Socket Programming: HTTP Web Proxy Server (without POST)

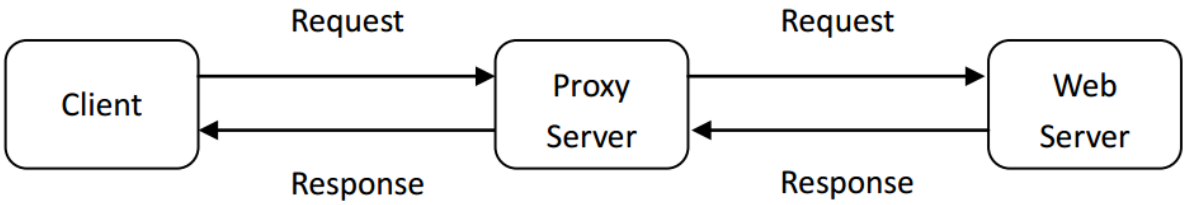
In this project, you will learn how web proxy servers work and one of their basic functionalities – caching.

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

Generally, when the client 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.

With respect to the client, the proxy server behaves like a server, and with respect to the web server, the proxy server behaves like a client. In what follows, the web server will also be referred to as "original server".

Your code has to implement inter-process communication at the socket level: It has to explicitly create sockets, send data to the sockets, and receive data from the sockets. You are not allowed to use higher level mechanisms and you won't get credit if you do so.



1. Programming Language

There is some flexibility for the programming language, but you need to get the TA's approval.

2. Running the Proxy Server

To run the proxy server program locally on your computer, start the proxy using your command prompt. Then launch your browser. For the browser to direct requests to the proxy server, use "localhost", along with the destination port number.

For example, to go to google.com, instead of typing www.google.com, you would type http://localhost:5005/www.google.com

5005 is an arbitrarily chosen port number where the client can reach the proxy server. The only requirement is that the port number should not coincide with any of the reserved port numbers. To use the proxy server with browser and proxy on separate computers, you will need the IP address on which your proxy server is running. In this case, while running the proxy, you will have to replace the "localhost" with the IP address of the computer where the

proxy server is running. Also note the port number used. You will replace the port number used here “5005” with the port number you have used in your server code at which your proxy server is listening.

3. Outline for the Proxy Server Logic

HTTP runs over TCP, so you will be using TCP sockets. You should insert print statements in your code as shown below.

Create a server socket welcomeSocket, bind to a port p # This is the welcoming socket used to listen to requests coming from a client

Listen on welcomeSocket

While True

    # print ‘WEB PROXY SERVER IS LISTENING’

    Wait on welcomeSocket to accept a client connection

    When a connection is accepted, a new client connection socket is created. Call that socket clientSocket

    Read the client request from clientSocket

    # Print the message received from the client (see note below)

    Process the request

        Split the message to extract the header

        Parse the header to extract the method, dest address, HTTP version

        # Print the extracted method, dest address and HTTP version (see note below)

        if method is GET

            Look up the cache to determine if requested object is in the cache

            if object is not in the cache # Need to request object from the original server

                # Print “Object not found in the cache” message

                Create a serverSocket to send request to the original server

                Compose the request header, send request

                # Print the request sent to the original server (see note below)

                Read response from the original server

                Parse it to extract the relevant components

                # Print the response header from the original server (see note below)

                if response is 200 OK

                    Write object into cache

                Close serverSocket

                Compose response and send to client on clientSocket

                # Print the response header from the proxy to the client (see note below)

            if object is in the cache

                # Print “Object found in the cache” message

                Read from the cache, compose response, send to client on clientSocket

                # Print the response header from the proxy to the client (see note below)

        else # Method is not GET

Create a serverSocket to send request to the original server
Compose the request header, send request
Print the request sent to the original server
Read response from the original server
Parse it to extract the relevant components
Print the response header from the original server
Compose response to the client and send on clientSocket
Print the response header from the proxy to the client
Close clientSocket
End of while True
Close welcomeSocket

Note: Examples can be found in the screenshots below

HTTP Processing done by Proxy

Examples of parsing done by the Proxy to extract the method, dest address, HTTP version, hostname, URL, file name from the client HTTP request, and examples of the Proxy building the request sent to the original server can be found in the screenshots below.

4. Required Extensions

In addition to the above, you are required to implement the following.

Error Handling

Currently the proxy server does no error handling. This can be a problem especially when the client requests an object which is not available, since the "404 Not found" response usually has no response body and the proxy assumes there is a body and tries to read it. Extend the code to be able to handle the "404 Not found" case.

Caching

Caching: A typical proxy server will cache the web pages each time the client makes a particular request for the first time. The basic functionality of caching works as follows. When the proxy gets a request, it checks if the requested object is cached, and if yes, it returns the object from the cache, without contacting the server. If the object is not cached, the proxy retrieves the object from the server, returns it to the client and caches a copy for future requests. In practice, the proxy server must verify that the cached responses are still valid and that they are the correct responses to the client's requests. You can read more about caching and how it is handled in HTTP in RFC 2068. Add the simple caching functionality described above. You do not need to implement any replacement or validation policies. Your implementation, however, will need to be able to write responses to the disk (i.e., the cache) and fetch them from the disk when you get a cache hit. For this you need to implement some internal data structure in the proxy to keep track of which objects are cached and where they are on the disk. You can keep this data structure in main memory; there is no need to make it persist across shutdowns.

Not in the scope of the assignment: Support for POST

The simple proxy server supports only HTTP GET method. Add support for POST, by including the request body sent in the POST-request.

5. Validation Scenarios

In order to validate your code, use the following scenarios. Note the details of HTTP messages may depend on the browser used. The screenshots are not intended to be normative, but to give you an idea of the type of info the proxy should display.

gaia.cs.umass.edu/wireshark-labs/HTTP-wireshark-file4.html, response is 200 OK

Do the following for <WebSite> = gaia.cs.umass.edu/wireshark-labs/HTTP-wireshark-file4.html

Step	Action	Expected behavior
0	Clear the browser's cache and make sure the proxy's cache is empty	
1	Start your proxy	Proxy should display "WEB PROXY SERVER LISTENING" or similar message
2	Launch the browser and type http://localhost:5005/<WebSite>	The browser should display the page requested (screenshot #3). In addition, the Proxy should display the info shown in screenshot #1. In particular, it should indicate the objects requested by the client are not found in the cache. At the end of this step, check that the objects requested by the client are in the cache.
3	Clear the browser's cache, start a new tab and type again http://localhost:5005/<WebSite>	The browser should display the page requested (screenshot #3). In addition, the Proxy should display the info shown in screenshot #2. In particular, it should indicate the objects requested by the client are found in the cache.

Screenshot #1 (Objects not in cache)

```
MESSAGE RECEIVED FROM CLIENT:
GET /gaia.cs.umass.edu/wireshark-labs/HTTP-wireshark-file4.html HTTP/1.1
Host: localhost:5005
Connection: keep-alive
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/80.0.3987.163 Safari/537.36
Sec-Fetch-Dest: document
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Sec-Fetch-Site: none
Sec-Fetch-Mode: navigate
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9,fr;q=0.8,vi;q=0.7

END OF MESSAGE RECEIVED FROM CLIENT

[PARSE MESSAGE HEADER]:
 METHOD = GET, DESTADDRESS = gaia.cs.umass.edu/wireshark-labs/HTTP-wireshark-file4.html, HTTPVersion = HTTP/1.1

[LOOK UP IN THE CACHE]: NOT FOUND, BUILD REQUEST TO SEND TO ORIGINAL SERVER
[PARSE REQUEST HEADER] HOSTNAME IS gaia.cs.umass.edu
[PARSE REQUEST HEADER] URL IS wireshark-labs/HTTP-wireshark-file4.html
[PARSE REQUEST HEADER] FILENAME is HTTP-wireshark-file4.html
```

```
REQUEST MESSAGE SENT TO ORIGINAL SERVER:
GET /wireshark-labs/HTTP-wireshark-file4.html HTTP/1.1
Host: gaia.cs.umass.edu
Connection: close
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/80.0.3987.163 Safari/537.36
Sec-Fetch-Dest: document
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Sec-Fetch-Site: none
Sec-Fetch-Mode: navigate
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9,fr;q=0.8,vi;q=0.7


END OF MESSAGE SENT TO ORIGINAL SERVER

RESPONSE HEADER FROM ORIGINAL SERVER:
HTTP/1.1 200 OK
Date: Sat, 18 Apr 2020 01:59:32 GMT
Server: Apache/2.4.6 (CentOS) OpenSSL/1.0.2k-fips PHP/5.4.16 mod_perl/2.0.11 Perl/v5.16.3
Last-Modified: Fri, 17 Apr 2020 05:59:03 GMT
ETag: "2ca-5a3763f1bb9d9"
Accept-Ranges: bytes
Content-Length: 714
Connection: close
Content-Type: text/html; charset=UTF-8
END OF HEADER

[WRITE FILE INTO CACHE]:  cache/HTTP-wireshark-file4.html
```

```
RESPONSE HEADER FROM PROXY TO CLIENT:
HTTP/1.1 200 OK
Content-Type: text/html; charset=UTF-8

END OF HEADER
```

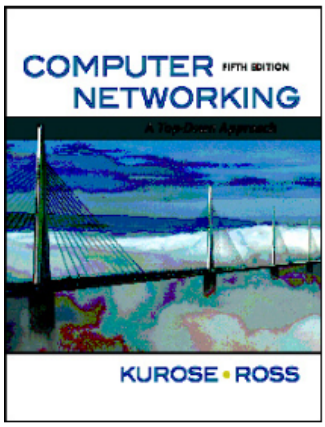
Screenshot # 2 (Objects in cache)

```
MESSAGE RECEIVED FROM CLIENT:
GET /gaia.cs.umass.edu/wireshark-labs/HTTP-wireshark-file4.html HTTP/1.1
Host: localhost:5005
Connection: keep-alive
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/80.0.3987.163 Safari/537.36
Sec-Fetch-Dest: document
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Sec-Fetch-Site: none
Sec-Fetch-Mode: navigate
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9,fr;q=0.8,vi;q=0.7

END OF MESSAGE RECEIVED FROM CLIENT

[PARSE MESSAGE HEADER]:
 METHOD = GET, DESTADDRESS = gaia.cs.umass.edu/wireshark-labs/HTTP-wireshark-file4.html, HTTPVersion = HTTP/1.1

[LOOK UP THE CACHE]: FOUND IN THE CACHE: FILE =cache/HTTP-wireshark-file4.html



RESPONSE HEADER FROM PROXY TO CLIENT:
HTTP/1.1 200 OK
Content-Length: 714
Content-Type: text/html;

END OF HEADER
```

Screenshot # 3

Pearson

This little HTML file is being served by gaia.cs.umass.edu. It contains two embedded images.
The image above, also served from the gaia.cs.umass.edu web site, is the logo of our publisher, Pearson.
The image of our 5th edition book cover below is stored at, and served from, the www server caite.cs.umass.edu:



www.google.com/unknown, response is 404 not found

Do the following for <WebSite> = www.google.com/unknown (assume the Proxy is already running)

Step	Action	Expected behavior

	Clear the browser's cache and type http://localhost:5005/<WebSite>	The browser should display the "404 not found" page. In addition, the Proxy should display the info shown in screenshot #4.

Screenshot #4

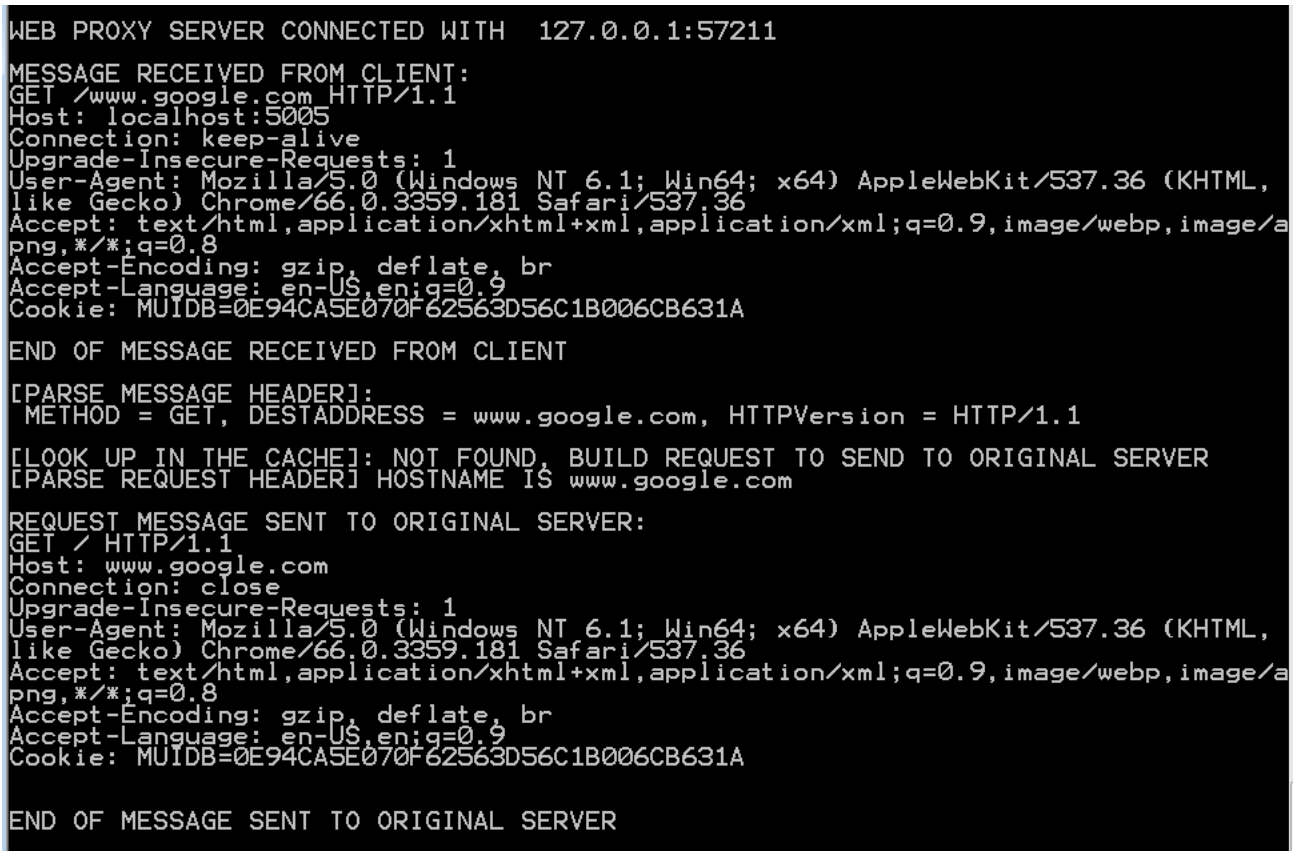
www.google.com, response is 302 found (redirection)

Do the following for <WebSite> = www.google.com (assume the Proxy is already running)

Step	Action	Expected behavior

	Clear the browser's cache and type http://localhost:5005/<WebSite>	The browser should display the page requested. In addition, the Proxy should display the info shown in screenshot #5.

Screenshot #5



```
RESPONSE HEADER FROM ORIGINAL SERVER:
HTTP/1.1 302 Found
Location: https://www.google.com/?gws_rd=ssl
Cache-Control: private
Content-Type: text/html; charset=UTF-8
P3P: CP="This is not a P3P policy! See g.co/p3phelp for more info."
Date: Sat, 26 May 2018 14:40:25 GMT
Server: gws
Content-Length: 231
X-XSS-Protection: 1; mode=block
X-Frame-Options: SAMEORIGIN
Set-Cookie: 1P_JAR=2018-05-26-14; expires=Mon, 25-Jun-2018 14:40:25 GMT; path=/;
 domain=.google.com
Set-Cookie: MUIDB=; expires=Mon, 01-Jan-1990 00:00:00 GMT; path=/; domain=www.go
ogle.com
Set-Cookie: MUIDB=; expires=Mon, 01-Jan-1990 00:00:00 GMT; path=/; domain=.www.g
oogle.com
Set-Cookie: MUIDB=; expires=Mon, 01-Jan-1990 00:00:00 GMT; path=/; domain=google
.com
Set-Cookie: MUIDB=; expires=Mon, 01-Jan-1990 00:00:00 GMT; path=/; domain=.googl
e.com
Set-Cookie: NID=131=NjGT4IE0R9146sxDA4wdTiGC34N98XIAZwjEwDAsJ5AEglCXMzP6IVaTt30Z
VSp4YKqmW3v16e4p-3qZniZWfMgUlLDBYldbPlKD9vsEcuaPK7Z24msz0cRbYw199XI7; expires=Su
n, 25-Nov-2018 14:40:25 GMT; path=/; domain=.google.com; HttpOnly
Connection: close
END OF HEADER


RESPONSE HEADER FROM PROXY TO CLIENT:
HTTP/1.1 302 Found
Location: https://www.google.com/?gws_rd=ssl
Cache-Control: private
Content-Type: text/html; charset=UTF-8
P3P: CP="This is not a P3P policy! See g.co/p3phelp for more info."
Date: Sat, 26 May 2018 14:40:25 GMT
Server: gws
Content-Length: 231
X-XSS-Protection: 1; mode=block
X-Frame-Options: SAMEORIGIN
Set-Cookie: 1P_JAR=2018-05-26-14; expires=Mon, 25-Jun-2018 14:40:25 GMT; path=/;
 domain=.google.com
Set-Cookie: MUIDB=; expires=Mon, 01-Jan-1990 00:00:00 GMT; path=/; domain=www.go
ogle.com
Set-Cookie: MUIDB=; expires=Mon, 01-Jan-1990 00:00:00 GMT; path=/; domain=.www.g
oogle.com
Set-Cookie: MUIDB=; expires=Mon, 01-Jan-1990 00:00:00 GMT; path=/; domain=google
.com
Set-Cookie: MUIDB=; expires=Mon, 01-Jan-1990 00:00:00 GMT; path=/; domain=.googl
e.com
Set-Cookie: NID=131=NjGT4IE0R9146sxDA4wdTiGC34N98XIAZwjEwDAsJ5AEglCXMzP6IVaTt30Z
VSp4YKqmW3v16e4p-3qZniZWfMgUlLDBYldbPlKD9vsEcuaPK7Z24msz0cRbYw199XI7; expires=Su
n, 25-Nov-2018 14:40:25 GMT; path=/; domain=.google.com; HttpOnly
Connection: close

END OF HEADER
```

6. Research Paper Topics

Write a research paper on one of the below topics. The research paper is essentially a compilation from the public literature, and you need to write it for your class mates as target audience. You may be asked to make an in-class presentation of your paper (or a subset of it). More details on the presentation will be provided at a later stage.

In the first phase, you are required to submit for approval a proposed outline listing the specific items in your paper before you work on the detailed paper to be submitted in the final team report. Below are the topics, please refer to the "Research-paper" document for details.

- Overview of SPDY/HTTP 2 and Google's QUIC
- SSL/TLS

7. What to Turn In

1. A proposed action plan by the due date specified in "ProjectTimeline" in "Projects Overview".

2. A proposed outline of the research paper for approval by the due date specified in "ProjectTimeline" in "Projects Overview".

3. A team report by the due date specified in "ProjectTimeline" in "Projects Overview".

   a) Including the complete code for including the extensions. The complete code includes the pieces you modified or wrote and the pieces you did not have to modify. Include a README file that describes how to compile/run the program.
   b) Providing a background summary of HTTP and caching
   c) Describing your hardware setup and configuration
   d) Including the design document of your code. This should give a description of the various design choices you have made, for example the error handling function, caching mechanism, etc.
   e) Providing screenshots at the client side verifying that you indeed get the web page via the proxy server in the normal case, and screenshots of other cases such as "404 Not found". Also include screenshots of the info displayed by the Proxy (HTTP messages, results of parsing, result of cache look up, etc.)
   f) Include screenshots of the client side browser verifying that you indeed get the web page via the proxy server in the normal case,.
   g) Describing what issues, if any, the team encountered during the project, how the team overcame the issues and what the team learned from the project. You can also provide suggestions on how the projects in Computer Networks could be improved in the future.
   h) Including a video clip to demo the code running
   i) Including the research paper

4. Individual reports, one for each team member by the due date specified in "ProjectTimeline" in "Projects Overview". The individual report is confidential and not shared with the other team members.

   a) If you, as an individual team member, have anything specific to add to 1.f) in the team report, please do it in your individual report. Describe what issues, if any, you, as an individual team member, encountered during the project, how you overcame the issues and what you learned from the project (this is not necessarily just about the topic, could be related to teamwork, etc.). You can also provide suggestions on how the projects in Computer Networks could be improved in the future. This complements the team report with any individual viewpoint not included in the team report.
   b) Describe what each team member (including yourself) did and contributed to the project, and assign a numerical score from 1 to 10 to each team member, including yourself. 1 is the poorest, and 10 is the best.

5. Powerpoint slides for an in-class presentation, by the date specified in "ProjectTimeline" in "Projects Overview". The scope of the presentation will be specified at a later date, but it is a subset of the research paper.

Note: There will be a separate session (taking place outside of lecture hours) for you to demo your code is running and answer questions about your code and design. The in-class presentations and the code demo sessions will take place towards the end of the semester.

8. Grading Criteria

General Note: The following are criteria used to come up with a team grade. Your final individual project score is not necessarily the team grade, it may be flexed up or down, depending on your individual contribution to the team and the quality of your individual report.

Coding (50%)

Source code of your well-structured and well-documented program. Include comments on your codes for clarification. In addition, your code has to meet the validation scenarios in the project description. **You should be**

able to demonstrate good understanding of the code and be able to answer specific questions on the code and design.

Documents (50%)

Group Report (24%)
The group report will be evaluated not only on its content, but also the professionalism of its appearance.

Research Paper and Presentation (26%)
Refer to the “Research-paper” for details on the criteria.