Wesleyan University, Fall 2022, COMP 411

Homework 3: HTTP and Web Proxies

Due by 5pm on September 30, 2022

---

1. Written problems (5 points)

Problem 1. *In this problem, you will compare persistent vs. non-persistent HTTP. Suppose your browser downloads a webpage. The webpage html object is 7 Kbits in length and additionally contains 5 embedded images, each 2.5 Kbits in length. The webpage and the 5 images are all stored on the same server. We will abstract the network path between your browser and the web server as a 50 Mbps link. Assume your browser has a 100 ms RTT to the server, which includes propagation, processing, and queuing delays. Assume also that the time to transmit a GET message is zero. You should, however, account for the time needed to setup up a TCP connection (i.e., 1 RTT) and the time to transmit the html file and the embedded images.*

a: *Assume one non-persistent HTTP connection. What is the response time? I.e., the time from when the user requests the webpage to the time when the webpage and its embedded images are displayed? Make sure you describe the various components that contribute to the response time.*

b: *Assume non-persistent HTTP and that the browser can open two parallel TCP connections to the server. What is the response time?*

c: *Assume one persistent HTTP connection with no pipelining. What is the response time?*

d: *Assume one persistent HTTP connection with pipelining. What is the response time?*

e: *Are non-persistent HTTP connections ever preferable to persistent HTTP connections?*

Solution: First, we compute some useful quantities to use to answer the questions.

$$
\begin{aligned}
d_{trans}^{(w)} &= \text{Webpage transmission delay} = 7\text{Kb}/50\text{Mbps} = 0.14 \text{ ms} \\
d_{trans}^{(i)} &= \text{Image transmission delay} = 2.5\text{Kb}/50\text{Mbps} = 0.05 \text{ ms} \\
d_{tcp} &= \text{TCP connection setup delay} = 100 \text{ ms} \\
d_{RTT} &= \text{Time to send HTTP Request and receive HTTP Response} = 100 \text{ ms}
\end{aligned}
$$

a: We compute the response time as the sum of the time to get the webpage html object and the time to get each of the images.

$$\begin{aligned}\text{Response Time} &= d_{tcp} + d_{trans}^{(w)} + d_{RTT} + 5[d_{tcp} + d_{trans}^{(i)} + d_{RTT}] \\ &= 100 \text{ ms} + 0.14 \text{ ms} + 100 \text{ ms} + 5[100 \text{ ms} + 0.05 \text{ ms} + 100 \text{ ms}] \\ &= 1200.39 \text{ ms}\end{aligned}$$

b: We have a total of six objects to get, which we can get in parallel over two sets of connections. In order to know that we need to get the images, we must first get the HTML object. Once we have the HTML object, we can split getting the images over the two sets of connections: two images on one set of connection and three images on the other. The set of connections getting three images will take longer, so that is the connection delay we use in the equation.

$$\begin{aligned}\text{Response Time} &= d_{tcp} + d_{trans}^{(w)} + d_{RTT} + 3[d_{tcp} + d_{trans}^{(i)} + d_{RTT}] \\ &= 100 \text{ ms} + 0.14 \text{ ms} + 100 \text{ ms} + 3[100 \text{ ms} + 0.05 \text{ ms} + 100 \text{ ms}] \\ &= 800.29 \text{ ms}\end{aligned}$$

c: If there is no pipelining, this means we must wait the full RTT to receive an object before sending a request for another object. Since we have a persistent HTTP connection, this means we only need to set up one TCP connection.

$$\begin{aligned}\text{Response Time} &= d_{tcp} + d_{trans}^{(w)} + d_{RTT} + 5[d_{trans}^{(i)} + d_{RTT}] \\ &= 100 \text{ ms} + 0.14 \text{ ms} + 100 \text{ ms} + 5[0.05 \text{ ms} + 100 \text{ ms}] \\ &= 700.39 \text{ ms}\end{aligned}$$

d: With pipelining, we can send requests for all of the objects without waiting a full RTT to receive an object before sending the next request. However, we must first receive the HTML webpage before requesting any other objects.

$$\begin{aligned}\text{Response Time} &= d_{tcp} + d_{trans}^{(w)} + d_{RTT} + 5d_{trans}^{(i)} + d_{RTT} \\ &= 100 \text{ ms} + 0.14 \text{ ms} + 100 \text{ ms} + 5[0.05 \text{ ms}] + 100 \text{ ms} \\ &= 300.39 \text{ ms}\end{aligned}$$

e: From the user perspective, non-persistent HTTP connections are never preferable since they result in larger delays than do persistent HTTP connections. This assumes that both non-persistent and persistent connections are allowed parallel connections.

2. Coding and hands-on problems (15 points)

Problem 2. *The purpose of this question is to have you setup Wireshark on your personal computer (or figure out how to get access to it on a lab computer), and to give you some familiarity with using Wireshark. For the following questions, you will record some basic information to show you have completed the assigned tasks using Wireshark.*

a: *Follow the instructions at* `https://www.wireshark.org/download.html` *to download and setup Wireshark. Once you have Wireshark setup, start it and select an interface on which to record. What is the interface on which you are recording traffic? Why did you choose the interface that you did?*

b: *While Wireshark is recording a trace, open an Internet browser and load the webpage* `www.nytimes.com`*. After the webpage has loaded, stop the trace recording. List the different protocols that you see. In which layer of the network protocol stack does each protocol belong? Are there protocols for which you cannot determine the appropriate layer? If so, which protocols?*

c: *Start Wireshark recording a trace. Enter the display filter* `tcp.port==80 and http` *into Wireshark. While Wireshark is running, open a terminal. Type* `nc www.nytimes.com 80` *at the terminal prompt. From the nc manual page (type man nc in a Linux or MAC terminal) we know that:*

"The nc (or netcat) utility is used for just about anything under the sun involving TCP or UDP. It can open TCP connections, send UDP packets, listen on arbitrary TCP and UDP ports, do port scanning, and deal with both IPv4 and IPv6. Unlike telnet, nc scripts nicely, and separates error messages onto standard error instead of sending them to standard output, as telnet does with some."

As the `nc` *command hangs, type the following HTTP GET request. Make sure to press enter twice after typing it.*

```
GET / HTTP/1.0
Host: www.nytimes.com
```

Take a screenshot of the packet you see in Wireshark that is generated by this command, and include the screenshot in your homework assignment.

Solution:

a: Any reasonable answer here was given full credit. I expected that most or all of you would use something like `eth0` or `en0`. You should have said that you chose the interface that connected you to the Internet in order to receive full credit.

b: I expected you to list the protocols you saw in the Protocol column in Wireshark's GUI. Any valid protocols listed here were given full credit.

For protocols that we covered in class and assigned to layers when we covered the Internet protocol stack, I expected you to correctly identify the layer for the protocol. For other protocols that we did not cover, you were not penalized for where you placed them in the network layer.

One protocol that a number of people had trouble placing, and which we didn't formally cover in class yet, was TLS/SSL. TLS uses TCP as its transport layer protocol, which puts

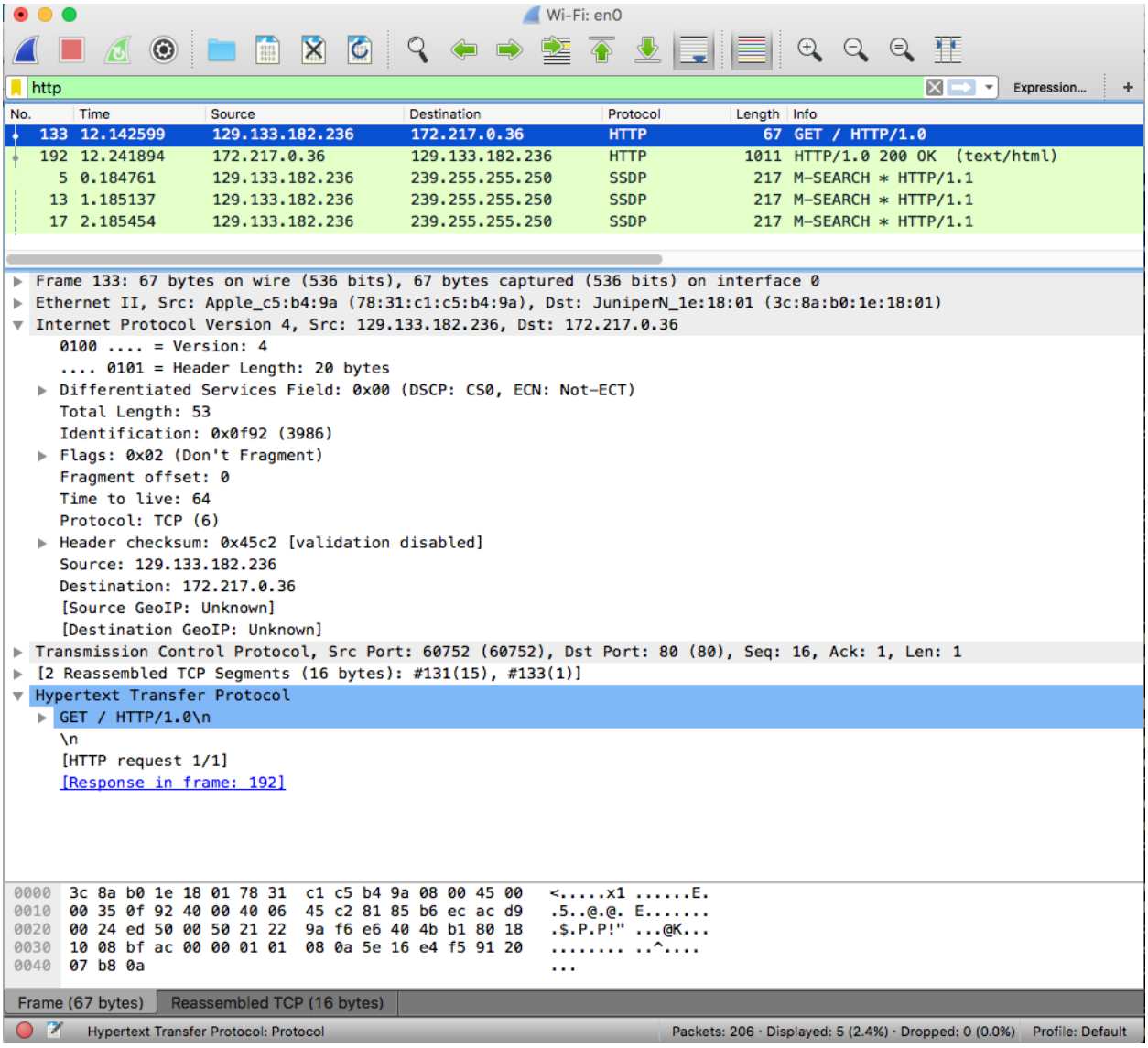

FIGURE 1. Example answer to question 5(c) showing HTTP request captured in Wireshark.

it in the application layer. However, TLS itself is used as a transport layer protocol by applications. I.e., an application can choose between using UDP, TCP or using TLS over TCP.

c: An example screen shot is shown in Figure 1.

PROBLEM 3. *The goal of this problem is to give you experience working with TCP sockets and creating and parsing HTTP requests and responses. You will implement a simple web client and web proxy using Python. The client will request a webpage via the web proxy, and the web proxy will return the webpage to the client. Project 1 is broken into 3 parts, with each part adding further complexity.*

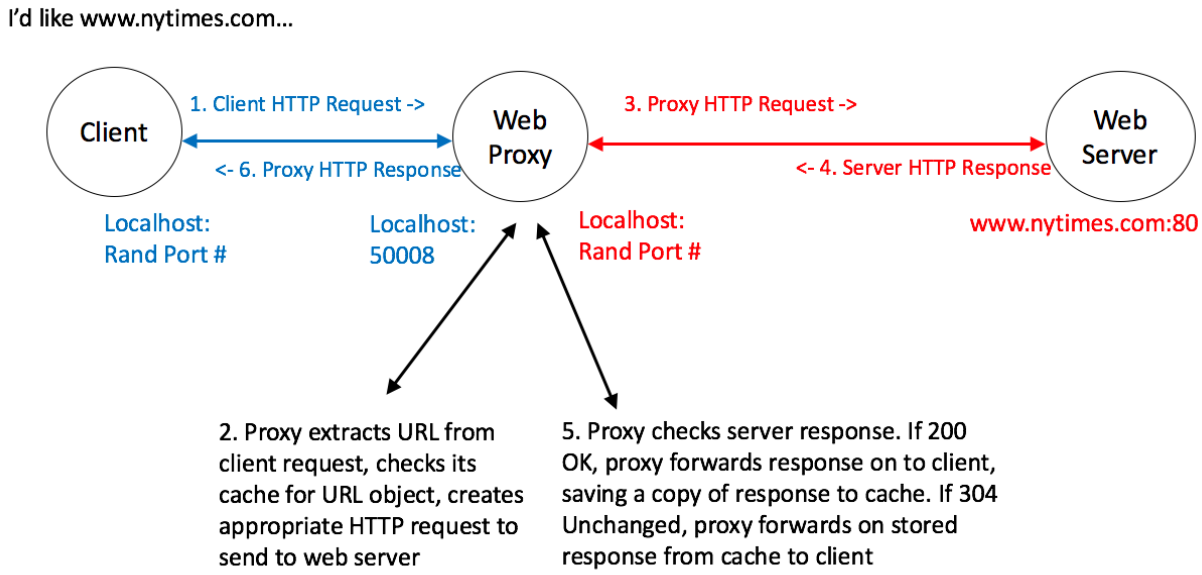

FIGURE 2. Architecture for Project 1.

In this homework assignment you will create the web client and a simple web proxy (part 1), while in the next homework assignment you will extend your program to include a cache (part 2), use conditional GET requests, and work with a real web browser (part 3). Starter code has been provided for you as a base for your code: `web_client.py` *and* `web_proxy.py`.

Part 1: URLs and GET requests. *You will create the web client and web proxy according to the setup shown in Figure 2. In Figure 2, the client and web proxy both run on your local machine using address* `localhost`*. The web server is any website on the Internet that accepts* `HTTP` *connections on port* `80`*. Note that many websites only accept* `HTTPS` *connections on port* `443`*: you will not want to use such websites for your testing. Your web proxy will listen on port* `50008` *for connections from a web client.*

- **Web client.** *The goal of the web client is to connect to a URL, such as* `www.nytimes.com` *as in Figure 2 or, for instance,* `www.wesleyan.edu/mathcs/index.html`*. This URL should be passed to the client program via the command-line when the client starts up. For testing, you may want to use a hard-coded default URL in the client code. To connect to the web proxy, the client will connect to the TCP socket on which the web proxy is listening. The client will then send an HTTP GET request into this socket to the web proxy. For instance, suppose the client wishes to connect to* `http://www.wesleyan.edu/mathcs/index.html`*. Then the client GET request will have the following format, where* `\r\n` *represents the carriage return and new line characters. The URL information contained in the host and path in the GET request indicates the desired object to download, as well as the host to which the web proxy will need to connect.*

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

- **Web proxy.** *The web proxy, upon receipt of this GET request, will parse out the hostname and open a TCP connection to that host. The web proxy will then forward the GET request to the host over this TCP connection and receive back the host's HTTP response. The web proxy will then forward the received response back to the client, over the TCP connection that the web proxy has with the client.*

I recommend writing out pseudocode for what you need to do, then adding comments for the pieces to fill in. That way, if something isn't working or you don't have time to finish something, I can see what you were trying to do and possibly give you partial credit. I suggest putting any decoding of binary data in a try-except block: that way you can still forward on the raw bytes even if you cannot decode them. You may also wish to add the following header line to the HTTP request.

```
Connection: close \r\n
```

Add this line after the `Host` line; this will close the connection after each response from the server, rather than requiring you to reassemble messages and determine whether you have received a complete message. You may also 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`

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 `hw3.pdf`, all files (*.py, README, and your design) to the Google Drive directory I have created for you named `comp411-f22-USERNAME/hw3/`. 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.