Comp 7005 - Data Communication Principles

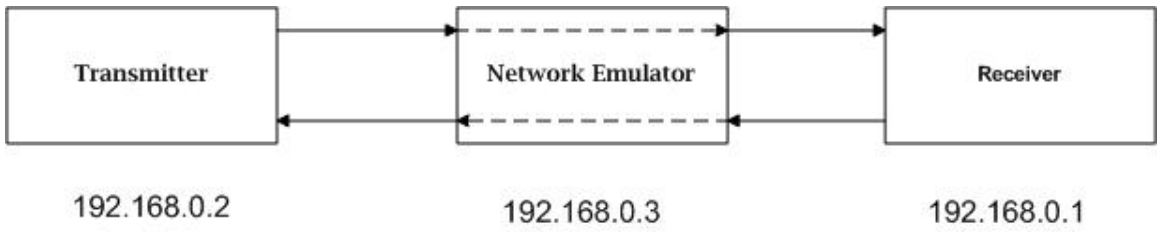
Final Project

Due: November 27, 2019 at 0800 hrs. Late submissions will not be accepted.

You may work in groups of two.

Objective

The objective of this project is to design and implement a basic Send-And-Wait protocol simulator. The protocol will be half-duplex and use sliding windows to send multiple packets between to hosts on a LAN with an "unreliable network" between the two hosts. The following diagram depicts the model:



Your Mission

- You may use any language of your choice to implement the three components shown in the diagram above. It is strongly recommended that you use your code from the first assignment to implement the peer stations.

- You will be designing an application layer protocol in this case on top of UDP (in keeping with the wireless channel model). The protocol should be able to handle network errors such as packet loss and duplicate packets. You will implement timeouts and ACKs to handle retransmissions due to lost packets (ARQ).

- The network emulator will act as an unreliable channel over with the packets will be sent. This means that transmitter will send the packets to the network emulator which in turn will forward them to the receiver. The receiver in turn will send ACKs back to the transmitter via the network emulator.

- Your implementation of the network emulator will include a "noise" component which will randomly discard packets (and ACKs as well) to achieve a specified bit error rate. This can be specified as part of command line arguments.

- Your overall application architecture will have a minimum of three source modules: transmitter, receiver, and network as well as any associated include files and libraries if necessary. For the purposes of simplicity it is recommended that the IP addresses and of the transmitter, sender and network be extracted from a common configuration file. The port numbers can also be part of the configuration file.

- One side will be allowed to acquired the channel first and send all of its packets. An End of Transmission (EOT) will indicate that it has completed sending all of its packets, after which the other side can start sending packets.

- The following structure depicts a suggested packet format:

```
struct packet
{
        int PacketType;
        int SeqNum;
        char data[PayloadLen];
        int WindowSize;
        int AckNum;
}
```

- The **PacketType** field indicates the type (numeric code) of the packet, i.e., ACK or Data or EOT.

- The **SeqNum** field is a sequence number used to number data packets.

- The **AckNum** field is used to indicate the previous data packet data packet being acknowledged and the next expected sequence number.

- The **WindowSize** field would typically be used at the start of the session to establish the number of packets that will be sent from the transmitter to the receiver.

Constraints

- The basic protocol is Send-and-Wait, however it is a modified version in that it will use a sliding window to send multiple frames rather than single frames. You will still have to implement a timer to wait for ACKs or to initiate a retransmission in the case of a no response for each frame in the window.

- Your window will slide forward with each ACK received, until all of the frames in the current window have been ACK'd.

- Both the transmitter and receiver will print out ongoing the session as simple text lines containing the type of packet sent, type of packet received, the status of the window, sequence numbers, etc. The format of this display will be left up to you.

- Your application will maintain a log file at both the transmitter and the receiver. This can be used for both troubleshooting and for validating your protocol.

- Your network module will take arguments such as the BER (Bit Error Rate), average delay per packet, and will also have a configuration file specifying IP addresses and port numbers for the transmitter and receiver.

- You are required to submit an extensive test document complete with screen shots validating all of the protocol characteristics you have implemented. Examples are successful transactions, retransmissions, timeouts, etc. You may also want to include Wireshark captures for this.

- Lastly, I have kept the project itself fairly open-ended to allow you to be as creative as you wish. In other words other than the half-duplex, multiple packet ARQ protocol requirement I am quite flexible in allowing you to add features and choose your own implementation method. As always if have ideas on how you will like to implement this by all means discuss it with me in class.

- You will be required to demo your project in the lab on the date it is due.

- Bonus marks will be awarded for implementations that show creativity and imagination.

To Be Submitted:

To Be Submitted:

- Hand in a complete document package in pdf format.
- Submit a **zip** file containing the documents as well as your Wireshark captures in the sharein folder for this course under "**Final Project-FT**".

Marking Guide:

Design work and instructions:	15
Test Document:	15
Protocol Functionality:	60
Report Format:	10

Total:	100