CS 513/EE 506 Introduction to Local and Wide Area Networks WPI, Spring 2007
Craig E. Wills Standard Project Description
Monday, January 29, 2007

Introduction

This assignment is a course project to design and implement a three layer communication model which permits two or more application processes to communicate over a network. The expectation is that the program will be done in a UNIX environment with processes using TCP/IP to exchange messages. The assignment can also be attempted in other environments, but there is no guarantee as to how difficult the project becomes when you use another network context.

The project is intended to expose you to at least three aspects of computer networks—the client/server paradigm, working with a real network protocol (TCP/IP) and issues in the design of the data link layer. 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 but does not specify all the details. This approach is used to give you choices in the functionality of the model and to force you to make some of the design decisions.

Throughout the description it is important to separate the abstraction of a three level model, from the details of interfacing to TCP/IP.

The project consists of three layers—application, data link, and physical. The application layer consists of a protocol between a client/server pair of processes. Communication is in the form of request and response messages. The client sends requests to the server and receives responses to those requests. The data link layer provides frame-level communication between client and server hosts. The data link layer operates over an unreliable channel subject to lost and garbled data. The physical layer provides a full-duplex, communication channel on which the 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 fields 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 proposal includes 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 a large transfer 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.

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()*. More details on using this call will also be made available by the instructor.

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. As in Tanenbaum's discussion, the data link layer processes must respond to timeout, frame_arrival and frame_error events.

Physical Layer

The actual communication with the server takes place using Unix sockets and TCP/IP. Use your Unix uid as the port number for your server unless this value conflicts with another server in which case pick a a random value. 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 error rate to be used by the client and the server should be given on the command line when each of these is started up. 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.

The sending data link layer will need to indicate the size of frames as part of the DLL header so that the receiving DLL knows the size of each frame. For more realism, you can add framing, character stuffing and checksumming to your data link layer, but in the simplest case you can assume the physical layer either transmits an entire frame reliably or drops the entire frame.

Input

It is the group'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.

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. 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 (You should add your own which are appropriate to your specific proposal):

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 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.

Project Deadlines

The project has two deadlines:

1. Design Report (Due: Monday, February 12, 2007)

   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: Monday, April 16, 2007)

   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 primarily graded via an in-person demonstration with the instructor during the last week of the semester. 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 data 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 project must. It also is more complex in the parameters it uses than your project 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.