COMP/ELEC 429/556 - Introduction to Computer Networks
Project 1 - Ping-Pong and World Wide Web

Handed out: Thursday, January 23

Due: Thursday, February 13 at 11:55pm

I. Introduction

The primary goal of this project is to let students get some experience with the design and implementation of networked applications and protocols, including how to format data and use sockets. Through this project, students will also gain hands-on experience with measuring network performance, concurrent software design, and I/O handling.

In this project, you are to develop a "ping-pong" client program and a server program. The "ping-pong" client and the server together are used to measure the network's performance in transmitting messages of various sizes. Further details are described in Section II.

In addition to responding to your "ping-pong" client's messages, your server should have an alternate execution mode in which it acts as a simple web server. This functionality is described in Section IV. Your server must be able to handle multiple concurrent connections from "ping-pong" clients in the default mode or web browsers in the "www" mode.

II. The Ping-Pong Client and Server (60 points)

The "ping-pong" client program should take 4 command line parameters, in the following order:

- *hostname*

  The host where the server is running. You should support connecting to a server by domain name (current list of CLEAR servers: ring.clear.rice.edu, sky.clear.rice.edu, glass.clear.rice.edu, water.clear.rice.edu).

- *port*

  The port on which the server is running (on CLEAR, the usable range is 18000 <= port <= 18200).

- *size*

The size in bytes of each message to send (10 <= size <= 65,535).

- *count*

  The number of message exchanges to perform (1 <= count <= 10,000).

The server program should take 1 mandatory parameter and 2 optional parameters, in the following order:

- *port*

  The port on which the server should run (on CLEAR, the usable range is 18000 <= port <= 18200).

- *mode* (optional)

  The mode of the server. If this is "www", then the server should run in web server mode (described in Section IV). If it is anything else, or left out, then the server should run in ping-pong mode.

- *root_directory* (required for www mode)

  This is the directory where the web server should look for documents. If the server is not in web server mode, ignore this parameter. (e.g. /home/student/comp429/project1/html)

The ping/pong message should be formatted as follows, with the first byte starting from the left:

size (2 bytes)	timestamp tv_sec (4 bytes)	timestamp tv_usec (4 bytes)	data (size - 10 bytes)

The size and the two timestamp fields are numeric values and therefore should be stored in network-byte order in the message. The data portion can contain arbitrary byte values of your choice. The size lets the receiver of the message know how much data there is to read from the socket to completely receive the message.

The "ping-pong" client should establish a SOCK_STREAM connection with the server. Once the connection is established, the client should send a "ping" message to the server. Upon receiving the "ping" message, the server should reply with that same message without modifying its content, a.k.a. a "pong" message, immediately. The client waits for the reply, then repeats such message exchange for "count" number of times, then closes the connection. The client should measure the latency of each such exchange and printf() to the screen the average latency of the exchanges (expressed in millisecond, with up to 3 decimal places of precision) and then terminate.

In order to measure latency, the client should use gettimeofday() to obtain the current time before sending a ping message and put it as a timestamp in the ping message. gettimeofday() fills in a struct timeval, which consists of two 32bit fields, tv_sec and tv_usec, the second and microsecond components of the time since 00:00:00 UTC, January 1, 1970. See "man gettimeofday" for details. Subsequently, when a pong message is received, the timestamp embedded in the pong message can be compared to the current time (again, use gettimeofday()) to compute the latency experienced by this ping-pong exchange.

III. Measurement - Mandatory for COMP/ELEC 556; Optional for COMP/ELEC 429 (15 points)

The ping-pong client/server measures the total latency of transferring the specified amount of data twice, once from the client to the server, and once from the server to the client. The data transfer latency measured can be decomposed into two components: (1) data size and bandwidth dependent transmission delay (which is approximated by data size divided by network link bandwidth), and (2) data size and bandwidth independent delay caused by the time it takes for data to physically propagate over the length of network links, the time overhead of making software function calls, etc.; this bandwidth independent delay exists no matter how high the underlying network link bandwidth is. Using your ping-pong client/server, if you measure the time it takes to send 65,535 bytes of data between two CLEAR servers, you might find that it takes nearly 1.5ms per iteration on average (this figure is approximate, your exact measurement may vary). If you perform the calculation for the data transmission rate achieved, you'll get (65535*2*8)/0.0015 bps or 699 Mbps. That is a rather low achieved transmission rate given the underlying network links are 1000 Mbps (i.e. 1 Gbps). The main reason for the low achieved transmission rate is that a lot of that 1.5ms is coming from the bandwidth independent delay.

For this part of the assignment, design and write up a method for measuring this bandwidth independent delay across two CLEAR servers as accurately as possible using only the ping-pong client/server software you have developed; no change to the software is allowed. Because you are limited by the ping-pong client/server's design, you will not be able to obtain a perfect answer; instead, the goal is to estimate the bandwidth independent delay as closely as possible. You should then perform the corresponding measurements according to your method using your ping-pong client/server, and report your estimate of this bandwidth independent delay. Finally, use this estimated bandwidth independent delay to adjust your estimate of the true bandwidth of the network link. You must clearly show your measurement results and calculations in your write-up.

The grading of this part will depend on the method design and the underlying reasoning to support your method, as well as your actual measurements and calculations. That means your grade heavily depends on the write-up. Logical reasoning and precise wording in the write-up are expected. Your write-up document should be a PDF file named "part3.pdf".

IV. The Web Server (40 points)

When the server's optional *mode* parameter is set to "www", the server should run in web server mode. In this mode, the server should answer HTTP GET requests from web browsers. The server should open files relative to the root directory specified as a parameter to the server, in order to respond to GET requests. The full specification for HTTP can be seen at http://www.w3.org/Protocols/. Because HTTP is an extremely complicated protocol, we only expect you to implement a subset, described here.

Special Cases

Note that you want all requests to be restricted to under the root directory. This means you should disallow requests containing "../" path elements which could potentially cause the server to send documents not under the root directory. You can allow symbolic links which leave the specified root, since the user explicitly put them there and knows what to expect. Return an error message with code 400 if "../" appears in the path.

If a file is not found, you need to return an appropriate HTTP error code and message. How you handle directories is largely up to you; you can return a file-not-found message, a more specialized error message, or convert the request to index.html within the specified directory. It is **NOT** OK to just open the directory as a file and return the contents.

V. Administrivia

Failure to adhere to the following rules may result in a zero grade for your project.

You may form a group of up to 3 students to work on this project. If you need assistance finding group-mates, email the instructor as soon as possible.

You are allowed to use the sample C programs and Makefile provided by the instructor as starting points for your project.

To facilitate the grading process you must put your work under a directory called project1, then create a tar archive of it (i.e. tar cvzf project1.tar.gz project1). Submit the tar archive to Canvas. One submission per group is enough, unless there is a special situation such as members of a group want to submit on different days due to slip day usage differences.

In the project1 directory, in addition to any software code files, there must be a file entitled README which states the names of the students who worked together on the project, and documents how you tested your client and server, any known problems, and anything the graders should know. Also, include your "part3.pdf" document as appropriate. You must use *make* to build your project. There must be a Makefile in the project directory, and running make in the project directory must build both the client and the server. The grading will be done on CLEAR, so make sure that your code works on CLEAR. If your code does not compile with make on CLEAR, you may receive a zero grade.

You must write your programs in C and use the Linux networking API.

Appendix: HTTP Protocol Overview

The Hypertext Transfer Protocol (HTTP) is an application-level protocol for distributed, collaborative, hypermedia information systems. HTTP has been in use by the World-Wide Web global information initiative since 1990. The first version of HTTP, referred to as HTTP/0.9, was a simple protocol for raw data transfer across the Internet. HTTP/1.0 improved the protocol by allowing messages to be in the format of MIME-like messages, containing meta information about the data transferred and modifiers on the request/response semantics. Latest HTTP Version is 1.1, which is documented in RFC 2616; version 1.0 is documented in RFC 1945.

HTTP operates over TCP connections, usually to port 80, though this can be overridden and another port used. After a successful connection, the client transmits a request message to the server, which sends a reply message back. HTTP messages are human-readable.

The simplest HTTP request message is "GET url", to which the server replies by sending the named document. If the document doesn't exist, the server will send an error message stating this.

The format for GET command is "GET url HTTP/version". The client may then send additional header fields, one per line, terminating the message with a blank line. The server replies in a similar vein, first with a series of header lines, then a blank line, then the document proper.

Here a sample HTTP 1.1 exchange represented using strings in the C programming language:

Client sends:

```
"GET /index.html HTTP/1.1 \r\n"

"\r\n"
```

Server replies:

```
"HTTP/1.1 200 OK \r\n"

"Content-Type: text/html \r\n"

"\r\n"

HTML document body follows
```

Simply speaking, HTTP protocol operates in 4 steps:

Step 1 'CONNECTION': Web browser (the client) try to connect to the web server using socket.

Step 2 'REQUEST': Web client sends its request via the established socket. Here we only consider the request using GET command. The format for GET command is:

```
"GET path/file HTTP/version \r\n\r\n"
```

the 'path/file' indicates the requested file; 'HTTP/version' specifies the HTTP version used by browser.

Step 3 'RESPONSE': After the server receives the request, it will do some necessary processing then send back the results to clients. Then, the client can display the requested web page. For example, if the client wants to visit the webpage www.mycomputer.com:80/mydir/index.html, it will send a GET command: GET /mydir/index.html HTTP/1.1 to the server. The webserver with domain name www.mycomputer.com will search for 'index.html' from its subdirectory 'mydir'.

The server needs to send the client some necessary information regarding the requested file. So the server will first transmit some HTTP header followed by the document body. HTTP header is separated from the document body by a blank line.

Some useful HTTP header message:

1)"HTTP/1.1 200 OK \r\n"

This is the first line of the response of the server, which lists the HTTP version and response code. The response code "200 OK" means the request is successful. Other return codes are as follows:

200 OK

400 Bad Request

404 Not Found

500 Internal Server Error

501 Not Implemented

2)"Content-Type: type\r\n"

This specifies the MIME type of the HTTP body, e.g., Content-Type: text/html means that the data sent is a HTML document. You only need to support text/html in your server. You do not need to deal with any other type. A blank line "\r\n" is then sent, followed by the document body.

Step 4 'DISCONNECTION': After the response, web server should disconnect with the client.

Needless to say, the aspect of HTTP we are concerned with in this project is a very tiny subset of the HTTP protocol. If you want to see the full gory detail of the HTTP protocol, you may wish to observe real HTTP protocol traffic generated from your own computer by using a traffic analysis tool like "wireshark" as you use your web browser. Note that you cannot do this on CLEAR servers because it requires administrative privilege.