**CS 313 Project: Network Chatbot over Noisy Channel (Deadline: 11:00AM 4/23/2018)**

(This is an individual project, not a group project. Please review Computer Science Department’s Collaboration Policy on the syllabus for guideline.)

**Objective**

To develop a network Chatbot application by implementing a client-server chat protocol for noisy channels

**Project Description**

This project is to develop a network Chatbot application over a noisy channel under the client-server model. The development implements a simple chat protocol similar to the Stop-and-Wait protocol studied in the class. In the client-server model, the reply from the server usually serves as the positive acknowledgement, and therefore no separate positive acknowledgement is required. Also, to simplify the task, timers are not used and a negative acknowledgement instead is sent to the client for errors detected in the client’s request.

The two major programs in the application are the client and the server. The client sends a message to the server through a noisy channel. Errors are purposely generated randomly in the CRC-checksummed message based on the channel’s error rate. When the server receives a message, it calculates the CRC checksum to detect errors. If no error, a reply generated from the Chatbot’s *rules engine* in response to the client’s message is sent to the client. Otherwise, a negative acknowledgment is sent back and the client retransmits the request. The chat continues until a termination message is sent by the client.

For the purpose of proof-of-concept and simplification, we unrealistically assume that no data is lost. Also, we assume that no error occurs in the reply and negative acknowledgements.

**Protocol Description**

**Message Formats**

Three types of messages are used in the protocol. Their formats are described as follows:

1. Client request (fixed size = 45 bytes), sent from client to server

[ ID | LENGTH | PAYLOAD | CHECKSUM ]

* + ID: client identification; size = 3 bytes
  + LENGTH: actual length of payload in bytes; size = 1 bytes; value is 0 to 40
  + PAYLOAD: client message; size = 40 bytes
    - Special value: a termination message contains “BYE” in any combination of cases.
  + CHECKSUM: calculated using CRC-8; size = 1byte

1. Server reply, sent from server to client

[“Hi”|CLIENTID|“! ”|PAYLOAD]

* + CLIENT ID: client identification; size = 3 bytes
  + PAYLOAD: server message; variable size

1. Negative acknowledgment, sent from server to client

[ NAK ]

* ACK:size=1byte;value=1

**Client Procedure X DONE**

1. Prompts for client ID
2. Prompts for client chat message up to 40 characters long
3. Puts the message into the PAYLOAD field and completes the header information of the request.
4. Calculates CRC checksum for the request and places it in the CHECKSUM field.
5. Use an *error function* to introduce errors based on error rate
6. Transmits the request.
7. Receives reply from the server.
8. If the server reply is NAK then  retransmits: go to step 3 (you may go to step 5 if the request is "buffered") else  displays the message  go to step 2
9. The procedure from step 2 to step 8 repeats until a termination message is sent by the client.

**Server Procedure X DONE**

1. Receives a request and calculates its checksum.
2. If “remainder” is 0 (i.e., no error) then

generates a chat message from the rules engine

sends reply (i.e., server’s chat message) to the client

* + - * 1. else

sends NAK to client

1. The procedure from step 1 to step 2 repeats until the server receives a termination message.

**Requirements**

1. Use the Link class provided by the instructor for the data communication.  -Copied and renamed CBLinkReceiver and CBLinkSender
2. Use the CRC-8 function provided by the instructor.
3. Implement the protocol exactly as described, including the data format and procedure.
4. Design and develop a Chatbot rules engine that generates a message to respond to the client chat message.  The rules engine contains at least 5 different rules for the server to chat back, possibly based on the content, prefix suffix, length of the client chat message. The rules engine also handles the termination message properly.
5. Include a trace option.

* For client trace on, the client program displays:
  + “OK”, if no error generated
  + “Error, need retransmission”, if error generated
* For server trace on, the server displays the following information:
  + Message sender is: *client\_ID*
  + Message length is: *client\_message\_length*
  + Checksum is: *checksum\_value*

1. Error function uses a random number generator to determine whether to damage a request.

* If a request is to be damaged, the probability of 1 error is 0.75 and 2 errors is 0.25.
* Theoretically, every bit in a damaged message is equally likely to be altered. You may alter bits in the  first byte of PAYLOAD without the loss of generality.

1. Input from user:

* Client: probability that a request will be damaged (i.e. error rate), trace on/off, client ID
* Server: trace on/off

1. Test your programs 5 times, each time with a different error rate: 0%, 50%, 60%, 75%, and 90%. 10 client chat messages should be sent in each test. Of course, the last message should be a termination message.
2. Display the following statistics after the communication terminates:

* Total number of client chat messages –(# messages sent)
* Total number of client request transmitted  –(# transmissions include retransmitted
* Theoretical total number of transmission (= Total number of client chat messages / (1 - error rate))
* Total number of chat messages damaged  –(#transmissions – #actual messages ) basically # of errors created
* Maximum number of retransmission for any single chat message 10.

This project is to be done individually. Do not copy any other student's work or copy from other sources.

**Submission**

1. Compress your code, report, the executable files, and results of test runs into one single zip file. Submit the zip file to D2L Assignments. Submission of multiple files will not be accepted.
2. The report should include
   1. Instructions to run and test your program
   2. Description of chat rules
   3. Analysis on the statistics from Requirement 8
   4. What you learned from this project
   5. Obstacles encountered during working on this project and how they are tackled
   6. Comments and suggestions
   7. Known bugs

**Grading**

* Correctness, robustness, documentation, and report.
* The instructor may contact you to ask for a demonstration.

**Suggested Milestones**

* X Week 1: Understand the project and its requirements, and can run the programs provided to you without problems. Figure out how to work with byte arrays. Design and develop chat rules engine.
* Weeks 2: Implement the protocol and test it with no errors first, and then with errors introduced. Implement trace option at this stage may help debugging.
* Weeks 3: Continue the tasks from Week 2. Add other requirements.
* Weeks 4: Test. Wrap up. Write report.

**Development Tips:**

* Start as early as possible!
* You may modify the provided LinkSender and LinkRecevier for client and server, respectively.
* Use iterative development approach.
* Ask/post questions whenever you have one.