To Kim Bao Pham and Omar Resendiz

Proxy Server Project Team Report

I. HTTP and Caching:

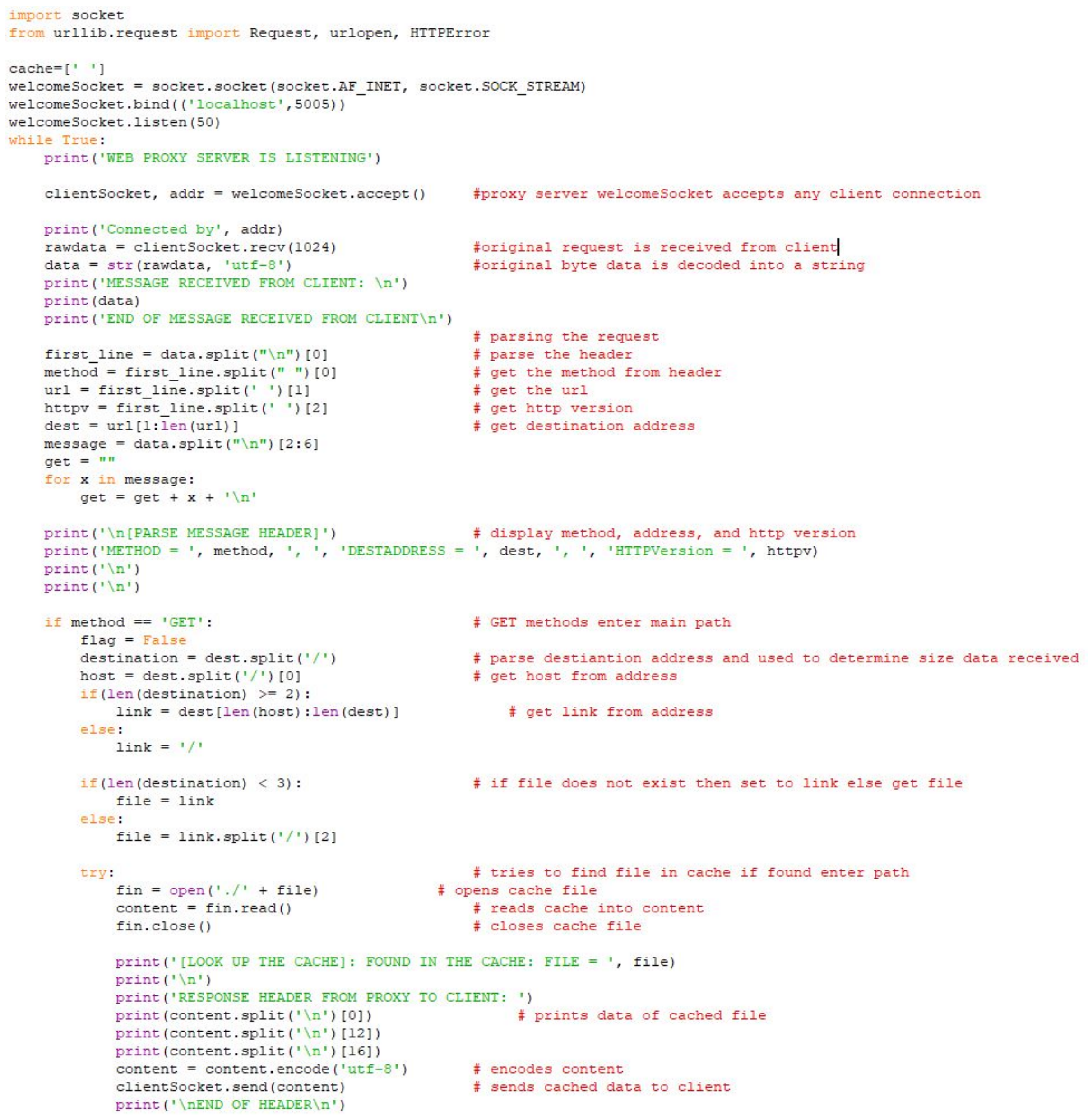
HTTP is the message protocol that is used for communication between clients and the web servers. This protocol is applicable for the application layer. HTTP has two types of messages, which are request and response. The request message can have different methods such as GET, POST or HEAD, which are the ways the client requests objects from the web. The response message is from the web server, which has different number codes to describe the status of the request such as 200, 404, 302,... The web proxy server project focuses on the GET request message and various different types of response statuses.

Caching is a popular mechanism of a proxy server and web browsers for storing recently requested objects. The cache is searched for every request, and if there is a cache hit, the server would send the stored object to the client without the need to contact the web server.

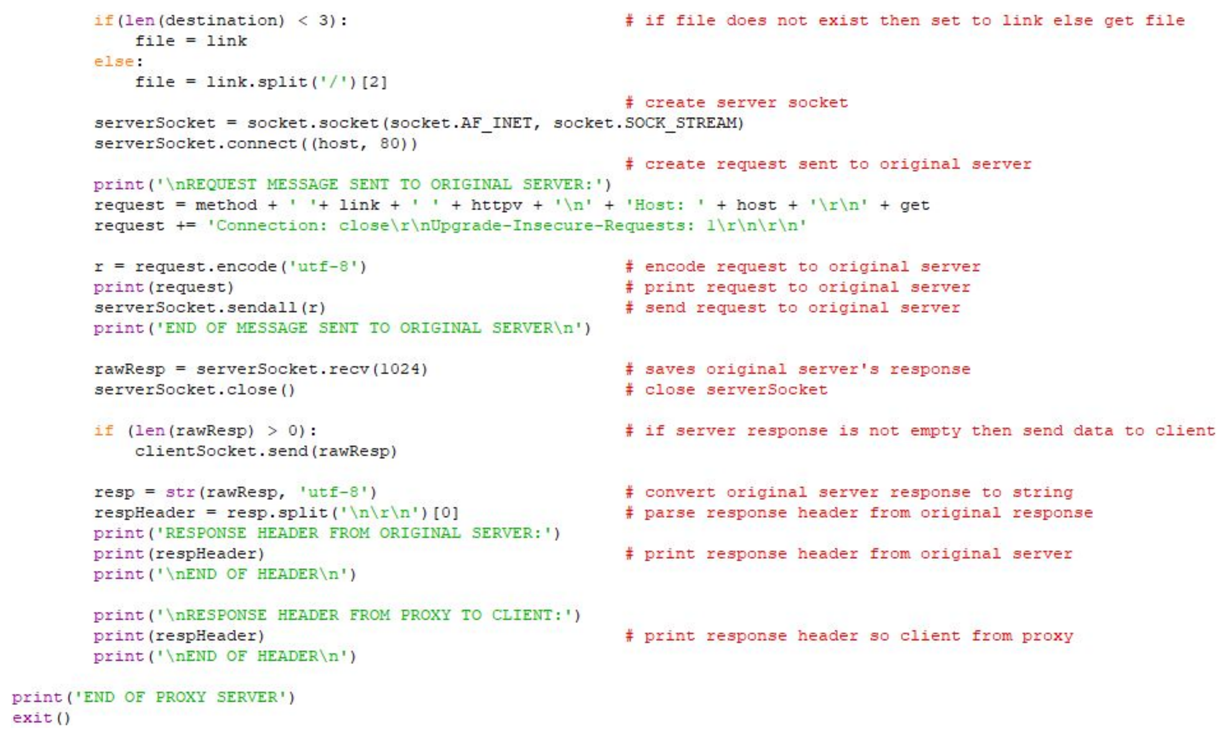
II. Hardware Setup and Configuration:

The only configuration we had to apply for the web proxy server is to turn the proxy option for our laptop’s settings on before execution. The proxy address is set to ‘localhost’ and the port number is 5005. The program can be run with the Python Shell and a browser with a clear cache to send requests to the proxy.

III. Complete Code:



```
import socket
from urllib.request import Request, urlopen, HTTPError

cache=[' ']
welcomeSocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
welcomeSocket.bind(('localhost',5005))
welcomeSocket.listen(50)
while True:
    print('WEB PROXY SERVER IS LISTENING')

    clientSocket, addr = welcomeSocket.accept()     #proxy server welcomeSocket accepts any client connection

    print('Connected by', addr)
    rawdata = clientSocket.recv(1024)               #original request is received from client
    data = str(rawdata, 'utf-8')                    #original byte data is decoded into a string
    print('MESSAGE RECEIVED FROM CLIENT: \n')
    print(data)
    print('END OF MESSAGE RECEIVED FROM CLIENT\n')
                                                    # parsing the request
    first_line = data.split("\n")[0]                # parse the header
    method = first_line.split(" ")[0]               # get the method from header
    url = first_line.split(' ')[1]                  # get the url
    httpv = first_line.split(' ')[2]                # get http version
    dest = url[1:len(url)]                          # get destination address
    message = data.split("\n")[2:6]
    get = ""
    for x in message:
        get = get + x + '\n'

    print('\n[PARSE MESSAGE HEADER]')               # display method, address, and http version
    print('METHOD = ', method, ', ', 'DESTADDRESS = ', dest, ', ', 'HTTPVersion = ', httpv)
    print('\n')
    print('\n')

    if method == 'GET':                             # GET methods enter main path
        flag = False
        destination = dest.split('/')               # parse destiantion address and used to determine size data received
        host = dest.split('/')[0]                   # get host from address
        if(len(destination) >= 2):
            link = dest[len(host):len(dest)]            # get link from address
        else:
            link = '/'

        if(len(destination) < 3):                   # if file does not exist then set to link else get file
            file = link
        else:
            file = link.split('/')[2]

        try:                                        # tries to find file in cache if found enter path
            fin = open('./' + file)             # opens cache file
            content = fin.read()                    # reads cache into content
            fin.close()                             # closes cache file

            print('[LOOK UP THE CACHE]: FOUND IN THE CACHE: FILE = ', file)
            print('\n')
            print('RESPONSE HEADER FROM PROXY TO CLIENT: ')
            print(content.split('\n')[0])                # prints data of cached file
            print(content.split('\n')[12])
            print(content.split('\n')[16])
            content = content.encode('utf-8')       # encodes content
            clientSocket.send(content)              # sends cached data to client
            print('\nEND OF HEADER\n')
    except IOError:
        print('[LOOK UP THE CACHE]: NOT FOUND, BUILD REQUEST TO SEND TO ORIGINAL SERVER')
        print('[PARSE REQUEST HEADER] HOSTNAME IS ', host)
        if len(destination) > 1:                # if url exist then set
            print('[PARSE REQUEST HEADER] URL IS ', link[1:len(link)])
        if len(destination) > 2:                # if file exist then set
            print('[PARSE REQUEST HEADER] FILESNAME IS ', file)

        serverSocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)    # create serverSocket
        serverSocket.connect((host, 80))


        print('\nREQUEST MESSAGE SENT TO ORIGINAL SERVER:')
                                                # compose request sent to original server
        request = method + ' '+ link + ' ' + httpv + '\n' + 'Host: ' + host + '\r\n' + get
        request += 'Connection: close\r\nUpgrade-Insecure-Requests: 1\r\n\r\n'

        r = request.encode('utf-8')             # encodes request
        print(request)                          # prints request sent
        serverSocket.sendall(r)                 # sends request to original server
        print('END OF MESSAGE SENT TO ORIGINAL SERVER\n')

        rawResp = serverSocket.recv(1024)       # saves original server's response
        serverSocket.close()                    # closes serverSocket

        if (len(rawResp) > 0):                  # if server response is not empty then send data to client
            clientSocket.send(rawResp)



        resp = str(rawResp, 'utf-8')            # convert original server response to string
        respHeader = resp.split('\n\r\n')[0]    # parse response header from original response
        print('RESPONSE HEADER FROM ORIGINAL SERVER:')
        print(respHeader)                       # print response header
        print('\nEND OF HEADER\n')
        cacheFile = 'cache/' + file
        header_first_line = respHeader.split('\n')[0]   # parses first line from response header
        resp_code = header_first_line.split(' ')[1]     # parses code from first line of response header

        if resp_code == '200':                          # if response code is 200 OK then enter path
            print('[WRITE FILE INTO CACHE]: ', cacheFile, '\n')
            cache_file = open("./" + file, "w")         # open cache file
            cache_file.write(resp)                      # save file into cache
            cache_file.close()                          # close cache file
            print('\nRESPONSE HEADER FROM PROXY TO CLIENT:')
            print(header_first_line)
            print(resp.split('\n')[8])
            print('\nEND OF HEADER\n')
        if resp_code == '404':                          # if response code is 404 NOT FOUND then enter path
            print('\nRESPONSE HEADER FROM PROXY TO CLIENT:')
            print(header_first_line)
            print(resp.split('\n')[1])
            print('\nEND OF HEADER\n')
        if resp_code == '302':                          # if response code is 302 OK then enter path
            print('\nRESPONSE HEADER FROM PROXY TO CLIENT:')
            print(respHeader)
            print('\nEND OF HEADER\n')
else:                                                   # if request method is not GET then enter path
    host = dest.split('/')[0]                           # get host from address
    link = dest[len(host):len(dest)]                    # get link from address
```



```
        if(len(destination) < 3):                           # if file does not exist then set to link else get file
            file = link
        else:
            file = link.split('/')[2]
                                                            # create server socket
        serverSocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        serverSocket.connect((host, 80))
                                                            # create request sent to original server
        print('\nREQUEST MESSAGE SENT TO ORIGINAL SERVER:')
        request = method + ' '+ link + ' ' + httpv + '\n' + 'Host: ' + host + '\r\n' + get
        request += 'Connection: close\r\nUpgrade-Insecure-Requests: 1\r\n\r\n'

        r = request.encode('utf-8')                         # encode request to original server
        print(request)                                      # print request to original server
        serverSocket.sendall(r)                             # send request to original server
        print('END OF MESSAGE SENT TO ORIGINAL SERVER\n')

        rawResp = serverSocket.recv(1024)                   # saves original server's response
        serverSocket.close()                                # close serverSocket

        if (len(rawResp) > 0):                              # if server response is not empty then send data to client
            clientSocket.send(rawResp)

        resp = str(rawResp, 'utf-8')                        # convert original server response to string
        respHeader = resp.split('\n\r\n')[0]                # parse response header from original response
        print('RESPONSE HEADER FROM ORIGINAL SERVER:')
        print(respHeader)                                   # print response header from original server
        print('\nEND OF HEADER\n')

        print('\nRESPONSE HEADER FROM PROXY TO CLIENT:')
        print(respHeader)                                   # print response header so client from proxy
        print('\nEND OF HEADER\n')

print('END OF PROXY SERVER')
exit()
```

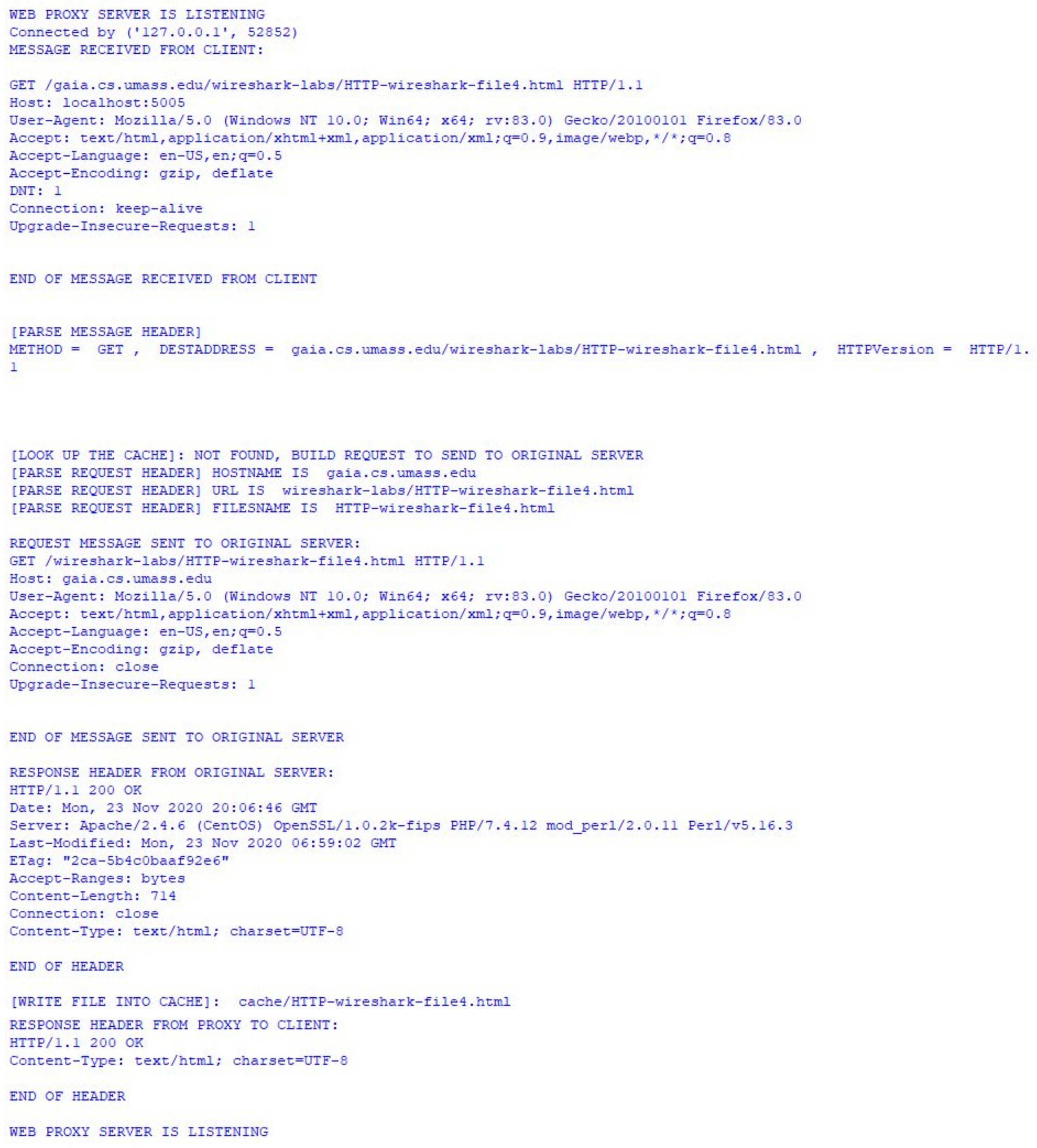
IV. Design Document:

1. Caching mechanism: We use the try block to open the file in the cache directory. If the opening is successful, that means that there is a cache hit and the program will retrieve the object to send back to the client. If the file cannot be open, then it will be handled by an exception, which connects to the web server to get the object needed. The exception will also store the object in cache.
2. Coding Language and IDE: We decided to use Python as the programming language because of its robustness and support for socket programming. The language also has a Google Drive IDE that allows multiple people to edit the program at the same time.
3. 404 Not Found handling: To handle 404 errors we first need to make sure the request sent by the client has the required fields such as host, link, and file within the destination address. Once this has been verified then the proxy can build the request sent from the proxy. After receiving the original server response we then parse the response header to

acquire the response code. When the code is a 404 Not Found error then we send the response to the client without adding anything to the cache.

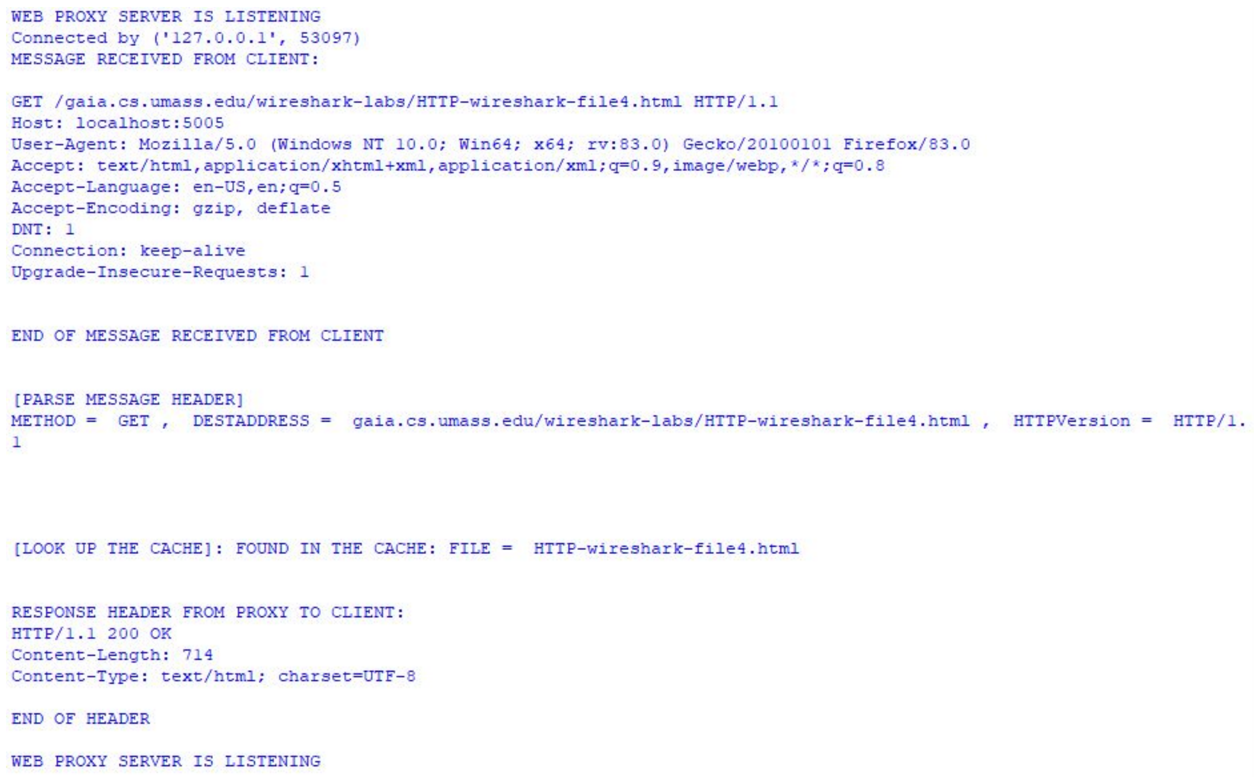
V. Screenshots:

Screenshot #1 (Object is not in cache)



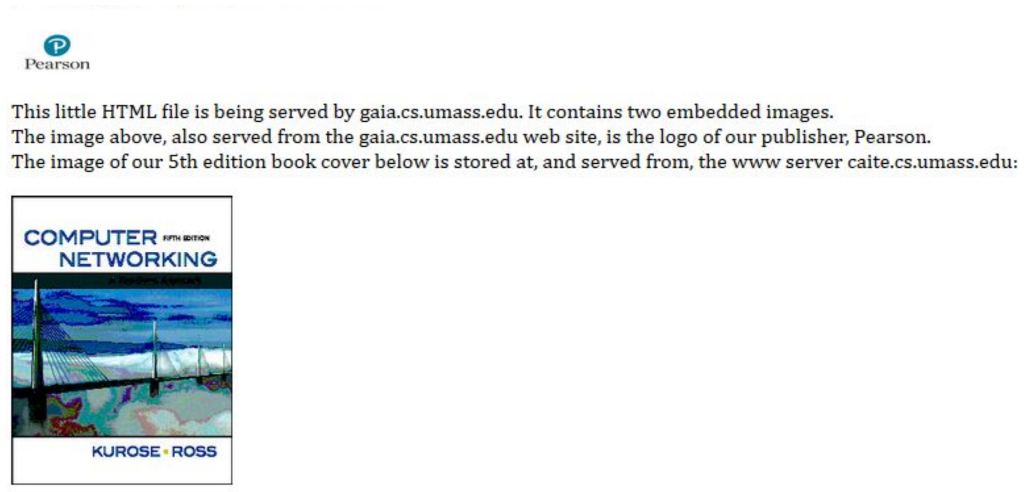
```
WEB PROXY SERVER IS LISTENING
Connected by ('127.0.0.1', 52852)
MESSAGE RECEIVED FROM CLIENT:

GET /gaia.cs.umass.edu/wireshark-labs/HTTP-wireshark-file4.html HTTP/1.1
Host: localhost:5005
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:83.0) Gecko/20100101 Firefox/83.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
DNT: 1
Connection: keep-alive
Upgrade-Insecure-Requests: 1


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]: 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] FILESNAME 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
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:83.0) Gecko/20100101 Firefox/83.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: close
Upgrade-Insecure-Requests: 1


END OF MESSAGE SENT TO ORIGINAL SERVER

RESPONSE HEADER FROM ORIGINAL SERVER:
HTTP/1.1 200 OK
Date: Mon, 23 Nov 2020 20:06:46 GMT
Server: Apache/2.4.6 (CentOS) OpenSSL/1.0.2k-fips PHP/7.4.12 mod_perl/2.0.11 Perl/v5.16.3
Last-Modified: Mon, 23 Nov 2020 06:59:02 GMT
ETag: "2ca-5b4c0baaf92e6"
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

WEB PROXY SERVER IS LISTENING
```

Screenshot #2 (Object is in cache)



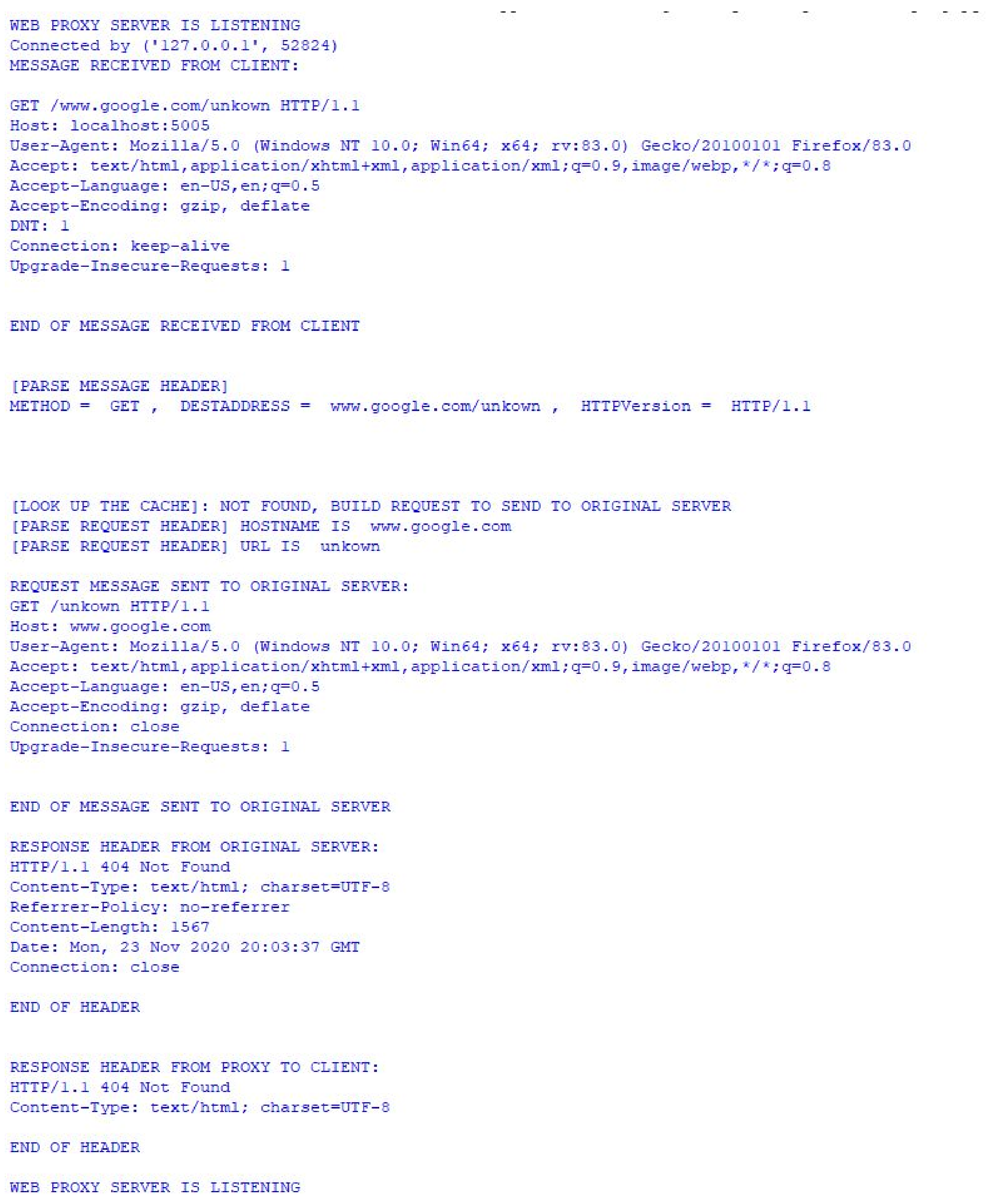
```
WEB PROXY SERVER IS LISTENING
Connected by ('127.0.0.1', 53097)
MESSAGE RECEIVED FROM CLIENT:

GET /gaia.cs.umass.edu/wireshark-labs/HTTP-wireshark-file4.html HTTP/1.1
Host: localhost:5005
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:83.0) Gecko/20100101 Firefox/83.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
DNT: 1
Connection: keep-alive
Upgrade-Insecure-Requests: 1


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 =  HTTP-wireshark-file4.html


RESPONSE HEADER FROM PROXY TO CLIENT:
HTTP/1.1 200 OK
Content-Length: 714
Content-Type: text/html; charset=UTF-8

END OF HEADER

WEB PROXY SERVER IS LISTENING
```

Screenshot #3 (Browser display for Screenshot #2 and #1)



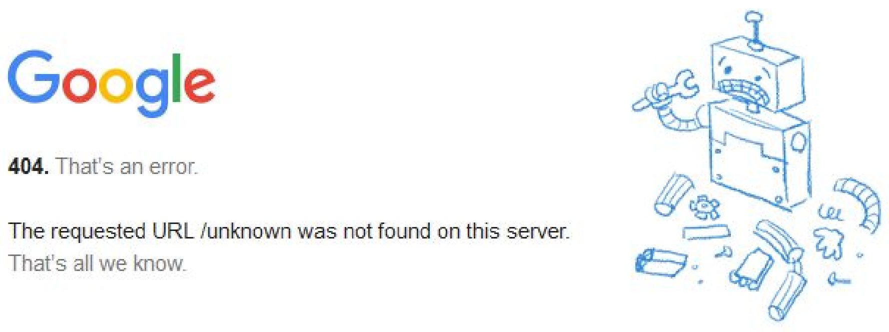
This little HTML file is being served by gaia.cs.umass.edu. It contains two embedded images.
The image above, also served from the gaia.cs.umass.edu web site, is the logo of our publisher, Pearson.
The image of our 5th edition book cover below is stored at, and served from, the www server caite.cs.umass.edu:

Screenshot #4 (404 Not Found)



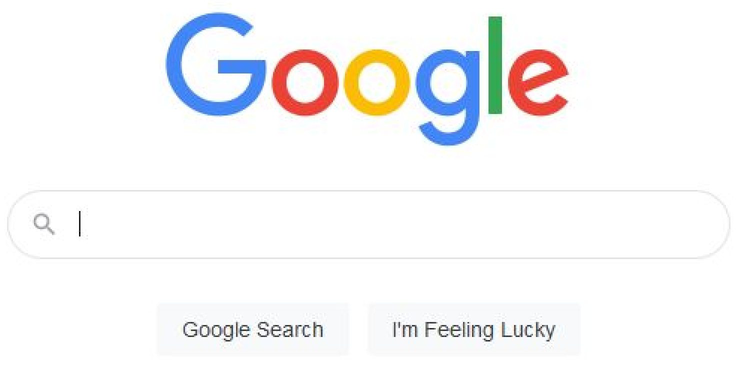
```
WEB PROXY SERVER IS LISTENING
Connected by ('127.0.0.1', 52824)
MESSAGE RECEIVED FROM CLIENT:

GET /www.google.com/unkown HTTP/1.1
Host: localhost:5005
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:83.0) Gecko/20100101 Firefox/83.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
DNT: 1
Connection: keep-alive
Upgrade-Insecure-Requests: 1


END OF MESSAGE RECEIVED FROM CLIENT


[PARSE MESSAGE HEADER]
METHOD =  GET ,  DESTADDRESS =  www.google.com/unkown ,  HTTPVersion =  HTTP/1.1




[LOOK UP THE CACHE]: NOT FOUND, BUILD REQUEST TO SEND TO ORIGINAL SERVER
[PARSE REQUEST HEADER] HOSTNAME IS  www.google.com
[PARSE REQUEST HEADER] URL IS  unkown

REQUEST MESSAGE SENT TO ORIGINAL SERVER:
GET /unkown HTTP/1.1
Host: www.google.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:83.0) Gecko/20100101 Firefox/83.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: close
Upgrade-Insecure-Requests: 1


END OF MESSAGE SENT TO ORIGINAL SERVER

RESPONSE HEADER FROM ORIGINAL SERVER:
HTTP/1.1 404 Not Found
Content-Type: text/html; charset=UTF-8
Referrer-Policy: no-referrer
Content-Length: 1567
Date: Mon, 23 Nov 2020 20:03:37 GMT
Connection: close

END OF HEADER


RESPONSE HEADER FROM PROXY TO CLIENT:
HTTP/1.1 404 Not Found
Content-Type: text/html; charset=UTF-8

END OF HEADER

WEB PROXY SERVER IS LISTENING
```

Screenshot #5 (Browser display for Screenshot #4)

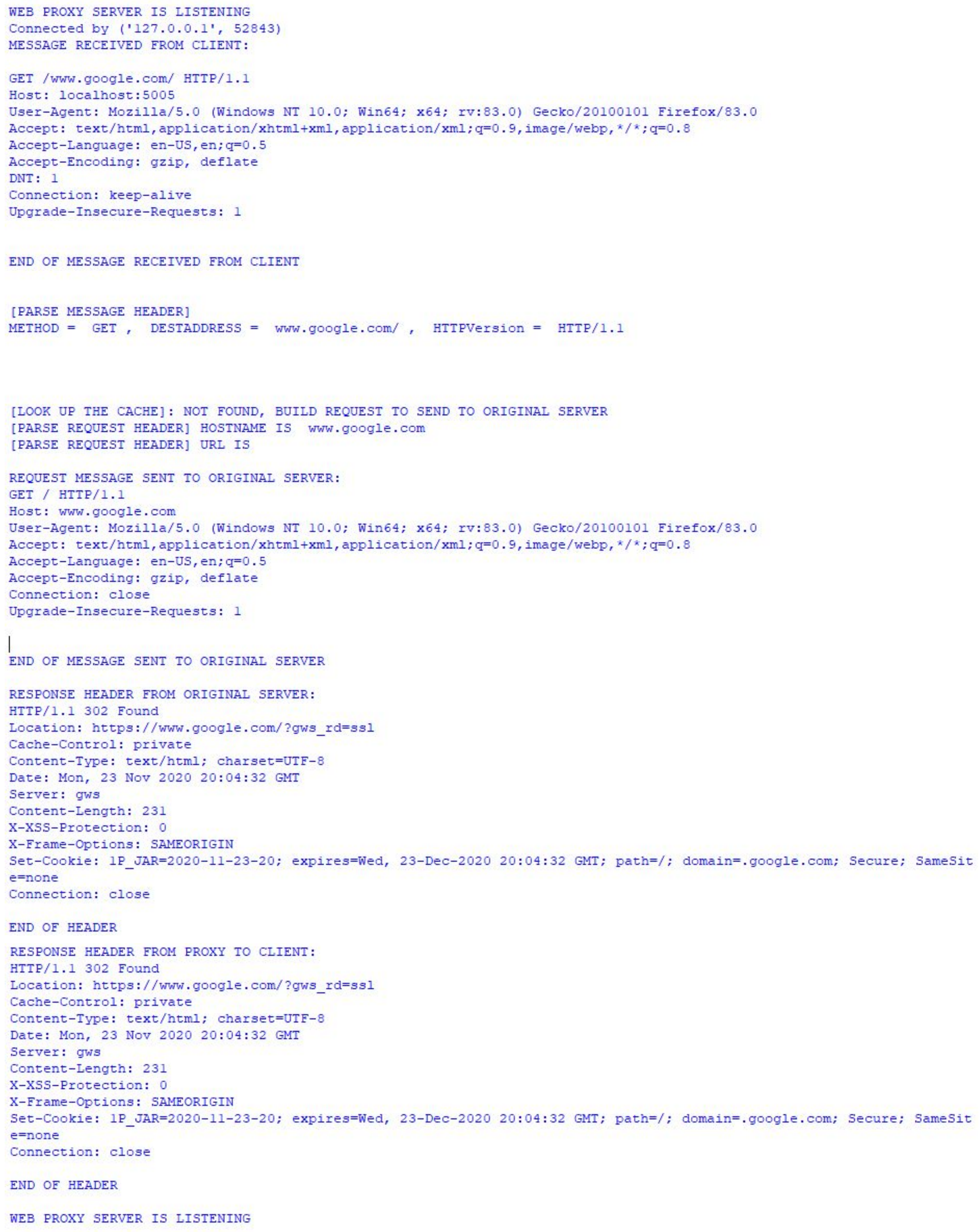

404. That's an error.

The requested URL /unknown was not found on this server. That's all we know.

Screenshot #6 (Browser display for Screenshot #7)



Screenshot #7 (302 found redirection)



```
WEB PROXY SERVER IS LISTENING
Connected by ('127.0.0.1', 52843)
MESSAGE RECEIVED FROM CLIENT:

GET /www.google.com/ HTTP/1.1
Host: localhost:5005
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:83.0) Gecko/20100101 Firefox/83.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
DNT: 1
Connection: keep-alive
Upgrade-Insecure-Requests: 1


END OF MESSAGE RECEIVED FROM CLIENT


[PARSE MESSAGE HEADER]
METHOD =  GET ,  DESTADDRESS =  www.google.com/ ,  HTTPVersion =  HTTP/1.1




[LOOK UP THE CACHE]: NOT FOUND, BUILD REQUEST TO SEND TO ORIGINAL SERVER
[PARSE REQUEST HEADER] HOSTNAME IS  www.google.com
[PARSE REQUEST HEADER] URL IS

REQUEST MESSAGE SENT TO ORIGINAL SERVER:
GET / HTTP/1.1
Host: www.google.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:83.0) Gecko/20100101 Firefox/83.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: close
Upgrade-Insecure-Requests: 1


END OF MESSAGE SENT TO ORIGINAL SERVER

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
Date: Mon, 23 Nov 2020 20:04:32 GMT
Server: gws
Content-Length: 231
X-XSS-Protection: 0
X-Frame-Options: SAMEORIGIN
Set-Cookie: 1P_JAR=2020-11-23-20; expires=Wed, 23-Dec-2020 20:04:32 GMT; path=/; domain=.google.com; Secure; SameSit
e=none
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
Date: Mon, 23 Nov 2020 20:04:32 GMT
Server: gws
Content-Length: 231
X-XSS-Protection: 0
X-Frame-Options: SAMEORIGIN
Set-Cookie: 1P_JAR=2020-11-23-20; expires=Wed, 23-Dec-2020 20:04:32 GMT; path=/; domain=.google.com; Secure; SameSit
e=none
Connection: close

END OF HEADER

WEB PROXY SERVER IS LISTENING
```

VI. Issues:

1. Parsing the request header: When parsing the request header we ran into the issue of some headers lacking the information we were trying to parse. The final solution we decided to go with was measuring the length of the first line in the header, which in the scope of this project allows us to know if the header contains the data we wanted to parse. An issue that we could see from this solution is that when the header does not exist then there could be potential errors that would halt the program.
2. Composing the request to send from proxy to server: When composing the request we ran into two different issues, first issue being that our request would not send successfully and would result in the program halting, the second issue being encoding the request. The solution used to handle the first issue was examining the end of the original request which showed us how we needed to construct the end of the request. The second issue was solved after we found several methods used to encode strings and decode bytes. The method agreed upon to handle encoding was object.encode() as it would return the encoded request.
3. Cache objects: At first, we recorded the filename strings of the request URLs in an array called “cache”. The array did not have anything else, and it did not function as a legitimate cache. We then learned how to open and write objects into the current working directory of the computer. To check for the cache directory, we try to open the file with the same path, and if we cannot open it, it is not in the cache. If we can open the file, we will retrieve the object and send it to the client.