved : {'ack': '1', 'checksum': 49}
SENDER Expecting sequence number: 0
SENDER sending: {'sequence_number': '0', 'data': 'M', 'checksum': 77}
RECIEVER expecting seq_num: 0
RECIEVER reply with: {'ack': '0', 'checksum': 48}
SENDER recieved : {'ack': '0', 'checksum': 48}
Sender Done!
Receiver received: ['K', 'A', 'M']
```



```
C:\Users\Kamal Said\Network_python>python main.py msg=KAM rel=0.75 delay=0 debug=0
{'msg': 'KAM', 'rel': '0.75', 'delay': '0', 'debug': '0'}
Sender is sending:KAM
SENDER Expecting sequence number: 0
SENDER sending: {'sequence_number': '0', 'data': 'K', 'checksum': 75}
RECIEVER expecting seq_num: 0
RECIEVER reply with: {'ack': '0', 'checksum': 48}
SENDER recieved : {'ack': '0', 'checksum': 48}
SENDER Expecting sequence number: 1
SENDER sending: {'sequence_number': '1', 'data': 'A', 'checksum': 65}
RECIEVER expecting seq_num: 1
RECIEVER reply with: {'ack': '1', 'checksum': 49}
SENDER recieved : {'ack': '1', 'checksum': 49}
SENDER Expecting sequence number: 0
SENDER sending: {'sequence_number': '0', 'data': 'M', 'checksum': 77}
RECIEVER expecting seq_num: 0
RECIEVER reply with: {'ack': '0', 'checksum': 48}
SENDER recieved : {'ack': '0', 'checksum': 48}
network_layer:corruption occured {'ack': '0', 'checksum': '3'}
SENDER Expecting sequence number: 0
SENDER sending: {'sequence_number': '0', 'data': 'M', 'checksum': 77}
RECIEVER expecting seq_num: 1
network_layer:corruption occured {'sequence_number': '0', 'data': 'M', 'checksum': 77}
RECIEVER reply with: {'ack': '0', 'checksum': 48}
SENDER recieved : {'ack': '0', 'checksum': 48}
Sender Done!
Receiver received: ['K', 'A', 'M']
```



```
C:\Users\Kamal Said\Network_python> python main.py msg=KAM rel=0.5 delay=0 debug=0
{'msg': 'KAM', 'rel': '0.5', 'delay': '0', 'debug': '0'}
Sender is sending:KAM
SENDER Expecting sequence number: 0
SENDER sending: {'sequence_number': '0', 'data': 'K', 'checksum': 75}
RECIEVER expecting seq_num: 0
network_layer:corruption occured {'sequence_number': '5', 'data': 'K', 'checksum': 75}
RECIEVER reply with: {'ack': '1', 'checksum': 49}
SENDER recieved : {'ack': '1', 'checksum': 49}
network_layer:corruption occured {'ack': '1', 'checksum': '7'}
SENDER Expecting sequence number: 0
SENDER sending: {'sequence_number': '0', 'data': 'K', 'checksum': 75}
RECIEVER expecting seq_num: 0
RECIEVER reply with: {'ack': '0', 'checksum': 48}
SENDER recieved : {'ack': '0', 'checksum': 48}
SENDER Expecting sequence number: 1
SENDER sending: {'sequence_number': '1', 'data': 'A', 'checksum': 65}
RECIEVER expecting seq_num: 1
RECIEVER reply with: {'ack': '1', 'checksum': 49}
SENDER recieved : {'ack': '1', 'checksum': 49}
network_layer:corruption occured {'ack': '\t', 'checksum': 49}
SENDER Expecting sequence number: 1
SENDER sending: {'sequence_number': '1', 'data': 'A', 'checksum': 65}
RECIEVER expecting seq_num: 0
network_layer:corruption occured {'sequence_number': '1', 'data': 'A', 'checksum': 65}
RECIEVER reply with: {'ack': '1', 'checksum': 49}
SENDER recieved : {'ack': '1', 'checksum': 49}
SENDER Expecting sequence number: 0
SENDER sending: {'sequence_number': '0', 'data': 'M', 'checksum': 77}
RECIEVER expecting seq_num: 0
RECIEVER reply with: {'ack': '0', 'checksum': 48}
SENDER recieved : {'ack': '0', 'checksum': 48}
Sender Done!
Receiver received: ['K', 'A', 'M']
```

PSEUDOCODE:

RDT_SEND

For date inside the process buffer

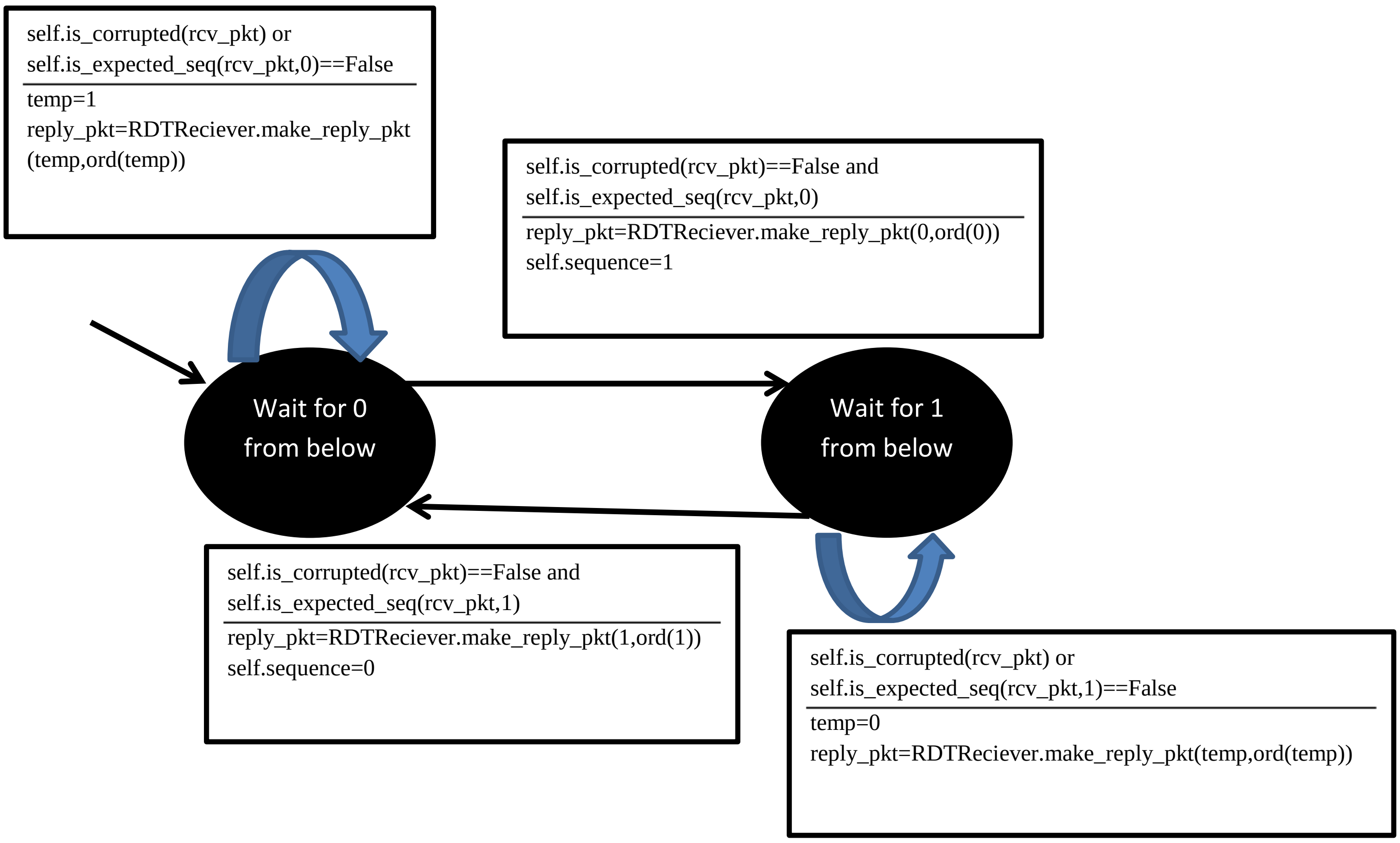
Checksum should be equal to the data of checksum result given by the method get_checksum implemented in skeleton code

Packet equals to the self sequence , data , and checksum given by make_pkt method implemented in skeleton code

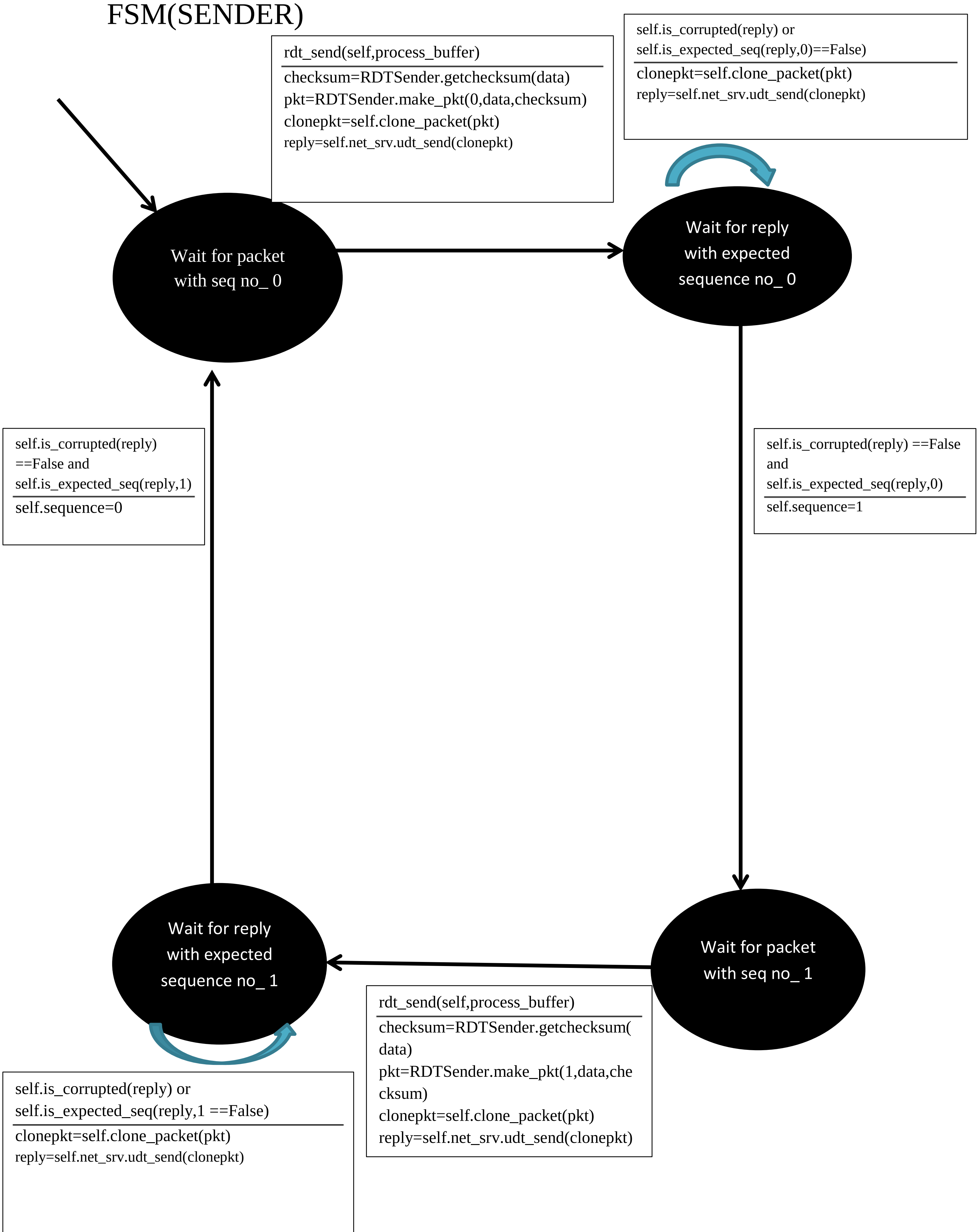
```
While (True)
        New variable clonepkt equals the packet result of method
clone_packet(pkt) implemented in skeleton code
        Print the line "sender expecting sequence number" + self sequence
        Print the line "sender sending" + clonepkt
        Variable reply equals to the cloned packet
         If the method is_corrupted (takes reply as a parameter) is TRUE
OR method is_excepted (takes reply and self sequence as
parameter)=FALSE
              Print network layer : corruption occurred
         Else
              Let self sequence = 0 when 1 or self sequence =1 when 0
               Break the loop
Print Sender Done


RDT_RCV
Print Reciever expecting seq_num:self.sequence
If method is_corrupted implemented in skeleton code equals TRUE OR
is_expected equals FALSE
   Print Network layer:corruption occurred:rcv_pkt
   Let temp=0
   If self sequence =0 let temp = 1
   Else
     Let temp=0
```

```
    Let variable reply_pkt= the packet made with method make_reply_pkt
with paramaters temp and the ascii code of temp.



Else
        Let variable reply_pkt= the packet made with method
        make_reply_pkt  with paramaters self.sequence and the ascii code of
        self.sequence.
        If self sequence equal 0 , let self sequence =1
        Else
                let self sequence =0
      Insert data of received packet in to the buffer



Print Reciever reply with: reply_pkt
Print Sender received : reply_pkt
Return ‘reply_pkt’
```

FSM(RECIEVER)


self.is_corrupted(rcv_pkt) or
self.is_expected_seq(rcv_pkt,0)==False
temp=1
reply_pkt=RDTReciever.make_reply_pkt
(temp,ord(temp))
self.is_corrupted(rcv_pkt)==False and
self.is_expected_seq(rcv_pkt,0)
reply_pkt=RDTReciever.make_reply_pkt(0,ord(0))
self.sequence=1
Wait for 0
from below
Wait for 1
from below
self.is_corrupted(rcv_pkt)==False and
self.is_expected_seq(rcv_pkt,1)
reply_pkt=RDTReciever.make_reply_pkt(1,ord(1))
self.sequence=0
self.is_corrupted(rcv_pkt) or
self.is_expected_seq(rcv_pkt,1)==False
temp=0
reply_pkt=RDTReciever.make_reply_pkt(temp,ord(temp))

FSM(SENDER)


Wait for packet with seq no_ 0
Wait for reply with expected sequence no_ 0
Wait for packet with seq no_ 1
Wait for reply with expected sequence no_ 1

```
rdt_send(self,process_buffer)
-----------------------------
checksum=RDTSender.getchecksum(data)
pkt=RDTSender.make_pkt(0,data,checksum)
clonepkt=self.clone_packet(pkt)
reply=self.net_srv.udt_send(clonepkt)
```

```
self.is_corrupted(reply) or
self.is_expected_seq(reply,0)==False)
-----------------------------
clonepkt=self.clone_packet(pkt)
reply=self.net_srv.udt_send(clonepkt)
```

```
self.is_corrupted(reply) ==False
and
self.is_expected_seq(reply,0)
-----------------------------
self.sequence=1
```

```
rdt_send(self,process_buffer)
-----------------------------
checksum=RDTSender.getchecksum(
data)
pkt=RDTSender.make_pkt(1,data,che
cksum)
clonepkt=self.clone_packet(pkt)
reply=self.net_srv.udt_send(clonepkt)
```

```
self.is_corrupted(reply) or
self.is_expected_seq(reply,1 ==False)
-----------------------------
clonepkt=self.clone_packet(pkt)
reply=self.net_srv.udt_send(clonepkt)
```

```
self.is_corrupted(reply)
==False and
self.is_expected_seq(reply,1)
-----------------------------
self.sequence=0
```