Network Application with Multiple Clients and Concurrent Server {February 7, 2012}

Course Project Points and Due Dates

Proposal 2 points Due: 8 p.m.Tuesday February 14, 2012

Each project group must submit a typed project proposal. The proposal includes team members and an explanation of the specific application that you propose to implement on MININET.

Design Report 40 points Due: 4 p.m. Friday March 2, 2012

Program teams must deliver a hard copy typed design report to my office. Pseudo-code is **unacceptable** for your design report. Professional technical prose is expected. This report will receive a letter grade and a point grade based on a grading rubric that includes all the standard criteria of a professional technical report (i.e., grammar, writing style, typos/misspellings, clarity and content will **ALL** be considered).

Final Project Demo 80 points Between April 16 – April 27, 2012

Each project must schedule a one hour time slot for a LIVE demonstration of their project during this period. Bring to your demo: a "cheat-sheet" style help form that lists the commands and explains the parameters used; a hard copy of your program which must conform to standard commenting expectations; the original marked-up copy of your design report, a short addendum that explains any significant changes and improvements made to your design; and a **README** document that itemizes those aspects of the project that are not working correctly and those which are not fully working as of the demo.

Introduction

The networks course project is a two-member team assignment to design and implement a three layer network model which permits two or more application client processes to communicate over the Internet with a concurrent application server. The expectation is that these programs will be written in C or C++ on CCC Linux hosts and will use TCP/IP to exchange messages. This assigned is intended to expose students to at least three aspects of computer networks - the client/server paradigm, key issues in the design of the data link layer, and working with a real network protocol (TCP/IP).

Once Program 1 has been completed, students can submit their preferences for a team partner. Note – both partners need to send an email indicating their preference! I will assign team pairs for those students who have not submitted a partner preference by **February 7th**. Students who not do complete Program 1 successfully **MUST** schedule a meeting with me before being assigned to a project team.

Each team must submit a typed project proposal (one page max). Each project team must then schedule a half hour **required office hour** to review project design issues by **February 27th**.

The following description outlines the project but deliberately does not specify all the details. This provides student teams with choices in the functionality of the model implemented and puts some of the design decision in the hands of the students. Throughout the description it is important to separate the abstraction of a three layer model, henceforth referred to as MININET, from the details of the UNIX socket interface to TCP/IP.

The MININET protocol 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 request and response messages. Clients send requests to a concurrent server and receive responses to these requests from the server. The data link layer provides an ***emulated*** frame-level communication between client and server processes on separate CCC computers. The data link layer operates over an emulated unreliable channel subject to lost and garbled frames. The physical layer provides the illusion of a two-way communication channel on which the data link layer can send an arbitrary byte stream. The physical layer is emulated using TCP/IP.

MININET Application Layer

The application layer implements an interactive request/response protocol and provides an interface to the data link layer. An application layer client reads commands from standard input, converts them into server requests, sends the request messages to the server, and prints the response messages returned by the server. The concurrent server listens to establish a TCP connection with each client, interprets the received client requests and reacts to client requests by sending back appropriate responses.

Each project team MUST propose, design and implement their specific application layer. Your design report includes an explanation the functionality of both the client and the server and a precise and detailed specification of the application layer 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 application **MUST** include at least six distinct client commands and one of the commands must involve a file transfer over the network. Your command set must involve at least one large transfer in both directions (e.g. a 1 Mbyte jpeg image). These transfers should be of sufficient size to test your sliding window mechanism. Your application specification needs at least a primitive login command that verifies that a client belongs to the set of authorized users of the service.

Messages, the data units between the application and data link layer on MININET, are limited in size to **256 bytes**. This includes application layer protocol bytes and any overhead bytes specified in your design. Note, your design must be able to handle multi-message application level peer communications such as file transfers.

If the server encounters an application-level error while processing a request, it returns an appropriate error message. The application client process then prints out appropriate error explanations to the user.

The critical MININET abstraction is that the application layer interfaces with the data link layer ONLY via two entities ***dl_send*** and ***dl_recv***. Both the client and server application layers interact with the data link layer via ***dl_send*** and ***dl_recv***.

Note: choices made on how to control the concurrency of the layers is the KEY design decision for this project. It is essential that each project team discuss and decide upon their approach to this problem PRIOR TO writing the design report.

MININET Data Link Layer

The data link layer accepts data passed via ***dl_send***, places the data into a data link frame, and sends the frame to the physical layer for transmission. The data link layer communicates with its peer server process over a two-way communications channel accessed through UNIX file descriptions (see the physical layer).

Your data link layer must include the ability to handle arbitrary data streams and to emulate an error-prone channel. Thus, the data link layer must include considerations for framing, data stuffing, buffering, retransmissions, checksums, and some form of flow control. Retransmissions will require a timer mechanism to detect transmission errors.

You must implement a sliding window protocol (see the Tanenbaum handout for details). Whether **Go Back N** or **Selective Repeat** is used is a team design choice. However, you must plan on using a sender sliding window size of at least four frames. Conceptually, the

data link layer buffers the data internally and allows the application to continue processing. If the caller attempts to transmit frames beyond the window size, the data link layer should block (and disable the application layer process) until space becomes available.

In the ***dl_recv*** functionality, a new data frame may arrive before the application layer actually calls ***dl_recv***. Because frames must be processed when they arrive (otherwise you might ignore an acknowledgement), received data frames will have to queued. When ***dl_recv*** is called, it must check for the presence of data in the received 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.

MININET Physical Layer

The actual communication between the client and the server takes place using Unix sockets and TCP. The physical layer of MININET handles the details of establishing a TCP connection and sending real messages over the WPI campus network. Assume the maximum size of the frame payload is **128 bytes**. That is, the data link layer hands off frames to the physical layer for transmission. To simplify matters, you may make reasonable assumptions on the maximum amount of data stuffing needed. The design of the full frame is the students' choice but must include "artificial" framing bytes at each end of the frame sent.

The physical layer is responsible for inducing errors in the transmission. Your design must include an adjustable error rate input as a command line argument for both the client and the server. The error rate is used to ***randomly*** insert a single bit error into the frame checksum field before a frame is sent by the client or the server.

Test Data

Since the students are specifying the application layer, it is also the group's responsibility to provide interesting and meaningful test data to show that your MININET works correctly. If the final project turned in does not work completely, then provide ways to demonstrate which modules in fact are working. All routines MUST have a single, primary author.

Output Logs and Monitoring

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 monitor on both the client and server machines.

The best form of monitor is one that records both ongoing and final statistics. When you demonstrate your working project, it is advantageous to have the client and server monitors outputting information to separate screen windows. The following are some suggested statistics that should be maintained for both clients and the server. The important idea is that the totals that you provide on the clients and server should be such that one can add and subtract some of the totals to show that your sliding window scheme works in the face of errors. (You should include any additional statistics that are germaine to the specific application that your MININET is supporting):

1. the total number of data frames sent
2. the total number of retransmissions sent
3. the total number of acknowledgments sent
4. the total number of data frames received correctly
5. the total number of acknowledgments received correctly
6. the total number of data frames received with errors
7. the total number of acknowledgements received with errors
8. the total number of duplicate frames received
9. the number of times the data link layer blocks due to a full window.

While debugging you may wish to show the size of frames sent and received. It is wise to have a verbose enable/disable switch such that you can turn on and off debug messages inside your programs. For performance analysis, you should measure the time required to satisfy individual client requests.

Comments and Suggestions

There are three aspects of this project which require specialized systems programming knowledge - the timer mechanism, using processes or threads to implement ***dl_send*** and ***dl_recv***, and the UNIX socket interface. When making design decisions, it is valid to propose a *less-than-elegant* solution due to your time constraints. However, your design report needs to explain and defend such choices.

Note, this document outlines the project without taking up the issues of suggested strategies to build your MININET. Consider developing your network by gradually adding complexity to the problem. For example, build and test an errorless data link layer first. Start with a simple stop-and-wait protocol and subsequently switch to the more difficult sliding window scheme.

Design Report

Your design report will be graded based on basic technical writing standards including: grammar, organization, formal presentation style, typos and clarity of writing. The three key points to focus on in your design report are: a detailed specification of your proposed application layer; a thorough explanation of the mechanisms to be used when

communicating between the layers; and a discussion about the concurrency control mechanisms that you will employ (e.g., semaphores, shared memory, pipes, processes and/or threads). Your design proposal must identify your specific choices. A Warning: do not get too fancy and avoid using mechanisms that you have not used previously.

Your design report must include: the detailed specification of the application protocol, a diagram that explains the components and interfaces in your design, an allocation of work between team members, a milestones schedule, and a bibliography. The design paper should be 10-20 typed pages not including pictures and diagrams.