

Lab 3 Principles of Reliable Data Transfer

Team Members:

1. Name : Abhinav Reddy Gutha

Roll No. : 2103102

2. Name : Nidamanuri Sai Adarsh

Roll No. : 2103123

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.

The file Protocol_rdt1.py implements the trivial protocol rdt1.0 which works only if the channel is assumed to be ideal. Run the simulation first with Pc=0, and then Pc=0.5. Check that the protocol fails in the second case, and list the failure symptoms. [Note: The channel parameters such as Pc and Pl can be set in the file Testbench.py]

Here's the output when Pc of the data channel is 0:

```
abhinav@abhinav-Inspiron-5402: ~/Downloads
rrupted=False)
TIME: 987 SENDING APP: sent data 338
TIME: 988 RECEIVING APP: received data 337
TIME: 989 RECEIVING APP: received data 338
TIME: 991 DATA_CHANNEL : udt_send called for Packet(seq_num=339, payload=339, co
rrupted=False)
TIME: 991 SENDING APP: sent data 339
TIME: 992 DATA_CHANNEL : udt_send called for Packet(seq_num=340, payload=340, co
rrupted=False)
TIME: 992 SENDING APP: sent data 340
TIME: 993 RECEIVING APP: received data 339
TIME: 993 DATA_CHANNEL : udt_send called for Packet(seq_num=341, payload=341, co
rrupted=False)
TIME: 993 SENDING APP: sent data 341
TIME: 994 RECEIVING APP: received data 340
TIME: 995 RECEIVING APP: received data 341
TIME: 996 DATA_CHANNEL : udt_send called for Packet(seq_num=342, payload=342, co
rrupted=False)
TIME: 996 SENDING APP: sent data 342
TIME: 998 RECEIVING APP: received data 342
TIME: 999 DATA_CHANNEL : udt_send called for Packet(seq_num=343, payload=343, co
rrupted=False)
TIME: 999 SENDING APP: sent data 343
abhinav@abhinav-Inspiron-5402:~/Downloads$
```

Here's the output when Pc of the data channel is 0.5:

```
abhinav@abhinav-Inspiron-5402:~/Downloads$ python3 Testbench.py
TIME: 4 DATA_CHANNEL : udt_send called for Packet(seq_num=0, payload=0, corrupte
d=False)
TIME: 4 SENDING APP: sent data 0
TIME: 5 DATA_CHANNEL : udt_send called for Packet(seq_num=1, payload=1, corrupte
d=False)
TIME: 5 SENDING APP: sent data 1
TIME: 5 DATA_CHANNEL : Packet(seq_num=1, payload=$H!T, corrupted=True) was corru
pted!
TIME: 6 RECEIVING APP: received data 0
TIME: 10 DATA_CHANNEL : udt_send called for Packet(seq_num=2, payload=2, corrupt
ed=False)
TIME: 10 SENDING APP: sent data 2
TIME: 12 DATA_CHANNEL : udt_send called for Packet(seq_num=3, payload=3, corrupt
ed=False)
TIME: 12 SENDING APP: sent data 3
TIME: 12 RECEIVING APP: received data 2
ERROR!! RECEIVING APP: received wrong data: 2 ,expected: 1
Halting simulation...
abhinav@abhinav-Inspiron-5402:~/Downloads$
```

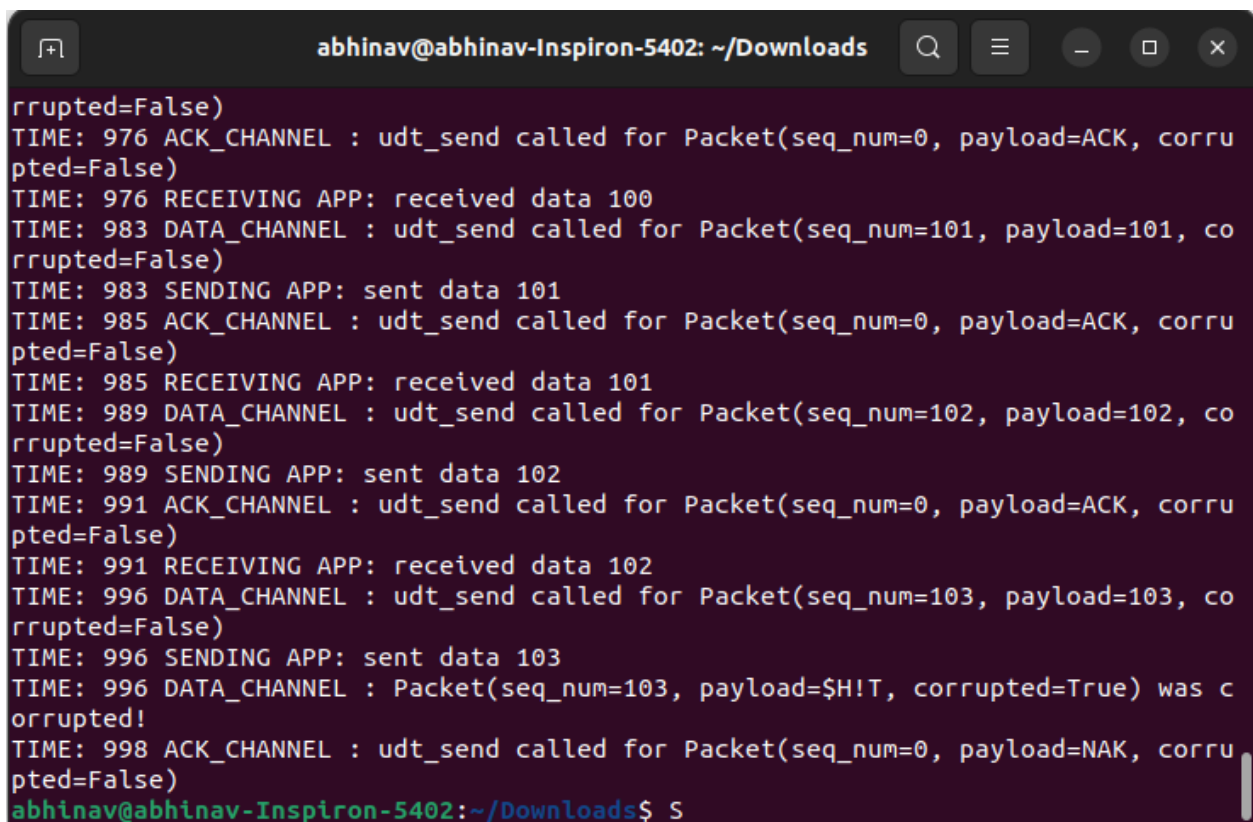
In RDT 1.0 we don't consider the case where the packet will get corrupted/ lost. Whenever there is a call from above for the sender and a packet in flight, the packet will flow through the data channel. As $P_c = 0.5$, sometimes the packet

getting corrupted becomes true. This will halt the transfer of packets at some moment of time.

Therefore, the protocol fails in the second case.

Question 2) 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 $P_c > 0$ 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 *]

Here's the output of the following code:

A terminal window with a dark background and light-colored text. The title bar shows the user 'abhinav' on a machine named 'abhinav-Inspiron-5402' in the directory '~/Downloads'. The terminal output shows a sequence of log messages for a data transfer protocol. It starts with an ACK packet, followed by data packets 100, 101, 102, and 103. At the end of the sequence, a data packet with payload '\$H!T' is marked as corrupted. This triggers a NAK packet to be sent back to the sender. The prompt at the bottom is 'abhinav@abhinav-Inspiron-5402:~/Downloads\$ S'.

```
rrupted=False)
TIME: 976 ACK_CHANNEL : udt_send called for Packet(seq_num=0, payload=ACK, corru
pted=False)
TIME: 976 RECEIVING APP: received data 100
TIME: 983 DATA_CHANNEL : udt_send called for Packet(seq_num=101, payload=101, co
rrupted=False)
TIME: 983 SENDING APP: sent data 101
TIME: 985 ACK_CHANNEL : udt_send called for Packet(seq_num=0, payload=ACK, corru
pted=False)
TIME: 985 RECEIVING APP: received data 101
TIME: 989 DATA_CHANNEL : udt_send called for Packet(seq_num=102, payload=102, co
rrupted=False)
TIME: 989 SENDING APP: sent data 102
TIME: 991 ACK_CHANNEL : udt_send called for Packet(seq_num=0, payload=ACK, corru
pted=False)
TIME: 991 RECEIVING APP: received data 102
TIME: 996 DATA_CHANNEL : udt_send called for Packet(seq_num=103, payload=103, co
rrupted=False)
TIME: 996 SENDING APP: sent data 103
TIME: 996 DATA_CHANNEL : Packet(seq_num=103, payload=$H!T, corrupted=True) was c
orrupted!
TIME: 998 ACK_CHANNEL : udt_send called for Packet(seq_num=0, payload=NAK, corru
pted=False)
abhinav@abhinav-Inspiron-5402:~/Downloads$ S
```

Whenever the packet gets corrupted, there will be a NAK packet sent across the ACK channel. Once it reaches the sender, the sender will resend the packets so that the non corrupted packet will be received at the receiver side.

Therefore, this protocol is designed in such a way that it can handle the packets which are corrupted and doesn't throw an error and ensures all packets are received at the receiver side.

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).

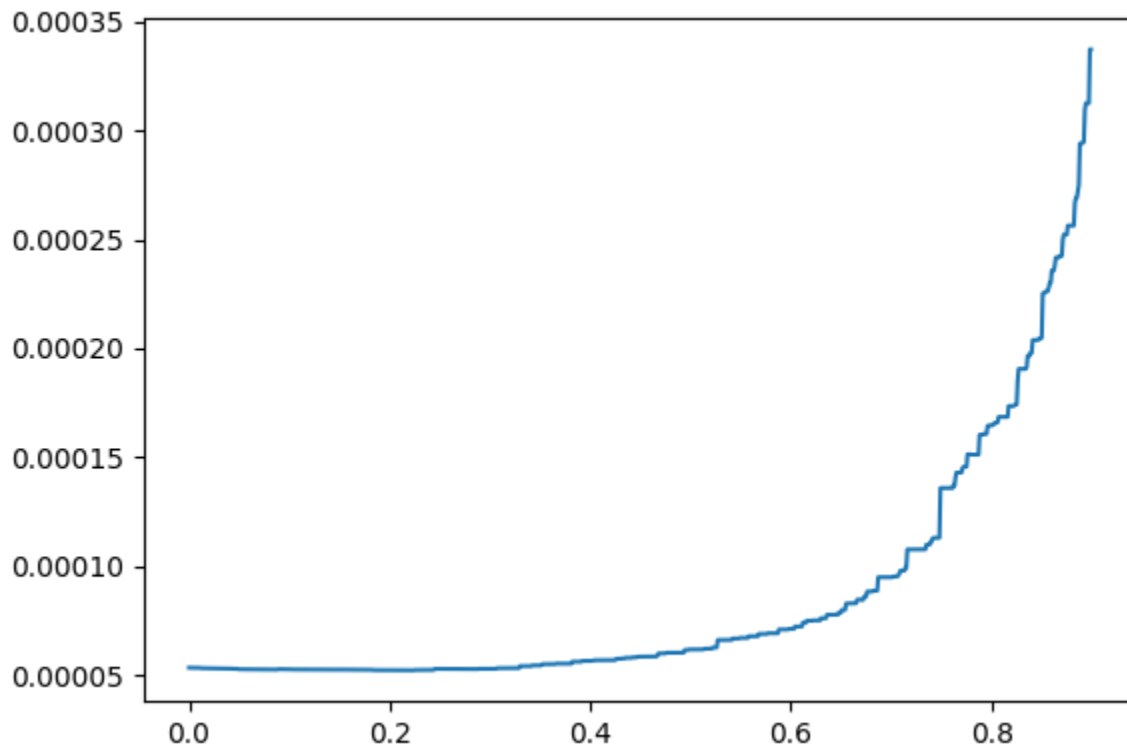
(Note that you need to run each simulation as long as necessary for all 1000 messages to be acknowledged.)

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 ($0 \leq Pc \leq 0.9$) and explain what trend you observe and why.

Here's the output of the following code:

```
abhinav@abhinav-Inspiron-5402: ~/Desktop/question3
TIME: 43528 DATA_CHANNEL : Packet(seq_num=1005, payload=$H!T, corrupted=True) wa
s corrupted!
TIME: 43530 ACK_CHANNEL : udt_send called for Packet(seq_num=0, payload=NAK, cor
rupted=False)
TIME: 43532 DATA_CHANNEL : udt_send called for Packet(seq_num=1005, payload=1005
, corrupted=False)
TIME: 43534 ACK_CHANNEL : udt_send called for Packet(seq_num=0, payload=ACK, cor
rupted=False)
TIME: 43534 RECEIVING APP: received data 1005
The round trip time is: 0.0001227855682373047
TIME: 43538 DATA_CHANNEL : udt_send called for Packet(seq_num=1006, payload=1006
, corrupted=False)
TIME: 43538 SENDING APP: sent data 1006
TIME: 43538 DATA_CHANNEL : Packet(seq_num=1006, payload=$H!T, corrupted=True) wa
s corrupted!
TIME: 43540 ACK_CHANNEL : udt_send called for Packet(seq_num=0, payload=NAK, cor
rupted=False)
TIME: 43542 DATA_CHANNEL : udt_send called for Packet(seq_num=1006, payload=1006
, corrupted=False)
TIME: 43544 ACK_CHANNEL : udt_send called for Packet(seq_num=0, payload=ACK, cor
rupted=False)
TIME: 43544 RECEIVING APP: received data 1006
The round trip time is: 7.748603820800781e-05
[5.342406233773556e-05, 5.337791292464104e-05, 5.337427542808021e-05, 5.33704041
9256693e-05, 5.3365672946172e-05, 5.3360982396062066e-05, 5.3355745477106566e-05]
```

When we run the code for different values of P_c , here's what we got when we plot a graph between T_{avg} and P_c . The graph we obtained is Geometric (Hyperbolic Function).



The logic behind this is we have noted the time at the sender side when the packet is being sent across the data channel. We have stored this value in the variable `start_time`. Similarly, we have noted the time again at the sender side when the ACK corresponding to the packet sent by the sender is received. We have stored this value in another variable `end_time`. We have calculated the RTT (Round Trip Time) by subtracting `end_time` with the start time. This is the value of the RTT corresponding to this packet.

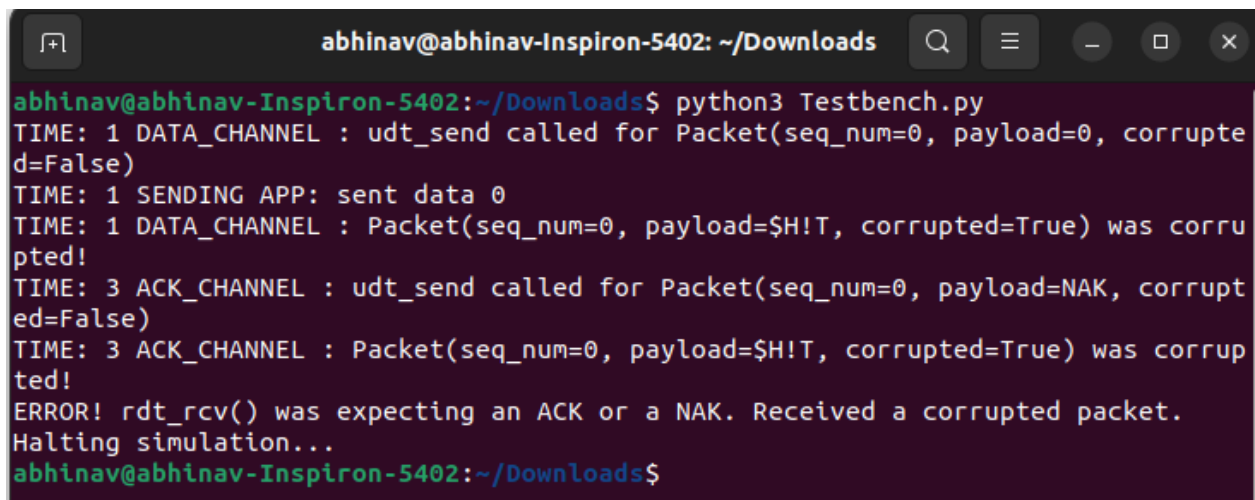
Now we would send 1000 such packets to the receiver and calculate the RTT values corresponding to each one of the packets and store it in a numpy array. Once all these 1000 packets are sent across the channels, we would calculate the mean value of the numpy array we obtained. This mean value is nothing but the avg RTT (T_{avg}) which we require for a given value of P_c .

We repeat this process by taking different values of P_c and calculate the T_{avg} .

Whenever the packet is getting corrupted, we would resend the packet. This would result in the increase of the RTT value. Therefore, the overall T_{avg} value also increases.

Question 4) Protocol_rdt2.py will not work if ACK/NAK can also get corrupted. Check this by setting $P_c > 0$ for the ACK channel and state the symptoms you observe.

Here's the output of the following code:

A terminal window titled 'abhinav@abhinav-Inspiron-5402: ~/Downloads' shows the execution of 'python3 Testbench.py'. The output logs the following: 'TIME: 1 DATA_CHANNEL : udt_send called for Packet(seq_num=0, payload=0, corrupted=False)', 'TIME: 1 SENDING APP: sent data 0', 'TIME: 1 DATA_CHANNEL : Packet(seq_num=0, payload=\$H!T, corrupted=True) was corrupted!', 'TIME: 3 ACK_CHANNEL : udt_send called for Packet(seq_num=0, payload=NAK, corrupted=False)', 'TIME: 3 ACK_CHANNEL : Packet(seq_num=0, payload=\$H!T, corrupted=True) was corrupted!', and finally 'ERROR! rdt_rcv() was expecting an ACK or a NAK. Received a corrupted packet. Halting simulation...'. The prompt returns to 'abhinav@abhinav-Inspiron-5402: ~/Downloads\$'.

This protocol doesn't work whenever an ACK packet is sent across the ACK channel. As the ACK packet got corrupted, the program will stop. This is because of the improper communication between sender and receiver and the packet count got disturbed during the process.

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 $P_c > 0$ for both the data and ack channels. You need to submit only the protocol as a file named Protocol_rdt22.py.

Here's the output of the following code:


```

adash@adash-Dell-G15-5515: ~/Desktop/question 4
$ python3 Testbench.py
TIME: 1 DATA_CHANNEL : udt_send called for Packet(seq_num=0, payload=0, corrupted=False)
TIME: 1 SENDING APP: sent data 0
TIME: 3 ACK_CHANNEL : udt_send called for Packet(seq_num=0, payload=ACK, corrupted=False)
TIME: 3 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=1, payload=ACK, corrupted=False)
TIME: 11 RECEIVING APP: received data 1
TIME: 16 DATA_CHANNEL : udt_send called for Packet(seq_num=0, payload=2, corrupted=False)
TIME: 16 SENDING APP: sent data 2
TIME: 18 ACK_CHANNEL : udt_send called for Packet(seq_num=0, payload=ACK, corrupted=False)
TIME: 18 RECEIVING APP: received data 2
TIME: 22 DATA_CHANNEL : udt_send called for Packet(seq_num=1, payload=3, corrupted=False)
TIME: 22 SENDING APP: sent data 3
TIME: 24 ACK_CHANNEL : udt_send called for Packet(seq_num=1, payload=ACK, corrupted=False)
TIME: 24 RECEIVING APP: received data 3
TIME: 29 DATA_CHANNEL : udt_send called for Packet(seq_num=0, payload=4, corrupted=False)
TIME: 29 SENDING APP: sent data 4
TIME: 31 ACK_CHANNEL : udt_send called for Packet(seq_num=0, payload=ACK, corrupted=False)
TIME: 31 RECEIVING APP: received data 4
TIME: 35 DATA_CHANNEL : udt_send called for Packet(seq_num=1, payload=5, corrupted=False)
TIME: 35 SENDING APP: sent data 5
TIME: 37 ACK_CHANNEL : udt_send called for Packet(seq_num=1, payload=ACK, corrupted=False)
TIME: 37 RECEIVING APP: received data 5
TIME: 37 ACK_CHANNEL : Packet(seq_num=1, payload=SHIT, corrupted=True) was corrupted!
TIME: 39 DATA_CHANNEL : udt_send called for Packet(seq_num=1, payload=5, corrupted=False)
TIME: 41 ACK_CHANNEL : udt_send called for Packet(seq_num=1, payload=ACK, corrupted=False)
TIME: 45 DATA_CHANNEL : udt_send called for Packet(seq_num=0, payload=6, corrupted=False)
TIME: 45 SENDING APP: sent data 6
TIME: 47 ACK_CHANNEL : udt_send called for Packet(seq_num=0, payload=ACK, corrupted=False)
TIME: 47 RECEIVING APP: received data 6
TIME: 50 DATA_CHANNEL : udt_send called for Packet(seq_num=1, payload=7, corrupted=False)
TIME: 50 SENDING APP: sent data 7
TIME: 52 ACK_CHANNEL : udt_send called for Packet(seq_num=1, payload=ACK, corrupted=False)
TIME: 52 RECEIVING APP: received data 7
TIME: 57 DATA_CHANNEL : udt_send called for Packet(seq_num=0, payload=8, corrupted=False)
TIME: 57 SENDING APP: sent data 8
TIME: 59 ACK_CHANNEL : udt_send called for Packet(seq_num=0, payload=ACK, corrupted=False)
TIME: 59 RECEIVING APP: received data 8
TIME: 61 DATA_CHANNEL : udt_send called for Packet(seq_num=1, payload=9, corrupted=False)
TIME: 61 SENDING APP: sent data 9
TIME: 63 ACK_CHANNEL : udt_send called for Packet(seq_num=1, payload=ACK, corrupted=False)
TIME: 63 RECEIVING APP: received data 9
TIME: 66 DATA_CHANNEL : udt_send called for Packet(seq_num=0, payload=10, corrupted=False)
TIME: 66 SENDING APP: sent data 10
TIME: 68 DATA_CHANNEL : Packet(seq_num=0, payload=SHIT, corrupted=True) was corrupted!
TIME: 68 ACK_CHANNEL : udt_send called for Packet(seq_num=1, payload=ACK, corrupted=False)
TIME: 68 ACK_CHANNEL : Packet(seq_num=1, payload=SHIT, corrupted=True) was corrupted!
TIME: 70 DATA_CHANNEL : udt_send called for Packet(seq_num=0, payload=10, corrupted=False)
TIME: 72 ACK_CHANNEL : udt_send called for Packet(seq_num=0, payload=ACK, corrupted=False)
TIME: 72 RECEIVING APP: received data 10
TIME: 72 ACK_CHANNEL : Packet(seq_num=0, payload=SHIT, corrupted=True) was corrupted!
TIME: 74 DATA_CHANNEL : udt_send called for Packet(seq_num=0, payload=10, corrupted=False)
TIME: 76 ACK_CHANNEL : udt_send called for Packet(seq_num=0, payload=ACK, corrupted=False)
TIME: 76 ACK_CHANNEL : Packet(seq_num=0, payload=SHIT, corrupted=True) was corrupted!

```

Here's the protocol which we designed for rdt2.2 of the sender and the receiver:

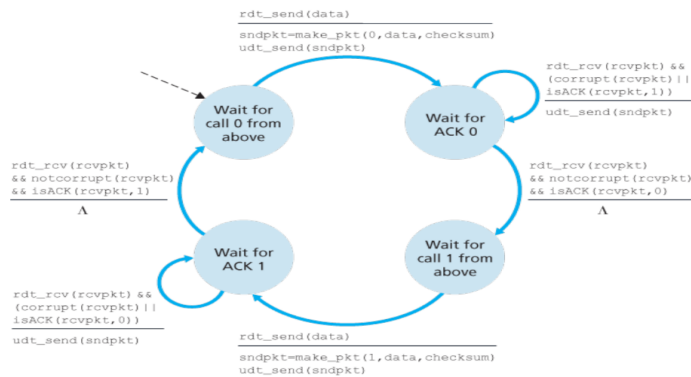


Figure 3.13 rdt2.2 sender

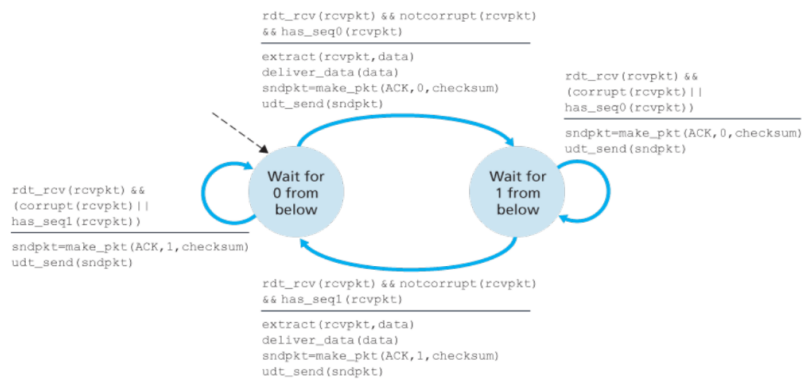


Figure 3.14 `rdt2.2` receiver

Based on the conditions in each and every state, using conditional statements in python we have modified the code in such a way that we handle both the cases where the ACK or the packet got corrupted.

Question 6) The protocol rdt2.2 will not work if packets can be lost.

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

Here's the output of the following code:

```

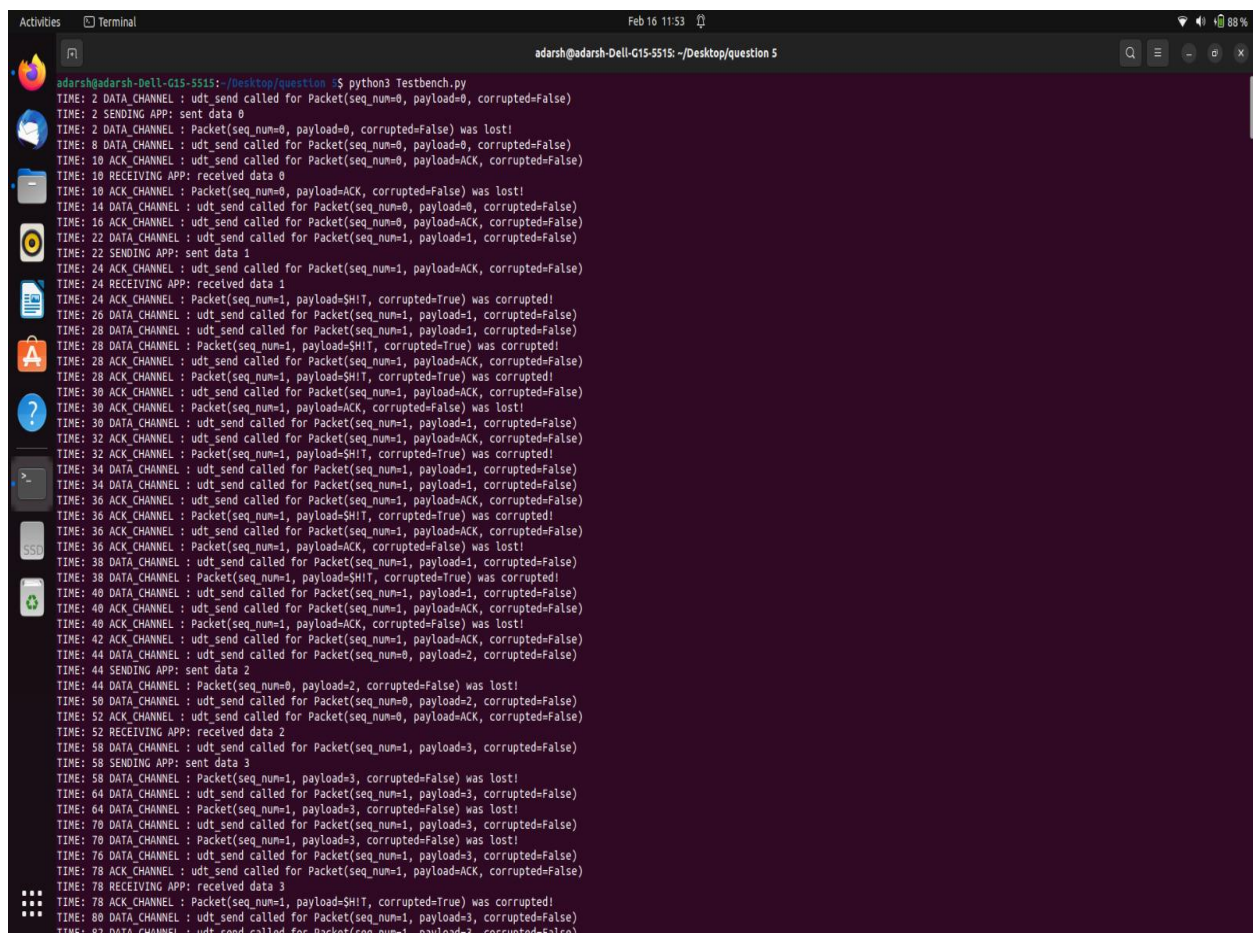
adash@adarsh-Dell-G15-5515: ~/Desktop/question 4
$ python3 Testbench.py
TIME: 2 DATA_CHANNEL : udt_send called for Packet(seq_num=0, payload=0, corrupted=False)
TIME: 2 SENDING_APP: sent data 0
TIME: 4 ACK_CHANNEL : udt_send called for Packet(seq_num=0, payload=ACK, corrupted=False)
TIME: 4 RECEIVING_APP: received data 0
TIME: 10 DATA_CHANNEL : udt_send called for Packet(seq_num=1, payload=1, corrupted=False)
TIME: 10 SENDING_APP: sent data 1
TIME: 12 ACK_CHANNEL : udt_send called for Packet(seq_num=1, payload=ACK, corrupted=False)
TIME: 12 RECEIVING_APP: received data 1
TIME: 12 ACK_CHANNEL : Packet(seq_num=1, payload=SHIT, corrupted=True) was corrupted!
TIME: 14 DATA_CHANNEL : udt_send called for Packet(seq_num=1, payload=1, corrupted=False)
TIME: 14 DATA_CHANNEL : Packet(seq_num=1, payload=1, corrupted=False) was lost!
adash@adarsh-Dell-G15-5515: ~/Desktop/question 4 $
  
```


As we have not defined any functions for defining PI based on timeouts, this protocol won't work when there is a possibility of packet loss happening.

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 $3 \times \text{Delay}$. Test that your protocol indeed works by simulating for a large amount of time or a large number of packets sent, with $P_c > 0$ and $P_l > 0$ 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.]

Here's the output of the following code:



```
adarsh@adarsh-Dell-G15-5515: ~/Desktop/question 5
$ python3 Testbench.py
TIME: 2 DATA_CHANNEL : udt_send called for Packet(seq_num=0, payload=0, corrupted=False)
TIME: 2 SENDING APP: sent data 0
TIME: 2 DATA_CHANNEL : Packet(seq_num=0, payload=0, corrupted=False) was lost!
TIME: 8 DATA_CHANNEL : udt_send called for Packet(seq_num=0, payload=0, corrupted=False)
TIME: 10 ACK_CHANNEL : udt_send called for Packet(seq_num=0, payload=ACK, corrupted=False)
TIME: 10 RECEIVING APP: received data 0
TIME: 10 ACK_CHANNEL : Packet(seq_num=0, payload=ACK, corrupted=False) was lost!
TIME: 14 DATA_CHANNEL : udt_send called for Packet(seq_num=0, payload=0, corrupted=False)
TIME: 16 ACK_CHANNEL : udt_send called for Packet(seq_num=0, payload=ACK, corrupted=False)
TIME: 22 DATA_CHANNEL : udt_send called for Packet(seq_num=1, payload=1, corrupted=False)
TIME: 22 SENDING APP: sent data 1
TIME: 24 ACK_CHANNEL : udt_send called for Packet(seq_num=1, payload=ACK, corrupted=False)
TIME: 24 RECEIVING APP: received data 1
TIME: 24 ACK_CHANNEL : Packet(seq_num=1, payload=SHIT, corrupted=True) was corrupted!
TIME: 26 DATA_CHANNEL : udt_send called for Packet(seq_num=1, payload=1, corrupted=False)
TIME: 28 DATA_CHANNEL : udt_send called for Packet(seq_num=1, payload=1, corrupted=False)
TIME: 28 DATA_CHANNEL : Packet(seq_num=1, payload=SHIT, corrupted=True) was corrupted!
TIME: 28 ACK_CHANNEL : udt_send called for Packet(seq_num=1, payload=ACK, corrupted=False)
TIME: 28 ACK_CHANNEL : Packet(seq_num=1, payload=SHIT, corrupted=True) was corrupted!
TIME: 30 ACK_CHANNEL : udt_send called for Packet(seq_num=1, payload=ACK, corrupted=False)
TIME: 30 ACK_CHANNEL : Packet(seq_num=1, payload=ACK, corrupted=False) was lost!
TIME: 30 DATA_CHANNEL : udt_send called for Packet(seq_num=1, payload=1, corrupted=False)
TIME: 32 ACK_CHANNEL : udt_send called for Packet(seq_num=1, payload=ACK, corrupted=False)
TIME: 32 ACK_CHANNEL : Packet(seq_num=1, payload=SHIT, corrupted=True) was corrupted!
TIME: 34 DATA_CHANNEL : udt_send called for Packet(seq_num=1, payload=1, corrupted=False)
TIME: 34 DATA_CHANNEL : udt_send called for Packet(seq_num=1, payload=1, corrupted=False)
TIME: 36 ACK_CHANNEL : udt_send called for Packet(seq_num=1, payload=ACK, corrupted=False)
TIME: 36 ACK_CHANNEL : Packet(seq_num=1, payload=SHIT, corrupted=True) was corrupted!
TIME: 36 ACK_CHANNEL : udt_send called for Packet(seq_num=1, payload=ACK, corrupted=False)
TIME: 36 ACK_CHANNEL : Packet(seq_num=1, payload=ACK, corrupted=False) was lost!
TIME: 38 DATA_CHANNEL : udt_send called for Packet(seq_num=1, payload=1, corrupted=False)
TIME: 38 DATA_CHANNEL : Packet(seq_num=1, payload=SHIT, corrupted=True) was corrupted!
TIME: 40 DATA_CHANNEL : udt_send called for Packet(seq_num=1, payload=1, corrupted=False)
TIME: 40 ACK_CHANNEL : udt_send called for Packet(seq_num=1, payload=ACK, corrupted=False)
TIME: 40 ACK_CHANNEL : Packet(seq_num=1, payload=ACK, corrupted=False) was lost!
TIME: 42 ACK_CHANNEL : udt_send called for Packet(seq_num=1, payload=ACK, corrupted=False)
TIME: 44 DATA_CHANNEL : udt_send called for Packet(seq_num=0, payload=2, corrupted=False)
TIME: 44 SENDING APP: sent data 2
TIME: 44 DATA_CHANNEL : Packet(seq_num=0, payload=2, corrupted=False) was lost!
TIME: 50 DATA_CHANNEL : udt_send called for Packet(seq_num=0, payload=2, corrupted=False)
TIME: 52 ACK_CHANNEL : udt_send called for Packet(seq_num=0, payload=ACK, corrupted=False)
TIME: 52 RECEIVING APP: received data 2
TIME: 58 DATA_CHANNEL : udt_send called for Packet(seq_num=1, payload=3, corrupted=False)
TIME: 58 SENDING APP: sent data 3
TIME: 58 DATA_CHANNEL : Packet(seq_num=1, payload=3, corrupted=False) was lost!
TIME: 64 DATA_CHANNEL : udt_send called for Packet(seq_num=1, payload=3, corrupted=False)
TIME: 64 DATA_CHANNEL : Packet(seq_num=1, payload=3, corrupted=False) was lost!
TIME: 70 DATA_CHANNEL : udt_send called for Packet(seq_num=1, payload=3, corrupted=False)
TIME: 70 DATA_CHANNEL : Packet(seq_num=1, payload=3, corrupted=False) was lost!
TIME: 76 DATA_CHANNEL : udt_send called for Packet(seq_num=1, payload=3, corrupted=False)
TIME: 76 ACK_CHANNEL : udt_send called for Packet(seq_num=1, payload=ACK, corrupted=False)
TIME: 78 RECEIVING APP: received data 3
TIME: 78 ACK_CHANNEL : Packet(seq_num=1, payload=SHIT, corrupted=True) was corrupted!
TIME: 80 DATA_CHANNEL : udt_send called for Packet(seq_num=1, payload=3, corrupted=False)
TIME: 82 DATA_CHANNEL : udt_send called for Packet(seq_num=1, payload=3, corrupted=False)
```

What this protocol does is we have set a timer initially when we send packets from the sender. If the time exceeds the timeout value, then it will resend the same packet. Here's the FSM diagram for the RDT 3.0 of the sender side. The FSM diagram for the receiver side will be the same as we have for RDT 2.2.

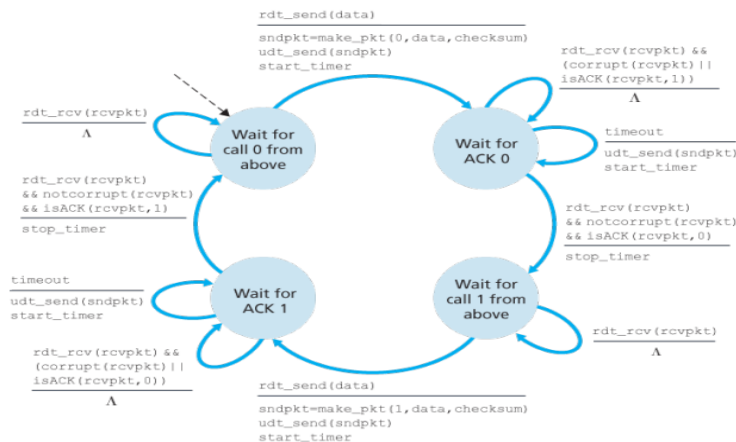
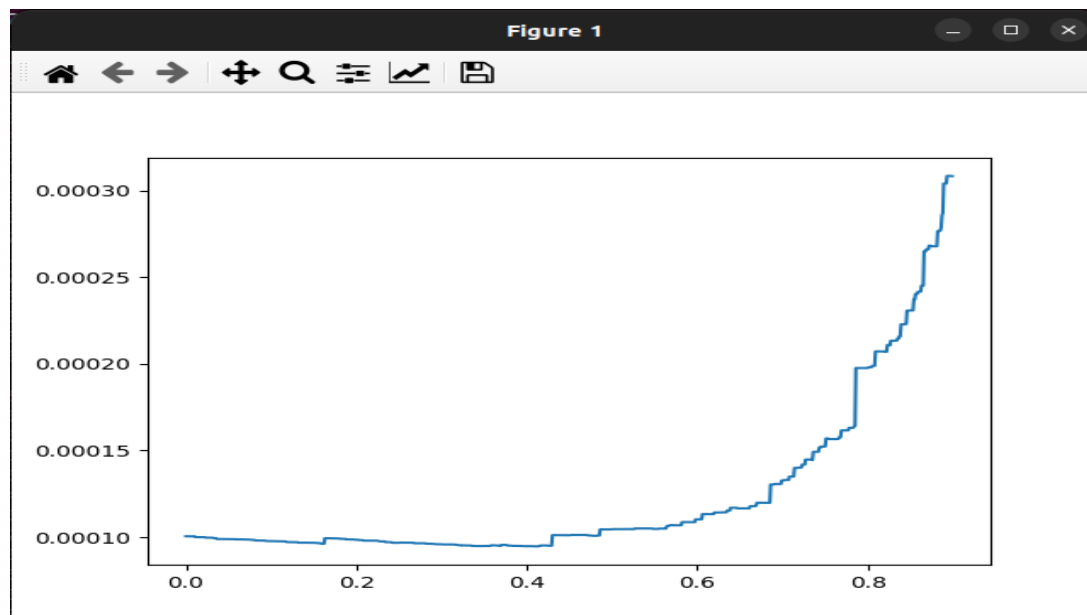


Figure 3.15 *rdt3.0* sender

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

Here's the output of the following code:



We can observe here that the T_{avg} increases as we increase the value of PI . Here the x-axis stands for PI and the y-axis stands for T_{avg} . The graph we obtained is hyperbolic, Whenever a packet is lost, the sender will resend the packet. This would definitely result in an increase of RTT value. Therefore the value of T_{avg} also increases.

d) [BONUS QUESTION] For the scenario where packet loss is possible (assume $P_c=0$), derive an analytical expression for how T_{avg} should vary with PI . Check if the trend observed from simulations matches this.

I have derived this equation based on the Memoryless-ness property and Total probability theorem which we studied on probability and statistics course.

As the packet sent at a moment doesn't affect the outcome of the other packet, this protocol is satisfying memoryless-ness property.
Based on total probability theorem,

$$T_{avg} = (1 - PI) * (c_1) + PI * (T_{avg} + c_2)$$

Where c_1 = Expected value of the time taken for the packet to be sent across the channels considering the packet is not lost.

c_2 = Expected value of the time taken for the duplicate packet to be sent across the channels considering this duplicate packet is not lost.

$$T_{avg} = c_1 - PI * c_1 + PI * T_{avg} + PI * c_2$$

$$T_{avg}(1 - PI) = c_1 + PI * (c_2 - c_1)$$

As this duplicate packet is nothing but a new packet and based on memoryless-ness property this outcome is not affected by the other outcomes of the packet. Therefore, we can consider that $c_2 \sim c_1$.

If we consider this, then we get

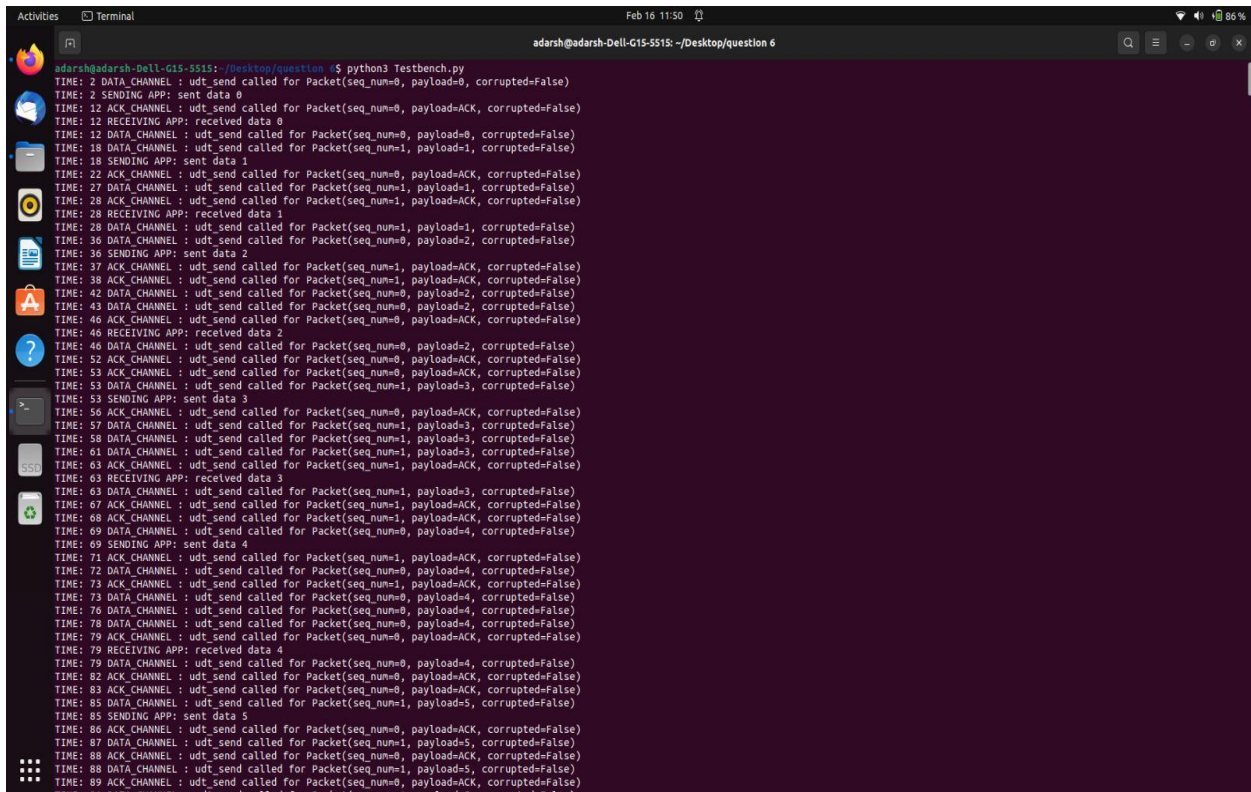
$$T_{avg}(1 - PI) = \text{constant}$$

The graph we obtained is hyperbolic and it is approximately matching with the graph which we plot for (c). You can observe that for the base case i.e, $PI = 0$;

we can observe that $T_{avg} = c_1$ which is nothing but the average expected value of the time taken by the packet considering the packet was not lost.

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 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 snippet that models this packet reordering.

Here's the output of the following code:



```
adarsh@adarsh-Dell-G15-5515: ~/Desktop/question 6
$ python3 Testbench.py
TIME: 2 DATA_CHANNEL : udt_send called for Packet(seq_num=0, payload=0, corrupted=False)
TIME: 2 SENDING APP: sent data 0
TIME: 12 ACK_CHANNEL : udt_send called for Packet(seq_num=0, payload=ACK, corrupted=False)
TIME: 12 RECEIVING APP: received data 0
TIME: 12 DATA_CHANNEL : udt_send called for Packet(seq_num=0, payload=0, corrupted=False)
TIME: 18 DATA_CHANNEL : udt_send called for Packet(seq_num=1, payload=1, corrupted=False)
TIME: 18 SENDING APP: sent data 1
TIME: 22 ACK_CHANNEL : udt_send called for Packet(seq_num=0, payload=ACK, corrupted=False)
TIME: 27 DATA_CHANNEL : udt_send called for Packet(seq_num=1, payload=1, corrupted=False)
TIME: 28 ACK_CHANNEL : udt_send called for Packet(seq_num=1, payload=ACK, corrupted=False)
TIME: 28 RECEIVING APP: received data 1
TIME: 28 DATA_CHANNEL : udt_send called for Packet(seq_num=1, payload=1, corrupted=False)
TIME: 36 DATA_CHANNEL : udt_send called for Packet(seq_num=0, payload=2, corrupted=False)
TIME: 36 SENDING APP: sent data 2
TIME: 37 ACK_CHANNEL : udt_send called for Packet(seq_num=1, payload=ACK, corrupted=False)
TIME: 38 ACK_CHANNEL : udt_send called for Packet(seq_num=1, payload=ACK, corrupted=False)
TIME: 42 DATA_CHANNEL : udt_send called for Packet(seq_num=0, payload=2, corrupted=False)
TIME: 43 DATA_CHANNEL : udt_send called for Packet(seq_num=0, payload=2, corrupted=False)
TIME: 46 ACK_CHANNEL : udt_send called for Packet(seq_num=0, payload=ACK, corrupted=False)
TIME: 46 RECEIVING APP: received data 2
TIME: 46 DATA_CHANNEL : udt_send called for Packet(seq_num=0, payload=2, corrupted=False)
TIME: 52 ACK_CHANNEL : udt_send called for Packet(seq_num=0, payload=ACK, corrupted=False)
TIME: 53 ACK_CHANNEL : udt_send called for Packet(seq_num=0, payload=ACK, corrupted=False)
TIME: 53 DATA_CHANNEL : udt_send called for Packet(seq_num=1, payload=3, corrupted=False)
TIME: 53 SENDING APP: sent data 3
TIME: 56 ACK_CHANNEL : udt_send called for Packet(seq_num=0, payload=ACK, corrupted=False)
TIME: 57 DATA_CHANNEL : udt_send called for Packet(seq_num=1, payload=3, corrupted=False)
TIME: 58 DATA_CHANNEL : udt_send called for Packet(seq_num=1, payload=3, corrupted=False)
TIME: 61 DATA_CHANNEL : udt_send called for Packet(seq_num=1, payload=3, corrupted=False)
TIME: 63 ACK_CHANNEL : udt_send called for Packet(seq_num=1, payload=ACK, corrupted=False)
TIME: 63 RECEIVING APP: received data 3
TIME: 63 DATA_CHANNEL : udt_send called for Packet(seq_num=1, payload=3, corrupted=False)
TIME: 67 ACK_CHANNEL : udt_send called for Packet(seq_num=1, payload=ACK, corrupted=False)
TIME: 68 ACK_CHANNEL : udt_send called for Packet(seq_num=1, payload=ACK, corrupted=False)
TIME: 69 DATA_CHANNEL : udt_send called for Packet(seq_num=0, payload=4, corrupted=False)
TIME: 69 SENDING APP: sent data 4
TIME: 71 ACK_CHANNEL : udt_send called for Packet(seq_num=1, payload=ACK, corrupted=False)
TIME: 72 DATA_CHANNEL : udt_send called for Packet(seq_num=0, payload=4, corrupted=False)
TIME: 73 ACK_CHANNEL : udt_send called for Packet(seq_num=1, payload=ACK, corrupted=False)
TIME: 73 DATA_CHANNEL : udt_send called for Packet(seq_num=0, payload=4, corrupted=False)
TIME: 76 DATA_CHANNEL : udt_send called for Packet(seq_num=0, payload=4, corrupted=False)
TIME: 78 DATA_CHANNEL : udt_send called for Packet(seq_num=0, payload=4, corrupted=False)
TIME: 79 ACK_CHANNEL : udt_send called for Packet(seq_num=0, payload=ACK, corrupted=False)
TIME: 79 RECEIVING APP: received data 4
TIME: 79 DATA_CHANNEL : udt_send called for Packet(seq_num=0, payload=4, corrupted=False)
TIME: 82 ACK_CHANNEL : udt_send called for Packet(seq_num=0, payload=ACK, corrupted=False)
TIME: 83 ACK_CHANNEL : udt_send called for Packet(seq_num=0, payload=ACK, corrupted=False)
TIME: 85 DATA_CHANNEL : udt_send called for Packet(seq_num=1, payload=5, corrupted=False)
TIME: 85 SENDING APP: sent data 5
TIME: 86 ACK_CHANNEL : udt_send called for Packet(seq_num=0, payload=ACK, corrupted=False)
TIME: 87 DATA_CHANNEL : udt_send called for Packet(seq_num=1, payload=5, corrupted=False)
TIME: 88 ACK_CHANNEL : udt_send called for Packet(seq_num=0, payload=ACK, corrupted=False)
TIME: 88 DATA_CHANNEL : udt_send called for Packet(seq_num=1, payload=5, corrupted=False)
TIME: 89 ACK_CHANNEL : udt_send called for Packet(seq_num=0, payload=ACK, corrupted=False)
TIME: 91 DATA_CHANNEL : udt_send called for Packet(seq_num=1, payload=5, corrupted=False)
```

If we zoom in and observe the packet flow with sequence numbering here, we can observe that the packets are not re-ordered and the receiver is collecting the packets without caring about the packet numbering. Our program won't halt in this case but there is an improper flow of data in the data channel.

