Fakharyar Khan
March 24th, 2023
Communication Networks
Professor Keene

Project #2: Reliable Data Transfer

Design Document

To implement rdt 3.0, I first gave the entity A class four attributes: seqnum_limit, the maximum number of distinct sequence numbers that can be used; ACK, this keeps track of which ACK the sender should look out for; seqnum, the sequence number of the current packet; and finally prevPack which holds the previous packet that was sent in case it needs to be retransmitted. Since I didn't implement Go-Back-N, the sequence number wasn't needed in my code but I did use it to generate the sample output and was very helpful for debugging.

In the output function of the entity A class, I first generated a packet with the current sequence number and the current ack. For the checksum, I took the data attribute of the message object received from layer 5 and hashed it and then also passed the data attribute as well. Once this was sent to layer 3, I set prevPack to the packet that was just sent and started the timer for a 100 time units. For the input function, I checked to see if the packet wasn't corrupted by seeing if the hash of the payload of the message sent by entityB was equal to its checksum and also if the receiver was acknowledging the packet that I had just sent. If both of these are true then the packet has been successfully sent and we move to the next packet by updating our ACK and sequence number and stopping the timer. If one of these conditions is false, then the sender will wait until a timeout occurs in which case the timer_interrupt function gets called which resends prevPacket and restarts the timer for 100 seconds.

Figure 1: Attributes and Methods Used for Implementing EntityA Class

For implementing the EntityB class, I only needed three attributes: the ACK number, the maximum sequence number, and the current sequence number. We don't need to keep track of the previous packet received because we never need to send it to the sender and once we send it to layer 5 we don't care about that

packet anymore. For the input function, we check if the packet sent hasn't been corrupted by comparing the checksum and the hash of the message body and also checking to see if the sender isn't sending a packet that we have already seen. If none of these occur then we update the sequence and ACK number and send out an ACK to layer 3. Since we didn't really talk about what goes in the payload of an ACK, I decided to just put in a message saying that the packet has been received. To implement the alternating bits protocol, the receiver didn't need to have a timer so I decided to leave the timer_interrupt method empty.

```
class EntityB:
    # The following method will be called once (only) before any other
    # EntityB methods are called.  You can use it to do any initialization.
    #
    # See comment above `EntityA.__init__` for the meaning of seqnum_limit.
    def __init__(self, seqnum_limit):

        self.seqnum_limit = seqnum_limit
        self.ACK = 0
        self.seqnum = 0
        #pass

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

        if packet.acknum == self.ACK and hash(packet.payload) == packet.checksum:
            print("------------RECEIVER-----------")
            print("RECEIVED PACKET " + str(self.seqnum) + " " + str(round(get_time(self))))
            to_layer5(self, Msg(packet.payload))

            print("SENDING OUT ACK " + str(self.ACK) + " FOR PACKET " + str(self.seqnum) + "\n")

            self.seqnum = (self.seqnum + 1) % self.seqnum_limit

            acknowledgement = Pkt(self.seqnum, self.ACK, hash(str.encode("RECEIVED! RECEIVED! ")), str.encode("RECEIVED! RECEIVED! "))

            to_layer3(self, acknowledgement)


            self.ACK = (self.ACK + 1) % 2


        else:
            print("------------RECEIVER-----------")
            print("PACKET " + str(self.seqnum) + " WAS EITHER CORRUPTED OR DUPLICATE PACKET" + " " + str(round(get_time(self))))
            print("RESENDING ACK " + str((self.ACK + 1) % 2) + "\n")
            acknowledgement = Pkt(self.seqnum, (self.ACK + 1) % 2, hash(str.encode("CRPT MSG | DUPLICATE")), str.encode("CRPT MSG | DUPLICATE"))

            to_layer3(self, acknowledgement)


    # Called when B's timer goes off.
    def timer_interrupt(self):
        pass
```

Figure 2: Attributes and Methods Used for Implementation of EntityB Class

Sample Output

To verify that my code was working properly I added print statements in all of my functions that would let me know the current step of the protocol that the program was in. As can be seen below, the sample output is a series of message sent by both the receiver and sender that say things like whether a packet or ACK was sent, if it was corrupted, and if their timer went off. Additionally, I used the get_time function to add a time stamp to each message which would make it easier for me to follow along with the trace of the code. The simulation that generated the sample output seen below used a packet loss probability of 10% and a packet corruption probability of 30%. So to go over a portion of the sample output, we see that the sender didn't receive the correct ACK for packet 2 and as a result waited until the timer went off at which point it resent packet 2. The receiver is then able to obtain packet 2 and it sends out an ACK for that packet. The sender sends out packet 3 and the receiver finds that the packet had been corrupted (it can't be a duplicate ACK since the sender had never sent packet 3 before) so it resends ACK 0. The sender being in state 1 ignores this ACK until timeout occurs and then resends packet 3 which comes to the receiver uncorrupted. Unfortunately, the sender wasn't able to get the ACK for packet 3 and so it sends packet 3 for a third time. But the receiver is waiting for packet 4 and so it sees packet 3 and resends ACK 1 which will show the sender that

packet 3 had been sent. Packet 4 gets lost though so the sender sends it again after timeout and the receiver is finally able to get it and sends the ACK for that packet. Overall, 10 messages were sent to entity A from layer 5 and it took 22 packets from entity A and 16 from B to deliver those 10 messages.

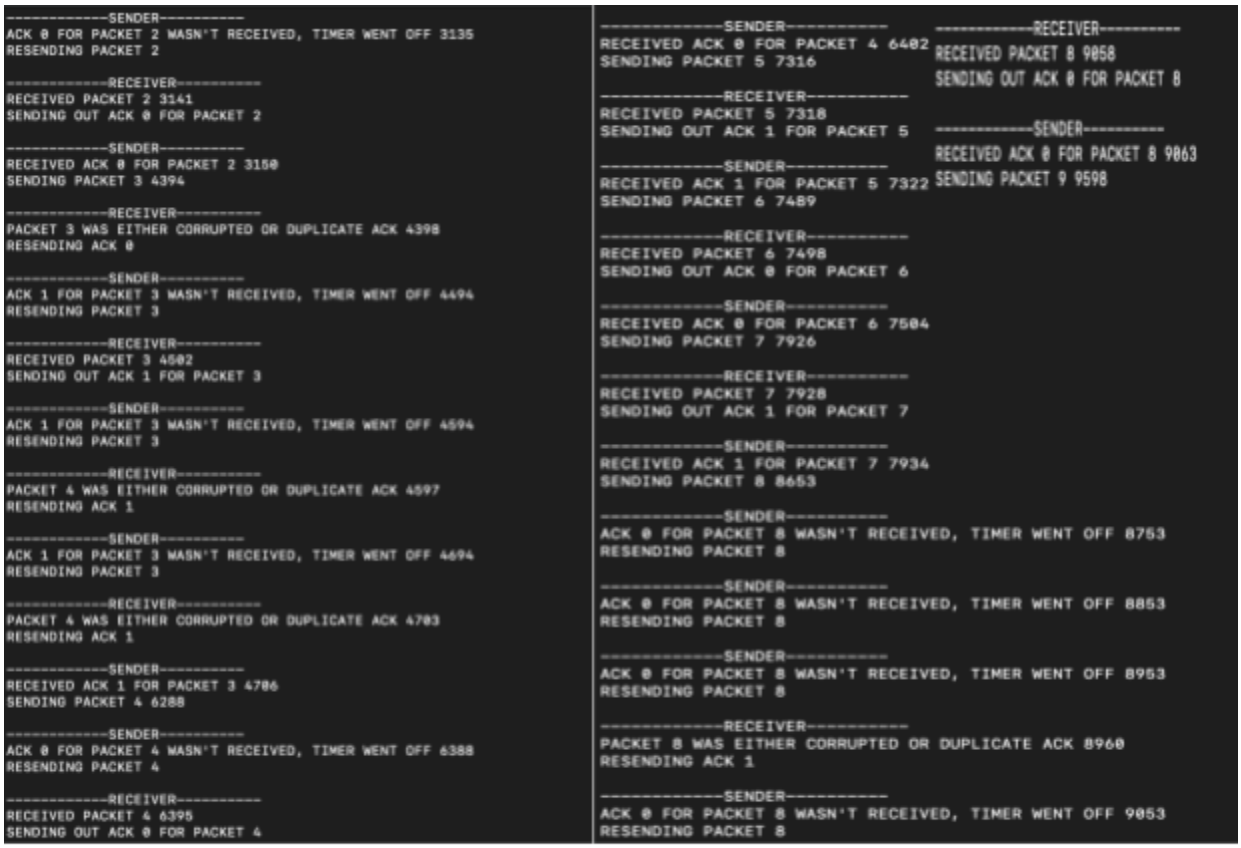

Figure 3: Sample Output of RDT 3.0 Implementation