CS513: Introduction to Local and Wide Area Networks
Spring 2014

Project 2: "Reliable Data Transfer Protocol"

DUE DATE: April 8, 2014 (11:59 AM)

Description

In this assignment you will be asked to design and implement a reliable data transfer protocol-using C/C++. You will implement a **3-layer network stack** consisting of: application, data link and physical layers. The reliable data transfer protocol will be implanted at the data link layer (instead of the transport layer, where we studied them in class) mainly because we are dealing with one-hop communication. This assignment is worth - **100 points.**

I recommend reading the assignment carefully multiple times to understanding exactly what needs to be done.

Submission

As a part of the submission you need to provide the following:

1. All the code for the three layers (.c and .h files)
2. A *makefile* that compiles the code (Tutorial on writing a makefile can be found here: http://mrbook.org/tutorials/make/)
3. A ReadMe (txt) file that spells out exactly how to compile and run the code.
4. A *Graph file* that shows the performance of each reliable data transfer protocol.

Combine (via zip or tar) everything up into a single archive file named "your-team-number_project2.zip".

Submit your document electronically via Instruct Assist (https://cerebro.cs.wpi.edu/cs513) by **11:59 AM** on the day the assignment is due. Make sure you choose "Project 2" under project drop-down in IA before uploading the zip file.

Requirements

This project consists of three layers – application, data link and physical layer. The application layer consists of a protocol between a client/server pair of processes. Communication is in the form of a request and response messages. The data link layer provides frame-level communication between the client and the server hosts. The data link layer operates over an unreliable channel subject to loss and corrupted data. The physical layer provides a **full-duplex**, communication channel on which data link layer can send an arbitrary byte stream. It is the physical layer that is actually implemented using the TCP/IP protocol.

Application Layer

The application layer implements an interactive request/response protocol. The client reads commands from standard input, converts them into server requests, sends them to the server, and prints the response returned by the server. The server sends client requests and sends back appropriate responses. This is the part of the project where you have the most choice. **You are to propose the application domain.** This includes defining the functionality of the server and the specification of the protocol:

- You must specify valid commands that the client accepts from input, the meaning of the command request to the server, and the server responses.
- Your protocol should have at least five distinct commands and at least one command must involve a file transfer over the network.
- Your commands should include transfers in both directions.
- You should define message types for each of the commands supported by your client and server. These messages should include header and data portions. However, the data portion of your application layer messages is limited to 100 bytes in size.
- You will need to define a means to accommodate commands requiring multiple messages for a single command.
- If the server encounters an application-level error while processing a request, it returns an error message. The client in turn prints out appropriate error explanations.

The application layer interfaces with the data link layer via only two routines: DataLinkSend() and DataLinkRecv(). Both the client and server processes can issue DataLinkSend() and DataLinkRecv() calls. You may also consider encapsulating these routines in a DataLink object. **Applications don't need to run on multiple machines.**

Data Link Layer

The data link layer accepts data passed via DataLinkSend(), places the data into a data link frame, and sends the frame on the TCP/IP link for transmission. The design of your data link layer must include the ability to handle an error-prone channel. Thus, the data link layer must include considerations for buffering, retransmissions and some form of flow control.

You must implement (have the ability to switch between the two as needed): Go-back-N protocol, and Selective repeat protocol

In the two pipelined protocol cases, your data link layer will have to handle:

- **Buffering** at the sender and/or receiver. If an application attempts to transmit messages beyond the window size, the data link layer will block until space becomes available.
- Retransmissions will require a timer mechanism to detect lack of acknowledgments.
- On the DataLinkRecv() side, note that a new data frame may arrive before the application layer actually calls DataLinkRecv(). Because frames must be processed when they arrive (otherwise you might ignore an

acknowledgment), received data frames will have to queued.

- When DataLinkRecv() is called, it must check for the presence of data in the receive queue. If none is present, it will have to suspend itself until an appropriate event occurs.
- The data link layer processes must respond to timeout, frame arrival and frame error events.
- Finally, as data can be corrupted, you need to add framing, character stuffing and checksums to your data link layer to detect packet corruption.

Physical Layer

The actual communication with the server takes place using Unix sockets and TCP/IP. The physical layer deals with the details of establishing a TCP/IP connection and sending messages over a TCP/IP network. The data link layer hands off frames to the physical layer for transmission. As part of introducing unreliability into the network transmission:

- You are responsible for artificially dropping frames in the transmission.
- Dropping a frame simply means not transmitting it for this assignment.
- Any frame that is sent (setup, data, ACK, NAK, etc.) can be potentially lost.
- The packet drop rate to be used by the client and the server should be given on the command line when each of these is started up (remember you have duplex communication with different error rates in each direction).
- Obviously your data link layer should not "know" whether a frame will be lost and will have to discover that problem via the protocol you implement.
- Data corruption in the channel should be simulated by the sender modifying the message contents before actual transmission.
- The packet corruption rate to be used by the client and the server should be given on the command line when each of these is started up (remember you have duplex communication with different error rates in each direction).
- The sending data link layer will need to indicate the size of frames as part of the data-link layer header so that the receiving data-link layer. knows the size of each frame.

Input and Output

It is the team's responsibility to provide interesting and meaningful test data to show that your project works. If the final project turned in does not work completely, then you should provide ways to demonstrate which modules in fact are working.

In order to observe the performance three reliability protocols and to check whether your implementation is working properly, you need to collect statistics for each of the reliable data transfer protocols. Include performance table with the submission.

- For debugging reasons, you should design a statistics gathering process on both the client and server machines.
- These processes record on-going and final statistics.

- When you demonstrate your project it is advantageous to have both monitoring processes outputting information into separate windows. The following are suggested statistics and **submit a document with graphs for average of the metrics below over 10 runs for different packet drop and packet corruption rates:**
    - The total number of data frames transmitted
    - The total number of retransmissions
    - The total number of acknowledgments sent
    - The total number of acknowledgments received
    - The total number of duplicate frames received
    - The total amount of data sent
    - Time required to classify client request for different combinations of packet drop and packet corruption rates

  Choose at least 10 combinations of packet drop and packet corruption rates over the entire range from 0 to 100%. You can plot multiple related metrics on a single graph, but don't make the graphs too crowded. Make sure you have appropriate legends in your graphs. You can organize the graphs (its dependent and independent variables) in whichever way you want.

Grading

The grade breakup (%) is as follows:

1. Full implementation of the Application Layer (Client And Server) – 10%
2. Correct implementation of Go-back-N protocol – 25%
3. Correct implementation of Selective Repeat protocol – 25%
4. Correct implementation of queuing, checksum, and the inter-layer interfaces – 20%
5. Full implementation of Physical Layer with data corruption and packet drop – 10%
6. Makefile + README = 5%
7. Graph file = 5%

The grade will be a **ZERO** if:

1. If *README* with detailed instructions on compiling the running the code is not present
2. If *makefile* is erroneous and the README does not provide a way tell us how to compile and run the code
3. If no *Graph* file is included

Teamwork

All submissions have to done in teams or 2 or 3 (preferable two). You can be in existing teams in the class or create new ones as needed. Please inform the instructor of your team members.

Teams will have to demo their project in the end. More details on the demo is forthcoming.