Lab 1: A Simple Web Proxy Server

50.012 Networks

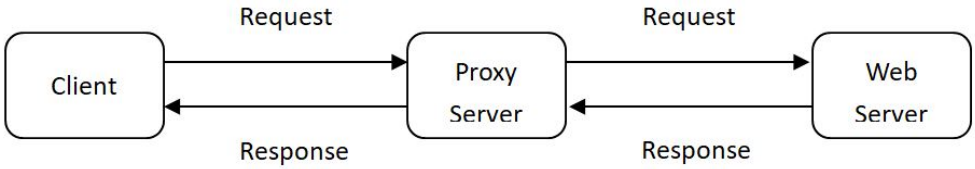
Hand-out: September 18
Hand-in: September 29 23:59

1 Objectives

In this lab, you will learn how web proxy servers work and one of their basic functionalities – caching.

Your task is to develop a simple web proxy server which is able to cache web pages. It is a very basic proxy which only understands HTTP GET requests, but is able to handle multiple kinds of objects - not just HTML pages, but also images.

Generally, when the client makes an HTTP request, the request is sent to the web server. The web server then processes the request and sends back a response message to the requesting client. In order to improve the performance we create a proxy server between the client and the web server. Now, both the request message sent by the client and the response message delivered by the web server pass through the proxy server. In other words, the client requests the objects via the proxy server. The proxy server will forward the client's request to the web server. The web server will then generate a response message and deliver it to the proxy server, which in turn sends it to the client.

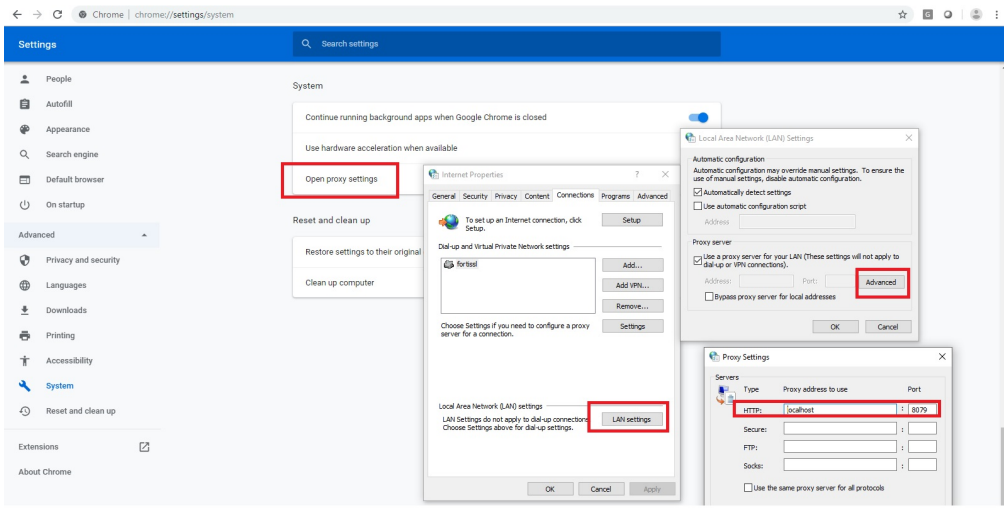


2 Code

You will be provided with the skeleton code (in Python 3) for the proxy. You are to complete the skeleton code. The places where you need to fill in your code are marked with #Fill in start and #Fill in end. Each place may require one or more lines of code. Feel free to make additional change to the code (outside the marked regions) to make it work better.

3 Running the Proxy Server

After you finish the coding, you can run the proxy. You can pass the port number you want your proxy to listen to as an optional argument. In the supplied code, it will listen to 8079 by default.



You can then configure your web browser to use your proxy. This depends on your browser. In Internet Explorer, you can set the proxy in Tools > Internet Options > Connections tab > LAN Settings. A snapshot of the configuration for chrome is given. You need to give the address of the proxy and the port number that you gave when you ran the proxy server. If you run the proxy and the browser on the same computer, you can simply use 127.0.0.1 or localhost for the proxy's address in your browser. If you run them in different computers, make sure the two computers can ping each other, and you need to find out the IP address of the computer that runs the proxy server, and use that address to configure your browser's proxy setup. After you start your proxy server and configure your browser, to get a web page, you simply provide the URL of the page you want in the browser, e.g., http://www.mit.edu (as our proxy focuses on http sites, this is one of the few stable HTTP websites that you can test against.)

Hint: after you visit a website, your browser may also keep local cache. To force your browser to ignore its local cache and contact the server, you can use Chrome's DevTools (press F12 to launch it), and from there you can first select Disable Cache, and then use Ctrl + R to refresh.

4 Hand-in

You will hand in your completed proxy server code (submitted as a single file proxy.py).

Before you submit your proxy server code, please verify that it can help your browser load the www.mit.edu web page (including both the HTML files and image contents) correctly.

5 Additional links (optional)

Note that after you putting in the minimum code to make our simple proxy run, the proxy still lacks many important features, e.g., the understanding of the conditional header field preconditions received in a request from the client. Your are encouraged to read RFC7234 `https://tools.ietf.org/html/rfc7234` to learn more about some complexity involved in implementing a proxy.

Here are some questions you can think about when reading the RFC:

- How the proxy should use the If−None−Match and ETag headers in the HTTP Request and Response respectively to decide its response to the client?

- How the proxy should use the If−Modified−Since header in the HTTP Request to decide its response to the client?
- How to support HTTP Requests with the Range header?

You are encouraged to think about how to implement some of these features in our simple proxy code.

In addition, you are encouraged to try out the following two popular tools that provide web proxy functionality as part of their feature offerings:

- Burp Suite
  - `https://portswigger.net/burp/communitydownload`
  - `https://portswigger.net/burp/documentation/desktop/tools/proxy/using`
- POSTMAN
  - `https://www.postman.com/downloads/`
  - `https://learning.postman.com/docs/sending-requests/capturing-request-data/capturing-http-requests/#built-in-proxy`

Some questions you can think about when trying these tools: What are the main use cases of these two tools? How a web proxy can play a role in their respective use cases? What are some different approaches that a proxy can take to handle Https connections?