ASSIGNMENT 2

Name: Imon Raj
Class: BCSE III
Roll: 002010501098
Section: A3
Subject: Computer Networks Lab Report

PROBLEM STATEMENT: Implement three data link layer protocols, Stop and Wait, Go Back N Sliding Window and Selective Repeat Sliding Window for flow control.

Sender, Receiver and Channel all are independent processes. There may be multiple Transmitter and Receiver processes, but only one Channel process. The channel process introduces random delay and/or bit error while transferring frames. Define your own frame format or you may use IEEE 802.3 Ethernet frame format.

DESIGN: This problem was quite interesting and hard to implement. For this assignment I have used Object-oriented approach with Java. In places, I have extensively used Multithreading of Java. For data sending and receiving purpose Socket is used. As it is an assignment on FlowControl, I haven't sent any actual binary data, rather I am only bothered about sending frames. Now, let me share my design approaches –

Stop and Wait: The general theory of this protocol says –

Sender sends frames one by one. After sending one packet sender must wait for the acknowledgement of that packet from the receiver. Sender will wait only for a previously decided timeout period, if no acknowledgement is got, sender will resend the packet, supposing that either packet or acknowledgement is lost. In this way sender continues. Sender assigns consecutive 0 and 1 as sequence numbers to the frames. That means if acknowledgement for 0 is not got, it will resend 0 and so on.

In case of receiver, if it gets a frame, it will instantly send an acknowledgement. It will expect frames in alternative sequence of 0 and 1. If it gets one frame twice it will send the acknowledgement again, as it means that previous acknowledgement is lost.

For implementation, the sender and receiver , each of them runs in one thread. Sender sends a frame to the socket and waits for acknowledgement from the receiver. The receiver sometimes sends acknowledgement for the frame, otherwise it sends no acknowledgement behaving as if it hasn't got the frame. We have used Java random int function to decide whether to send acknowledgement or not.

Go Back N ARQ: This protocol is a little complicated –

Sender at a time sends N number of frames and always maintains a N size window. As soon as it receives acknowledgement for the first frame of the window, it shifts the window by one position, and sends one new frame

maintaining N-sized window. But if after a timeout, it receives no acknowledgement, it will resend the all the frames of the window again.

Receiver sends acknowledgement after receiving each frame. If one frame is sent repeatedly, receiver sends the acknowledgement again, because probably sender is resending the complete window again.

Here for implementation, sender has two threads running concurrently- one for frame for sending frames, one for receiving acknowledgement. Receiver has one thread only. Receiver, here also, sends acknowledgements with a random probability. There are two sockets at two different ports. One socket is to send frames by sender, another to send acknowledgements by receiver.

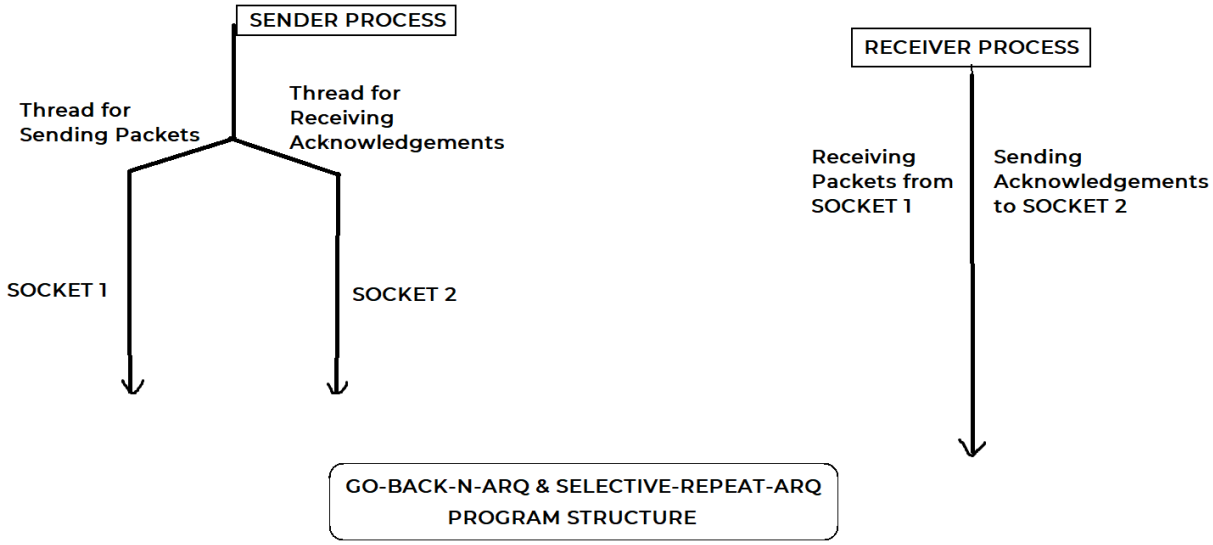
Selective Repeat ARQ: This is similar to the previous one. Here also a window is maintained of size N, but here if acknowledgement of first frame of window is not received by sender, sender only sends those frames within the window whose acknowledgements are not at all received.

Receiver sends an acknowledgement after receiving a frame. If one frame is sent twice, receiver sends acknowledgement again.

For implementation, most things will be same as the previous protocol. The change will be in resending frames. In this case, only unacknowledged frames are retransmitted after a timeout.

IMPLEMENTATION:

The threading and socket structure of Go-Back-N-Arq and Selective-Repeat-Arq is given below. As, Stop-and-wait is so simple, with sender-receiver each using one thread and one common socket, its figure is avoided.



All related codes written in Java are attached herewith (In the Drive-shared-folder).

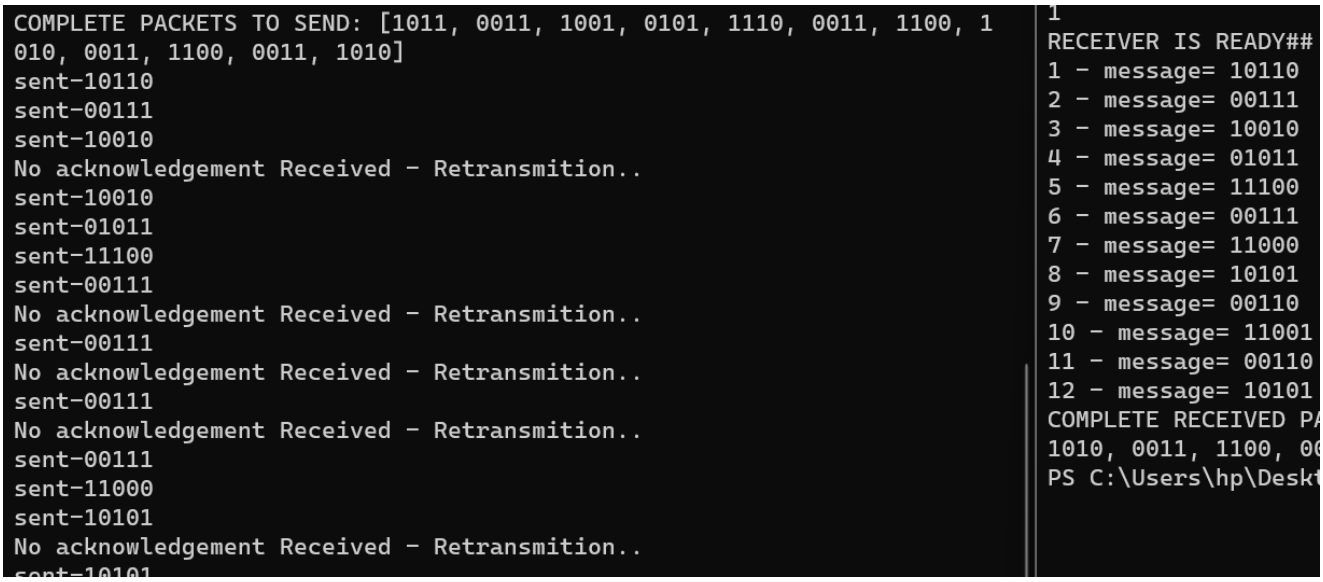
TEST CASES:

- There shouldn’t be any Deadlock condition, i.e-> In the multithreaded environment, No thread of either Sender or Receiver should wait indefinitely.
- The Program should correctly terminate, i.e-> All sockets used must be closed properly, there should not be any crash in the program.
- After all, the frame sending sequences should follow the rules of the protocols.

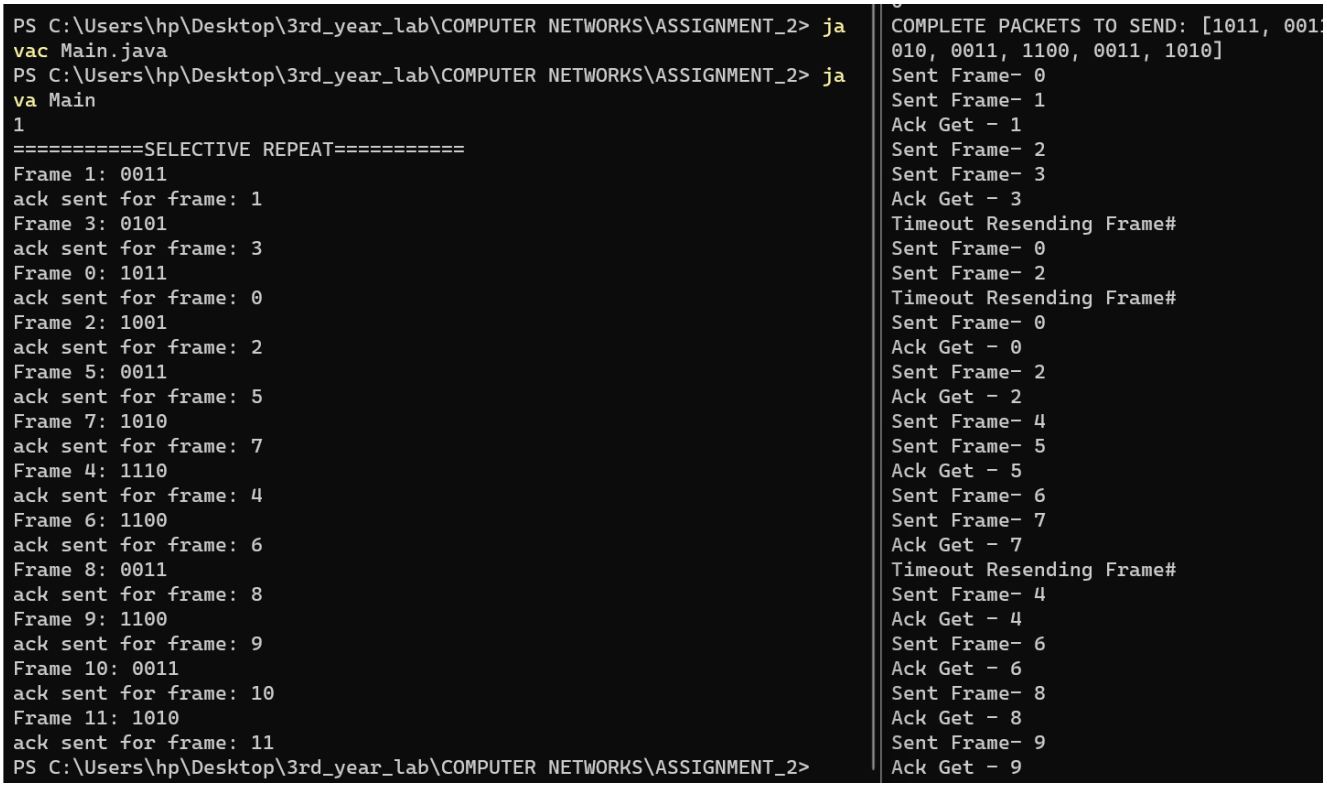
RESULTS & ANALYSIS:

Program can fulfil all the test cases conditions provided above. Given randomness in acknowledgement sending, different sequence of output is being generated, but all are correct. However, I can see improvement possibilities in the acknowledgement procedure.

SAMPLE OUTPUT(STOP AND WAIT)



SAMPLE OUTPUT(selective repeat)



Output of GO-back-N is somewhat similar to upper one.

COMMENTS:

This assignment took a lot of time for me to complete. The Stop-and-wait was simple but, the other two protocols' implementation made me think deeply about various approaches. I faced bug few times, and was able to fix them one by one. This assignment helped to polish my concepts on multithreading and synchronization techniques. According to me, this assignment was a little hard but yes, it was interesting to complete it.