Adam Liu

Professor Ryan Stutsman

Computer Networks

PA2 Report

DESIGN

Entity A:

Entity A initializes with instance variables including a packet sequence number, next sequence number, acknowledgment number, a packet queue, and a flight boolean. Entity A's output method creates a new packet with the passed in message data. Doing so the checksum is computed by the sum of all variables. The sequence number and ack reflect each other. The packet is then stored in a queue, if the flight variable has not been set to true, then no packets are being sent over the network so we transmit. The main advantage being denoted here is using a queue to store all the packets regardless of if they are in flight or not. This allows for easy retransmission of the packets which maintains the order they should be sent according to the call of the output method.

To transmit, a transmit method is called which will send the packet to Entity B, set the flight variable to true and then start a timeout timer. The timeout is calculated by the total RTT with an additional latency to account for simulated congestion / routing decisions. The acceptable timeout for my program worked out to be 18 units of time. If a timeout occurs, then the timeout method is called and the transmit method immediately gets called again to retransmit the lost/timed out packet. Abstracting away the transmit method is advantageous to reduce the redundancy of code where you may make mistakes in your checks and setting the flight variable.

Upon arrival of a packet to entity A's input method the timer is stopped, and the checksum is recomputed using the intended sequence and acknowledgement and payload. This is then compared to the checksum of the packet to detect corruption. Additionally, the sequence number and acknowledgement number are compared with the packet to the last item in the queue which will represent the intended packet variables to be acknowledged. If these checks correspond properly then the packet is successfully sent to layer 5 from B and we increase the next sequence and acknowledgment number, set the in-flight variable to false, pop the item on the queue, and transmit the next intended packet in the queue. Otherwise the packet is corrupted and the previous packet will be retransmitted. The main tradeoff here in the input method is acknowledging the packet that B will receive and send off to the next layer. This simplifies the logic so that the sequence number and acknowledgement number will always reflect each other instead of acknowledging the next packet to transmit.

Entity B:

Entity B starts with the intended sequence and acknowledgement number which will always reflect each other. Receiving a packet it will use these to compute the correct checksum. This is advantageous because it will always reflect the correct checksum since we will account for every intended packet that must come through B. If all values are proper then we send that message to layer 5 and send the same packet with the correct non corrupted values back to A. Otherwise a new packet with the precious sequence number, acknowledgement number and proper checksum will be created and sent back to A indicating that packet needs to be retransmitted.

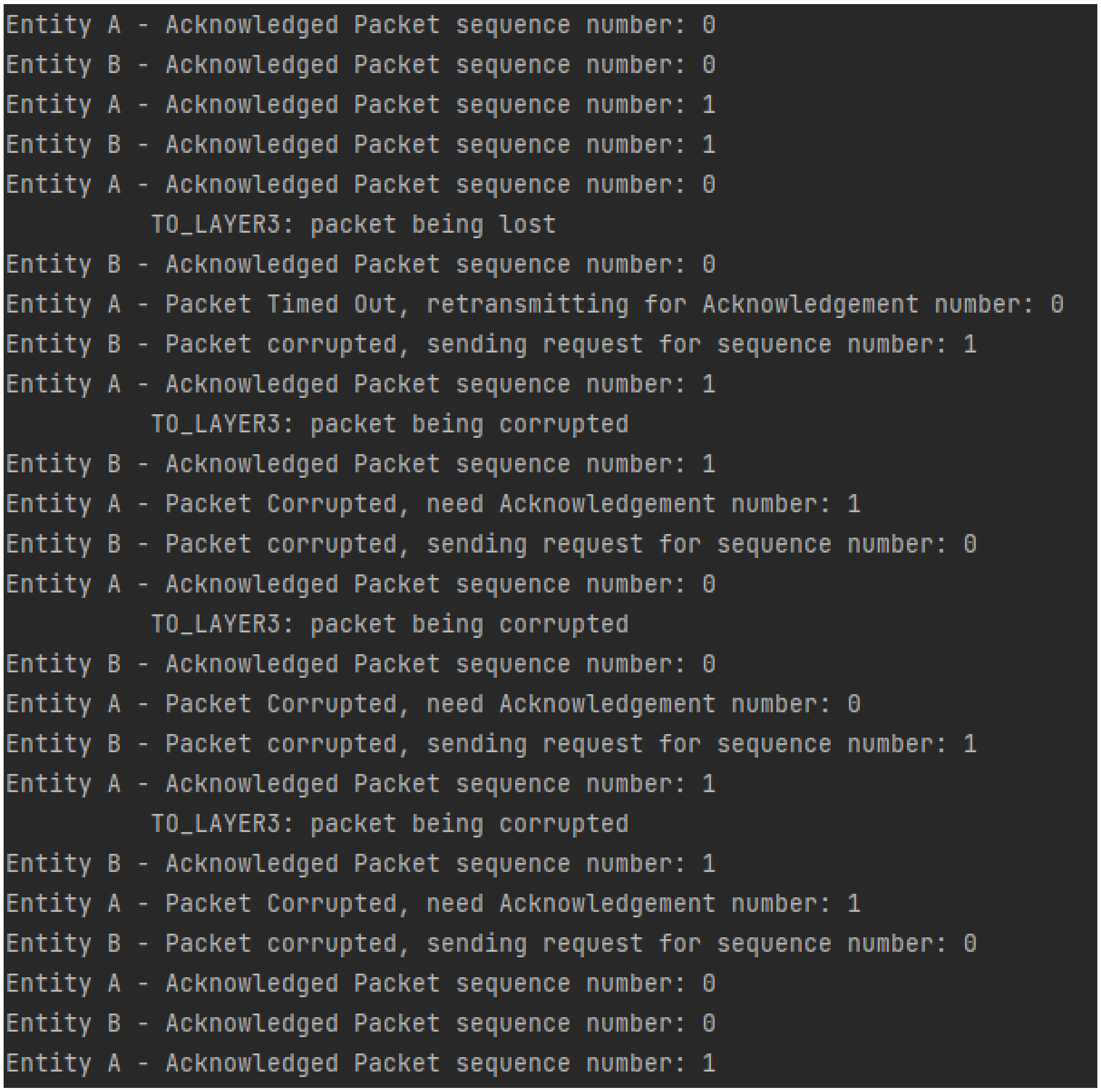
TESTING

Testing this file was done comprehensively through gradescope in which I passed all tests including some of my own which I will further discuss. To begin, testing the acknowledgement and sequence number checks in both Entity A and Entity B were done with no corruption and no packet lotto ensure that we will only have to first see if the program will follow the natural order which it has indeed done. Moving onto the packet loss, I made sure to note that even if a packet times out, we will still always check for the intended packet. One thing I found in my code that was done well is that I compute the expected acknowledgement and sequence number and the previous acknowledgement and sequence number to ensure if a packet is lost and in the case that a bit gets flipped on a preceding packet, that will still will not accept any packets that come in out of order. The check was done with (if self.acknum == packet.acknum and self.next_seqnum == packet.seqnum and self.q[0].checksum == checksum:) which led to a valid conclusion every time. The case in which I was not confident about my tests passing is when I set my timeout to a lower or higher timeout. My timeout seemed to work perfectly at the 18 units of time mark, however I tested all the way down to 12 which was much too infrequent and would often cause issues with the previous packet being stored in the queue for too short periods of time. Although, if the total timeout was much too long then we would be allowing for a much larger latency which ultimately made the tests report that even fewer packets were being delivered by B. View the next consecutive pages for a comprehensive analysis on the program output and the code liable for test completion.

OUTPUT

SIMULATION OUTPUT: python rdtsim.py -n 100 -d 100 -z 2 -l 0.05 -c 0.05 -s 686868686 -v 2

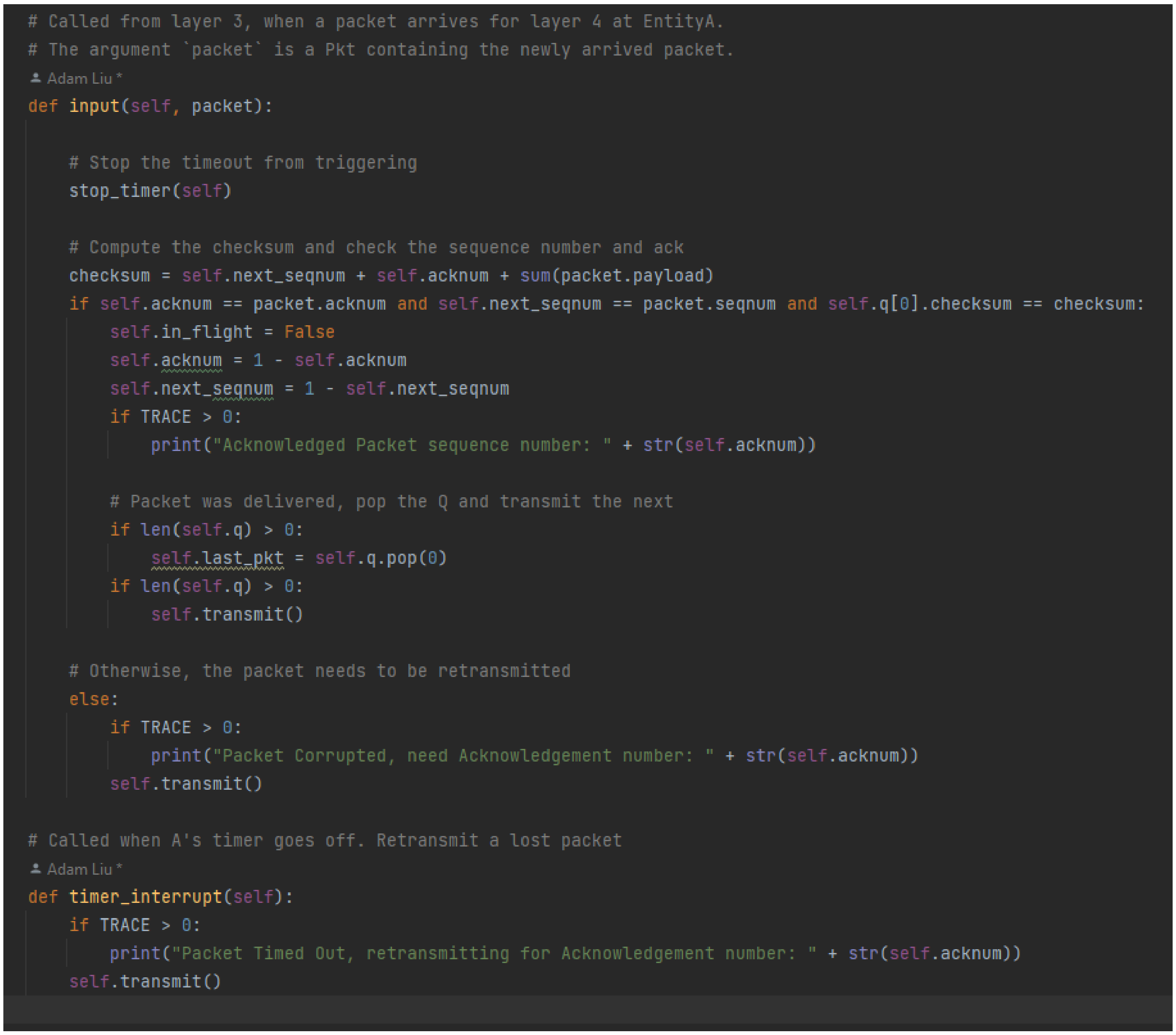
Below shows a few more than 10 correctly Acknowledged Packets after being either corrupted or lost.



```
Entity A - Acknowledged Packet sequence number: 0
Entity B - Acknowledged Packet sequence number: 0
Entity A - Acknowledged Packet sequence number: 1
Entity B - Acknowledged Packet sequence number: 1
Entity A - Acknowledged Packet sequence number: 0
         TO_LAYER3: packet being lost
Entity B - Acknowledged Packet sequence number: 0
Entity A - Packet Timed Out, retransmitting for Acknowledgement number: 0
Entity B - Packet corrupted, sending request for sequence number: 1
Entity A - Acknowledged Packet sequence number: 1
         TO_LAYER3: packet being corrupted
Entity B - Acknowledged Packet sequence number: 1
Entity A - Packet Corrupted, need Acknowledgement number: 1
Entity B - Packet corrupted, sending request for sequence number: 0
Entity A - Acknowledged Packet sequence number: 0
         TO_LAYER3: packet being corrupted
Entity B - Acknowledged Packet sequence number: 0
Entity A - Packet Corrupted, need Acknowledgement number: 0
Entity B - Packet corrupted, sending request for sequence number: 1
Entity A - Acknowledged Packet sequence number: 1
         TO_LAYER3: packet being corrupted
Entity B - Acknowledged Packet sequence number: 1
Entity A - Packet Corrupted, need Acknowledgement number: 1
Entity B - Packet corrupted, sending request for sequence number: 0
Entity A - Acknowledged Packet sequence number: 0
Entity B - Acknowledged Packet sequence number: 0
Entity A - Acknowledged Packet sequence number: 1
```

We see that when the packet will either get corrupted or timeout, my program will handle this by checking the checksum, sequence and acknowledgement numbers and retransmit if needed. Below shows the simple print statements that output the result above.

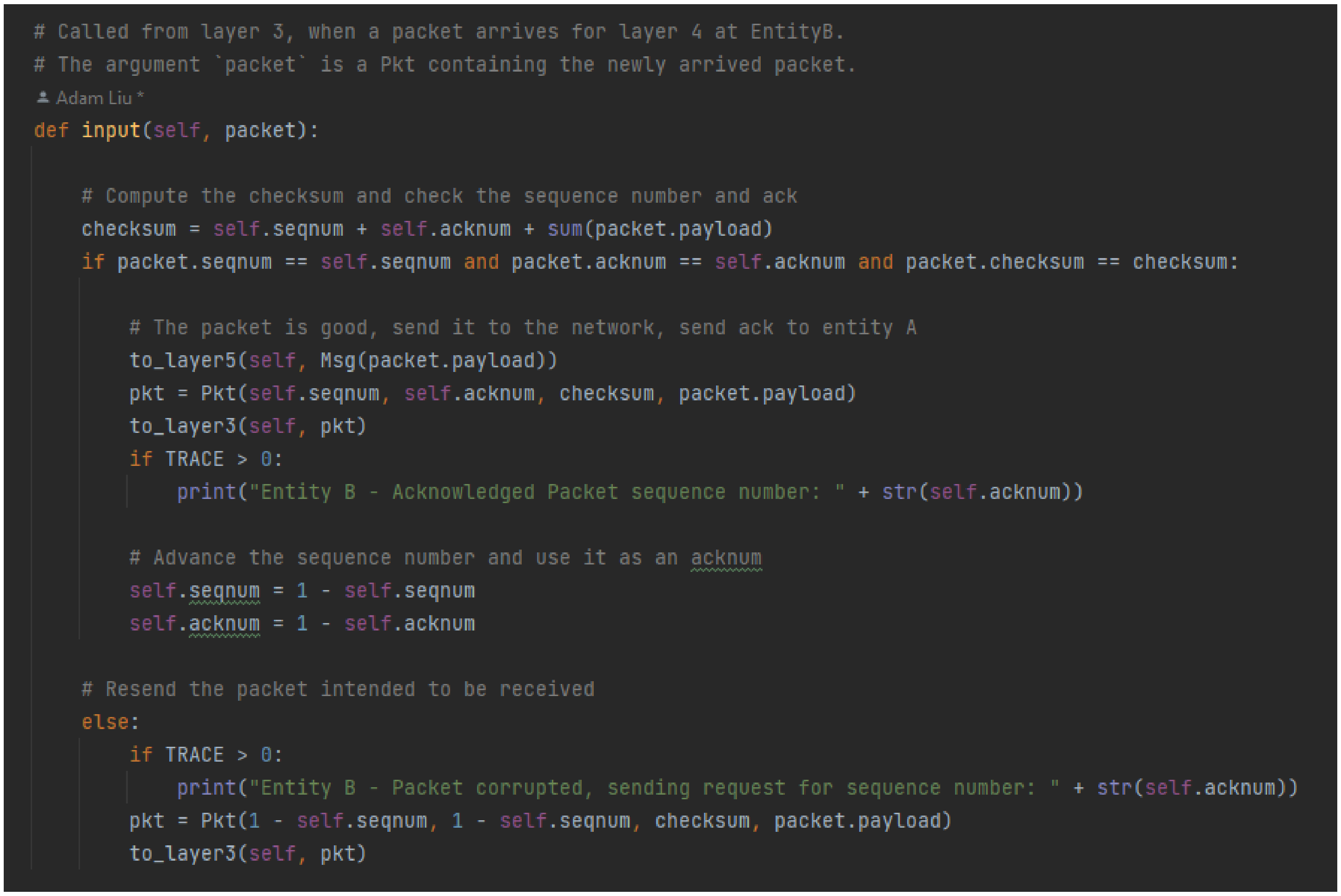
Entity A:



```
# Called from layer 3, when a packet arrives for layer 4 at EntityA.
# The argument `packet` is a Pkt containing the newly arrived packet.
Adam Liu *
def input(self, packet):

    # Stop the timeout from triggering
    stop_timer(self)

    # Compute the checksum and check the sequence number and ack
    checksum = self.next_seqnum + self.acknum + sum(packet.payload)
    if self.acknum == packet.acknum and self.next_seqnum == packet.seqnum and self.q[0].checksum == checksum:
        self.in_flight = False
        self.acknum = 1 - self.acknum
        self.next_seqnum = 1 - self.next_seqnum
        if TRACE > 0:
            print("Acknowledged Packet sequence number: " + str(self.acknum))

        # Packet was delivered, pop the Q and transmit the next
        if len(self.q) > 0:
            self.last_pkt = self.q.pop(0)
        if len(self.q) > 0:
            self.transmit()

    # Otherwise, the packet needs to be retransmitted
    else:
        if TRACE > 0:
            print("Packet Corrupted, need Acknowledgement number: " + str(self.acknum))
        self.transmit()

# Called when A's timer goes off. Retransmit a lost packet
Adam Liu *
def timer_interrupt(self):
    if TRACE > 0:
        print("Packet Timed Out, retransmitting for Acknowledgement number: " + str(self.acknum))
    self.transmit()
```

Entity A uses (if self.acknum == packet.acknum and self.next_seqnum == packet.seqnum and self.q[0].checksum == checksum:) to determine id the correct sequence number and acknowledgment number are being sent from the intended packets entity.

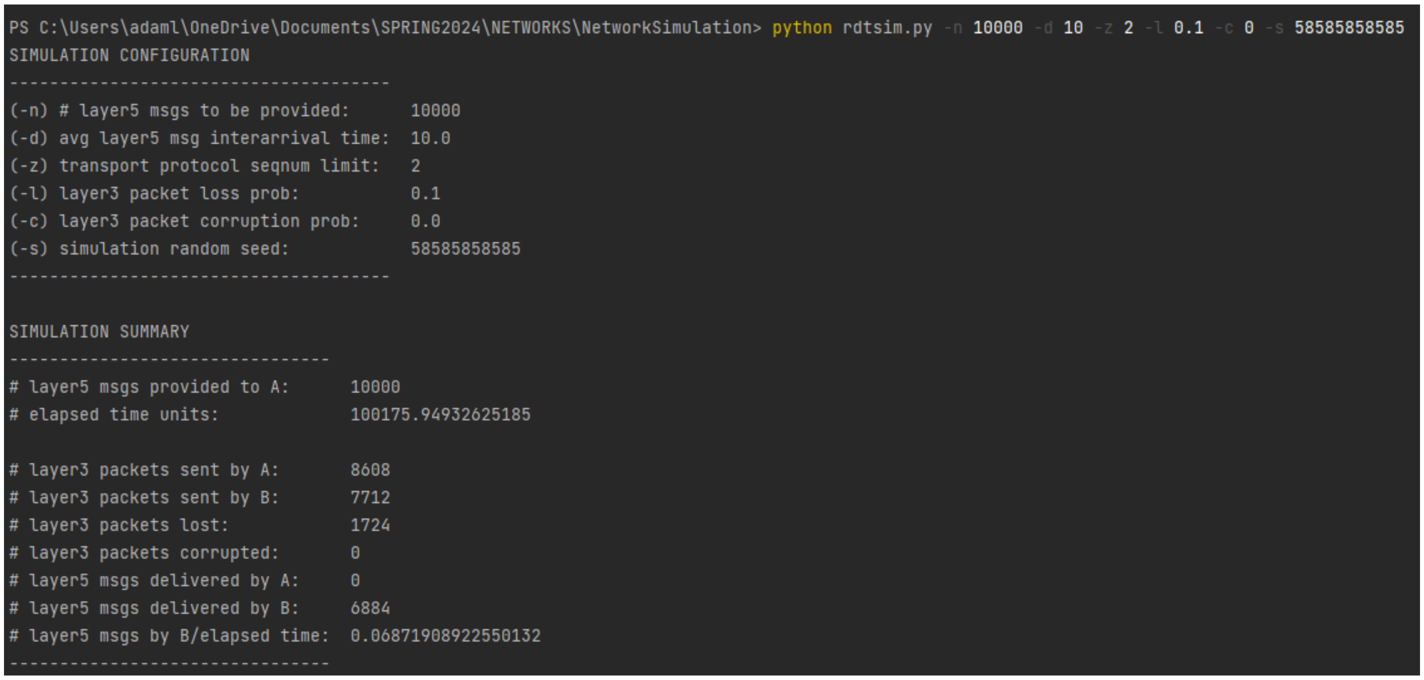
Entity B:



```
# Called from layer 3, when a packet arrives for layer 4 at EntityB.
# The argument `packet` is a Pkt containing the newly arrived packet.
Adam Liu *
def input(self, packet):

    # Compute the checksum and check the sequence number and ack
    checksum = self.seqnum + self.acknum + sum(packet.payload)
    if packet.seqnum == self.seqnum and packet.acknum == self.acknum and packet.checksum == checksum:

        # The packet is good, send it to the network, send ack to entity A
        to_layer5(self, Msg(packet.payload))
        pkt = Pkt(self.seqnum, self.acknum, checksum, packet.payload)
        to_layer3(self, pkt)
        if TRACE > 0:
            print("Entity B - Acknowledged Packet sequence number: " + str(self.acknum))

        # Advance the sequence number and use it as an acknum
        self.seqnum = 1 - self.seqnum
        self.acknum = 1 - self.acknum

    # Resend the packet intended to be received
    else:
        if TRACE > 0:
            print("Entity B - Packet corrupted, sending request for sequence number: " + str(self.acknum))
        pkt = Pkt(1 - self.seqnum, 1 - self.seqnum, checksum, packet.payload)
        to_layer3(self, pkt)
```

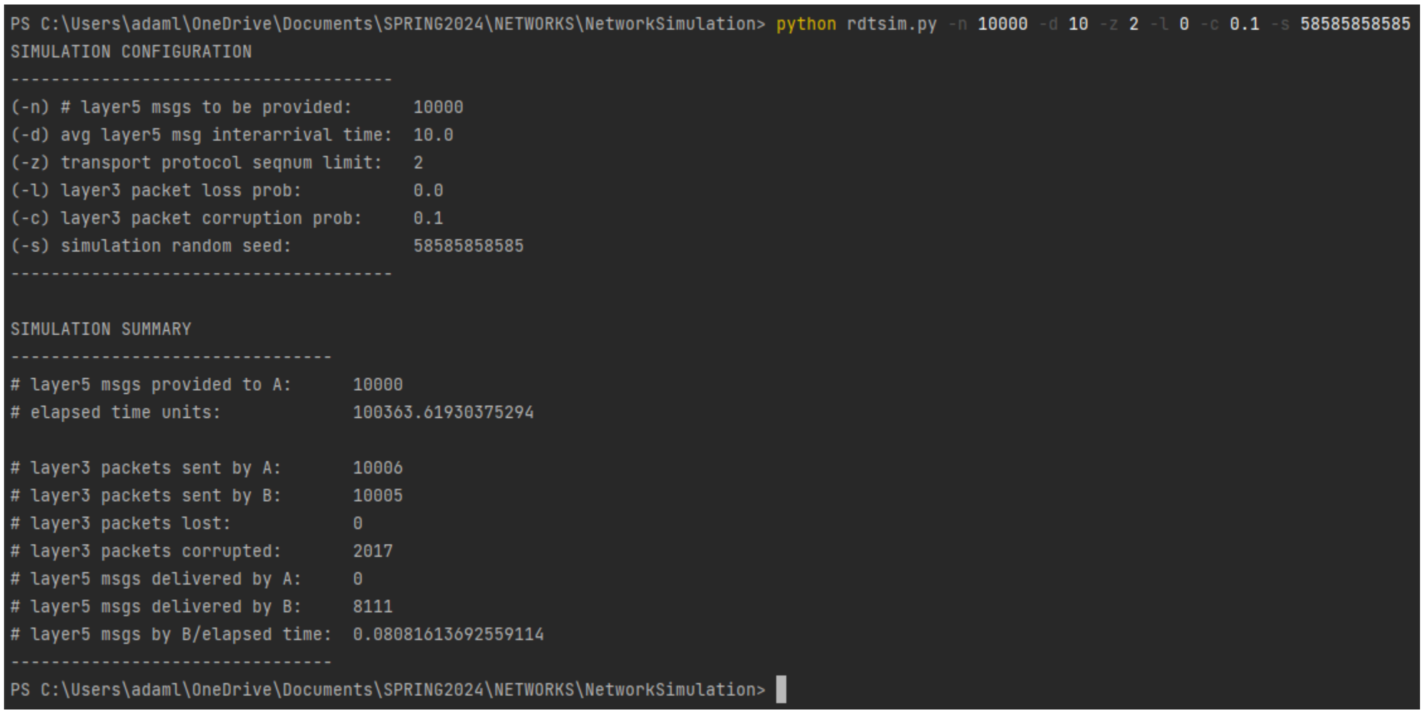
Similarly, Entity B uses (if self.acknum == packet.acknum and self.next_seqnum == packet.seqnum and self.q[0].checksum == checksum:) to determine id the correct sequence number and acknowledgment number are being sent from the intended packets entity. Otherwise it will send the intended packet with the previous acknowledgment number back insinuating the next one needs to be sent again.

10% Loss, No Corruption



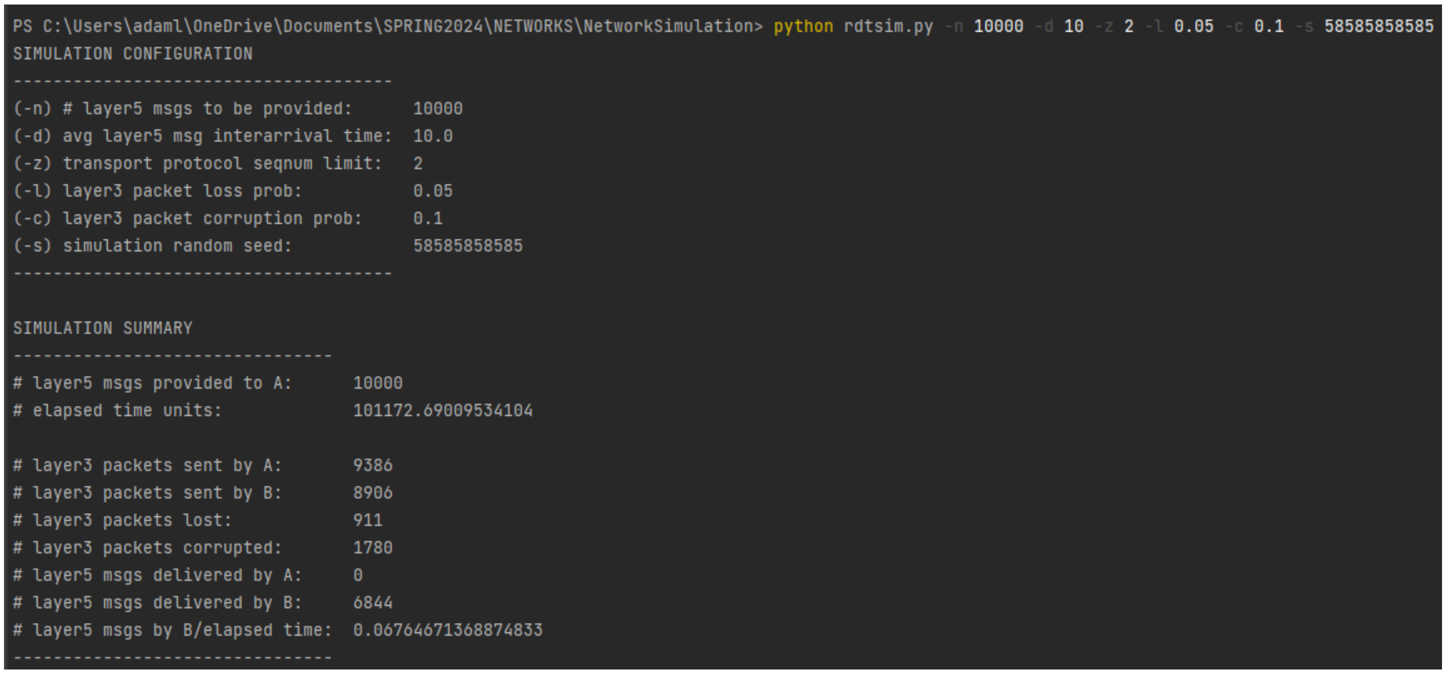
```
PS C:\Users\adaml\OneDrive\Documents\SPRING2024\NETWORKS\NetworkSimulation> python rdtsim.py -n 10000 -d 10 -z 2 -l 0.1 -c 0 -s 58585858585
SIMULATION CONFIGURATION
--------------------------------------
(-n) # layer5 msgs to be provided:      10000
(-d) avg layer5 msg interarrival time:  10.0
(-z) transport protocol seqnum limit:   2
(-l) layer3 packet loss prob:           0.1
(-c) layer3 packet corruption prob:     0.0
(-s) simulation random seed:            58585858585
--------------------------------------

SIMULATION SUMMARY
--------------------------------
# layer5 msgs provided to A:      10000
# elapsed time units:             100175.94932625185

# layer3 packets sent by A:       8608
# layer3 packets sent by B:       7712
# layer3 packets lost:            1724
# layer3 packets corrupted:       0
# layer5 msgs delivered by A:     0
# layer5 msgs delivered by B:     6884
# layer5 msgs by B/elapsed time:  0.06871908922550132
--------------------------------
```

No Loss, 10% Corruption



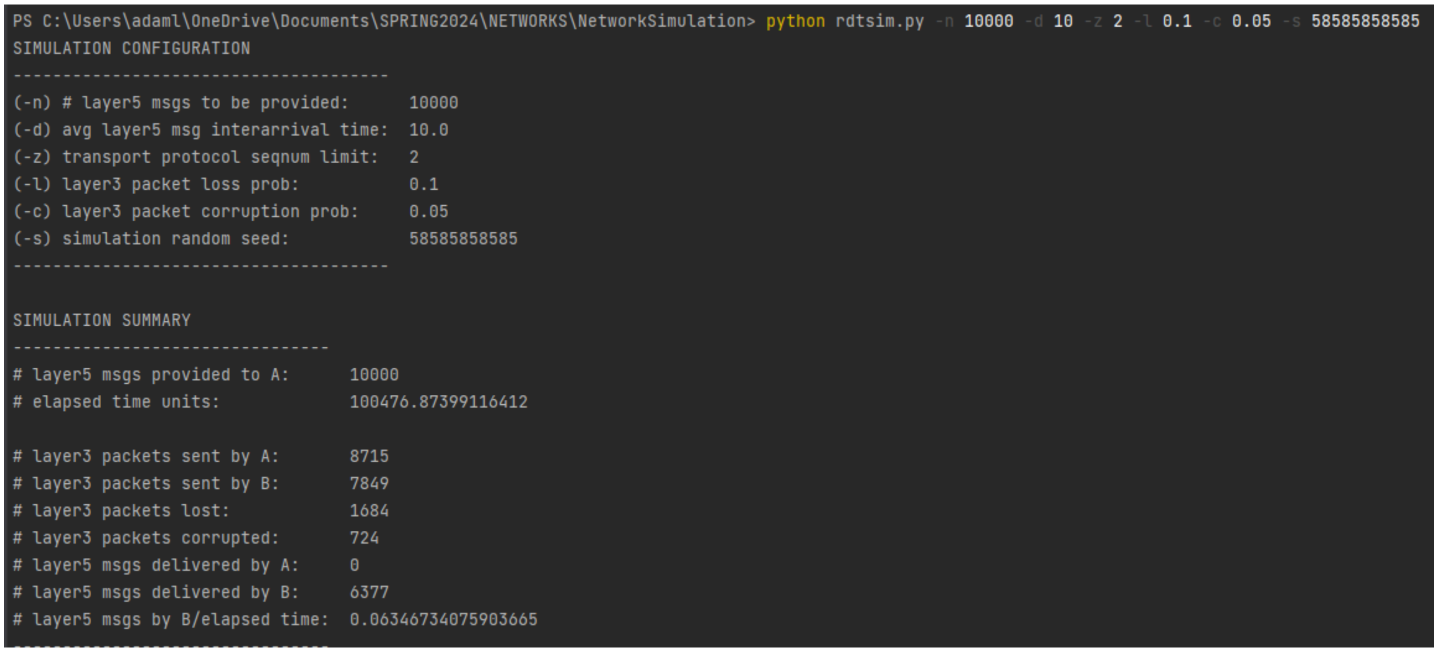
```
PS C:\Users\adaml\OneDrive\Documents\SPRING2024\NETWORKS\NetworkSimulation> python rdtsim.py -n 10000 -d 10 -z 2 -l 0 -c 0.1 -s 58585858585
SIMULATION CONFIGURATION
--------------------------------------
(-n) # layer5 msgs to be provided:      10000
(-d) avg layer5 msg interarrival time:  10.0
(-z) transport protocol seqnum limit:   2
(-l) layer3 packet loss prob:           0.0
(-c) layer3 packet corruption prob:     0.1
(-s) simulation random seed:            58585858585
--------------------------------------

SIMULATION SUMMARY
--------------------------------
# layer5 msgs provided to A:      10000
# elapsed time units:             100363.61930375294

# layer3 packets sent by A:       10006
# layer3 packets sent by B:       10005
# layer3 packets lost:            0
# layer3 packets corrupted:       2017
# layer5 msgs delivered by A:     0
# layer5 msgs delivered by B:     8111
# layer5 msgs by B/elapsed time:  0.08081613692559114
--------------------------------
PS C:\Users\adaml\OneDrive\Documents\SPRING2024\NETWORKS\NetworkSimulation>
```

5% Loss, 10% Corruption



```
PS C:\Users\adaml\OneDrive\Documents\SPRING2024\NETWORKS\NetworkSimulation> python rdtsim.py -n 10000 -d 10 -z 2 -l 0.05 -c 0.1 -s 58585858585
SIMULATION CONFIGURATION
--------------------------------------
(-n) # layer5 msgs to be provided:      10000
(-d) avg layer5 msg interarrival time:  10.0
(-z) transport protocol seqnum limit:   2
(-l) layer3 packet loss prob:           0.05
(-c) layer3 packet corruption prob:     0.1
(-s) simulation random seed:            58585858585
--------------------------------------

SIMULATION SUMMARY
--------------------------------
# layer5 msgs provided to A:      10000
# elapsed time units:             101172.69009534104

# layer3 packets sent by A:       9386
# layer3 packets sent by B:       8906
# layer3 packets lost:            911
# layer3 packets corrupted:       1780
# layer5 msgs delivered by A:     0
# layer5 msgs delivered by B:     6844
# layer5 msgs by B/elapsed time:  0.06764671368874833
--------------------------------
```

10% Loss, 5% Corruption



```
PS C:\Users\adaml\OneDrive\Documents\SPRING2024\NETWORKS\NetworkSimulation> python rdtsim.py -n 10000 -d 10 -z 2 -l 0.1 -c 0.05 -s 58585858585
SIMULATION CONFIGURATION
--------------------------------------
(-n) # layer5 msgs to be provided:      10000
(-d) avg layer5 msg interarrival time:  10.0
(-z) transport protocol seqnum limit:   2
(-l) layer3 packet loss prob:           0.1
(-c) layer3 packet corruption prob:     0.05
(-s) simulation random seed:            58585858585
--------------------------------------

SIMULATION SUMMARY
--------------------------------
# layer5 msgs provided to A:      10000
# elapsed time units:             100476.87399116412

# layer3 packets sent by A:       8715
# layer3 packets sent by B:       7849
# layer3 packets lost:            1684
# layer3 packets corrupted:       724
# layer5 msgs delivered by A:     0
# layer5 msgs delivered by B:     6377
# layer5 msgs by B/elapsed time:  0.06346734075903665
--------------------------------
```

In review, After showing these statistics and analyzing the tests done manually and through gradescope. We can see that this RDT simulation does indeed recover packets from being lost and or corrupted. Delivering it to layer 5 from B is successful with most packets being delivered to the intended destination.