Wesleyan University, Spring 2023, COMP 332

Homework 4: DNS and Web Proxies

Due by 11:59 pm on March 1, 2023

---

1. Written problems (5 points)

Problem 1. *This question will use the command-line tool* `dig` *to explore the DNS hierarchy of servers. To learn more about how* `dig` *works, type the command* `man dig` *in a terminal. Starting with a root DNS servers (e.g., choose one from* `https://www.iana.org/domains/root/servers`*), you will initiate a sequence of non-recursive DNS look-ups to resolve a hostname using dig. To help you get started, an example first command might be the following, using* `a.root-servers.net` *as the name server. Note the use of the* `@` *sign in the command.*

```
dig +norecurse @a.root-servers.net any www.wesleyan.edu
```

From this command you'll get a list of next name servers. Choose one of the name servers and execute another `dig` *command. Repeat this process until the address is resolved.*

(a): *Give the list of all name servers you use to resolve* `www.wesleyan.edu`*. List the name servers in the order that you accessed them. Describe anything interesting that you notice.*

(b): *Give the list of all name servers you use to resolve the host* `csail.mit.edu`*. List the name servers in the order that you accessed them. You will want to put an https:// in front of the ip address you get to see it in the browser.*

2. Coding (15 points)

Problem 2. *In Homework 3, you created a simple web client and web proxy (Part 1). In this homework you will extend your programs to include a cache (Part 2), use conditional GET requests (Part 3), and work with a real web browser (Part 4). For reference the web client and web proxy interactions are again as in the setup shown in Figure 1.*

Part 2: Proxy cache. *Part 2 is an intermediate step on the way to making the web proxy more realistic. The purpose of Part 2 is simply to store the responses from the various web servers to which clients connect.*

What to cache? *An HTTP response message has the format below. You may additionally see the header line,* `Last-Modified:<date>` *in the HTTP response. In a real web proxy, you would store only the entity-body along with the URL and date. However, for simplicity, and so that you don't have to parse the HTTP responses, you will save the entire HTTP response in your cache. You will want to keep track of the HTTP response itself, the URL it is associated with, and the* `Last-Modified` *date returned in the response indicating when it was last updated (or simply the* `Date` *value if there is no* `Last-Modified field`*). This information will be used in Part 3 of the project.*

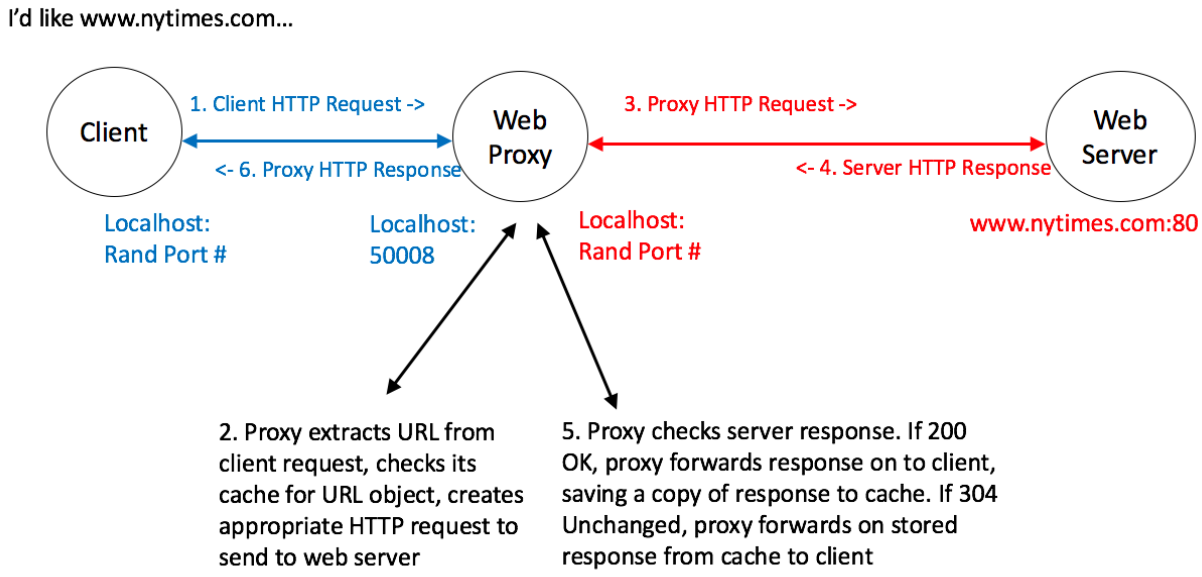

FIGURE 1. Architecture for Problem 3 programming.

```
HTTP/1.1 200 OK\r\n
Date: Tue, 20 Jan 2018 18:27:37 GMT \r\n
Connection: Close \r\n
Content-Type: text/plain \r\n
Content-Length: 5430 \r\n
Last-Modified: Sun, 18 Jan 2018 20:43:27 GMT\r\n\r\n
[ Entity-body of length 5430 bytes ]
```

Cache data structure. *You are free to choose whatever approach you'd like to organize your cache, which might range from a simple dictionary data structure, to storing webpages in files. I suggest starting with something very simple like a dictionary, so that you can move on to Part 3.*

Part 3: Conditional GET requests. *Rather than have the web proxy automatically forward on the client's GET request to the web server, the web proxy will now forward on a modified version of the GET request. How the request is modified depends on whether the webpage object for the desired URL is stored in the proxy cache. If the object for the desired URL* is not *stored in the proxy cache, the web proxy forwards on the client's GET request unmodified. If the object for the desired URL* is *stored in the proxy cache, the web proxy turns the client's GET request into a conditional GET request, using the date information stored in the cache. A conditional HTTP GET request has the following format:*

```
GET /mathcs/index.html HTTP/1.1\r\n
Host: www.wesleyan.edu\r\n
```

```
If-Modified-Since: Sun, 18 Jan 2018 20:43:27 GMT\r\n\r\n
```

In the response to the conditional GET request, one of two things may occur:

(1) **The URL object has changed, or the `If-Modified-Since` is ignored.** *In this case, the web server's response will be as in Part 1, and the web proxy forwards the web server's HTTP response directly to the client.*

(2) **The URL object has not changed.** *The web server will return a `304 Not Modified` status code in its response rather than a `200 OK` status code. In this case what the web proxy has stored in its cache is what is returned to the client. The web proxy creates an HTTP response containing the information in its cache, and forwards this response to the client. If the web proxy is storing entire responses in its cache, rather than just the entity-body, as you will be doing, then the web proxy simply returns what is stored in its cache, which should be a validly formatted HTTP Response.*

Several websites you might wish to use to test conditional requests include `example.com` and `httpforever.com`.

Part 4: Testing with a web browser. *In Part 4, you will test your web proxy using a web browser as a client instead of your web client. To do this, choose your favourite browser and change the proxy settings to use the web proxy listening on `localhost:50007`: i.e., to use your web proxy.*

Pathname parsing. *When a web browser uses a proxy, it includes the full URL ncluding the hostname as the path in the GET line of the request. This pathname, however, will not be parseable by web servers. Therefore, the web proxy must now check the pathname it is given, identify whether it is a complete URL, and if so create a new HTTP request that includes only the path and object portion of the URL in its GET line.*

You may find the following helpful when looking at HTTP requests and responses.

- **RFC 2616**: Hypertext Transfer Protocol – HTTP/1.1. `https://tools.ietf.org/html/rfc2616`
- **RFC 7232**: Hypertext Transfer Protocol (HTTP/1.1): Conditional Requests. `https://tools.ietf.org/html/rfc7232`

For this problem you should submit the following files. I recognize that as this is a two-part programming assignment, that things such as the design may change in the second part. However, part of my goal is to get you think about how you design, structure, and test your code.

- *All of your python files.* Your python code should include a header at the top of each file describing what the file does and your python code itself should be well-commented, check for exceptions and shut down nicely (e.g., clean up sockets and other state when the web client or web proxy terminates for some reason).

- *Readme file* indicating how to run your code and listing all of your python files and the purpose of each file.

- *A write-up* describing and motivating your program design, a list of web sites for which your program works, a list of example websites for which your program does not work (if any) along with an explanation why. For at least one test case for which your program works, include example screen shots of the output. I do not expect that your code will work with all web servers, since, for instance, some web servers do not use utf-8 encoding, and you are not fully parsing responses. However, in such cases you should catch whatever error is shown and continue cleanly or shutdown. Also include any test cases that cause the web client or web proxy to crash.

3. Submission

Upload your written work as `hw4.pdf`, all files (*.py, README, and your design) to the Google Drive directory I have created for you named `comp332-s23-USERNAME/hw4/`. You should replace `USERNAME` with your Wesleyan username.

Do not forget that your written work must be submitted as a PDF! Make sure that at the top of each file you have put your name! Do not, however, change the names of the files.