## Term Project
## Data Link Layer Protocols Simulation

In this project, you will develop, simulate and test data link layer protocols between peers that are connected with a noisy channel, where the transmission is not error-free, packets may get corrupted, duplicated, delayed, or lost, and the buffers are of limited sizes.
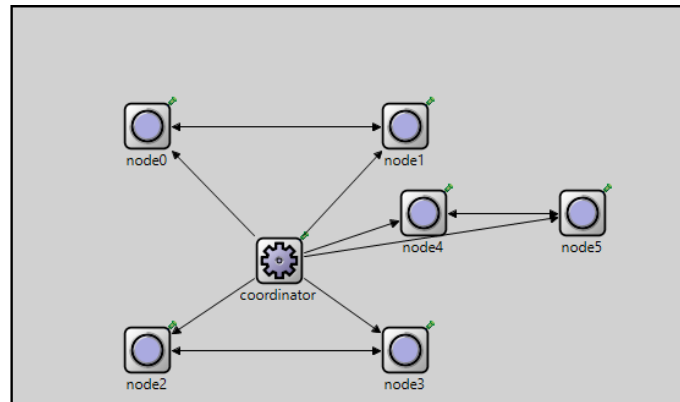


**Figure 1- The design of the system's network**

The system network topology is shown in figure1. It consists of three pairs of nodes [node0, node1], [node2, node3], and [node4, node5], and one coordinator that is connected to all the pairs.

By the end of phase2 of the project, each pair of nodes would communicate and exchange messages using the **Selective Repeat** algorithm with noisy channel and window of size M, using **Byte Stuffing** as a framing algorithm, and **CRC checksum** as an error detection algorithm.

## System inputs

1. Each node has a list of messages to send, and each node reads its list of messages from a different input text file. An example of such input file is shown in figure2.
   Each message starts in a new line, and there is a 4-bits binary prefix before each message. These 4-bits represent the possibility of  [*Modification, Loss, Duplication, Delay*] that would affect this message.
   For example "1010 Data Link" means that the message "Data Link" will have a modification to one of its bits while sending, and will be sent twice.
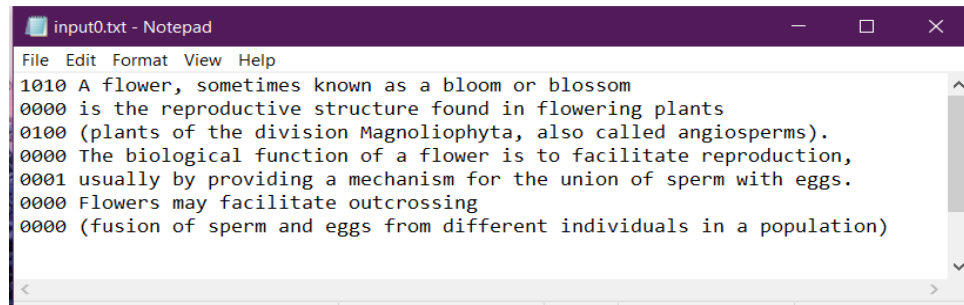
**Figure 2**

2. The coordinator starts working in the initialization stage, its main job is to assign input file names to their relative nodes, and choose which node of each pair should start, and when to start in seconds. The coordinator gets this information from an input file "coordinator.txt" and its format is as shown in figure3.
Every line contains whether ["*Node_id*" "*input_file_name*" "*start*" "*starting_time*"] if it is the starting node, or ["*Node_id*" "*input_file_name*" ] if it is the other node in the pair.

The start time is the exact starting time from the beginning of the simulation, i.e. it is not the cumulative time.

After the coordinator sends the initialization messages, each peer should start exchanging their messages on the specified starting time **directly**, i.e. messages between peers don't pass through the coordinator.
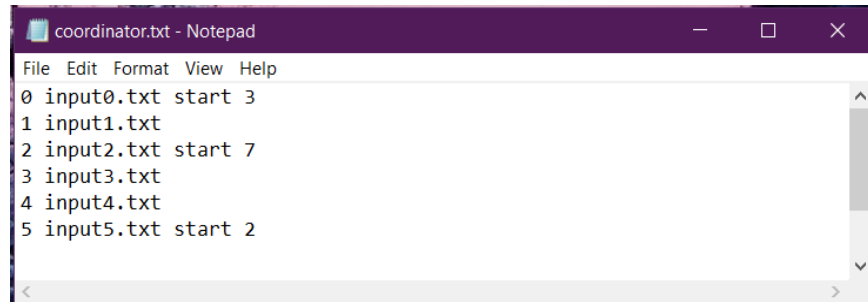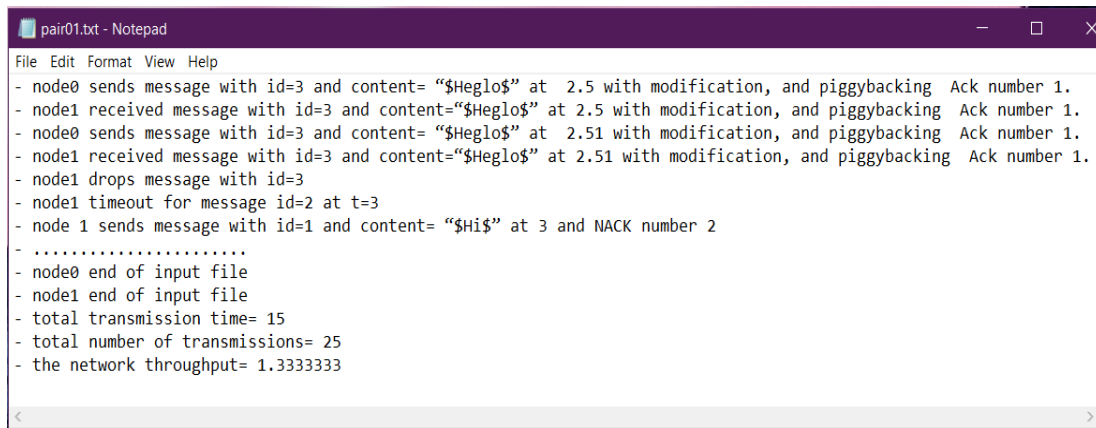


**Figure 3**

3. Parameters that are set in the .ini file:
   ● The timeout interval in seconds for the selective repeat protocol.
   ● The window size M for the selective repeat protocol.
   ● The delay time in seconds for any delayed packet.

## System outputs

1. For each pair, the system should print one log file named [pair01.txt / pair23.txt / pair45.txt] respectively, containing:
   ● The details for each message transmission from both nodes.
   ● The total transmission time.
   ● The total number of transmissions, including the [correct messages, duplications, retransmissions, and losses].

● The network throughput= number of correct messages/ total transmission time.

An example for the pair01.txt file is shown in figure4.



```
pair01.txt - Notepad                                                    —    □    ✕
File  Edit  Format  View  Help
- node0 sends message with id=3 and content= "$Heglo$" at  2.5 with modification, and piggybacking  Ack number 1.
- node1 received message with id=3 and content="$Heglo$" at 2.5 with modification, and piggybacking  Ack number 1.
- node0 sends message with id=3 and content= "$Heglo$" at  2.51 with modification, and piggybacking  Ack number 1.
- node1 received message with id=3 and content="$Heglo$" at 2.51 with modification, and piggybacking  Ack number 1.
- node1 drops message with id=3
- node1 timeout for message id=2 at t=3
- node 1 sends message with id=1 and content= "$Hi$" at 3 and NACK number 2
- .....................
- node0 end of input file
- node1 end of input file
- total transmission time= 15
- total number of transmissions= 25
- the network throughput= 1.3333333
```

Figure 4

2. Print the details for each message transmission from both nodes to the simulation console. "I.e. the same upper messages in the log file".

## Messages
Every message contains the following fields:

● Header: the message-id, sending time.
● Payload: the message contents after byte stuffing (in characters or bytes).
● Trailer: the CRC checksum byte.
● Piggybacking: ACK /NACK.
● Piggybacking id

## Noisy channel errors
For the simulation purpose, the errors are done on the sender's side before actually sending the packet. If more than one error happens, the execution of the errors should follow the exact same order below.

1. Modification: This is done to any randomly selected single bit on the **payload** after the byte stuffing. The modification could not happen to other message fields.
2. Duplicated: the whole message should be sent twice with a small difference in time (0.01s).
3. Delay: the whole message should be delayed using the delay value from the .ini file. [dont use busy waiting].
4. Loss: the whole message should not be sent using the send function, but it should be included in the log file and the system calculations.

## Framing
● Is done using the byte stuffing algorithm with starting and ending bytes.
● The flag byte is '$' and the escape character is '/'.
● The framing is applied to the message payload only.

- There is no specific maximum transmission size for the frame, each message is represented with one line in the input file.

## Error detection
- Is done using the CRC checksum algorithm.
- The CRC byte checks for the payload after applying the byte stuffing.
- The CRC byte is added to the message trailer field.
- The receiver uses the CRC byte to detect and print log messages if there is/isn't any single bit error during the transmission.

# Deliverables:
**I- Teams' formation deadline –week7**

**II- Phase 1 [7 Pts] – end of week 9: a modified simplex protocol**
- In this phase, the network consists of the coordinator and one pair of nodes only [node0 and node1].
- **[1 Pt]** The coordinator reads the initialization from the 'coordinator.txt' file and sends the information to the pair.
- **[1 Pt]** The starting node reads the input file and starts sending at the specific given time, **[2 Pts]** and takes into account the four types of errors. It will not send the next message until it receives a response from the other node.
- **[2 Pts]** The other node will only confirm receiving the messages, detect the errors if any, and respond with "ACK" if no error and with "NACK" if there is an error.
- The types of errors that the receiver node detects are single bit modification, and receiving a message out of order "duplicates".
- For simplicity, Whenever the sender receives "NACK" or "ACK" it starts to send the next message anyways, i.e., there is no special effect of receiving "NACK" in this phase.
- **[1 Pt]** print the log file at the end of the session.
- In this phase, the session ends when the starting node finishes sending all the messages in its input file.
- The window size in this phase is infinity, and packets take increasing id numbers starting with zero.
- There is only one node sending payloads in this phase.
- There is no selective repeat protocol in this phase.
- There is no timeout in this phase.
- Implement the message with the full fields even if you will not use all of them in this phase.

## III- Phase 2 [13 Pts] – start of week 13: Selective repeat protocol
- **[2 Pts]** Add the other two pairs to the network topology and allow for simultaneous execution of more than one pair at the same time.
- **[6 Pts]** Implement the selective repeat protocol taking into consideration the time-out and the window size with the noisy channel.
  **[2 Pts]** Make both nodes in any pair being able to send messages to each other.

**[2 Pts]** The messages are piggybacked, and each ACK or NACK is given an id according to the selective repeat protocol and according to the window size.

- **[1 Pt]** The session of each peer ends when both the nodes in any peer finish sending all the messages in their input files even if one file is shorter than the other.
- **Bonus [2 Pts]**: implement the Hamming code algorithm for single-bit errors correction, print the error position and the corrected string in the log file and in the simulation, and make a parameter at the .ini that is used to select between the CRC checksum and the hamming code.

The exact submission dates and guidelines will be posted on google classroom later.