CSC358H5: Principles of Computer Networking — Winter 2025

Programming Assignment 3: Proxy Server

Due Date: Friday, April 4, 11:59:59 PM

In PA1 and PA2, you implemented a simplified networking stack for hosts, capable of encapsulation, ARP, and simple routing functionalities in the Data Plane. In this assignment, you will learn about proxy servers, a.k.a., web cache. Your task is to develop a small web proxy server which is able to cache web pages. It is a very simple proxy server which only understands GET-requests, but is able to handle all kinds of objects- not just HTML pages, but also images, Javascripts, etc. It is worth noting that this assignment is adapted from an assignment made by Dr. Larry Yueli Zhang.

Collaboration

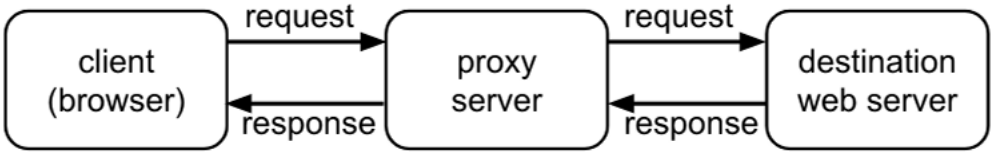
All programming assignments should be done individually. All codes that you use in your submissions should be written by you. Do NOT simply copy-and-paste code from websites such as Stack Overflow, GitHub, or even AI generating agents. It is OK to see the sample code snippets on such places, but you should write the code yourself and give the credit to them by citing the source that you used (*e.g.*, by mentioning the page URL).

You should not share your code with others or see other student codes. It is OK to discuss the assignment or potential solutions for them with another student. However, in addition to writing the whole code yourself, you should also mention this (the discussion with other students in finding the answer to the assignments) in the submissions.

You can also ask your questions on the course discussion board publicly, as long as you do not reveal your code. If there is a specific question related to a part of your code that needs revealing it, either ask it in a tutorial session or an office hour from a TA or via a private question on the discussion website.

1 Background

Generally, when the client (web browser) makes a 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. A proxy server can also be used to anonymize web surfing. The destination server receives requests from the proxy server, and thus does not receive information about the end user's address.



2 Getting Started

In this assignment, you will use Python 3's socket API to implement the TCP connections used by the proxy server.

2.1 Socket Programming in Python

To get started with Python socket programming, please review the TCP client/server examples provided in PyMOTW-3. Furthermore, you may find the official documentation of the Python 3 socket library helpful. Once your are caught up with socket programming, you may start completing your tasks in this assignment in the following steps.

2.2 Step 1: Forward HTTP requests and responses without caching

First, implement a simple TCP server (on localhost:8888) that can receive HTTP requests sent from the browser clients. Once an HTTP request is received, your proxy server needs to create a new client socket that connects to the destination web server, and forward the HTTP request there. Print out the HTTP request and take a close look: does it need to be modified before being forwarded to the destination?

After sending the request to the destination web server, you'll receive an HTTP response. You'll then forward this response back to the browser client so that the requested web page can be displayed in the browser. Things to observe and think about here: How many HTTP requests are issued in order to retrieve one web page? How do you tell that you have received the complete response from the destination?

Below are some URLs that you can use to test this step. Enter them in a web browser (Firefox or Google Chrome) and see whether you're getting the expected web page back.

`http://localhost:8888/www.example.org` (simple and small)
`http://localhost:8888/www.cs.toronto.edu/~ylzhang/` (with CSS and Javascript)
`http://localhost:8888/www.cs.toronto.edu/~arnold/` (a giant HTML to receive. Thanks, Arnold!)
`http://localhost:8888/www.pexels.com/search/nature%20wallpaper/` (one with many images)

If your proxy server works correctly for the above test cases, it's a very good start!

Pro Tip: Use the private (incognito) mode to visit these URLs, and close-reopen the window, to avoid the interference of browser caching.

2.3 Step 2: Enable the TCP server to handle simultaneous connections

If you haven't done so already, you now need to make your proxy server become capable of handling multiple incoming connections at the same time. The way to achieve this is to use the select.select() method. Below are links to a tutorial and the Python documentation. You'll figure it out!

How to work with TCP sockets in Python (with Select example)
select- waiting for I/O completion- Python 3.12.2 documentation

Make sure to maintain the select list properly by removing sockets from the list whenever they become inactive.

Note: There are other possible approaches to support simultaneous connections (such as multi-threading, forking). However, in order to meet the learning expectations of this assignment, you must use the select-based approach.

2.4 Step 3: Enable caching

Now let's add the cache. For each requested URL, we save their response from the destination in a **file on the disk** (so the cache persists when we terminate and restart the proxy server). Next time the same URL gets

requests, we will load the response from the corresponding file on the disk rather than creating a connection to the destination server. You'll see that this greatly improves the page loading speed on the browser side.

Things to think about in this step: How do we name the cache files, i.e., how do convert the URLs into proper filenames? Things not to worry about: you don't need to worry about replacement policies, i.e., how to evict an item out of the cache when the cache becomes "full". We simply assume that we have enough disk space and we never need to evict any item.

Note: To make your program portable, the cache files should be located under the current folder (where proxy.py is). Your program should NOT rely on the existence of any folder that's outside the current folder (e.g., /tmp).

2.5 Step 4: Make cache items expire

In this step, we add a parameter that specifies how long a cached item stays valid.
This parameter is passed to the program as a command argument,
e.g., `python proxy.py 120`,
which means that the cached item expires 120 seconds (2 minutes) after it's created. To implement this, you'll need to check the last-modified time of a file (using `os.path.getmtime()`) and compare it with the current time (`time.time()`). If the item expires, you need to fetch it from the destination server again and update the cache accordingly.

3 Requirements

Below are some specific requirements your code needs to satisfy just so that it can be properly marked by the TA.

1. Your code must be written in Python 3.
2. You are only allowed to have the following import statement in your code:
   `import sys, os, time, socket, select`
   No other import statement is allowed.
3. Your proxy server must be started by a command like the following:
   `python proxy.py 120`
   where 120 is the maximum age (in seconds) for an item in the cache, i.e., when set to 120, a cached item expires 120 seconds (2 minutes) after it's created. **No other action (e.g., creating a folder with a certain name) should be required to start the program.**
4. The URL entered in the browser to visit a web page via the proxy server must be like the following:
   `http://localhost:8888/the.web.page/to/visit/`
   i.e., the host name must be localhost and the port number must be 8888
5. Your code must work as expected on the Linux lab computers in DH-2020 or DH-2026, using Firefox or Chrome as the web browser.
6. Your proxy server only need to be able to handle `GET` requests
7. Your proxy server does not need to be able to handle `https` connections.

Submitting The Assignment

You can access the MarkUs submission website of the course at https://markus.teach.cs.toronto.edu/utm-2025-01/main/login_remote_auth.

We prepared the submission assignment for Programming Assignment 3 there (PA3). You will submit your "proxy.py" by using the web submission interface of MarkUs. You can submit the same filename multiple times

and only the latest version will be marked, so it is a good practice to submit your first version well before the deadline and then submit a newer version to overwrite when you make some more progress. Make sure your code runs as expected on a lab computer.
A3 is graded with a total of 100 marks and it contributes 10% towards your course grade. Step 1, 2, 3, and 4 are worth 40, 20, 30, and 10 marks, respectively.
Note that you are allowed to use up to a maximum of two grace tokens for this assignment.

Using Git

You are welcome to store your code in a private repository on GitHub, GitLab, Bitbucket, etc., but please make sure your code is not publicly accessible.