

Programming Assignment 1: HTTP Web Proxy Server Programming Assignment (Python based)

Due 26 Mar by 17:00 **Points** 24

Adapted from Kurose & Ross - Computer Networking: a top-down approach featuring the Internet

CBOK categories: Abstraction, Design, Data & Information, Networking, Programming

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

Task

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 simple HTTP/1.1 GET-requests but is able to handle all kinds of objects - not just HTML pages, but also images.

Very Important

Although this practical is marked automatically, all marks will be moderated by a marker using the following information.

You must add a comment explaining the changes you've made to **each** of your svn repository commits. In other words, each commit should explain what was changed in the code and why briefly (e.g. it may be a bug fix or a protocol behaviour fix).

We should be able to review your development process and see the changes made to code with each version you have committed. **Please keep in mind that we can check the number of lines of code as well as actual code changes made with each commit.**

During manual marking and plagiarisms checks, we will look at your development process. If we do not see a clear path to a solution (i.e. code changes and regular commits and comments reflecting your learning to develop the protocol you **will forfeit 50% of the marks allocated for this programming assignment and in the extreme case you will be allocated a mark of 0%. An example case of a 0% would be the sudden appearance of working code passing multiple tests.**

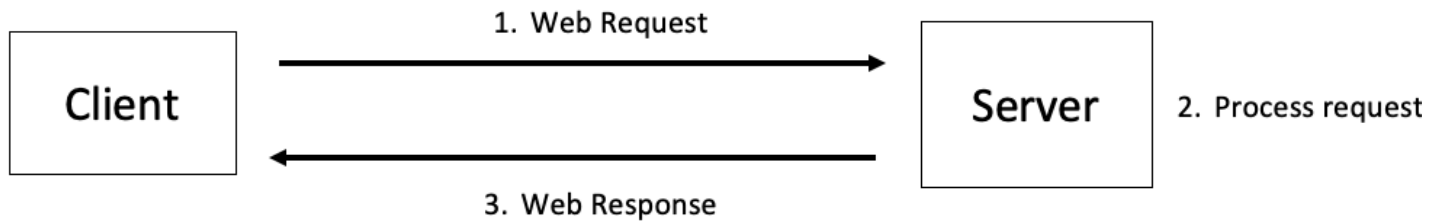
It is up to you to provide evidence of your development process.

Introduction

In this practical, you will learn how web proxy servers work and one of their basic functionalities, **caching**. Generally, when a client (e.g. your browser) makes a web request the following occurs:

1. The client sends a request to the web server
2. The web server then processes the request
3. The web server sends back a response message to the requesting client

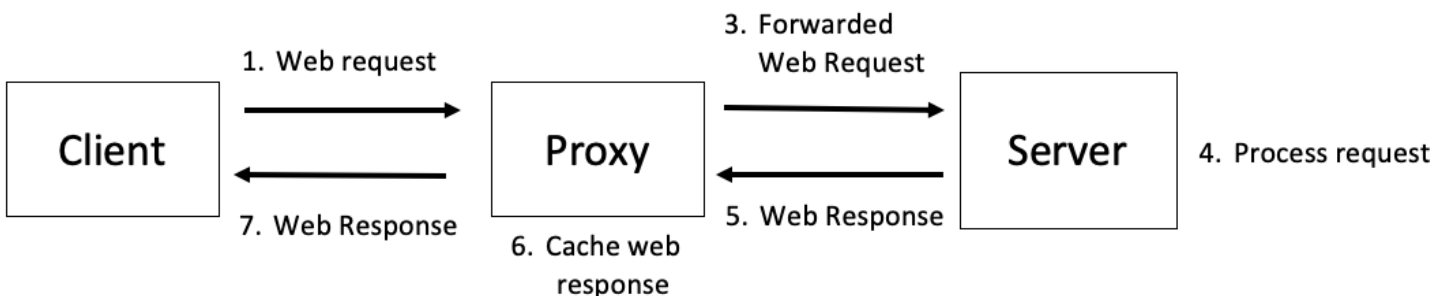
And let's say that the entire transaction takes 500 ms.



In order to improve the performance, we create a proxy server between the client and the web server. The proxy server will act as a middle-man for the web transactions. Requesting a web request will now occur in the following steps:

1. The client sends a request to the proxy server
2. Skip to step 7 If the proxy server has cached the response
3. Forward the request to the web server
4. The web server processes the request
5. The web server sends a response back to the proxy server
6. The proxy server caches the response
7. The proxy server returns the cached response to the client

On the first request, the transaction may be a fraction longer than the previous example. However, subsequent requests will be significantly faster due to reduced network latency and server load (sometimes less than 10 ms).



The mechanism for caching can be as simple as storing a copy of a resource on the proxy servers file system.

The standard you are using, HTTP1.1, in your programming assignment is described in detail in RFCs (request for comments) [see: https://en.wikipedia.org/wiki/Request_for_Comments] (https://en.wikipedia.org/wiki/Request_for_Comments%5D).

HTTP1.1 RFC we use is 2616 [<https://tools.ietf.org/html/rfc2616>]

<https://tools.ietf.org/html/rfc2616>].

Please read **Section 5.1.2** in RFC 2616.

You can read more about caching and how it is handled in HTTP in RFC 2616 in **Section 13**.

Now we want you to aim for the correct protocol/proxy behavior (see the marking rubric at the end of the practical description for what we will look for).

Steps for approaching the practical

Step 1

Understand the HTTP/1.1 requests/responses of the proxy. Your proxy **MUST** be able to handle GET requests from a client. Your proxy may handle other request types (such as POST) but this is not required.

You need to know the following:

1. What HTTP request will the browser send to the proxy?
2. What will HTTP response look like?
3. In what ways will the response look different if it comes from the proxy than if it comes from the origin server (i.e. the server where the original page is stored?). You will not be able to test this yet, but what do you think would happen?

Step 2

Understand the socket connections:

1. How will the client know to talk to the proxy?
2. What host and port number will the proxy be on?
3. The proxy may need to connect to the origin server (if the web object is not in the cache), what host and port number will the proxy connect to?
4. Where will the proxy get the web object information?


Step 3: Checkpoint

Make sure you have the answers to steps 1 & 2 before you go any further.

You can't code it if you don't know what should happen.

Ask questions on the discussions forum if you are unsure above the above.

Step 4: Python Code

Now that you know what should be happening, have a look at the code [here](https://myuni.adelaide.edu.au/courses/64831/files/8387551/download?download_frd=1)  https://myuni.adelaide.edu.au/courses/64831/files/8387551/download?download_frd=1 .

Look at the interactions you identified in steps 1 & 2 and see where they would occur in the code.

Review the python code presented in the sockets lecture. You'll find details of the socket calls in the

Python Socket library documentation <https://docs.python.org/2.7/library/socket.html>
(<https://docs.python.org/2.7/library/socket.html>).

[_ \(https://docs.python.org/2.7/library/socket.html\)](https://docs.python.org/2.7/library/socket.html) *You won't just be able to copy the lecture code*, but it shows you the key steps; creating sockets, connecting sockets, sending data on sockets and receiving data on sockets.

Your task is to make the correct socket calls and supply the correct arguments for those calls.

You will **only** need to fill in the code between the comment lines.

```
1 | # ~~~~ INSERT CODE ~~~~  
2 | ...  
3 | # ~~~~ END CODE INSERT ~~~~
```

The comments above the comments lines give you hints to the code to insert.

Step 5: Differences in Python and C

If you are new to python, look at the code structure. Most of the code is given to you with your focus just on adding the networking code. A couple of things that are different in Python than in the C derived languages:

1. Python uses whitespace to indicate code blocks and end of lines. Note there are no brackets or braces `{ }` around code blocks and no semi-colons `;` at the end of lines. The indentation isn't just important for readability in Python, it affects how your code runs. Make sure you indent code blocks correctly.
2. Python has a tuple data structure. So functions can return more than one value, or more precisely return one tuple with multiple values, but the syntax allows the parenthesis to be left off. Tuples appear in the lecture slides in the line:

```
1 | clientSocket.connect((serverName,serverPort))
```

`(serverName, serverPort)` is a tuple of two values that are passed as the argument to the `connect()` function.

```
1 | connectionSocket, addr= serverSocket.accept()
```

The `accept()` call returns a tuple of two values, the first value is the new socket and the second is the address information. This is the same as:

```
1 | (connectionSocket, addr)= serverSocket.accept()
```

The use of the `()` around the tuple is optional.

Step 6

Start with getting the proxy working when a cached file is requested. Where it will return the response

itself and does not have to contact the origin server

Step 7

Once that is working, add the code to handle the case where the file is not cached by the proxy and the proxy must request it from the origin server.

Step 8

In both steps 6 & 7, make use of both *telnet* utility *and* a browser to test your proxy server. You can also use **Wireshark** to capture what is being sent/received from the origin server.

Running your proxy in the labs and Websub

The labs and test machines are running Python 2.7. Make sure that your submission is compatible with Python 2 and does not use Python 3 features that are not backward compatible with Python 2.7.

The University proxy will intercept any attempt to connect to a host outside of the University and require authentication (which is outside the scope of this practical), thus, your proxy will not be able to connect to origin servers outside of the University when running on computers inside the University. When testing inside the University, use URLs within the University infrastructure. This should not affect you when running your proxy outside the University.

Running the Proxy Server

Download the [template file](https://myuni.adelaide.edu.au/courses/64831/files/8387551/download?download_frd=1)  (https://myuni.adelaide.edu.au/courses/64831/files/8387551/download?download_frd=1) and save it as `Proxy.py`.

Run the following command in terminal to run the proxy server

```
1 | $ python Proxy.py localhost 8888
```

You can change `localhost` to listen on another IP address for requests, localhost is the address for your machine.

You can change `8888` as the port to listen to.

The skeleton code without any modifications should (tested on Python 2.7.5 on university systems) should go into an infinite loop. You can use Ctrl+C to terminate a program.

When you are ready to test your proxy server, open a Web browser and navigate to

`http://localhost:8888/http://autoidlab.cs.adelaide.edu.au`

Note that when typing the proxy into the browser request in this way, the browser may only display the main page and may fail to download associated files such as images and style sheets. This is due to a difference in URI requirements when sending requests to a proxy vs sending to an origin Web server (we'll look at this in the tutorials).

Configuring your Browser (Optional)

You can also directly configure your web browser to use your proxy if you want without using the URI. This depends on your browser.

- In Internet Explorer, you can set the proxy in *Tools > Internet Options > Connections tab > LAN Settings*.
- In Netscape (and derived browsers such as Mozilla), you can set the proxy in *Tools > Options > Advanced tab > Network tab > Connection Settings*.

In both cases, you need to give the address of the proxy and the port number that you set when you ran the proxy server. You should be able to run the proxy and the browser on the same computer without any problem. With this approach, to get a web page using the proxy server, you simply provide the URL of the page you want.

For example, running `http://autoidlab.cs.adelaide.edu.au` would be the same as running `http://localhost:8888/http://autoidlab.cs.adelaide.edu.au`

Set up like this, the browser should successfully load both the main web page and all associated files due to reasons we'll discuss in tutorials.

Testing your code

- When it comes time to test your code, `cUrl` and `telnet` are useful tool to use.
- If you want to test on an image that is the same as the web submission system, you can ssh into: `uss.cs.adelaide.edu.au` and then run and test your code.

Let's have a look at these two tests:

Obtaining a remote web page

```
1 | $ curl -si http://localhost:8080/http://autoidlab.cs.adelaide.edu.au/ | head -n 1
2 | HTTP/1.1 200 OK
```

The above command requests `http://autoidlab.cs.adelaide.edu.au` (`http://autoidlab.cs.adelaide.edu.au`) via the Web proxy. `-i` prints out response with headers and removes additional output and `head -n 1` extracts the first line from the cUrl command.

The result is the first line in the response from the proxy. This response should match if you were talking directly to the origin server, like so

```
1 | $ curl -sI http://autoidlab.cs.adelaide.edu.au/ | head -n 1
2 | HTTP/1.1 200 OK
```

Handle page that does not exist

```
1 | $ curl -sI http://localhost:8080/http://autoidlab.cs.adelaide.edu.au/fakefile.html | head -n 1
2 | HTTP/1.1 404 Not Found
```

The response for a path that doesn't exist shows the above status code. Your proxy to work correctly will also need to handle this case too.

Using telnet

```
1 [prompt]$ telnet localhost 8080
2 Trying 127.0.0.1...
3 Connected to localhost.
4 Escape character is '^]'.
5 GET http://autoidlab.cs.adelaide.edu.au HTTP/1.1
```

Note that we used the absolute URL because telnet here is connected to you proxy and not the origin server.

If you are very unfamiliar and having trouble, this video demo recorded by David showing an example use of both tools to connect and send GET requests will be handy. ***Note that in the video, David is not connecting to a proxy but connecting to a web server and issuing HTTP requests to the web server.***

<https://tinyurl.com/cm79nyb> [_\(https://tinyurl.com/cm79nyb\)](https://tinyurl.com/cm79nyb)

Be aware! Using arbitrary URLs to for testing

When testing, be aware that for certain URLs, although you issue http:// requests, for security reasons, they might get directed to https://. It has now become common practice for most websites to redirect http:// to https://. So, you may have seen 30x like responses. This redirects your browser to goto the HTTPS secure web server. So when you tried to do something like <http://localhost:8888/http://ecms.adelaide.edu.au/> [\(http://localhost:8888/http://ecms.adelaide.edu.au/\)](http://localhost:8888/http://ecms.adelaide.edu.au/) the browser will read the response and load the HTTPS URL <https://ecms.adelaide.edu.au/>, [_\(https://ecms.adelaide.edu.au%2C/\)](https://ecms.adelaide.edu.au/%2C/) thus navigating away from your proxy.

Here are two you can use where there are no redirects.

- <http://autoidlab.cs.adelaide.edu.au/> [_\(http://autoidlab.cs.adelaide.edu.au/\)](http://autoidlab.cs.adelaide.edu.au/)
- <http://jsonplaceholder.typicode.com/todos/1> [_\(http://jsonplaceholder.typicode.com/todos/1\)](http://jsonplaceholder.typicode.com/todos/1)

Here is a link to an image you can use to test on a file other than an html file.

- <http://autoidlab.cs.adelaide.edu.au/sites/default/files/projectimages/PeerTrack.jpg>

Submission

Your proxy server will be inside the `Proxy.py` file only.

Your work will need to be submitted to **Websub** [_\(https://cs.adelaide.edu.au/services/websubmission/\)](https://cs.adelaide.edu.au/services/websubmission/) with the assignment path `/YYYY/s1/cna/webproxy` Here, replace YYYY with year.

The Web submission system will perform a static analysis of your code. Your code will be run and <https://myuni.adelaide.edu.au/courses/64831/assignments/211809>

The VCS submission system will perform a static analysis of your code. Your code will be run and marked after the deadline.

Bonus Marks Question (Optional)

Your mission, should you chose to accept it, will be rewarding!

If you have completed Prac 1 and would like some extension work. Here are three questions to tackle for 6 bonus marks (*so if you get full marks for the prac, the 6 bonus marks will be on top of that*).

Bonus mark questions (2 marks for each component that is correctly implemented)

Our current proxy doesn't check if the cached file is still **fresh**. In practice, the proxy server must verify that the cached responses are still valid and that they are consistent with the responses the client would receive from the origin server. You can read more about caching and how it is handled in HTTP in RFC 2616.

Modify your web proxy to handle the following:

1. Check the Expires header of cached objects to determine if a new copy is needed from the origin server instead of just sending back the cached copy (2 marks)
2. Respond correctly to the cache-control header `max-age=<seconds>` sent by the client. (2 marks)
3. The current proxy only handles URLs of the form `hostname/file` Add the ability to handle origin server ports that are specified in the URL, i.e. `hostname:portnumber/file` (2 marks)

Hand in: please hand in your `Proxy.py` and `Proxy-bonus.py` for the version that contains the solution to the bonus questions. You can edit any part of the code necessary for `Proxy-bonus.py`. (Please note that you are handing in two files). However

1. At the start of the `Proxy-bonus.py` file explain which of the above you have implemented.
2. For each implementation and any code added, please clearly document your code so that it is clear how you built your solution for each of the three extension questions above. **No comments or no explanation implies no marks, sorry! So please document your work so we can easily understand your implementation.**

Prac 1: Web Proxy

Criteria	Ratings		Pts
Proxy server started	0 Pts Full Marks	0 Pts No Marks	0 pts
Connected to Proxy server	10 Pts Full Marks	0 Pts No Marks	10 pts
Obtained remote homepage	2 Pts Full Marks	0 Pts No Marks	2 pts
Obtained remote file	2 Pts Full Marks	0 Pts No Marks	2 pts
Handle page that does not exist	2 Pts Full Marks	0 Pts No Marks	2 pts
Cache requested webpages	2 Pts Full Marks	0 Pts No Marks	2 pts
Read from a cached file	1 Pts Full Marks	0 Pts No Marks	1 pts
Redownloaded the file from server after file was removed	1 Pts Full Marks	0 Pts No Marks	1 pts
Handles internal server error	4 Pts Full Marks	0 Pts No Marks	4 pts
Total points: 24			