Computer Networks Project 2
My Reliable Data Transfer Service

Note: Please carefully read through it! If you find any potential errors, please let me know so that I can update this document.

In Lecture 7 we learned how to design a reliable data transfer (rdt) server from scratch. In this project, you will implement rdt3.0, the final version of the stop-and-wait approach. Recall that rdt3.0 can handle bit errors and packet loss. If you need details about rdt3.0, refer back to the textbook or the slides.

In this project, you will implement the services on both the sender and receiver ends. This rdt3.0 runs in the application layer, and you can choose either TCP or UDP as the underlying transport service. Even if you chose TCP, you still need to implement all the features in rdt3.0 such as error detection, retransmission, acknowledgment, etc. That is, we assume neither of them is reliable in this project.

A. Project Requirement

1. Required packet format

For all the packets in this project, we use the following packet format. The first 8 bytes always contain the value "COMPNETW". The 9th and 10th bytes together are for the checksum of our packet, which is calculated exactly the same as the UDP checksum calculation. For the 11th and 12th bytes, the first 14 bits are used to save the length (header + data) of the current packet, the 15th bit is used to indicate if this packet is an ACK packet, and the 16th bit saves the sequence number of this packet. Regarding the length of the data, you can assume that it will be at most 1,000 bytes. Note that CRLF is not needed for this packet format.

```
0                7 8              15 16              23 24              31
+----------------+----------------+----------------+----------------+
|       C        |       O        |       M        |       P        |
+----------------+----------------+----------------+----------------+
|       N        |       E        |       T        |       W        |
+----------------+----------------+----------------+----------------+
|            checksum             |          length        |ACK|Seq|
+----------------+----------------+----------------+----------------+
|                            Data (if any)                          |
+                                                                   +
|                                                                   |
+----------------+----------------+----------------+----------------+
```

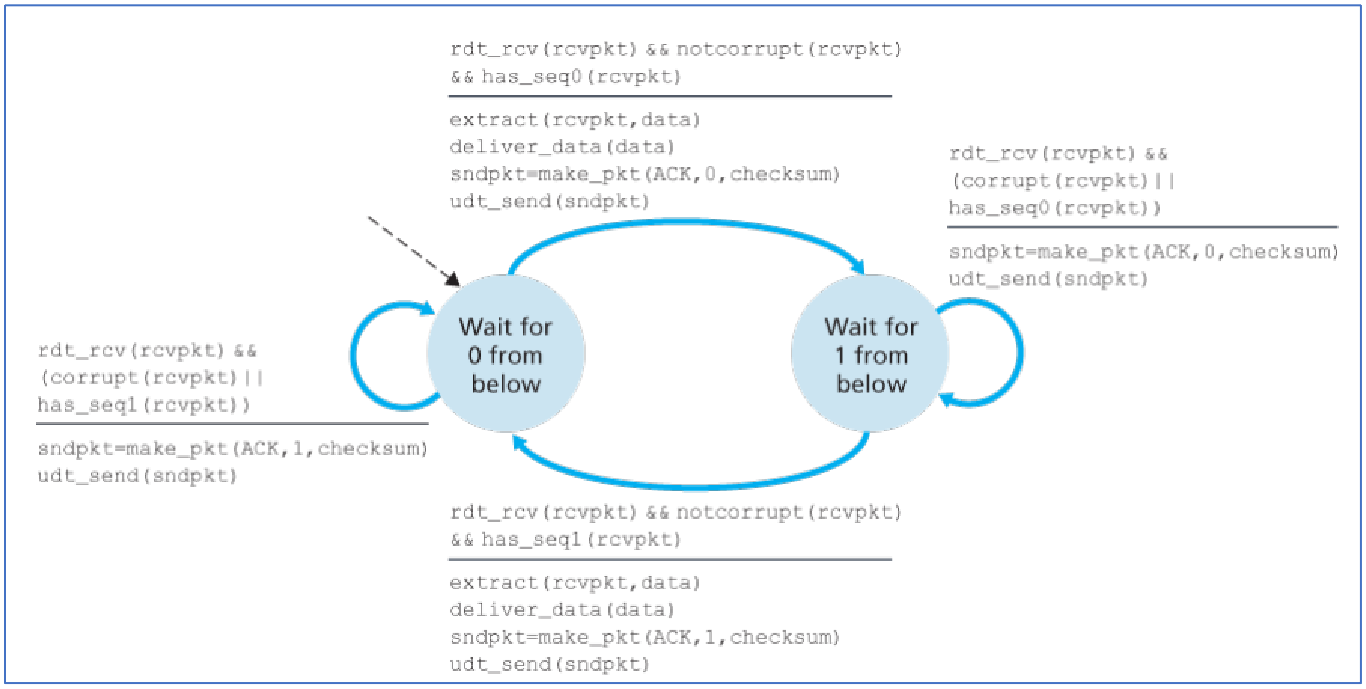
2. Three required Python files: sender.py, util.py, receiver.py

Please download the skeleton code and read the comments carefully. Do not change the function name or arguments as stated in the comment. In the zip file, a main.py is also included for your reference in in terms of how your sender service will be called. Note that you're not required to include this main.py in your submission.

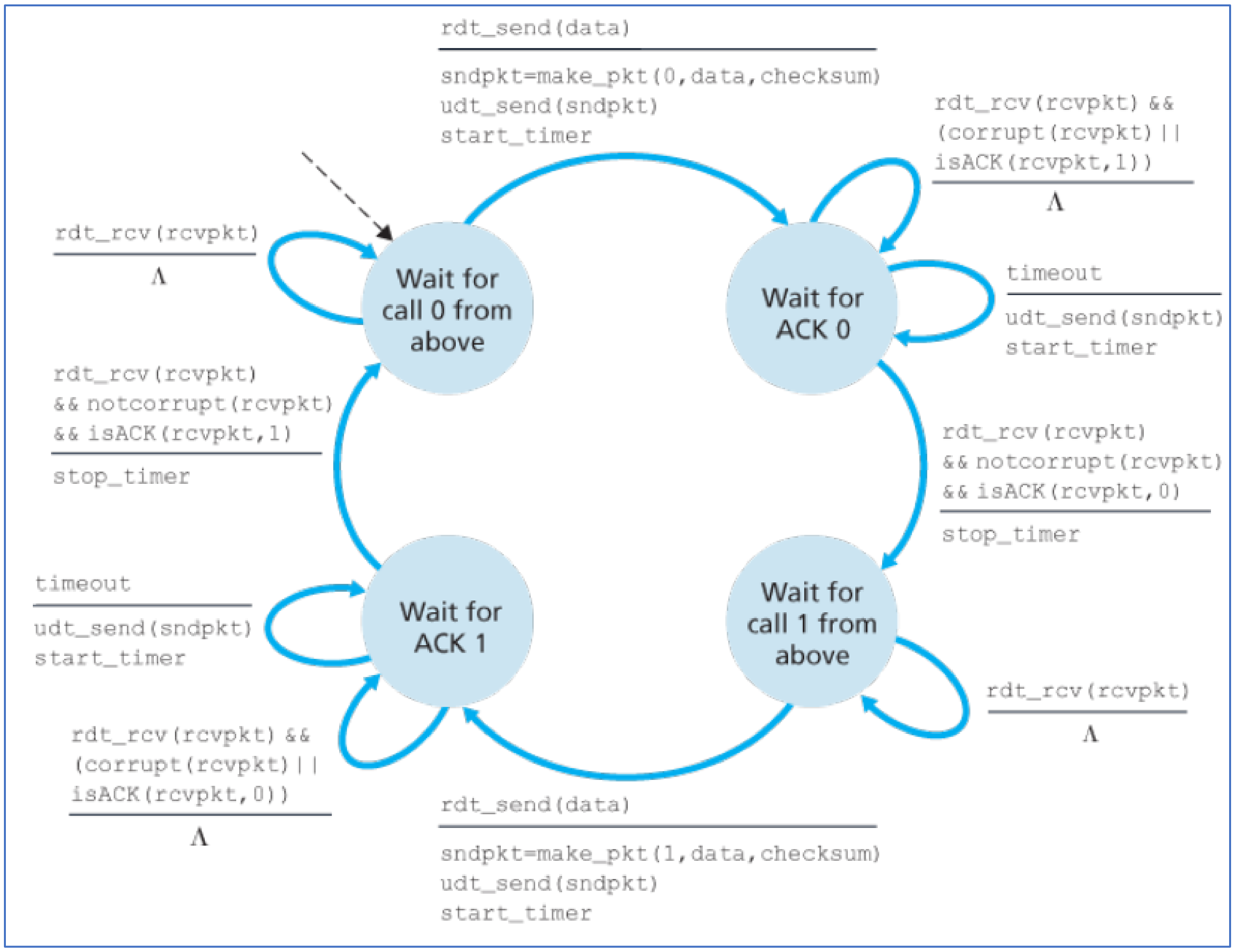
3. Required functionality

Your rdt implementation should have all the functionalities as shown in the FSM below, except for the timer and corruption detection. You don't need to have a dedicated timer. Instead, you can leverage socket timeout to simulate a timeout event. That is, to deterministically have a timeout event or packet corruption (e.g., bit errors), you are required to adopt the following strategies.

1) Number all the received packet from **1** on the receiver side
2) Simulate timeout for all the packets whose number is divisible by 6
   For the 6th, 12th, 18th, 24th … packet, you can use a time.sleep() in your receiver.py to intentionally sleep a while before responding back. This should be able to trigger a socket timeout on the sender side.
3) Simulate packet corruption for all the packets whose number is divisible by 3
   Bit errors don't happen quite a lot. Therefore, similar to the timeout simulation, for a packet whose number is divisible by 3, you should consider it as corrupted (even though you verified the checksum is valid) and process this 'corrupted' packet correspondingly. Note that if its number is divisible by both 3 and 6, simulate a timeout only.



Receiver side FSM (Figure 3.14 in the textbook)


rdt_send(data)
sndpkt=make_pkt(0,data,checksum)
udt_send(sndpkt)
start_timer
rdt_rcv(rcvpkt) && (corrupt(rcvpkt) || isACK(rcvpkt,1))
Λ
rdt_rcv(rcvpkt)
Λ
Wait for call 0 from above
Wait for ACK 0
timeout
udt_send(sndpkt)
start_timer
rdt_rcv(rcvpkt) && notcorrupt(rcvpkt) && isACK(rcvpkt,1)
stop_timer
rdt_rcv(rcvpkt) && notcorrupt(rcvpkt) && isACK(rcvpkt,0)
stop_timer
timeout
udt_send(sndpkt)
start_timer
Wait for ACK 1
Wait for call 1 from above
rdt_rcv(rcvpkt)
Λ
rdt_rcv(rcvpkt) && (corrupt(rcvpkt) || isACK(rcvpkt,0))
Λ
rdt_send(data)
sndpkt=make_pkt(1,data,checksum)
udt_send(sndpkt)
start_timer

Sender side FSM (Figure 3.15 in the textbook)

4. Required output format

Please see the example output below and use the **same** message format (including empty lines) to demonstrate your code step by step.

5. Allowed libraries

In this project, only the socket module and the sleep function in the time module are allowed. If you believe you need more libraries, post a discussion on Slack or Canvas discussion (so that everyone can see it), I'll address case by case.

from socket import *	It's okay to use all the functions in the socket lib
from time import sleep	-

6. Submissions

Create a zip file containing sender.py, util.py, and receiver.py and name it as *YOUR_SU_USERNAME*.zip (replace *YOUR_SU_USERNAME* with your real SU username). Submit this zip file to Canvas.

C. Grading Rubrics

All the submissions will be graded on CS1 server. Make sure your code can be executed on this server with the default python3 (version 3.6.8). Otherwise, it will result in zero points on this assignment. Note that the following rubrics are subject to minor changes (to cover exceptions) during grading.

Label	Notes
Functionality (24 pts)	1. [2 pts] sender and receiver can send each other packets 2. util.py implementation (1) [5 pts] create_checksum() and verify_checksum() function correctly (2) [5 pts] packet format is correctly followed (3) [2 pts] three required functions are available to import and test and test 3. [2 pts] ACK and sequence number are handled correctly on both ends 4. [6 pts] timeout and corruption simulations are correctly simulated 5. [2 pts] program output should have the similar (identical output is preferred) format, for example, packet is numbered, ACK and seq numbers are printed out, etc.
Code style (6 pts)	1. [2 pts] Code has appropriate modules/functions; do not use a single/long while loop or main function 2. [2 pts] Code is appropriately spaced and indented 3. [2 pts] Code is appropriately commented
Overriding policy	1. If the code cannot be executed or can be barely executed (exceptions, for instance), it results in zero points on this assignment. 2. The code that does not follow the packet format or use unallowed libraries will result in zero points
Late submission	Please refer to the late submission policy on Syllabus.
Academic Integrity	Strictly enforced. Please check more on Syllabus.

D. Development Tips

Port Numbers [Different from Project 1]

The port # (10101 - 10300) on CS1 are assigned to this course. In order to reduce the risk of using the same port #, we develop a simple algorithm to calculate the port # your proxy will use.

Port # = 10100 + (your student ID) % 200

If you still believe this port # is in use, then do a linear probing to try subsequent port numbers.

Little-endian or Big endian?

Please use big-endian if applicable.

Q&A

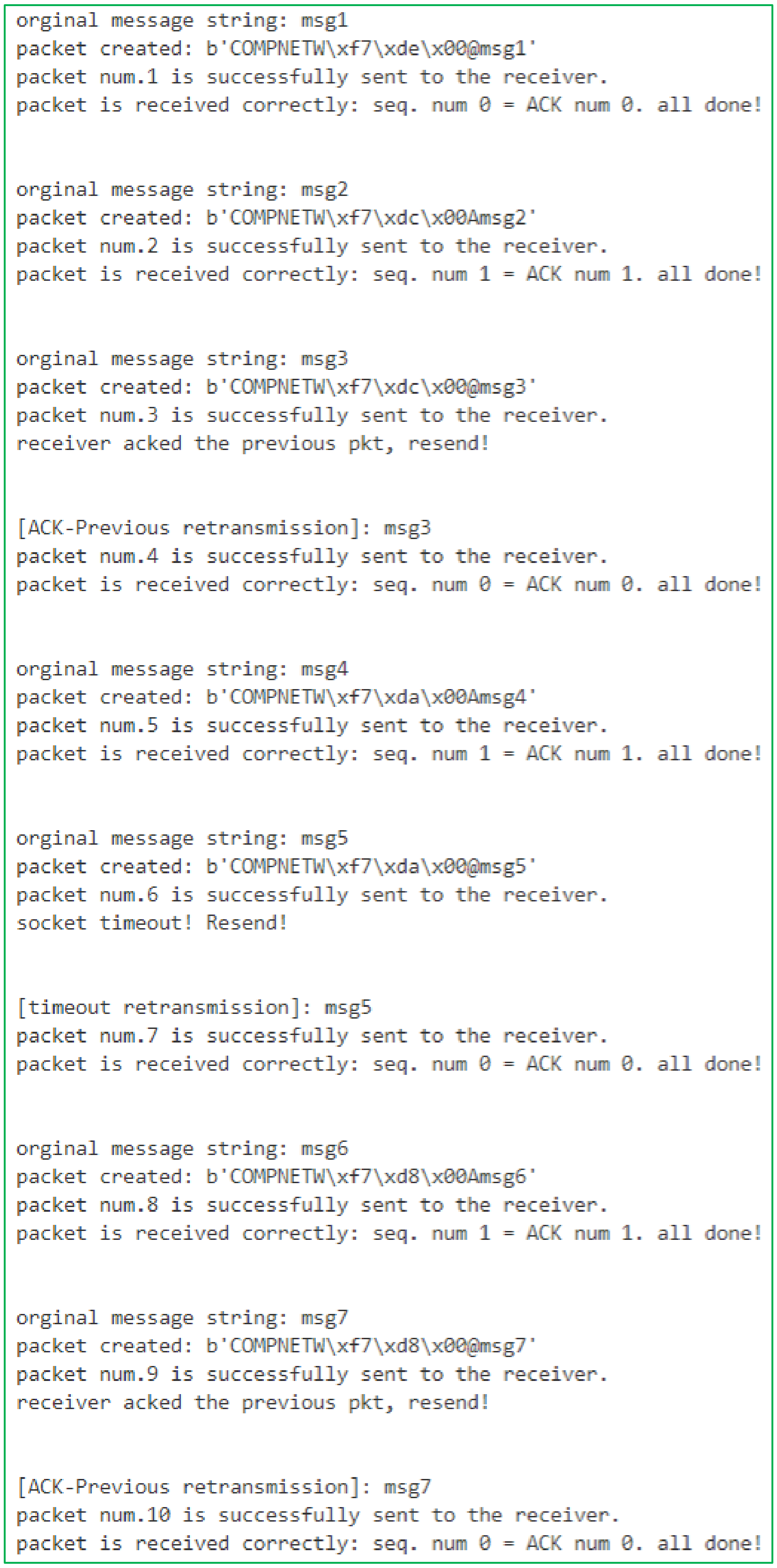
First of all, if you have a question, post it on Slack or Canvas discussion so that others could also be helped. Meanwhile, you're strongly encouraged to answer questions from your classmates.

1. Do we need multiprocessing or multithreading?
   No
2. Can I use any timeout related libraries?
   No. For simplicity, use socket timeout for this project.

3. How can I deliver the received packet to the targeted application?
   Simply print out a message saying something like ‘message delivered’ is good enough.
4. Academic integrity concerns. To what extent can we collaborate with my fellow classmates? Can I search partial solutions online?
   Yes, collaboration is encouraged. You can discuss the idea with each other, however, when it is the time to writing the code, you’re not expected to collaborate anymore. That is, you shouldn’t copy any code from your fellow classmates. Similarly, you can also search ideas online and you might find some code snippets. It is fine to get inspiration from those code. But you shouldn’t simply copy and paste the code into your solutions. Oftentimes, certain API calls tend to be identical. You’re not expected to make up new ways to call them. However, the line is that you don’t largely copy and paste from any sources.

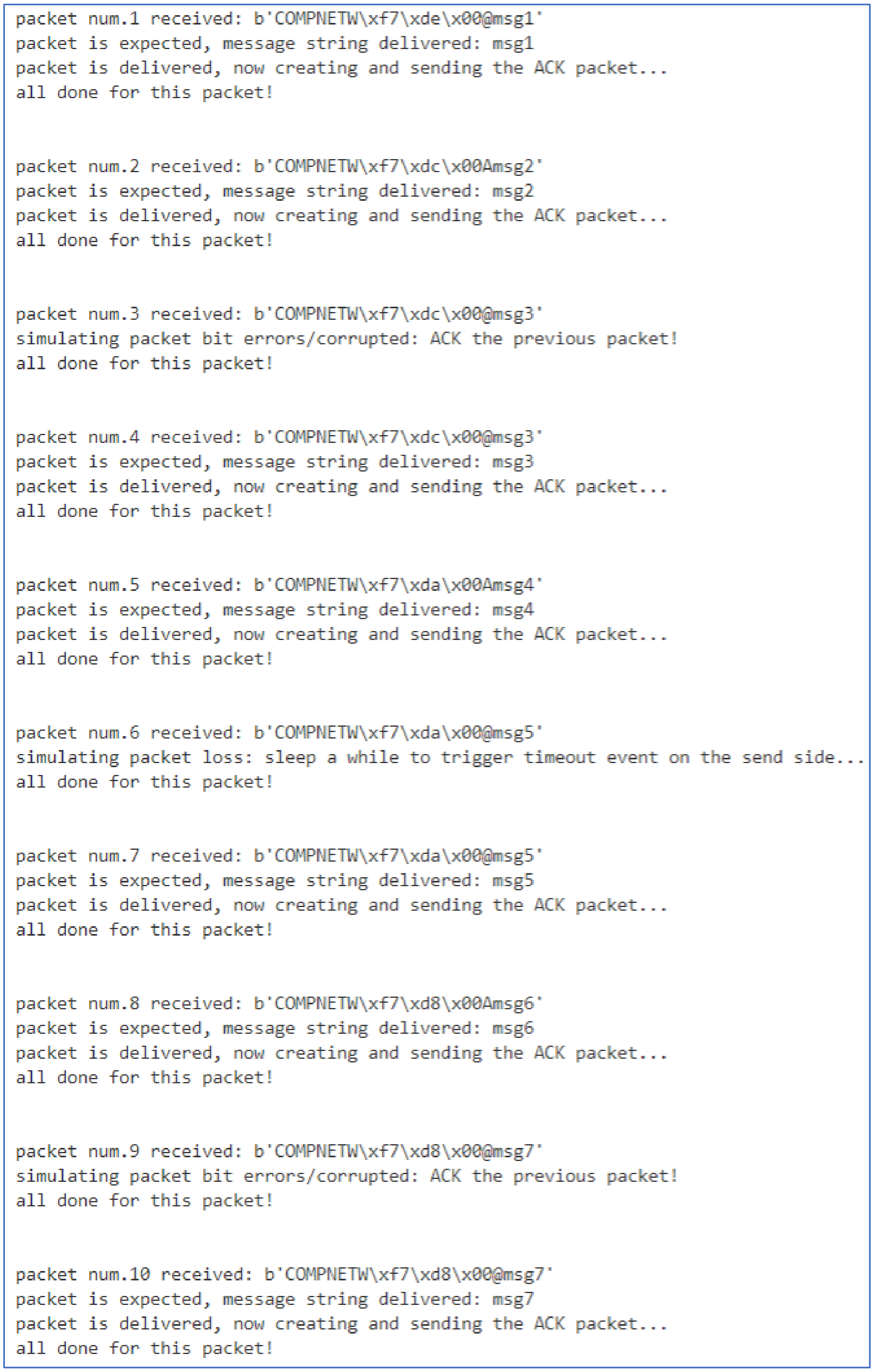
Example output

The following screenshot shows the example output on the sender and receiver sides. Note that your program will be tested with different application messages.



```
orginal message string: msg1
packet created: b'COMPNETW\xf7\xde\x00@msg1'
packet num.1 is successfully sent to the receiver.
packet is received correctly: seq. num 0 = ACK num 0. all done!


orginal message string: msg2
packet created: b'COMPNETW\xf7\xdc\x00Amsg2'
packet num.2 is successfully sent to the receiver.
packet is received correctly: seq. num 1 = ACK num 1. all done!


orginal message string: msg3
packet created: b'COMPNETW\xf7\xdc\x00@msg3'
packet num.3 is successfully sent to the receiver.
receiver acked the previous pkt, resend!


[ACK-Previous retransmission]: msg3
packet num.4 is successfully sent to the receiver.
packet is received correctly: seq. num 0 = ACK num 0. all done!


orginal message string: msg4
packet created: b'COMPNETW\xf7\xda\x00Amsg4'
packet num.5 is successfully sent to the receiver.
packet is received correctly: seq. num 1 = ACK num 1. all done!


orginal message string: msg5
packet created: b'COMPNETW\xf7\xda\x00@msg5'
packet num.6 is successfully sent to the receiver.
socket timeout! Resend!


[timeout retransmission]: msg5
packet num.7 is successfully sent to the receiver.
packet is received correctly: seq. num 0 = ACK num 0. all done!


orginal message string: msg6
packet created: b'COMPNETW\xf7\xd8\x00Amsg6'
packet num.8 is successfully sent to the receiver.
packet is received correctly: seq. num 1 = ACK num 1. all done!


orginal message string: msg7
packet created: b'COMPNETW\xf7\xd8\x00@msg7'
packet num.9 is successfully sent to the receiver.
receiver acked the previous pkt, resend!


[ACK-Previous retransmission]: msg7
packet num.10 is successfully sent to the receiver.
packet is received correctly: seq. num 0 = ACK num 0. all done!
```

Sender side example output



```
packet num.1 received: b'COMPNETW\xf7\xde\x00@msg1'
packet is expected, message string delivered: msg1
packet is delivered, now creating and sending the ACK packet...
all done for this packet!


packet num.2 received: b'COMPNETW\xf7\xdc\x00Amsg2'
packet is expected, message string delivered: msg2
packet is delivered, now creating and sending the ACK packet...
all done for this packet!


packet num.3 received: b'COMPNETW\xf7\xdc\x00@msg3'
simulating packet bit errors/corrupted: ACK the previous packet!
all done for this packet!


packet num.4 received: b'COMPNETW\xf7\xdc\x00@msg3'
packet is expected, message string delivered: msg3
packet is delivered, now creating and sending the ACK packet...
all done for this packet!


packet num.5 received: b'COMPNETW\xf7\xda\x00Amsg4'
packet is expected, message string delivered: msg4
packet is delivered, now creating and sending the ACK packet...
all done for this packet!


packet num.6 received: b'COMPNETW\xf7\xda\x00@msg5'
simulating packet loss: sleep a while to trigger timeout event on the send side...
all done for this packet!


packet num.7 received: b'COMPNETW\xf7\xda\x00@msg5'
packet is expected, message string delivered: msg5
packet is delivered, now creating and sending the ACK packet...
all done for this packet!


packet num.8 received: b'COMPNETW\xf7\xd8\x00Amsg6'
packet is expected, message string delivered: msg6
packet is delivered, now creating and sending the ACK packet...
all done for this packet!


packet num.9 received: b'COMPNETW\xf7\xd8\x00@msg7'
simulating packet bit errors/corrupted: ACK the previous packet!
all done for this packet!


packet num.10 received: b'COMPNETW\xf7\xd8\x00@msg7'
packet is expected, message string delivered: msg7
packet is delivered, now creating and sending the ACK packet...
all done for this packet!
```

Receiver side example output