CS4316: Computer Networks
Spring 2021

Instructor: Dr. Deepak Tosh
Due: 03/11/2021 (Part-1)
03/21/2021 (All Parts)

Assignment # 3 [Reliable Data Transfer]

GOAL:

The goal of this assignment is to implement both non-pipelined and pipelined versions of the reliable data transfer protocol: (1) Stop-and-wait (SnW) protocol and (2) Go-Back-N (GBN) Protocol. Additionally, interested students can attempt to implement the Selective Repeat (SR) protocol for **bonus**. We know that GBN and SR use sliding window to send more than one packets to a receiver over an unreliable medium where packets can get corrupted and lost too. However, the Stop-and-Wait protocol only sends one packet at a time and waits for the ACK to be received until it attempts to send the next packet.

Disclaimer: This is a group assignment. Each group has two members. Your **groups are pre-decided and are activated in** the **Blackboard**, where you can interact with your teammate.

There are two parts of the assignment and the deadline for both parts are different. The first part asks to implement the SnW protocol which is due on 11th March. And the second part includes implementation of GBN protocol (with bonus task of implementing SR protocol) which is due on 21st March 2021.

Environment Setup:

- The starter code (in Python3) is provide. There are a total of 5 important python programs. You can run them on same machine under two terminals/command prompts or on two separate VMs.
- The "receiver.py" needs to be running first, and then you can run the sender program ("sender.py") to send any number of packets.
- When you run both programs for the first time, you will observe that the sender sends 20 packets of random payload, and receiver prints them upon receival. This is given to make you understand how those two programs use the other auxiliary programs (*udt, packet, and timer* modules).

You are given with 3-auxiliary helper modules: ***timer***, ***packet***, ***udt***.

- Timer module: Provides *start(), stop(), timeout(), running()* functions
- Packet module: Provides *make(seqNo, data), extract(packet), make_empty()*
- udt module: Provides *send(packet, sock, addr), rcv(sock)* to send/recv from unreliable channel

Important Note: You are given with two text files: one is a small file (bio.txt) and other one is a large file of 10MB (10MB.txt). You will need to show that your implemented protocol is able to successfully send both files to the receiver. In order to check whether the "sentFile" and "recvdFile" are same, you can use the **diff** command in Unix/Linux/Mac, or "**FC**" command in

Windows. These commands will allow is there is difference in the sent and received files. If both files do not have any difference, you can assume that your protocol is correctly implemented.

Part-1 [File Transfer using Stop-and-Wait (SnW) Protocol] (35%)

In this part, your job is to implement the stop-and-wait protocol that we discussed in the class to transfer two files ("*Bio.txt*" and 100MB.txt). The state diagram of this protocol is same to **RDT3.0**, which you can refer from Section 3.4 of our textbook.

(i) Implementing SnW Sender

In this task, you will program following python functions in *"Sender.py"*.

a. Modify the function "*send_snw(sock)*" that will read the file, create packet, and transfer it to the receiver by following the Stop-and-Wait protocol. Note that each packet should have a payload of 512 Bytes, which is defined in PACKET_SIZE. You need to handle packet losses by using the timer module.
b. Implement the function "*receive_snw(sock, pkt)*" to handle the acknowledgements in Stop-and-Wait protocol. If the ACK gets lost, this function will retransmit the *pkt* again.

(ii) Implementing SnW Receiver

a. Modify the function "*receive_snw(sock)*" to receive all the payloads in correct sequence and write them into a file "*receiver_file_x.txt*" at the receiver, reconstructing the sender's sent file.

(iii) Measurements

As you are implementing SnW sender, you would need to count the following attributes to understand how this protocol is performing (conduct separately for transfer of both files):

a. Total number of packets sent (Both freshly transmitted and retransmissions)
b. Number of retransmitted packets (only when time expires, packet gets retransmitted)
c. Time taken to complete the file transfer

Part-2 [File Transfer using Go-Back-N (GBN) Protocol] (45%)

(i) Implementing GBN-Sender

a. Implement *send_gbn(sock)* that will read the file to transfer it to the GBN receiver. You will need to use the window size (N) send N packets (default 5) at a time. Ensure to implement the actions when packets get lost happens.
b. Implement *receive_gbn(sock)* that checks if the expected ACKs are being received or not. Based on that the sender should take necessary actions.

(ii) Implementing GBN-Receiver

a. Implement *receive_gbn(sock)* at the receiver which receives the packets and writes them into the file *"receiver_bio.txt"*. Ensure that if the expected sequence number is not received, it sends the last in-ordered packet's sequence # as ACK.

(iii) Measurements

As you are implementing GBN sender, you would need to count the following attributes to understand how this protocol is performing (conduct separately for transfer of both files):

a. Total number of packets sent (Both freshly transmitted and retransmissions)
b. Number of retransmitted packets (only when time expires, packet gets retransmitted)
c. Time taken to complete the file transfer

(iv) Comparison

You will need to make a comparison between SnW and GBN implementations based on the measurements you conducted. Although you can observe that GBN performs better, what other optimizations you can make on your GBN implementation to enhance its performance further, especially when sending large files?

(Bonus) File Transfer using Selective Repeat (SR) Protocol (+15%)

Repeat the same task of transferring the file between sender and receiver as above, but this time you need to implement the Selective Repeat (SR) protocol instead of the GBN.

a. Read the SR-sender's events and actions from Fig 3.24 in the Book and modify the "Sender.py" program accordingly.
b. Read the SR-receiver's events and actions from Fig 3.25 in the Book and modify the "Receiver.py" program accordingly.
c. **Measurements**

As you are implementing SR sender, you would need to count the following attributes to understand how this protocol is performing (conduct separately for transfer of both files):

i. Total number of packets sent (Both freshly transmitted and retransmissions)
ii. Number of retransmitted packets (only when time expires, packet gets retransmitted)
iii. Time taken to complete the file transfer

Other required items: **(15%)**

1. A **PDF formatted report** with all evidences, codes, execution samples, instructions for running the submitted programs, and references used. The title should be formatted as "*Group_x_Report_Assign2.pdf*". **(10%)**
2. Well-documented source codes and *README.txt* in the form of **one single Zip** file. **(5%)**

Group Activity: **(5%)**

- The students must work in group of 2. If you decide to work alone, you will have to complete part-1 and part-2 completely, but you may lose this 5% for group activity.
- Each student's contribution should also be reported with a comparative chart as shown below and should be reported through the report.

Task lists	Student-1's contribution	Student-2's contribution
…	…	…

Submission:

Submit your **Python codes** for all the implemented algorithms, *README.txt* and the report on **Blackboard only.**