Computer Networks
Spring 2024

Instructor: Dr. Deepak Tosh
Due Date: 03/22/2024

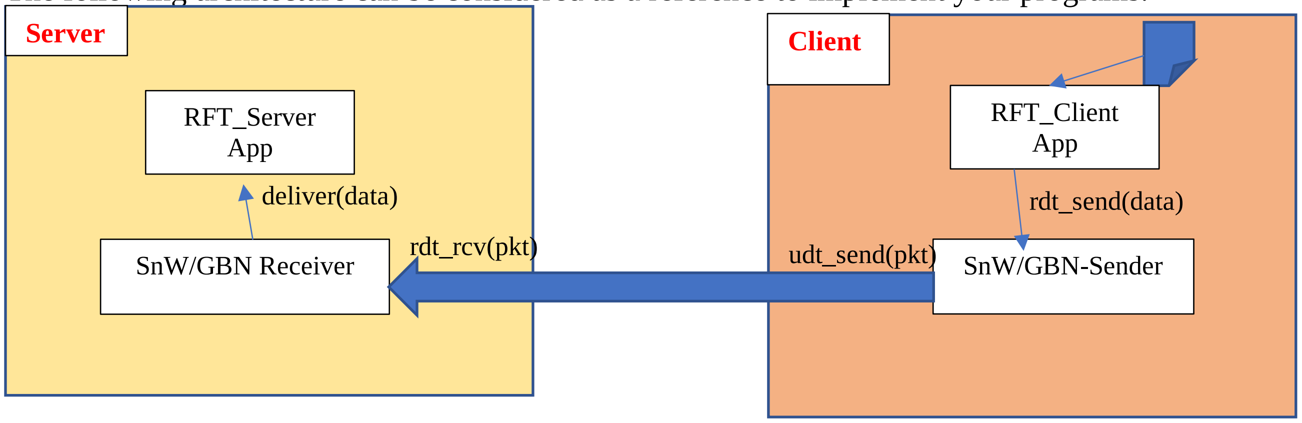
Assignment # 2b: Reliable File Transfer

In this assignment, your goal is to write reliable file transfer protocol that we discussed in class based on **Pipelined-RDT protocol** to ***upload*** a file from client to server. As you know the **Go-Back-N (GBN)** and Selective Repeat (SR) are the two protocols that are enhanced versions of the simple Stop-and-Wait (SnW) protocol, your objective will be to write transport layer programs that implement SnW and GBN protocols to upload the file reliably over unreliable medium. Please *note* that *the client must be able to transfer any kind of files (e.g., text, pdf, or media file) to the server*.

Task Details:

In the previous assignments you have learned on how to create a TCP socket and send data blocks between client and server. However, in this assignment, you will be developing your own pipelined reliable transport protocol using **Stop-and-Wait** (SnW) and **Go-Back-N** (GBN) protocols that will upload the file reliably from client to the server.

The following architecture can be considered as a reference to implement your programs.



Task – UG1 (File Upload from Server to Client using **Stop-and-Wait Protocol**)

Specifically, you should modify the server program to do the following,

Experimental Setup Hint: You can think that the client is seeking to upload a file to the server, which means, the client will implement the sender's logic and server will implement the receiver's logic. In this case, the receiver program must be started first and then the sender program.

1. When started, the server should ask for a port number and the protocol to use (such as SnW or GBN). Then the server will start listening for packets sent from clients and store it with a default file name based on client's IP/port#. The file could be a text, pdf, docx,

mp3, or mov file. You should choose different type of files to be sent in different execution runs and show evidence of file transfer.

2. Given the client is going to play the role of sender as per our SnW protocol's point-of-view, you will write programs/functions to ensure that client behaves like a SnW Sender (based on the protocol details discussed in class). In addition, you need to make sure the maximum segment size (MSS) is 1000 bytes at maximum.
3. As you may remember, the sequence numbers used for SnW protocol are either 0 or 1, so make sure to use these while creating segments.
4. Implement a **create_checksum() function** to include checksum in every data segment and ACK packet. You also need to implement a **verify_checksum()** function to verify if the checksum is valid. You may assume that the checksum is an 8-bit number that is calculated based on the sequence number, and payload. These functions need to be used in both sender and receiver programs.
5. Similarly, modify the server program to run the SnW-Receiver's logic, where it initially waits for a segment with sequence number 0. You can make some assumptions on what type of file is being transferred and accordingly create the file with appropriate extension.
6. After receiving the file, make sure to use **diff** command to check the similarity or dissimilarity between the sender and receiver files.

Note: When the file's data is completely transferred, you can use your own END-OF-FILE signal to identify this situation.

Task – UG2 (Measurements for Stop-and-Wait Protocol)

As you have implemented the SnW sender, you would need to add few counters in your program to measure the SnW's performance.

For reporting, you need to measure following attributes.

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

Below tasks are mandatory for 5313 students/groups.

Task-G1 (File Transfer from Server to Client using GBN)

Now, you can modify the above programs to add GBN's sender/receiver functionalities.

1. When started, the server should ask for a port number, the protocol to use (such as SnW or GBN), and window size (N). Then the server will start listening for packets sent from clients and store it with a default file name based on client's IP/port#. The file could be a text, pdf, docx, mp3, or mov file You should choose different type of files to be sent in different execution runs and show evidence of file transfer.
2. Here, the server is going to play the role of receiver as per our GBN protocol's point-of-view. So, you will write programs/functions to ensure server behaves like a GBN receiver.
3. The client should implement the GBN sender's logic to send packets in pipelined way (with Window Size = N, you should try with N=4, 8, 12, 16. In addition, you need to make sure the segment size is 1000 bytes at maximum.

4. To know if the file's data is completely received, you can use your own END-OF-FILE signal.
5. Make sure you incorporate the checksum creation and validation logic into both sender/receiver's program.
6. You need to carefully chose what sequence number range you would like to use and repeat using them after the segment uses the maximum value.

Note: You can leverage **concurrent programming** concepts like multi-threading, locks/semaphores to build your GBN sender functions. Or you can also leverage **select** function to handle socket read/write events asynchronously.

First, you need to modify the client program to do the following:

1. The client should ask the server's IP, port, protocol, and file_name to use as as part of reliable file transfer protocol.
2. As per the protocol chosen, you need to include the GBN receiver function in your client program.
3. As the client program sends the packets, it needs to include sequence number, checksum, and payload in network byte order before sending.

Task-G2 (Measurements)

As you have implemented the GBN sender, you would need to count the following attributes to understand how this protocol is performing (conduct separately for transfer of different files). You need to send a large file for the sake of better visualization with <u>N=4, 8, 12, 16, 24.</u> For each protocol (GBN and SR), you need to measure following attributes.

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

Then, use a spreadsheet or python's Matplotlib module to plot the following comparison figures

a. Window size (N) vs. time taken to complete file transfer
b. Window size (N) vs. # of retransmissions
c. Window size (N) vs. # of timeout

Sample Helper Module (Must be used)

No special python modules related to file transfer is allowed to be used, except the **sample helper modules given to you**.

You are given with 3-auxiliary helper modules: ***timer, packet, udt***. You must examine the helper modules to get acquainted on how and when to use them.

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

Note: Feel free to play with the server and client program to get familiar with the above modules. But it is not required that you will have to write programs on the server.py and client.py.

Environment to Execute the Programs

- Use the same environment that was used in Programming Assignment-2a.

After the client receives the file, it terminates. Then, you should test the correctness of the files by using the DIFF command - **diff -s recvdFile sentFile**

Submission:

As part of this assignment, you would need to submit the following:

1. **Python programs/modules** that result required outputs with proper code documentation. Lack of code documentation will cost 5% of the grade.
2. **You need to follow the provided report template** to document your strategy on solving the problem (with code snippets or algorithm), evidence, and multiple execution samples (i.e., sending different kinds of files), instructions for running the submitted programs, and references used. Comprehensive report with no grammatical error is expected and it carries 10% of this assignment's total grade. Also, provide evidence that the sent file and received files are exactly same (using **diff** command).

References:

[1] Socket programming in python: https://realpython.com/python-sockets/
[2] Python multi-threading: https://realpython.com/intro-to-python-threading/
[3] GBN and SR Animation: https://www2.tkn.tu-berlin.de/teaching/rn/animations/gbn_sr/