ECE 158b Spring'24 Homework Assignment 2

University of California San Diego

Instruction:

Submission deadline: **6pm, Wednesday 05/08/2024.** Submit a pdf report containing answers to the problems below. Submit your source code and data (if any) in a separate zip file. Include your name and PID in the pdf document. It is recommended that you include your name in the file name as well, e.g., "John_Smith_158b_HW2.pdf" and "John_Smith_158b_HW2.zip".

This homework must be done individually. Make sure your code is well commented to show your understanding of the problems/solutions. Copying answers from other students or from online solutions is prohibited, and such cases will be reported to the campus Academic Integrity Office.

Problem Set:

Problem 1 (22pt): True or false questions. If false, explain why.

1. TCP is a more sophisticated transport layer protocol compared with UDP. It has higher performance than UDP, and is preferred by latency sensitive applications such as interactive video and online gaming.
2. TCP can guarantee a specific level of throughput for the applications, whereas UDP cannot.
3. Client/server architecture is more robust than P2P architecture against hardware failure or attacks, mainly because the former relies on very powerful server machines, while the latter consists of a network of low-profile client machines.
4. Persistent HTTP is a stateful application-layer protocol, because it needs to remember clients' states.
5. The HTTP GET message is preferred for the users to input data into forms, because all the data can be carried through the URL which is more efficient.
6. Web cookies allow a client to retrieve recently accessed webpages from its local storage, thus speeding up the web access.
7. FTP protocol sends control commands and actual data through the same TCP connection so as to the save the overhead in opening multiple TCP connections.
8. Email applications follow a P2P architecture, because the email sender and receiver can directly exchange messages with each other.
9. The DNS system is a distributed database following a tree structure. Local DNS servers are at the root of the tree. So clients' DNS requests always go to the local DNS servers first.
10. BitTorrent uses a tit-for-tat strategy to completely remove free-riders.
11. DNS redirection can help CDN load balancing.

Problem 2 (8pt): HTTP interaction with TCP.

Suppose within your Web browser you click on a link to obtain a Web page. The IP address for the associated URL is not cached in your local host, so a DNS lookup is necessary to obtain the IP address. Suppose that n DNS servers are visited before your host receives the IP address from DNS; the successive visits incur an RTT of $RTT_1, \ldots, RTT_n$. Further suppose that the Web page associated with the link contains exactly one object, consisting of a small amount of HTML text. Let RTT_0 denote the RTT between the local host and the server containing the object.

1) Assuming zero transmission time of the object, how much time elapses from when the client clicks on the link until the client receives the object?

2) Suppose the HTML file references 4 very small objects on the same server. Neglecting transmission times, how much time elapses with

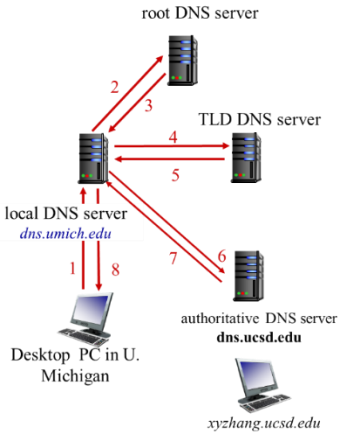
a. Non-persistent HTTP with no parallel TCP connections? (Hint: The website contains a base HTML and 4 additional objects. The browser does not know this structure before it reads the base HTML file.)

b. Non-persistent HTTP with the browser configured for 5 parallel TCP connections that can download different objects simultaneously?

c. Persistent HTTP?

Problem 3 (18pt): Answer the following questions regarding different application protocols.

1. Why does FTP separate the control and data channel? Does each channel use persistent or non-persistent TCP? Why? Is it stateful or stateless? Why?
2. Consider the example iterative DNS query in Lecture 08 (also shown below). What type of DNS records will be returned in step 5 and 7?



3. Besides host name to IP address mapping, what are other useful functionalities of DNS? How does DNS help realize these functionalities?
4. In BitTorrent, how does a peer decide on which chunks to download, and from which peers to download the chunks? Why does BitTorrent make the decision in this way?
5. How does a DHT assign (key, value) to peers? What are the advantages compared with a random assignment? What happens when a peer joins/leaves the DHT?
6. Why is CDN needed, given that web cache can already bring content very close to the users?

Problem 4 (6pt): File sharing application.

Consider distributing a file of F bits to N peers using a client-server architecture. Assume a fluid model where the server can simultaneously transmit to multiple peers, transmitting to each peer at different rates, as long as the combined rate does not exceed the server's uploading bandwidth u_s. Consider a distribution scheme in which the server sends the file to each client, in parallel, at a rate of u_s/N. Suppose that u_s/N $\leq d_{min}$, where d_{min} is the minimum downloading bandwidth of all clients. Then, how long does it take for the file to be distributed to all clients?

Problem 5 (48pt): HTTP server implementation.

In this programming assignment, you will develop a web server that handles one HTTP request at a time.

Task 1: Implement a basic Web Server. Your web server should accept and parse the HTTP request, get the requested file from the server's file system, create an HTTP response message consisting of the requested file preceded by header lines, and then send the response directly to the client. If the requested file is not present in the server, the server should send an HTTP "404 Not Found" message back to the client.
You can find the skeleton code for the Web server in the attached WebServer.py. You are to complete the skeleton code. The places where you need to fill in code are marked with "#Fill in start" and "#Fill in end". Each place may require one or more lines of code.

Here are the steps you need to follow to run and test your server implementation. Put an HTML file (e.g., HelloWorld.html) in the same directory that the server is in. Run the server program. Determine the IP address of the host that is running the server (e.g., 128.238.251.26). You are encouraged to use the AWS server (check HW1 handout), but it is fine if you use a single PC as both the client and the server. From the client host, open a browser and provide the corresponding URL. For example: http://128.238.251.26:6789/HelloWorld.html For a quick test, you can first use your localhost (corresponding to your PC's loopback IP address 127.0.0.1) as both server and client. In your report, explain your setup in detail.

'HelloWorld.html' is the name of the file you placed in the server directory. Note also the use of the port number after the colon. You need to replace this port number with whatever port you have used in the server code. In the above example, we have used the port number 6789. The browser should then display the contents of HelloWorld.html. If you omit ":6789", the browser will assume

port 80 and you will get the web page from the server only if your server is listening at port 80. Then try to get a file that is not present at the server. You should get a "404 Not Found" message.

Task 2: Persistent vs. non-persistent HTTP. Now, instead of using a browser, write your own HTTP client to test your server. Your client will connect to the server using a TCP connection, send an HTTP request to the server, and display the server response as an output. You can assume that the HTTP request sent is a GET method.

The client should take command line arguments specifying the server IP address or host name, the port at which the server is listening, and the path at which the requested object is stored at the server. The following is an input command format to run the client.

```
client.py server_host server_port HTMLfilename
```

Implement both persistent and non-persistent HTTP for this task. Specifically, suppose the server's base HTML file contains a list of objects (including 10 images and 10 text files). Non-persistent HTTP should request the base HTML file first, then it reopens a new TCP connection to download each of the objects. Whereas for persistent-HTTP, the client can send a GET message asking the server to send everything together including the base HTML and all objects it links to. You can decide on what kind of objects to use (just make sure there are 10 images and 10 text files).

For non-persistent HTTP, the HTML header should look like "GET /%s HTTP/1.1\r\n Connection: close\r\n\r\n" %(fileName).

For persistent HTTP, the HTML header should look like "GET /%s HTTP/1.1\r\n Connection: keep-alive\r\n Accept: image/*, text/*\r\n\r\n" %(fileName).

Here "fileName" is the name of the HTML file that the client wants from the server.

For both the request and response message format, check lecture 07 (the reference materials for lecture 06 contain more details).

Run and test your HTTP implementation. Meanwhile, use Wireshark to measure the total downloading time, for persistent and non-persistent HTTP, respectively. Explain your measurement *method* and *results*.

Deliverables:
For Task 1, you will hand in the complete server code along with the screen shots of your client browser, verifying that you actually receive the contents of the HTML file from the server.

For Task 2, you will hand in the complete server and client code. In addition, briefly report how you analyze the performance of persistent and non-persistent implementation, and answer the questions raised in Task 2.