Introduction

This assignment is to design and implement a reliable protocol on top of an unreliable delivery mechanism. You are to do this project in a Unix environment using the TCP/IP protocol to exchange messages.

The project is intended to expose you to at least four aspects of computer networks the client/server paradigm, working with a real network protocol (TCP/IP), the concept of layering and issues in the design of a reliable protocol. You have already done some work with the first two items, which you will be building on for this assignment. The following description outlines the project. It defines many of the details, but leaves other decisions on functionality and design up to you.

You are to create a client and server process that communicate at the lowest layer via a TCP connection. On top of this connection you will be sending messages. While the TCP connection will reliably deliver all messages you will introduce errors in message delivery so that some proportion of messages are simply dropped and not sent at all. This unreliability will require you to create a reliable delivery mechanism on top of the “unreliable” channel.

You will be implementing two layers: an application layer and a data link layer. The application layer will communicate (via procedure calls) directly with the data link layer for sending and receiving messages. The most straightforward project design will include the application and data link layers as part of the same Unix process, although they can be separated into two separate processes.

Application Layer

You will be creating a file transfer application that will allow the client to send a file to the server for storage. The file being transferred can be of arbitrary size and type (it is not restricted to be a text file). The client should initially send the name that the remote server should use to store the file.

You should define message types for all communication between your client and server (similar to defining message types defined for communication with the oracle server). However, the data portion of your messages is limited to 100 bytes in size. You will need to define a means to accommodate file transfers requiring more than a single message.

The application layer interfaces with the data link layer via the following routines: DataLinkInit(), DataLinkSend(), DataLinkRecv() and DataLinkCleanup().

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 some type of sliding window protocol (see Tanenbaum for details). At a minimum you must use a single-bit sliding window protocol, but you are encouraged to use a send window size greater than three. In the latter case 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. These can be done using the select() system call. See the man page for how to use the timer facility of select().

For the basic project, your server will continually call DataLinkRecv() to receive messages from the client. Initially it will receive the name of the file to create, followed by the contents of the file (in one or more messages). When the entire file has been transferred, the server should close the file for writing. If multiple messages can be outstanding at a time, messages received by the data link layer on the server may not be immediately passed up to the application layer because they are not the next one expected. Depending on the data link layer protocol you are implementing, these out-of-sequence messages will either be buffered or discarded. When the server receives a message it should generate an ACK or a NAK as appropriate to your protocol.

The actual communication with the server takes place using Unix sockets and TCP/IP. You will need to initialize these structures at the beginning of your client and server code in a routine written by you called DataLinkInit(). Use your Unix uid as the port number for your server unless this value conflicts with another server in which case pick a random value. Your data link layer routines DataLinkSend() and DataLinkRecv() will use the TCP connection for all data transmission. The link corresponds to the physical layer of a real network. When the client or server has completed its use of the channel, it should call DataLinkCleanup() to terminate the connection. Note that if the data link layer on the client is buffering then sent, but unacknowledged, messages that it will need to wait until all messages have been acknowledged before terminating the connection.

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 error rate to be used by the client and the server should be given on the command line when each is started up as indicated below. 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.

Interface

The command line interface for your client and server is as follows:

> server errorpct

> client errorpct put inputfile [outputfile]

where errorpct is an integer between 0 and 100. It represents the percent of frames that should be dropped by that process. An error percentage of zero means that all frames are correctly transmitted. An error percentage of 50 means that 50% of the frames are dropped and not sent. You should initially work with an error rate of zero and experiment with other values once you have your client and server working.
The put argument simply indicates the client is sending the file. It can be changed to get if you do more advanced work on the project. The inputfile is the file to read and send to the server. The optional argument outputfile is the name that the server should use in creating the file. If no outputfile argument is given then the server should use the same name as the client. Beware of this option if the client and server are running from the same directory.

Output
In order to observe the performance of the sliding window protocol and to check whether your implementation is working properly, you need to collect statistics. You should collect statistics for both the client and server. After transferring a file you should print out the following suggested statistics (you should add your own which are appropriate to your specific project):

1. the total number of data frames transmitted
2. the total number of retransmissions
3. the total number of acknowledgments sent
4. the total number of acknowledgments received
5. the total number of duplicate frames received
6. the total amount of data sent

For your debugging purposes you may wish to show the size of frames sent and received. For performance analysis purposes you should consider measuring the time required to satisfy a client request.

Design Decisions
Students are encouraged to work in two-person groups of your choosing, but individual projects are acceptable. You should select a partner who has similar goals for the project as yourself. Projects will be graded equally for all team members unless exceptional circumstances arise.

One issue to address in your design for the project is what protocol to use for handling errors. More credit will be given to projects that correctly implement a protocol that allows multiple outstanding messages (such as “go back n” or “selective repeat”) as opposed to projects that implement a simple “stop and wait” protocol. However projects that do not work will be penalized regardless of the protocol. Hence it is better to turn in a project that works with a simpler data link protocol than one that does not work with a more complex protocol.

An enhancement that you may implement for additional credit is to allow not only of sending of a file from the client to the server, but also allow retrieving a file from the server to the client. The command line interface is the same as given previously except the put command should be changed to get.

Another enhancement for additional credit is to send and receive frames byte-by-byte using character delimiters for frames along with character stuffing and checksums. If this
enhancement is used then your error rates will be on a per-byte rather than per-frame basis. In this case you should use a larger integer range for error rates.

Project Deadlines

The project has two deadlines:

1. Design Report (Due: Thursday, June 10, 2004)
   Each group will turn in a typed design report (one or two pages) defining the project and explaining the work to be done. The design should include the team members, the data link layer protocol to be used along with message structures and types. You should also indicate what statistics you intend to gather. This design should clearly explain the final product and include a schedule of work to be done. These designs will be reviewed and returned with comments, but not actually graded.

2. Final Project and Report (Due: Tuesday, June 29, 2004)
   The final report should be a well-presented technical report discussing your project. You should explain how the program works, give specific sample runs and analyze the results. Results indicating the relative efficiency of your protocol with varying error percentages are encouraged. The final report may include parts of your design report. The report should be 5-10 pages in length.

Grading

Projects will be turned using turnin with “proj2” and tested for grading. The final report will also be examined as part of the grade. Projects will be graded based on features and correctness. Correctly implemented projects using the “stop and wait” protocol will generally receive grades in the B range. Correctly implemented projects using the “go back n” and “selective repeat” protocols will generally receive grades in the low A range. Correctly implemented projects that allow both sending and receiving of files along with buffering in the data link layer and/or actual framing of data will generally receive A grades. Variation in grades will depend on the quality of the work, features provided, the analyses that were done and the final report.
Final Comments

This document outlines the project and a lot of the details on how to design and build it. You should be familiar with TCP/IP from the previous assignment and should begin in a simple manner by writing routines for transmitting data link information over the TCP/IP connection. From this start you can go on to more complex issues.

The project provides you latitude on how it is designed and implemented. Not all details are spelled out for you. Part of the assignment is to test how well you can take a more open design project and turn it into a finished product. Do not wait to get started and do not wait until the project design deadline to start coding. You have many decisions to make and much work to do.

You will obviously want to consult the data link layer protocols in the textbook for this assignment, although you probably do not want to use the code verbatim. Tanenbaum has made the data link layer protocol code available online at the text Web site (http://authors.phptr.com/tanenbaumcn4/) and included it as part of a simulation readers can obtain. This simulation does not actually transfer data as your simulation must. It also is more complex in the parameters it uses than your simulation needs to be. This simulation and its code are available for you to consult, but it is not suggested that you try to use this simulation code directly in your project. Any parts you do use should be documented in your design report and final project.