Program 2 {Updated November 15, 2010} 30 points

HTTP Request and Response
Due: Tuesday, November 23, 2010 at 4 p.m.

This assignment develops a tool that can be used to explore Internet and TCP performance. The initial task is to write a simple Web client that can communicate with any Web server via TCP.

HTTP is implemented on top of TCP and uses HTTP requests. The following is an example request generated by a real browser for **/** located on **www.cnn.com** at port 80. The first line of the request contains the type of request (you only need to use **GET**, but other types such as **HEAD** and **POST** are possible in HTTP). Following the **GET** request is the object to be requested. The remainder of the line identifies the HTTP version used by the browser. The remaining lines are HTTP request headers, which you will not need.

GET / HTTP/1.1
Host: www.cnn.com
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.8.0.3) Gecko/20060426 Firefox/1.5.0.3
Accept:text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive

Note that the HTTP specification expects all lines to be terminated with a CR (carriage return) and LF (line feed) characters. These characters are **\r\n** in C/C++. Note also that the HTTP specification expects a ``blank" line containing only **\r\n** at the end of the request headers.

There are a number of HTTP response codes, but for this project you will simply save the response code and content that is returned.

Client Testing

As a starting point you should build a simple Web client. This client should connect to a given port on a given host and send a minimal request string. You will use command line arguments to control your client where the first argument is a URL of the form **http://*server:port/path*** where the specification of **http://** and the port are optional (80 is the default port). For example

% webclient www.cnn.com/

can be used to request the object from port 80 on the machine **www.cnn.com**. Your simple webclient will need to connect to the port and send the **GET** line (ending in CR/LF) along with the **Host:** line followed by a blank line with CR/LF. You can use ``HTTP/1.0" as the version. Note – the machine goes on the **Host** line and the path goes in the **GET** line in the string buffer

sent to the server. Your client should then receive back the response headers and content from the server and save them to the file named **webout** by default. You should be able to use your Web client with any Web server by sending to the standard Web server port 80. Your program should support the **-f** option (specified after the URL) allowing an alternate file name to be specified for the output. For example

% webclient www.cnn.com/ -f cnn.save

will save the output in the file **cnn.save**. You should also implement a **-nf** option that causes the output to be read, but not saved in a file. The –**nf** option and the –**f** option are mutually exclusive in that only one of them can be invoked per execution. The **-nf** option is useful in connection with latter parts of the project. Completing the simple client with this functionality is worth 15 out of the 30 points for the project.

TCP Ping

A common measurement performed on the Internet is a ``ping" from one host to another to determine the round trip time (RTT) between the two hosts. Unix/Linux systems have a ***ping*** command, which uses the ICMP protocol to measure the RTT, but because of security concerns fewer and fewer servers respond to ping requests.

As an alternate approach, you can implement your own version of ping by adding a **-ping** option to measure the time in your web client to perform the TCP ***connect()***. The ***gettimeofday()*** systems call can be used to determine and record the time prior to your call to ***connect()*** and to determine the time after ***connect()*** has completed. The difference between these two times is an estimate of the RTT. You should print the RTT value in milliseconds (you will have to do conversion from the second and microsecond fields used by ***gettimeofday()***).

Once the **-ping** option is implemented, identify five Web servers from around the U.S. and five additional Web servers from around the world (and outside the US). Use the **-ping** option of your client to make **at least** 20 measurements for each server. After gathering this data, construct a table listing each server tested along with the minimum, maximum, median and mean ping time measured for each. Include this table in a brief report submitted with Program 2. This portion of the project is worth an additional five points.

TCP Packet Patterns

For the next five points of the project, you will examine the approach used by a TCP connection for transmission of data. The use of this feature is controlled by a **-pkt** option. When this option is specified, your client should record the time and number of bytes for each read in the main loop of your program. These values should be recorded in an array and not immediately output in order to avoid I/O during data collection. Use of the **-pkt** option should also automatically turn on the **-nf** option.

Once your program has read all data from the socket and closed it, the program should print out the time and number of bytes for each read. In many cases each read will correspond to the

reception of a single packet so that the number of bytes read is the size of a packet. In other cases, a read might contain the data of multiple packets. Your program should be able to read up to 10K bytes on each iteration.

Your report should print the maximum packet size for each of the servers in your test set. Is the size the same for all servers?

TCP sending behavior, which we will discuss in more detail later, often results in bursts of packets being sent. Each of these bursts is sometimes referred to as a ``flight." To better see this behavior select an object download from a server that requires 30-50 packets. For three such downloads create a graph where the x-axis is time (time begins just prior to the ***connect()*** call) and the y-axis is the cumulative number of bytes that have been read. How many packets are read in each flight for different servers? You should include graphs and analysis in the report you submit. How well formed are these flights in terms of distinct waves being visually evident in your graphs?

TCP Information

For the final five points on the project, you need to add an **-info** option to your client, which when invoked will display information about the TCP connection. TCP information maintained by the kernel is available on Linux systems via the TCP_INFO option to the ***getsockopt()*** system call. This information is returned in the **tcp_info** structure that is defined in the **<netinet/tcp.h>** include file. This structure contains much information about the TCP connection, but for this assignment retrieve and print out the RTT and RTT variance for the TCP connection after all data has been read from the socket, but before the socket is closed. Include these measured values in your report. Compare these measured RTT values with the results from the **-ping** option.

What to turnin for Program 2

Turn in your assignment using the ***turnin*** program. **You should turn in a tarred file that includes:** the final source program ***webclient.c*** , a README file and a ***make*** file. The final version of your webclient should contain the code for all the implemented options. You must submit a separate hard copy of your report to Professor Kinicki by the 4 p.m. deadline.