NAME - GAUTAM KUMAR MAHAR(2103114) AND AUMKAR LOREKAR (2003108)

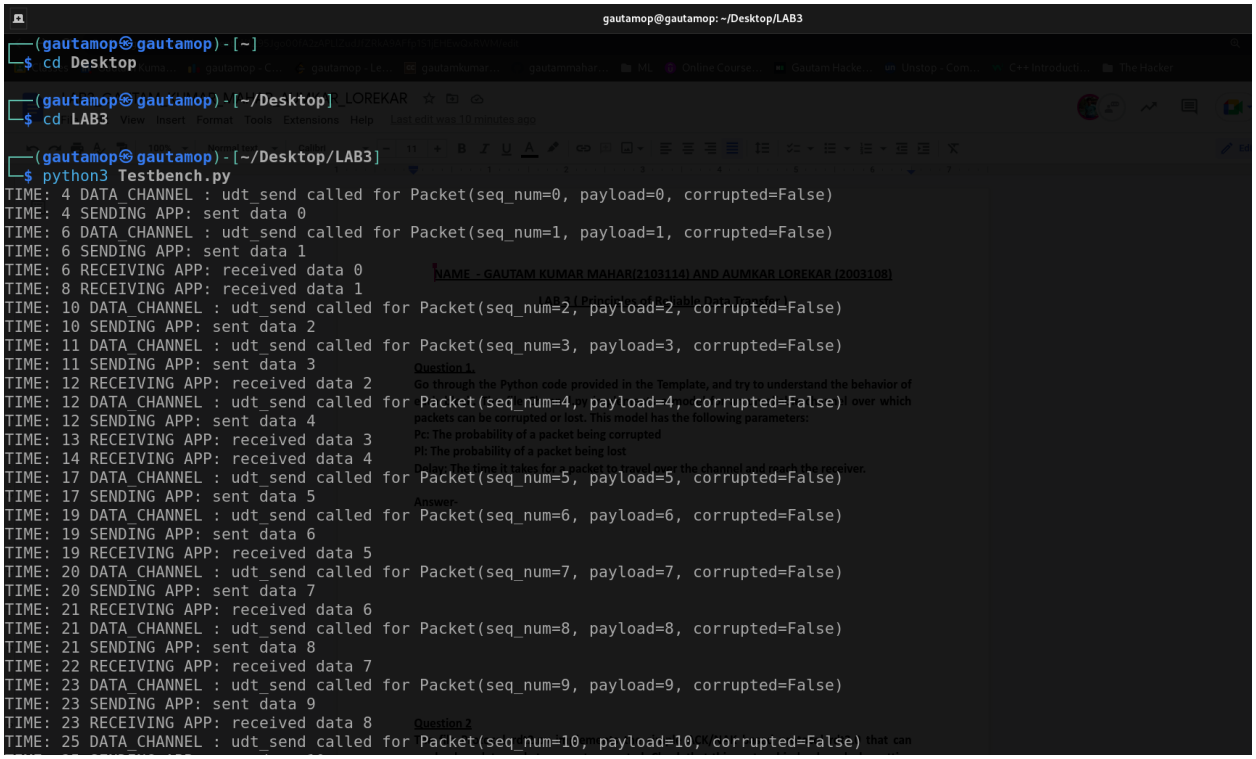
LAB 3 (Principles of Reliable Data Transfer)

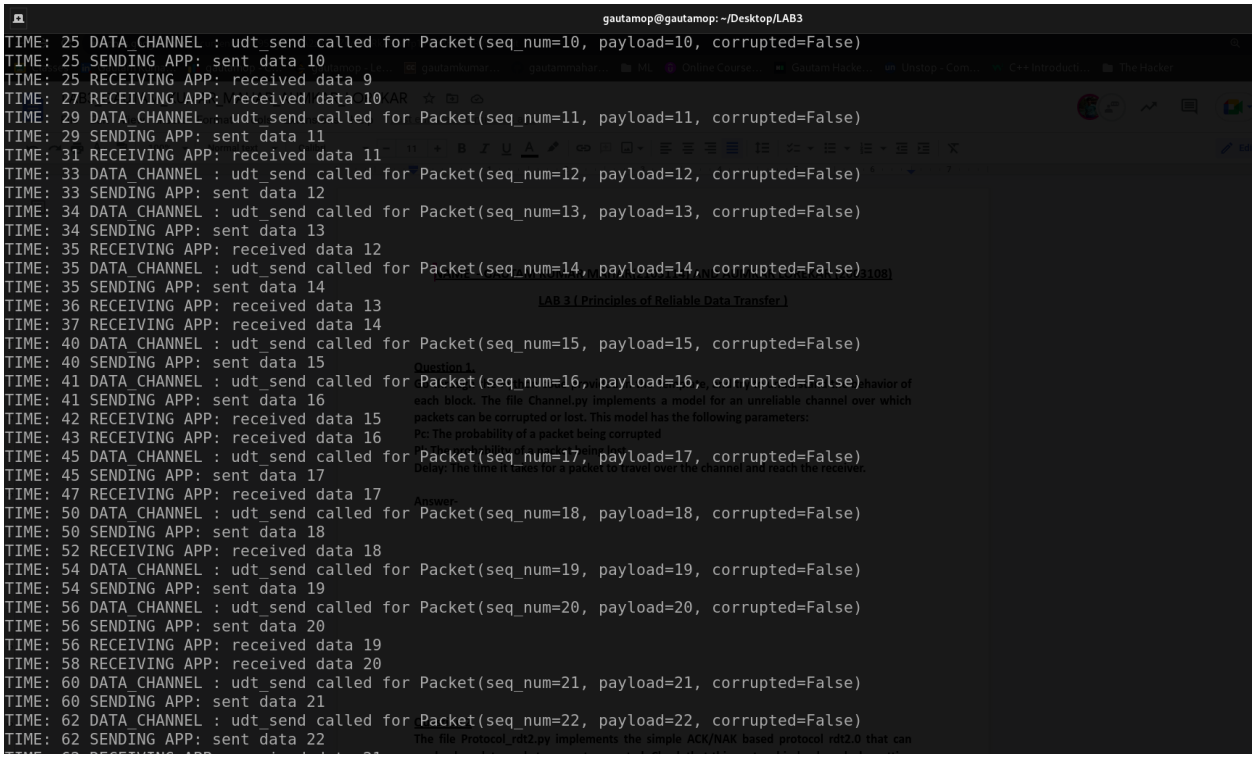
Question 1.

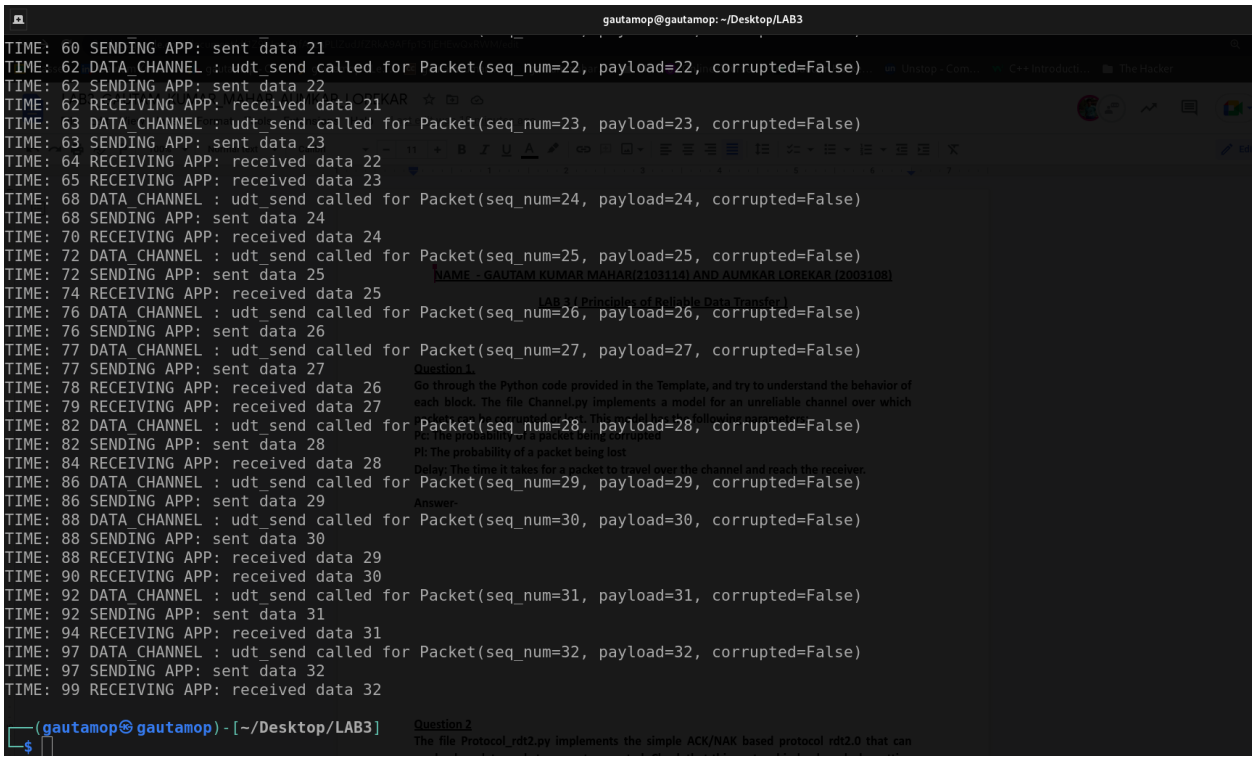
Go through the Python code provided in the Template, and try to understand the behavior of each block. The file Channel.py implements a model for an unreliable channel over which packets can be corrupted or lost. This model has the following parameters:
Pc: The probability of a packet being corrupted
Pl: The probability of a packet being lost
Delay: The time it takes for a packet to travel over the channel and reach the receiver.

Answer-

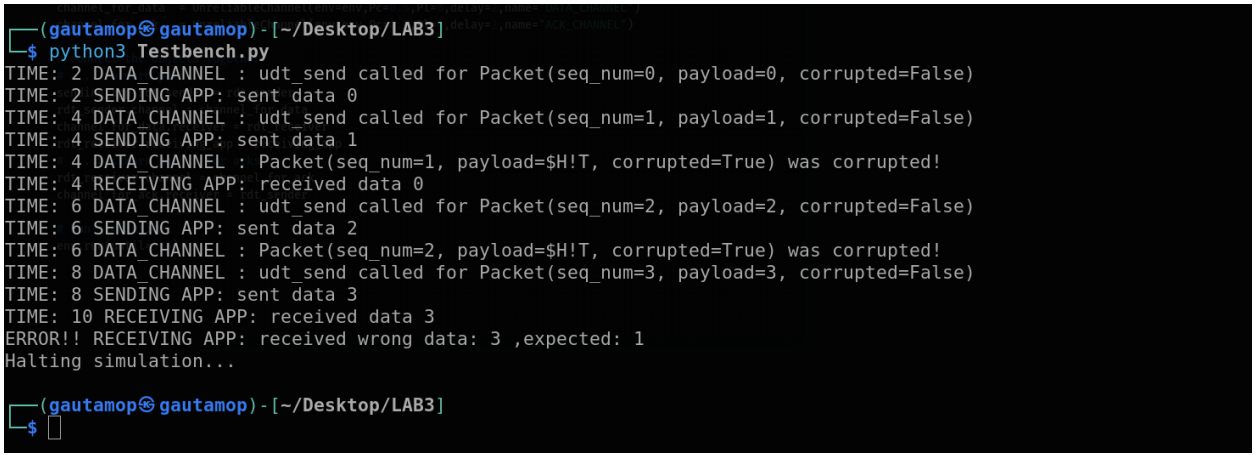
When Pc=0, rdt_1.0 protocol works perfectly fine.




gautamop@gautamop: ~/Desktop/LAB3
TIME: 25 DATA_CHANNEL : udt_send called for Packet(seq_num=10, payload=10, corrupted=False)
TIME: 25 SENDING APP: sent data 10
TIME: 25 RECEIVING APP: received data 9
TIME: 27 RECEIVING APP: received data 10
TIME: 29 DATA_CHANNEL : udt_send called for Packet(seq_num=11, payload=11, corrupted=False)
TIME: 29 SENDING APP: sent data 11
TIME: 31 RECEIVING APP: received data 11
TIME: 33 DATA_CHANNEL : udt_send called for Packet(seq_num=12, payload=12, corrupted=False)
TIME: 33 SENDING APP: sent data 12
TIME: 34 DATA_CHANNEL : udt_send called for Packet(seq_num=13, payload=13, corrupted=False)
TIME: 34 SENDING APP: sent data 13
TIME: 35 RECEIVING APP: received data 12
TIME: 35 DATA_CHANNEL : udt_send called for Packet(seq_num=14, payload=14, corrupted=False)
TIME: 35 SENDING APP: sent data 14
TIME: 36 RECEIVING APP: received data 13
TIME: 37 RECEIVING APP: received data 14
TIME: 40 DATA_CHANNEL : udt_send called for Packet(seq_num=15, payload=15, corrupted=False)
TIME: 40 SENDING APP: sent data 15
TIME: 41 DATA_CHANNEL : udt_send called for Packet(seq_num=16, payload=16, corrupted=False)
TIME: 41 SENDING APP: sent data 16
TIME: 42 RECEIVING APP: received data 15
TIME: 43 RECEIVING APP: received data 16
TIME: 45 DATA_CHANNEL : udt_send called for Packet(seq_num=17, payload=17, corrupted=False)
TIME: 45 SENDING APP: sent data 17
TIME: 47 RECEIVING APP: received data 17
TIME: 50 DATA_CHANNEL : udt_send called for Packet(seq_num=18, payload=18, corrupted=False)
TIME: 50 SENDING APP: sent data 18
TIME: 52 RECEIVING APP: received data 18
TIME: 54 DATA_CHANNEL : udt_send called for Packet(seq_num=19, payload=19, corrupted=False)
TIME: 54 SENDING APP: sent data 19
TIME: 56 DATA_CHANNEL : udt_send called for Packet(seq_num=20, payload=20, corrupted=False)
TIME: 56 SENDING APP: sent data 20
TIME: 56 RECEIVING APP: received data 19
TIME: 58 RECEIVING APP: received data 20
TIME: 60 DATA_CHANNEL : udt_send called for Packet(seq_num=21, payload=21, corrupted=False)
TIME: 60 SENDING APP: sent data 21
TIME: 62 DATA_CHANNEL : udt_send called for Packet(seq_num=22, payload=22, corrupted=False)
TIME: 62 SENDING APP: sent data 22


gautamop@gautamop: ~/Desktop/LAB3
TIME: 60 SENDING APP: sent data 21
TIME: 62 DATA_CHANNEL : udt_send called for Packet(seq_num=22, payload=22, corrupted=False)
TIME: 62 SENDING APP: sent data 22
TIME: 62 RECEIVING APP: received data 21
TIME: 63 DATA_CHANNEL : udt_send called for Packet(seq_num=23, payload=23, corrupted=False)
TIME: 63 SENDING APP: sent data 23
TIME: 64 RECEIVING APP: received data 22
TIME: 65 RECEIVING APP: received data 23
TIME: 68 DATA_CHANNEL : udt_send called for Packet(seq_num=24, payload=24, corrupted=False)
TIME: 68 SENDING APP: sent data 24
TIME: 70 RECEIVING APP: received data 24
TIME: 72 DATA_CHANNEL : udt_send called for Packet(seq_num=25, payload=25, corrupted=False)
TIME: 72 SENDING APP: sent data 25
TIME: 74 RECEIVING APP: received data 25
TIME: 76 DATA_CHANNEL : udt_send called for Packet(seq_num=26, payload=26, corrupted=False)
TIME: 76 SENDING APP: sent data 26
TIME: 77 DATA_CHANNEL : udt_send called for Packet(seq_num=27, payload=27, corrupted=False)
TIME: 77 SENDING APP: sent data 27
TIME: 78 RECEIVING APP: received data 26
TIME: 79 RECEIVING APP: received data 27
TIME: 82 DATA_CHANNEL : udt_send called for Packet(seq_num=28, payload=28, corrupted=False)
TIME: 82 SENDING APP: sent data 28
TIME: 84 RECEIVING APP: received data 28
TIME: 86 DATA_CHANNEL : udt_send called for Packet(seq_num=29, payload=29, corrupted=False)
TIME: 86 SENDING APP: sent data 29
TIME: 88 DATA_CHANNEL : udt_send called for Packet(seq_num=30, payload=30, corrupted=False)
TIME: 88 SENDING APP: sent data 30
TIME: 88 RECEIVING APP: received data 29
TIME: 90 RECEIVING APP: received data 30
TIME: 92 DATA_CHANNEL : udt_send called for Packet(seq_num=31, payload=31, corrupted=False)
TIME: 92 SENDING APP: sent data 31
TIME: 94 RECEIVING APP: received data 31
TIME: 97 DATA_CHANNEL : udt_send called for Packet(seq_num=32, payload=32, corrupted=False)
TIME: 97 SENDING APP: sent data 32
TIME: 99 RECEIVING APP: received data 32
(gautamop㉿gautamop)-[~/Desktop/LAB3]
$

When Pc=0.5, rdt_1.0 protocol fails due to lack of a reliable data channel.



<u>Question 2</u>

The file Protocol_rdt2.py implements the simple ACK/NAK based protocol rdt2.0 that can work when data packets can get corrupted. Check that this protocol indeed works by setting Pc>0 for the data-channel.

[Note: The protocol to be used can be specified in the file Testbench.py by modifying the line from Protocol_rdt import *]

Ans: Pc = 0.1

```
┌──(gautamop㉿gautamop)-[~/Desktop/LAB3]
└─$ python3 Testbench.py
TIME: 3 DATA_CHANNEL : udt_send called for Packet(seq_num=0, payload=0, corrupted=False)
TIME: 3 SENDING APP: sent data 0
TIME: 5 ACK_CHANNEL : udt_send called for Packet(seq_num=0, payload=ACK, corrupted=False)
TIME: 5 RECEIVING APP: received data 0
TIME: 8 DATA_CHANNEL : udt_send called for Packet(seq_num=1, payload=1, corrupted=False)
TIME: 8 SENDING APP: sent data 1
TIME: 10 ACK_CHANNEL : udt_send called for Packet(seq_num=0, payload=ACK, corrupted=False)
TIME: 10 RECEIVING APP: received data 1
TIME: 12 DATA_CHANNEL : udt_send called for Packet(seq_num=2, payload=2, corrupted=False)
TIME: 12 SENDING APP: sent data 2
TIME: 14 ACK_CHANNEL : udt_send called for Packet(seq_num=0, payload=ACK, corrupted=False)
TIME: 14 RECEIVING APP: received data 2
TIME: 21 DATA_CHANNEL : udt_send called for Packet(seq_num=3, payload=3, corrupted=False)
TIME: 21 SENDING APP: sent data 3
TIME: 21 DATA_CHANNEL : Packet(seq_num=3, payload=$H!T, corrupted=True) was corrupted!
TIME: 23 ACK_CHANNEL : udt_send called for Packet(seq_num=0, payload=NAK, corrupted=False)
TIME: 25 DATA_CHANNEL : udt_send called for Packet(seq_num=3, payload=3, corrupted=False)
TIME: 27 ACK_CHANNEL : udt_send called for Packet(seq_num=0, payload=ACK, corrupted=False)
TIME: 27 RECEIVING APP: received data 3
TIME: 32 DATA_CHANNEL : udt_send called for Packet(seq_num=4, payload=4, corrupted=False)
TIME: 32 SENDING APP: sent data 4
TIME: 34 ACK_CHANNEL : udt_send called for Packet(seq_num=0, payload=ACK, corrupted=False)
TIME: 34 RECEIVING APP: received data 4
TIME: 36 DATA_CHANNEL : udt_send called for Packet(seq_num=5, payload=5, corrupted=False)
TIME: 36 SENDING APP: sent data 5
TIME: 38 ACK_CHANNEL : udt_send called for Packet(seq_num=0, payload=ACK, corrupted=False)
TIME: 38 RECEIVING APP: received data 5
TIME: 41 DATA_CHANNEL : udt_send called for Packet(seq_num=6, payload=6, corrupted=False)
TIME: 41 SENDING APP: sent data 6
TIME: 43 ACK_CHANNEL : udt_send called for Packet(seq_num=0, payload=ACK, corrupted=False)
TIME: 43 RECEIVING APP: received data 6
TIME: 49 DATA_CHANNEL : udt_send called for Packet(seq_num=7, payload=7, corrupted=False)
TIME: 49 SENDING APP: sent data 7
TIME: 51 ACK_CHANNEL : udt_send called for Packet(seq_num=0, payload=ACK, corrupted=False)
TIME: 51 RECEIVING APP: received data 7
TIME: 54 DATA_CHANNEL : udt_send called for Packet(seq_num=8, payload=8, corrupted=False)
TIME: 54 SENDING APP: sent data 8
TIME: 56 ACK_CHANNEL : udt_send called for Packet(seq_num=0, payload=ACK, corrupted=False)
TIME: 56 RECEIVING APP: received data 8
TIME: 58 DATA_CHANNEL : udt_send called for Packet(seq_num=9, payload=9, corrupted=False)
TIME: 58 SENDING APP: sent data 9
TIME: 60 ACK_CHANNEL : udt_send called for Packet(seq_num=0, payload=ACK, corrupted=False)
TIME: 60 RECEIVING APP: received data 9
```

```
TIME: 68 SENDING APP: sent data 11
TIME: 70 ACK_CHANNEL : udt_send called for Packet(seq_num=0, payload=ACK, corrupted=False)
TIME: 70 RECEIVING APP: received data 11
TIME: 75 DATA_CHANNEL : udt_send called for Packet(seq_num=12, payload=12, corrupted=False)
TIME: 75 SENDING APP: sent data 12
TIME: 77 ACK_CHANNEL : udt_send called for Packet(seq_num=0, payload=ACK, corrupted=False)
TIME: 77 RECEIVING APP: received data 12
TIME: 80 DATA_CHANNEL : udt_send called for Packet(seq_num=13, payload=13, corrupted=False)
TIME: 80 SENDING APP: sent data 13
TIME: 82 ACK_CHANNEL : udt_send called for Packet(seq_num=0, payload=ACK, corrupted=False)
TIME: 82 RECEIVING APP: received data 13
TIME: 86 DATA_CHANNEL : udt_send called for Packet(seq_num=14, payload=14, corrupted=False)
TIME: 86 SENDING APP: sent data 14
TIME: 88 ACK_CHANNEL : udt_send called for Packet(seq_num=0, payload=ACK, corrupted=False)
TIME: 88 RECEIVING APP: received data 14
TIME: 93 DATA_CHANNEL : udt_send called for Packet(seq_num=15, payload=15, corrupted=False)
TIME: 93 SENDING APP: sent data 15
TIME: 95 ACK_CHANNEL : udt_send called for Packet(seq_num=0, payload=ACK, corrupted=False)
TIME: 95 RECEIVING APP: received data 15
TIME: 98 DATA_CHANNEL : udt_send called for Packet(seq_num=16, payload=16, corrupted=False)
TIME: 98 SENDING APP: sent data 16

┌──(gautamop㉿gautamop)-[~/Desktop/LAB3]
└─$
```

Question 3

For the testbench using Protocol_rdt2.py, modify the code such that:
a) The sending application generates a fixed total number of messages (say 1000), with a fixed time interval between each message (say 3 units of time)
b) As soon as the protocol at the sending-side (rdt_Sender) receives positive acknowledgements for all of the 1000 messages, the simulation ends, and a quantity "T_avg" is printed as output, where T_avg is the time between sending a packet and receiving a positive acknowledgement for it, averaged across all packets. This is, in essence the Average Round-Trip Time (RTT avg).
(Note that you need to run each simulation as long as necessary for all 1000 messages to be acknowledged.)

You would expect that T_avg should increase with Pc, but in what manner? (linearly? exponentially? geometrically?). Obtain a plot of T_avg versus Pc for (0<=Pc<=0.9) and explain what trend you observe and why

Answer a) - The Modifications In the File Applications.py

```
import simpy
import random
from Packet import Packet
import sys
class SendingApplication(object):

    def __init__(self,env):
        # Initialize variables
        self.env=env
        self.rdt_sender=None
        self.total_messages_sent=0

        # start behavior
        self.env.process(self.behavior())

    def behavior(self):

        while (self.total_messages_sent < 1000):
            # wait for a random amount of time
            t=3
            yield self.env.timeout(t)
```

This would make sure a 1000 messages are sent with a fixed time interval of 3 seconds between the messages.

Answer b) - To calculate the average round trip time we have made the following modifications to our code in protocol2

```
from tracemalloc import start
import simpy
import random
from Packet import Packet
import sys

packetCount = 0
Overall_Time = 0

# the sender can be in one of these two states:
WAITING_FOR_CALL_FROM_ABOVE =0
WAIT_FOR_ACK_OR_NAK=1



class rdt_Sender(object):
```

The global variables packetCount and overall time, which are both initially set to 0, will count the number of packets that are received at the program's receiver side. Overall time also stores the total time required to send all 1000 messages.

At the Sender Side in rdt_send() method —>

```
def rdt_send(self,msg):
    global starttime
    if self.state==WAITING_FOR_CALL_FROM_ABOVE:
        # This function is called by the
        # sending application.

        # create a packet, and save a copy of this packet
        # for retransmission, if needed
        self.packet_to_be_sent = Packet(seq_num=self.seq_num, payload=msg)
        self.seq_num+=1
        # send it over the channel
        self.channel.udt_send(self.packet_to_be_sent)
        starttime = self.env.now
        # wait for an ACK or NAK
        self.state=WAIT_FOR_ACK_OR_NAK
        return True
    else:
        return False

def rdt_rcv(self,packt):
```

In the rdt send method a start time global variable is kept which is set whenever udt_send is called for a packet.
Now, At the sender side in the rdt_rcv() method —>

```
def rdt_rcv(self,packt):
    global packetCount
    global timetaken
    global Overall_Time
    global average_Time

    # This function is called by the lower-layer
    # when an ACK/NAK packet arrives

    assert(self.state==WAIT_FOR_ACK_OR_NAK)
    if(packt.payload=="ACK"):
        # Received an ACK. Everything's fine.
        self.state=WAITING_FOR_CALL_FROM_ABOVE
        endtime = self.env.now
        packetCount = packetCount+1
        timetaken = endtime - starttime
        Overall_Time = Overall_Time + timetaken

        if(packetCount == 1000):
            average_Time = Overall_Time/1000
            print("Average Duration = ", average_Time)

    elif(packt.payload=="NAK"):
        # Received a NAK. Need to resend packet.
        self.channel.udt_send(self.packet_to_be_sent)
    else:
        print("ERROR! rdt_rcv() was expecting an ACK or a NAK. Received a corrupted packet.")
        print("Halting simulation ... ")
        sys.exit(0)
```

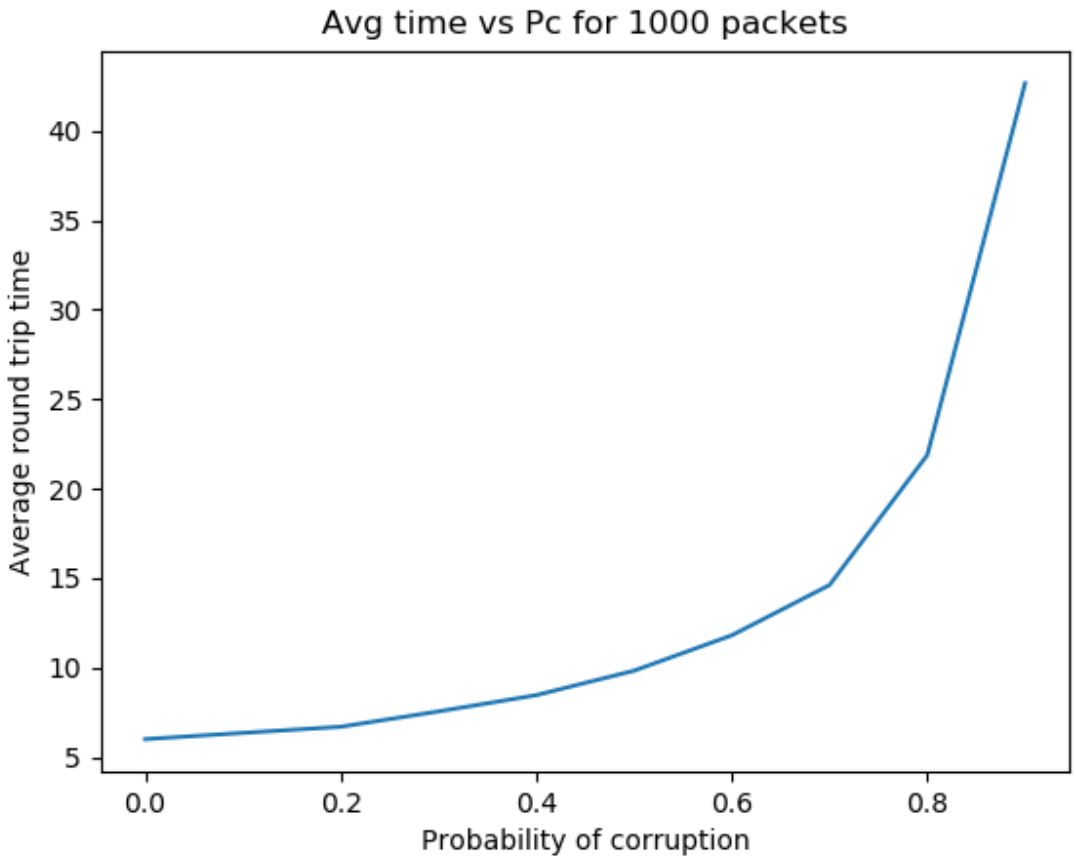
We delete until from our env.run because in the Testbench we have to let the simulation run until all 1000 packets are received.

```
# Run Simulation
env.run()
```

Here, if an ACK is received for a packet, it implies that the round trip time for that packet is endtime (which is calculated from the Simpy environment), minus starttime, and this is kept in the variable timetaken. We then update the total time by the rtt of the current packet, which is timetaken.
Then, if we have received all 1000 packets, we confirm it and publish the average time spent as Overall time/Number of packets = Overall time/1000.

Pc	Average Time Taken(in s)
0.0	5.999
0.1	6.347
0.2	6.689
0.3	7.547
0.4	8.456
0.5	9.822
0.6	11.799
0.7	14.611
0.8	21.879
0.9	42.655

Plot of Average Round Trip Time for 1000 Packets v.s Pc of Data Channel



Question 4 -

Protocol_rdt2.py will not work if ACK/NAK can also get corrupted. Check this by setting Pc>0 for the ack-channel and state the symptoms you observe.

When Pc = 0.1 and Ack Channel Corruption probablity of 0.1

```
┌──(gautamop㉿gautamop)-[~/Desktop/LAB3]
└─$ python3 Testbench.py
TIME: 3 DATA_CHANNEL : udt_send called for Packet(seq_num=0, payload=0, corrupted=False)
TIME: 3 SENDING APP: sent data 0
TIME: 5 ACK_CHANNEL : udt_send called for Packet(seq_num=0, payload=ACK, corrupted=False)
TIME: 5 RECEIVING APP: received data 0
TIME: 9 DATA_CHANNEL : udt_send called for Packet(seq_num=1, payload=1, corrupted=False)
TIME: 9 SENDING APP: sent data 1
TIME: 11 ACK_CHANNEL : udt_send called for Packet(seq_num=0, payload=ACK, corrupted=False)
TIME: 11 RECEIVING APP: received data 1
TIME: 15 DATA_CHANNEL : udt_send called for Packet(seq_num=2, payload=2, corrupted=False)
TIME: 15 SENDING APP: sent data 2
TIME: 17 ACK_CHANNEL : udt_send called for Packet(seq_num=0, payload=ACK, corrupted=False)
TIME: 17 RECEIVING APP: received data 2
TIME: 21 DATA_CHANNEL : udt_send called for Packet(seq_num=3, payload=3, corrupted=False)
TIME: 21 SENDING APP: sent data 3
TIME: 23 ACK_CHANNEL : udt_send called for Packet(seq_num=0, payload=ACK, corrupted=False)
TIME: 23 RECEIVING APP: received data 3
TIME: 27 DATA_CHANNEL : udt_send called for Packet(seq_num=4, payload=4, corrupted=False)
TIME: 27 SENDING APP: sent data 4
TIME: 29 ACK_CHANNEL : udt_send called for Packet(seq_num=0, payload=ACK, corrupted=False)
TIME: 29 RECEIVING APP: received data 4
TIME: 33 DATA_CHANNEL : udt_send called for Packet(seq_num=5, payload=5, corrupted=False)
TIME: 33 SENDING APP: sent data 5
TIME: 33 DATA_CHANNEL : Packet(seq_num=5, payload=$H!T, corrupted=True) was corrupted!
TIME: 35 ACK_CHANNEL : udt_send called for Packet(seq_num=0, payload=NAK, corrupted=False)
TIME: 37 DATA_CHANNEL : udt_send called for Packet(seq_num=5, payload=5, corrupted=False)
TIME: 39 ACK_CHANNEL : udt_send called for Packet(seq_num=0, payload=ACK, corrupted=False)
TIME: 39 RECEIVING APP: received data 5
TIME: 39 ACK_CHANNEL : Packet(seq_num=0, payload=$H!T, corrupted=True) was corrupted!
ERROR! rdt_rcv() was expecting an ACK or a NAK. Received a corrupted packet.
Halting simulation...
```

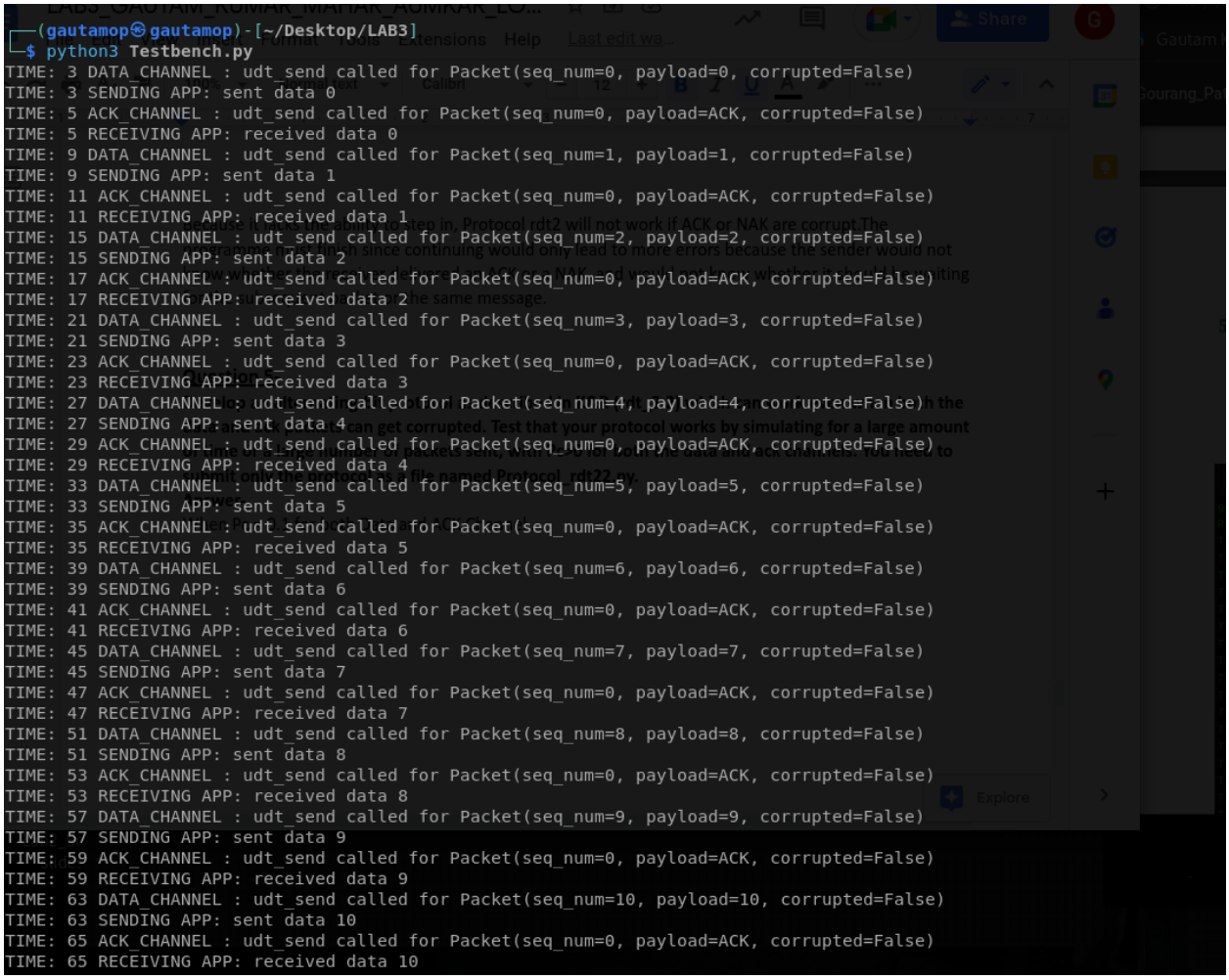
Because it lacks the ability to step in, Protocol rdt2 will not work if ACK or NAK are corrupt.The programme must finish since continuing would only lead to more errors because the sender would not know whether the receiver delivered an ACK or a NAK, and would not know whether it should be waiting for the subsequent packet or the same message.

Question 5-

Develop an alternating-bit protocol as described in K&R (rdt_2.2) which can work even when both the data and ack packets can get corrupted. Test that your protocol works by simulating for a large amount of time or a large number of packets sent, with Pc>0 for both the data and ack channels. You need to submit only the protocol as a file named Protocol_rdt22.py.

Answer-

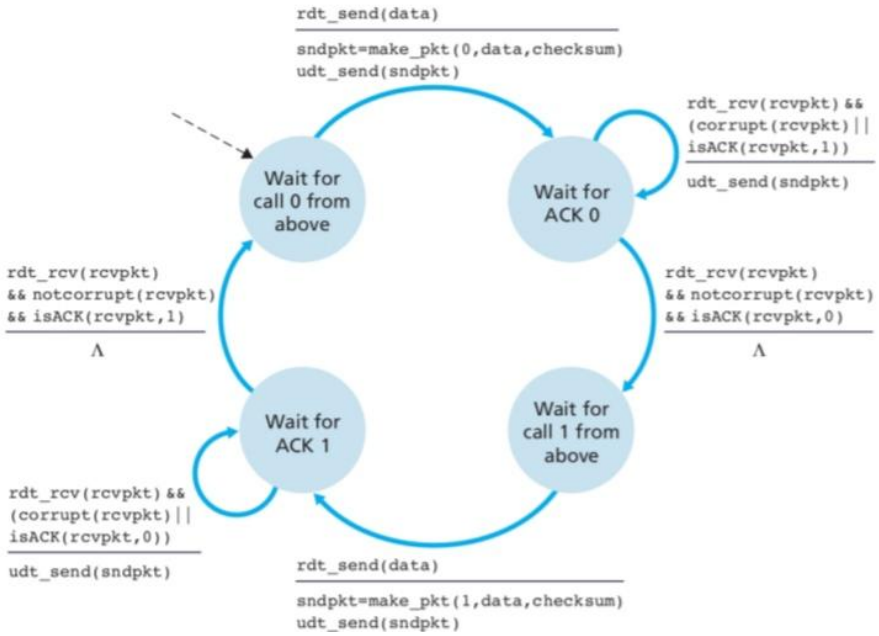
When Pc = 0.1 for both Data and ACK Channel .

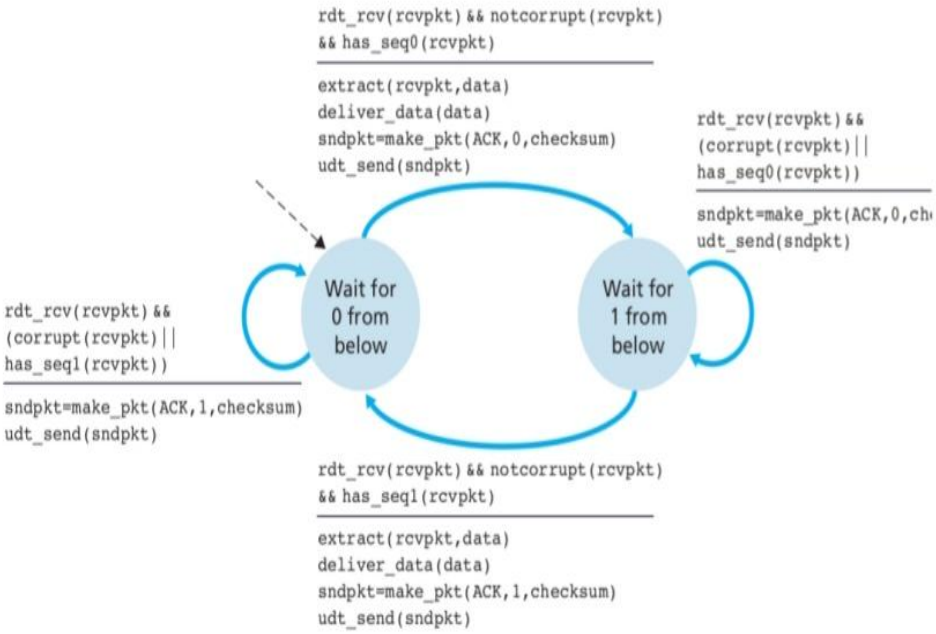


```
┌──(gautamop㉿gautamop)-[~/Desktop/LAB3]
└─$ python3 Testbench.py
TIME: 3 DATA_CHANNEL : udt_send called for Packet(seq_num=0, payload=0, corrupted=False)
TIME: 3 SENDING APP: sent data 0
TIME: 5 ACK_CHANNEL : udt_send called for Packet(seq_num=0, payload=ACK, corrupted=False)
TIME: 5 RECEIVING APP: received data 0
TIME: 9 DATA_CHANNEL : udt_send called for Packet(seq_num=1, payload=1, corrupted=False)
TIME: 9 SENDING APP: sent data 1
TIME: 11 ACK_CHANNEL : udt_send called for Packet(seq_num=0, payload=ACK, corrupted=False)
TIME: 11 RECEIVING APP: received data 1
TIME: 15 DATA_CHANNEL : udt_send called for Packet(seq_num=2, payload=2, corrupted=False)
TIME: 15 SENDING APP: sent data 2
TIME: 17 ACK_CHANNEL : udt_send called for Packet(seq_num=0, payload=ACK, corrupted=False)
TIME: 17 RECEIVING APP: received data 2
TIME: 21 DATA_CHANNEL : udt_send called for Packet(seq_num=3, payload=3, corrupted=False)
TIME: 21 SENDING APP: sent data 3
TIME: 23 ACK_CHANNEL : udt_send called for Packet(seq_num=0, payload=ACK, corrupted=False)
TIME: 23 RECEIVING APP: received data 3
TIME: 27 DATA_CHANNEL : udt_send called for Packet(seq_num=4, payload=4, corrupted=False)
TIME: 27 SENDING APP: sent data 4
TIME: 29 ACK_CHANNEL : udt_send called for Packet(seq_num=0, payload=ACK, corrupted=False)
TIME: 29 RECEIVING APP: received data 4
TIME: 33 DATA_CHANNEL : udt_send called for Packet(seq_num=5, payload=5, corrupted=False)
TIME: 33 SENDING APP: sent data 5
TIME: 35 ACK_CHANNEL : udt_send called for Packet(seq_num=0, payload=ACK, corrupted=False)
TIME: 35 RECEIVING APP: received data 5
TIME: 39 DATA_CHANNEL : udt_send called for Packet(seq_num=6, payload=6, corrupted=False)
TIME: 39 SENDING APP: sent data 6
TIME: 41 ACK_CHANNEL : udt_send called for Packet(seq_num=0, payload=ACK, corrupted=False)
TIME: 41 RECEIVING APP: received data 6
TIME: 45 DATA_CHANNEL : udt_send called for Packet(seq_num=7, payload=7, corrupted=False)
TIME: 45 SENDING APP: sent data 7
TIME: 47 ACK_CHANNEL : udt_send called for Packet(seq_num=0, payload=ACK, corrupted=False)
TIME: 47 RECEIVING APP: received data 7
TIME: 51 DATA_CHANNEL : udt_send called for Packet(seq_num=8, payload=8, corrupted=False)
TIME: 51 SENDING APP: sent data 8
TIME: 53 ACK_CHANNEL : udt_send called for Packet(seq_num=0, payload=ACK, corrupted=False)
TIME: 53 RECEIVING APP: received data 8
TIME: 57 DATA_CHANNEL : udt_send called for Packet(seq_num=9, payload=9, corrupted=False)
TIME: 57 SENDING APP: sent data 9
TIME: 59 ACK_CHANNEL : udt_send called for Packet(seq_num=0, payload=ACK, corrupted=False)
TIME: 59 RECEIVING APP: received data 9
TIME: 63 DATA_CHANNEL : udt_send called for Packet(seq_num=10, payload=10, corrupted=False)
TIME: 63 SENDING APP: sent data 10
TIME: 65 ACK_CHANNEL : udt_send called for Packet(seq_num=0, payload=ACK, corrupted=False)
TIME: 65 RECEIVING APP: received data 10
```

```
TIME: 63 SENDING APP: sent data 10
TIME: 65 ACK_CHANNEL : udt_send called for Packet(seq_num=0, payload=ACK, corrupted=False)
TIME: 65 RECEIVING APP: received data 10
TIME: 69 DATA_CHANNEL : udt_send called for Packet(seq_num=11, payload=11, corrupted=False)
TIME: 69 SENDING APP: sent data 11
TIME: 71 ACK_CHANNEL : udt_send called for Packet(seq_num=0, payload=ACK, corrupted=False)
TIME: 71 RECEIVING APP: received data 11
TIME: 75 DATA_CHANNEL : udt_send called for Packet(seq_num=12, payload=12, corrupted=False)
TIME: 75 SENDING APP: sent data 12
TIME: 77 ACK_CHANNEL : udt_send called for Packet(seq_num=0, payload=ACK, corrupted=False)
TIME: 77 RECEIVING APP: received data 12
TIME: 81 DATA_CHANNEL : udt_send called for Packet(seq_num=13, payload=13, corrupted=False)
TIME: 81 SENDING APP: sent data 13
TIME: 83 ACK_CHANNEL : udt_send called for Packet(seq_num=0, payload=ACK, corrupted=False)
TIME: 83 RECEIVING APP: received data 13
TIME: 87 DATA_CHANNEL : udt_send called for Packet(seq_num=14, payload=14, corrupted=False)
TIME: 87 SENDING APP: sent data 14
TIME: 89 ACK_CHANNEL : udt_send called for Packet(seq_num=0, payload=ACK, corrupted=False)
TIME: 89 RECEIVING APP: received data 14
TIME: 93 DATA_CHANNEL : udt_send called for Packet(seq_num=15, payload=15, corrupted=False)
TIME: 93 SENDING APP: sent data 15
TIME: 95 ACK_CHANNEL : udt_send called for Packet(seq_num=0, payload=ACK, corrupted=False)
TIME: 95 RECEIVING APP: received data 15
TIME: 99 DATA_CHANNEL : udt_send called for Packet(seq_num=16, payload=16, corrupted=False)
TIME: 99 SENDING APP: sent data 16
```

The aforementioned protocol deals with corruptions in both the DATA Channel and the ACK Channel.




rdt_rcv(rcvpkt) && notcorrupt(rcvpkt)
&& has_seq0(rcvpkt)
extract(rcvpkt,data)
deliver_data(data)
sndpkt=make_pkt(ACK,0,checksum)
udt_send(sndpkt)
rdt_rcv(rcvpkt) &&
(corrupt(rcvpkt)||
has_seq0(rcvpkt))
sndpkt=make_pkt(ACK,0,ch
udt_send(sndpkt)
Wait for
0 from
below
Wait for
1 from
below
rdt_rcv(rcvpkt) &&
(corrupt(rcvpkt)||
has_seq1(rcvpkt))
sndpkt=make_pkt(ACK,1,checksum)
udt_send(sndpkt)
rdt_rcv(rcvpkt) && notcorrupt(rcvpkt)
&& has_seq1(rcvpkt)
extract(rcvpkt,data)
deliver_data(data)
sndpkt=make_pkt(ACK,1,checksum)
udt_send(sndpkt)

Question 6-

The protocol rdt2.2 will not work if packets can be lost.

a) Check this by setting Pl>0 in the testbench with your implementation of Protocol_rdt22. What failure symptoms do you observe?

Ans.)

When Pc=0.1 for both Data and Ack Channel and pl = 0.1

```
┌──(gautamop㉿gautamop)-[~/Desktop/LAB3]
└─$ python3 Testbench.py
TIME: 3 DATA_CHANNEL : udt_send called for Packet(seq_num=0, payload=0, corrupted=False)
TIME: 3 SENDING APP: sent data 0
TIME: 5 ACK_CHANNEL : udt_send called for Packet(seq_num=0, payload=ACK, corrupted=False)
TIME: 5 RECEIVING APP: received data 0
TIME: 9 DATA_CHANNEL : udt_send called for Packet(seq_num=1, payload=1, corrupted=False)
TIME: 9 SENDING APP: sent data 1
TIME: 11 ACK_CHANNEL : udt_send called for Packet(seq_num=0, payload=ACK, corrupted=False)
TIME: 11 RECEIVING APP: received data 1
TIME: 15 DATA_CHANNEL : udt_send called for Packet(seq_num=2, payload=2, corrupted=False)
TIME: 15 SENDING APP: sent data 2
TIME: 17 ACK_CHANNEL : udt_send called for Packet(seq_num=0, payload=ACK, corrupted=False)
TIME: 17 RECEIVING APP: received data 2
TIME: 21 DATA_CHANNEL : udt_send called for Packet(seq_num=3, payload=3, corrupted=False)
TIME: 21 SENDING APP: sent data 3
TIME: 23 ACK_CHANNEL : udt_send called for Packet(seq_num=0, payload=ACK, corrupted=False)
TIME: 23 RECEIVING APP: received data 3
TIME: 27 DATA_CHANNEL : udt_send called for Packet(seq_num=4, payload=4, corrupted=False)
TIME: 27 SENDING APP: sent data 4
TIME: 29 ACK_CHANNEL : udt_send called for Packet(seq_num=0, payload=ACK, corrupted=False)
TIME: 29 RECEIVING APP: received data 4
TIME: 33 DATA_CHANNEL : udt_send called for Packet(seq_num=5, payload=5, corrupted=False)
TIME: 33 SENDING APP: sent data 5
TIME: 35 ACK_CHANNEL : udt_send called for Packet(seq_num=0, payload=ACK, corrupted=False)
TIME: 35 RECEIVING APP: received data 5
TIME: 39 DATA_CHANNEL : udt_send called for Packet(seq_num=6, payload=6, corrupted=False)
TIME: 39 SENDING APP: sent data 6
TIME: 39 DATA_CHANNEL : Packet(seq_num=6, payload=$H!T, corrupted=True) was corrupted!
TIME: 41 ACK_CHANNEL : udt_send called for Packet(seq_num=0, payload=NAK, corrupted=False)
TIME: 43 DATA_CHANNEL : udt_send called for Packet(seq_num=6, payload=6, corrupted=False)
TIME: 45 ACK_CHANNEL : udt_send called for Packet(seq_num=0, payload=ACK, corrupted=False)
TIME: 45 RECEIVING APP: received data 6
TIME: 48 DATA_CHANNEL : udt_send called for Packet(seq_num=7, payload=7, corrupted=False)
TIME: 48 SENDING APP: sent data 7
TIME: 50 ACK_CHANNEL : udt_send called for Packet(seq_num=0, payload=ACK, corrupted=False)
TIME: 50 RECEIVING APP: received data 7
TIME: 54 DATA_CHANNEL : udt_send called for Packet(seq_num=8, payload=8, corrupted=False)
TIME: 54 SENDING APP: sent data 8
TIME: 56 ACK_CHANNEL : udt_send called for Packet(seq_num=0, payload=ACK, corrupted=False)
TIME: 56 RECEIVING APP: received data 8
TIME: 60 DATA_CHANNEL : udt_send called for Packet(seq_num=9, payload=9, corrupted=False)
TIME: 60 SENDING APP: sent data 9
TIME: 62 ACK_CHANNEL : udt_send called for Packet(seq_num=0, payload=ACK, corrupted=False)
TIME: 62 RECEIVING APP: received data 9
TIME: 62 ACK_CHANNEL : Packet(seq_num=0, payload=ACK, corrupted=False) was lost!

┌──(gautamop㉿gautamop)-[~/Desktop/LAB3]
└─$ python3 Testbench.py
```

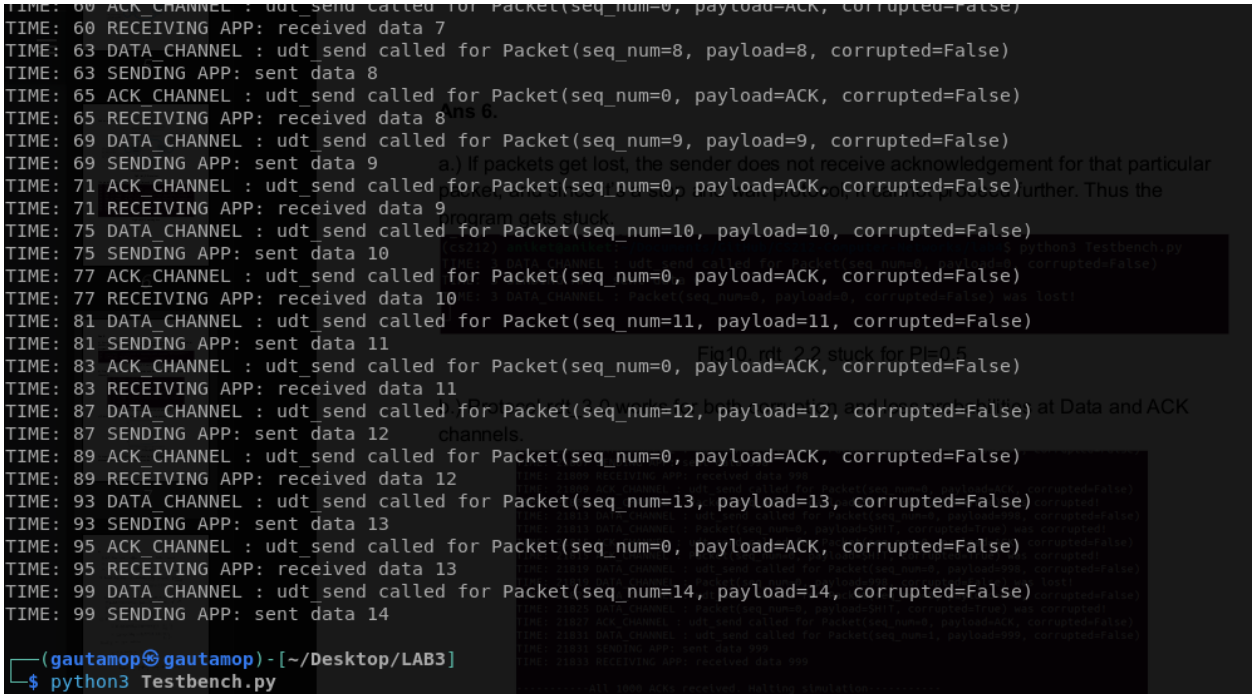
However, if packets can be lost, then RDT2.2 is unable to handle the situation. In this example, data up to packet 9 was successfully received, but as packet 9's ACK was lost, nothing will happen because the sender will keep waiting for packet 9's ACK to arrive, but it will never arrive. A similar scenario will be seen in the case where data channel loss occurs.

b) Implement an alternating-bit protocol with Timeouts (rdt3.0) that can work when data or ack packets can be corrupted or lost. Set the timeout value to 3*Delay. Test that your protocol indeed works by simulating for a large amount of time or a large number of packets sent, with Pc>0 and Pl>0 for both the data and ack channels. You need to submit the protocol as a file named "Protocol_rdt3.py".
[Hint: To implement timeouts, you may need to model a Timer in SimPy. A "skeleton" for implementing such a timer is provided here in Appendix A .]
Ans) -

Sample Output For the Case where Delay is set to 2s and Timeout as (3*Delay) 6s.
When Pc=0.1, pl=0.1 both Data and Ack Channel

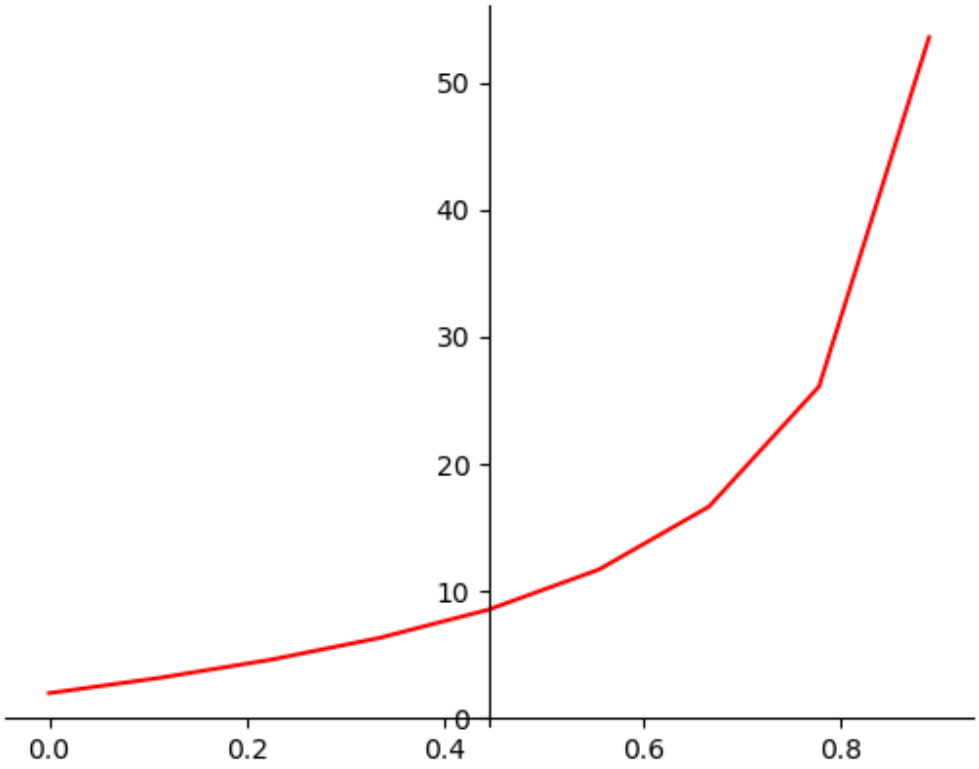
```
┌──(gautamop㉿gautamop)-[~/Desktop/LAB3]
└─$ python3 Testbench.py
TIME: 3 DATA_CHANNEL : udt_send called for Packet(seq_num=0, payload=0, corrupted=False)
TIME: 3 SENDING APP: sent data 0
TIME: 5 ACK_CHANNEL : udt_send called for Packet(seq_num=0, payload=ACK, corrupted=False)
TIME: 5 RECEIVING APP: received data 0
TIME: 9 DATA_CHANNEL : udt_send called for Packet(seq_num=1, payload=1, corrupted=False)
TIME: 9 SENDING APP: sent data 1
TIME: 11 ACK_CHANNEL : udt_send called for Packet(seq_num=0, payload=ACK, corrupted=False)
TIME: 11 RECEIVING APP: received data 1
TIME: 15 DATA_CHANNEL : udt_send called for Packet(seq_num=2, payload=2, corrupted=False)
TIME: 15 SENDING APP: sent data 2
TIME: 15 DATA_CHANNEL : Packet(seq_num=2, payload=$H!T, corrupted=True) was corrupted!
TIME: 17 ACK_CHANNEL : udt_send called for Packet(seq_num=0, payload=NAK, corrupted=False)
TIME: 19 DATA_CHANNEL : udt_send called for Packet(seq_num=2, payload=2, corrupted=False)
TIME: 19 DATA_CHANNEL : Packet(seq_num=2, payload=$H!T, corrupted=True) was corrupted!
TIME: 21 ACK_CHANNEL : udt_send called for Packet(seq_num=0, payload=NAK, corrupted=False)
TIME: 23 DATA_CHANNEL : udt_send called for Packet(seq_num=2, payload=2, corrupted=False)
TIME: 25 ACK_CHANNEL : udt_send called for Packet(seq_num=0, payload=ACK, corrupted=False)
TIME: 25 RECEIVING APP: received data 2
TIME: 30 DATA_CHANNEL : udt_send called for Packet(seq_num=3, payload=3, corrupted=False)
TIME: 30 SENDING APP: sent data 3
TIME: 32 ACK_CHANNEL : udt_send called for Packet(seq_num=0, payload=ACK, corrupted=False)
TIME: 32 RECEIVING APP: received data 3
TIME: 36 DATA_CHANNEL : udt_send called for Packet(seq_num=4, payload=4, corrupted=False)
TIME: 36 SENDING APP: sent data 4
TIME: 38 ACK_CHANNEL : udt_send called for Packet(seq_num=0, payload=ACK, corrupted=False)
TIME: 38 RECEIVING APP: received data 4
TIME: 42 DATA_CHANNEL : udt_send called for Packet(seq_num=5, payload=5, corrupted=False)
TIME: 42 SENDING APP: sent data 5
TIME: 44 ACK_CHANNEL : udt_send called for Packet(seq_num=0, payload=ACK, corrupted=False)
TIME: 44 RECEIVING APP: received data 5
TIME: 48 DATA_CHANNEL : udt_send called for Packet(seq_num=6, payload=6, corrupted=False)
TIME: 48 SENDING APP: sent data 6
TIME: 50 ACK_CHANNEL : udt_send called for Packet(seq_num=0, payload=ACK, corrupted=False)
TIME: 50 RECEIVING APP: received data 6
TIME: 54 DATA_CHANNEL : udt_send called for Packet(seq_num=7, payload=7, corrupted=False)
TIME: 54 SENDING APP: sent data 7
TIME: 54 DATA_CHANNEL : Packet(seq_num=7, payload=$H!T, corrupted=True) was corrupted!
TIME: 56 ACK_CHANNEL : udt_send called for Packet(seq_num=0, payload=NAK, corrupted=False)
TIME: 58 DATA_CHANNEL : udt_send called for Packet(seq_num=7, payload=7, corrupted=False)
TIME: 60 ACK_CHANNEL : udt_send called for Packet(seq_num=0, payload=ACK, corrupted=False)
TIME: 60 RECEIVING APP: received data 7
TIME: 63 DATA_CHANNEL : udt_send called for Packet(seq_num=8, payload=8, corrupted=False)
```

The Protocol Functions Perfectly. It can handle a large number of packets being sent from the sender side in the event of loss or corruption. To illustrate, let's look at the case of TIME: 15, where packet 3 was corrupted during transmission and the receiver responded with an acknowledgement but without the correct sequence number. As a result, after a timeout, the data for packet 3 was once again sent back at TIME: 21, which is the 6 second timeout we intended to have.



c) You would expect that T_avg (defined the same way as in question 1) would increase with Pl. For your implementation of the rdt3.0 protocol, with channel Delay=2, Pc=0.2, timeout=3*Delay and total packets generated =1000, plot T_avg versus Pl

Ans) Average time will increase as probability of loss increases.



Avg time vs Pc for 1000 packets

d) [BONUS QUESTION] For the scenario where packet loss is possible (assume Pc=0), derive an analytical expression for how T_avg should vary with Pl. Check if the trend observed from simulations matches this.
Ans) -

Packet loss can happen both on Data channel as well as Ack channel so, let P_e be the less probability

$\Rightarrow$ Round trip time = $\{(1-P_e)2*Delay + (P_e(timeout))\}$ + {this is for Data channel}

$\{(1-P_e)(2*Delay) + P_e[timeout]\}$ {Ack channel}

Data channel has GP because with P_e probability 1st packet transmission fails, P_e^2 2nd fails, P_e^3 3rd fails and so on

where timeout = $3*Delay$

$RTT = \Sigma(1-P_e)(2*Delay) + P_e(timeout) + timeout(P_e + P_e^2 + P_e^3$

$= 2(1-P_e)(2*Delay) + P_e(timeout\left(1 + \frac{1}{1-P_e}\right)$

Rewriting in g-x notation

$$g = C_1(1-x) + C_2 x\left(\frac{2-x}{1-x}\right)$$

This is a Hyperbola equation.

e) Protocol rdt3.0 will not work if packets can be re-ordered over the channel. This can be easily modeled by setting the channel delay for each packet to be a randomly chosen number instead of having the same value for all packets. Thus, packet-2 sent after packet-1, might experience a lower delay and arrive ahead of a packet-1 at the receiver. Implement this feature in the model, and check if the rdt3.0 protocol indeed fails. In your report, show the code snipette that models this packet reordering

Ans) - We can configure the delay to be random between 1 and 5 to represent a random delay for the channel each time it is invoked, which would guarantee packet reordering**.**

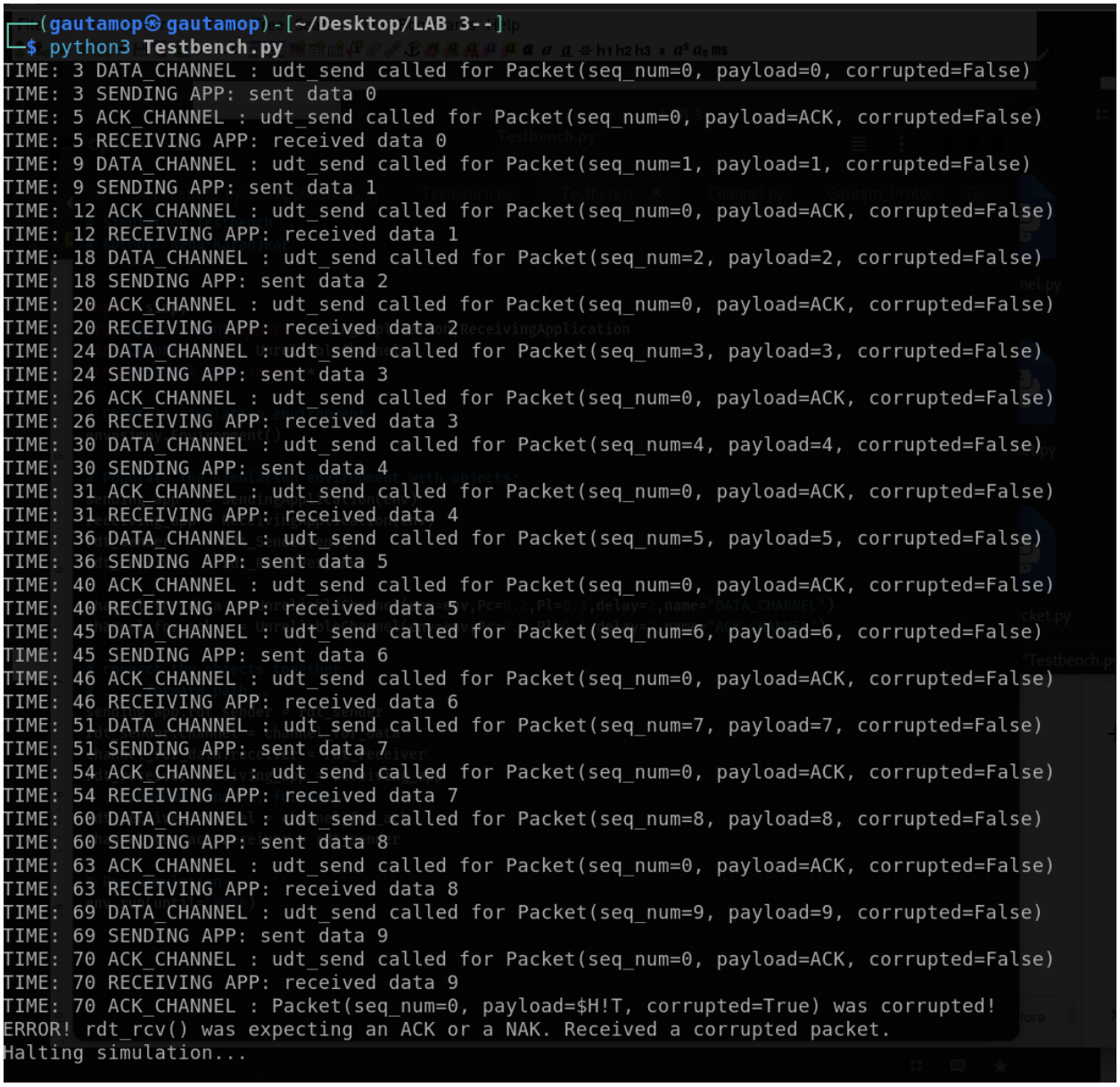
```
import simpy
import random
from Packet import Packet
import copy
class UnreliableChannel(object):

    def __init__(self,env,Pc,Pl,delay,name):
        # Initialize variables
        self.env=env
        self.Pc=Pc
        self.Pl=Pl
        self.delay=delay
        self.receiver=None
        self.name=name

    def udt_send(self,packt_to_be_sent):
        # this function is called by the sending-side
        # to send a new packet over the channel.
        packt=copy.copy(packt_to_be_sent) #!!! BEWARE: Python uses pass-by-reference by default.Thus a
copy() is necessary
        print("TIME:",self.env.now,self.name,": udt_send called for",packt)
        # start a process to deliver this packet across the channel.
        self.env.process(self.deliver_packet_over_channel(random.randint(1,5), packt))
        # self.env.process(self.deliver_packet_over_channel(self.delay, packt))
```

Output looks like -

When pc = 0.2, pl = 0.1 for both data and Ack channel



Here, the simulation is interrupted when the channel's packets are rearranged.

Thank You !