CS471/EE471/CS573: Fall 2021-2022
Zartash Uzmi

Programming Assignment 3

(Due 8th December 2021 Wednesday – **max 50% credit for up to 24hr late**)

This assignment is intended to show the implementation of a reliable protocol over an unreliable channel (which can corrupt but not lose packets).

About cooperation with other students:

This assignment is meant to be done alone. Absolutely **NO** cooperation is allowed. If there are questions, ask the course staff; they are there to help you. For this assignment, you must do all thinking, research, and coding by yourself. ***Do not even discuss with anyone how far you are with this assignment!***

ANY HELP (EXCEPT FROM COURSE STAFF) IS PROHIBITED!!!

Individuals found guilty of violating above policy will be referred to the disciplinary committee.

(Warning: We will use software to measure the code similarity)

More Important Note:

Once again, we are giving you the responsibility to REPORT all incidences of cooperation. You MUST report to the instructor (or the TAs) such incidences. If you do not, we will consider you as guilty as those who you witnessed cooperating. All coding, debugging, web search for functions, etc. must be done by individual students!

For syntax related issues, make sure to consult www.google.com before asking the course staff. They will do the same!

Preamble:

You will use **UNIX programming environment** for this assignment.

You will need to submit all the files zipped in a file whose name is the same as your student ID. **See submission instructions for more details and follow them precisely.**

The Assignment:

This assignment focuses on implementing the state machines at the sender and receiver side of a reliable data transfer protocol. For simplicity, we will consider:

- The protocol does not pipeline packets (i.e., no sliding window is used). The sequence numbers are 0 and 1 (alternating-bit protocol)
- A channel that does not lose packets
- Only the sent packet and/or the acknowledgment packet can get corrupted.
- The protocol is NACK-free and the ACKs are numbered.

As discussed in class, rdt2.2 (a stop-and-wait and alternating-bit protocol) will work with above assumptions. In this assignment, we will implement sender and receiver side of rdt2.2. Please go over the text and associated slides to learn about rdt2.2.

The focus of this assignment is to get familiarity with the sender and receiver state machines rather than implementing the networking code that actually transports data over the network. Thus, we work in a simulated environment where both the sender and the receiver are part of the same codebase.

Starter Code and what you will write:

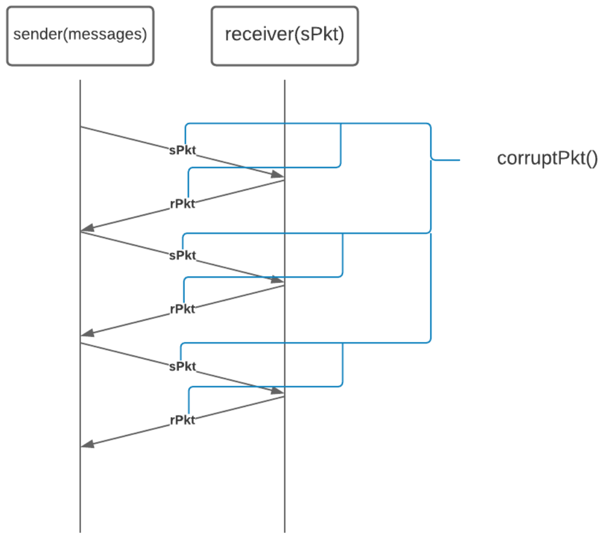
You are given two files: packet.py and rdt_2_2.py. You are required to add your code to both. You should not make any change to the class definition in packet.py. You may, of course, edit the calculateChecksum() function.

You will implement the following:

- The sender state machine – in the `sender()` function
- The receiver – in the `receiver()` function
- `calculateChecksum()` function

The Sender state machine is part of the `sender()` function, and you are responsible to write the code for this. A data packet (`sPkt`) must be created for each message in the message array and the `receiver(sPkt)` function is called to simulate the exchange of data and ack between the sender and the receiver. See figure for details.

The channel is simulated by the call to the receiver() function which takes a packet as an input and outputs the acknowledgment packet (which must be stored by the sender() that calls it). You may treat the functions as separate machines that pass packets through function calls and returns. Note that the acknowledgment packet can also get corrupted so you must calculate the checksum and resend packets accordingly. Even if your code outputs the correct message at the end, ensure that the ACKs received are correct and packets are resent when they are not.



You are also required to provide the code for computing the checksum. This code lies in the function `calculateChecksum()` and is executed every time a packet is sent to the receiver. First, when the packet is created, the checksum is computed using `calculateChecksum()` and is stored as a variable within the packet in the initialization. When the packet is received, the checksum is computed once again and compared against the stored value of checksum within the packet.

What and Where to submit?

1. Very importantly, **strictly** stick to the following submission guidelines.
2. Use the same starter files as provided on the course website. DO NOT change the **filenames** or our script may not accept your submission.
3. You need to submit all files separately. **Do not Zip or RAR your files.**

4. Submit each of the two files individually (packet.py and rdt_2_2.py) on LMS (use the ***Assignment*** link in your portfolio, and **NOT** the Dropbox).
5. **Do NOT email your assignment to us.**