Web-Proxy Server

By Tanmay Kaushik

Student Number - 18308341

INTRODUCTION 2
REQUIREMENTS 3
DESIGN OVERVIEW 4
IMPLEMENTATION and WORKING 5
CODE LISTING 11

INTRODUCTION

A Web-Proxy is a local server which acts as a gateway between a client application and a real server. For this project, we are asked to implement a Web-Proxy server which fetches items from the Web. This server will act as a gateway between the client which is the computer and the server which is the web.

This will allow us to add the functionality of Access Control and Caching of websites. This technology is used by companies, universities and schools all over the world.

REQUIREMENTS

Handle HTTP and HTTPS Requests

The project requires us to respond to HTTP & HTTPS requests. It also requires us to display each request on a management console. The Web-Proxy server should be able to forward the request to the Web server and relay the response to the browser.

Handle WebSocket Connections

The project requires us to use WebSocket protocols which attempts to open a connection between server and browser which will allow us to send and receive responses without polling the server

ACCESS CONTOL - Blocking and Unblocking

The project requires us to add functionality of access control. This includes blocking the user's access to a list of blocked URLs which is managed by the company or organisation in a real world.

CACHING

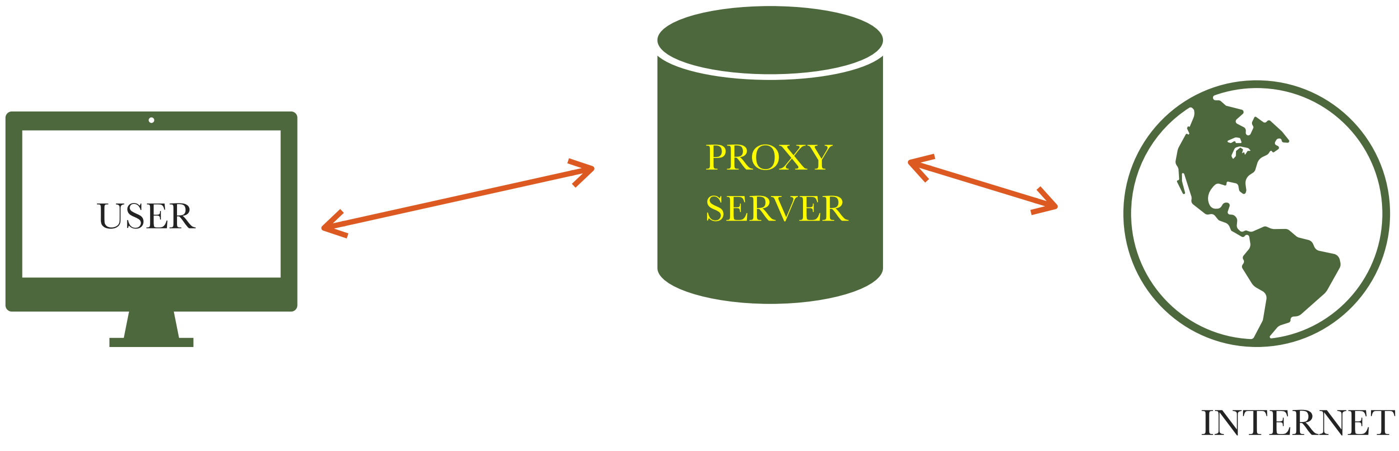
The project requires us to add the functionality of Caching requests locally so the bandwidth is saved. The time taken and bandwidth of the data must be provided which shows that caching is more efficient.

CREATING A MULTI-THREADED SERVER

The Web-Proxy server must be a multi-threaded server which has the ability to handle multiple requests simultaneously.

DESIGN OVERVIEW

The purpose of the web-proxy server is to intercept the request from the computer and retrieve the response from the server on behalf of the computer.



The Web-Proxy listens on a port and waits for a request from the computer. When a request is received, Proxy creates a new thread and the connection is handled. This makes our Proxy server a multi-threaded server as different requests are handled simultaneously because each thread handles a request separately. This allows our Proxy server to handle multiple connections at once which is a very important requirement.

There are two types of requests that our server can receive - HTTP and HTTPS Requests.

HTTP is an acronym for HyperText Transfer Protocol and HTTPS is a Secured HTTP request.

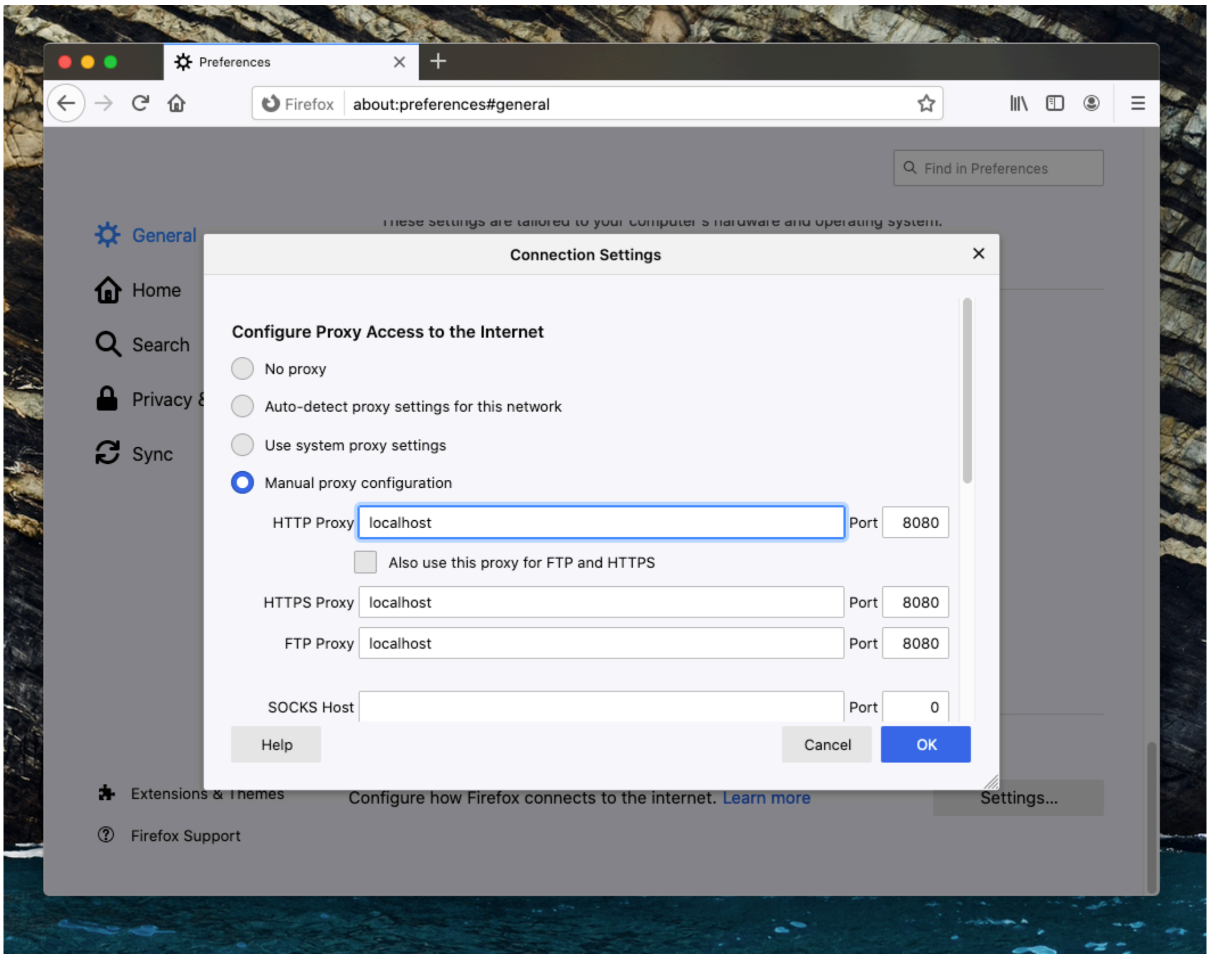
Both of the requests are handled quite differently so we implemented the server which handles both types of requests.

To enable the functionality of caching we maintain a Cache list which stores requests in that list. A HTTP request is cached by storing a copy of the request locally the first time that request is retrieved. This is done so that when that request is needed again, the local copy of the request is used which saves bandwidth and takes far less time.

Blocking and Unblocking of requests are implemented in such a way that when a HTTP and HTTPS request is decoded we lookup if that URL is a blocked URL or not. If the requested URL is blocked, the connection is closed.

IMPLEMENTATION and WORKING

Before starting the program, one has to set up the browser settings to configure manual proxy. I used Mozilla Firefox for my project. To set up the settings, one has to go to network settings and enter the Port number and Name in the fields shown below.



Configure Proxy Access to the Internet

No proxy

Auto-detect proxy settings for this network

Use system proxy settings

[✓ Manual proxy configuration]

HTTP Proxy localhost Port 8080

Also use this proxy for FTP and HTTPS

HTTPS Proxy localhost Port 8080

FTP Proxy localhost Port 8080

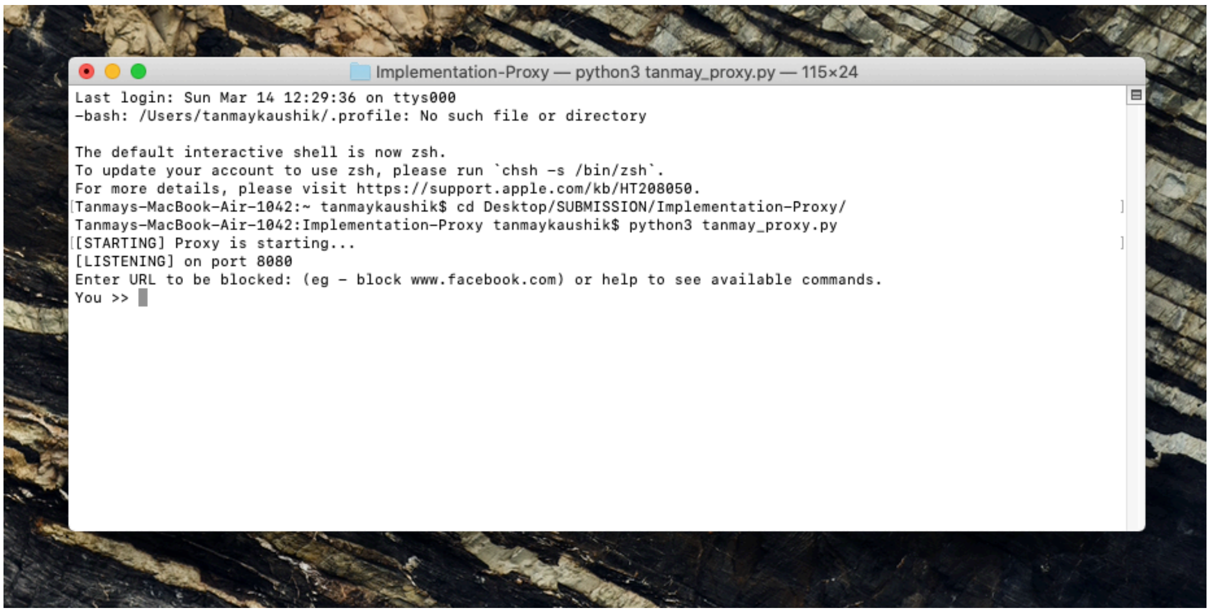
SOCKS Host Port 0

Configure how Firefox connects to the internet. Learn more

To develop this project, I used Python programming language and used libraries - Socket, Threading, Request and few other like datetime and Cmd.

The proxy is started by using the command `python3 tanmay_proxy.py`

This starts the server and the command line unit looks like this:

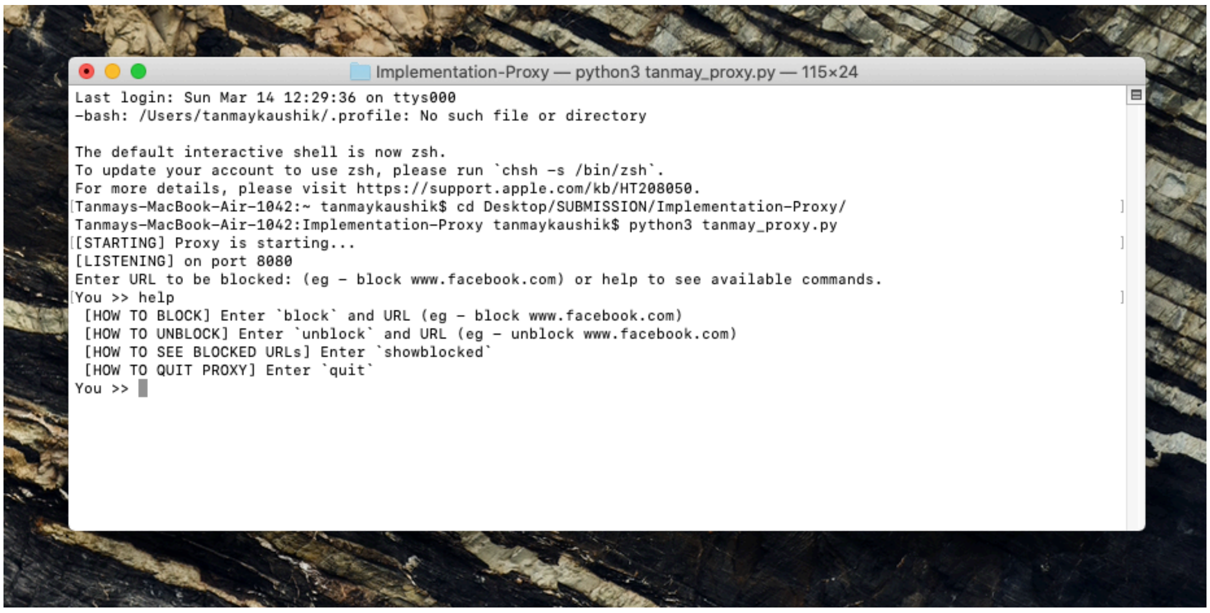


Implementation-Proxy — python3 tanmay_proxy.py — 115×24

```
Last login: Sun Mar 14 12:29:36 on ttys000
-bash: /Users/tanmaykaushik/.profile: No such file or directory

The default interactive shell is now zsh.
To update your account to use zsh, please run `chsh -s /bin/zsh`.
For more details, please visit https://support.apple.com/kb/HT208050.
Tanmays-MacBook-Air-1042:~ tanmaykaushik$ cd Desktop/SUBMISSION/Implementation-Proxy/
Tanmays-MacBook-Air-1042:Implementation-Proxy tanmaykaushik$ python3 tanmay_proxy.py
[STARTING] Proxy is starting...
[LISTENING] on port 8080
Enter URL to be blocked: (eg - block www.facebook.com) or help to see available commands.
You >> 
```

If the user wants to know what features he/she can use, one can type `help` in the control unit which provides the user with the instructions :

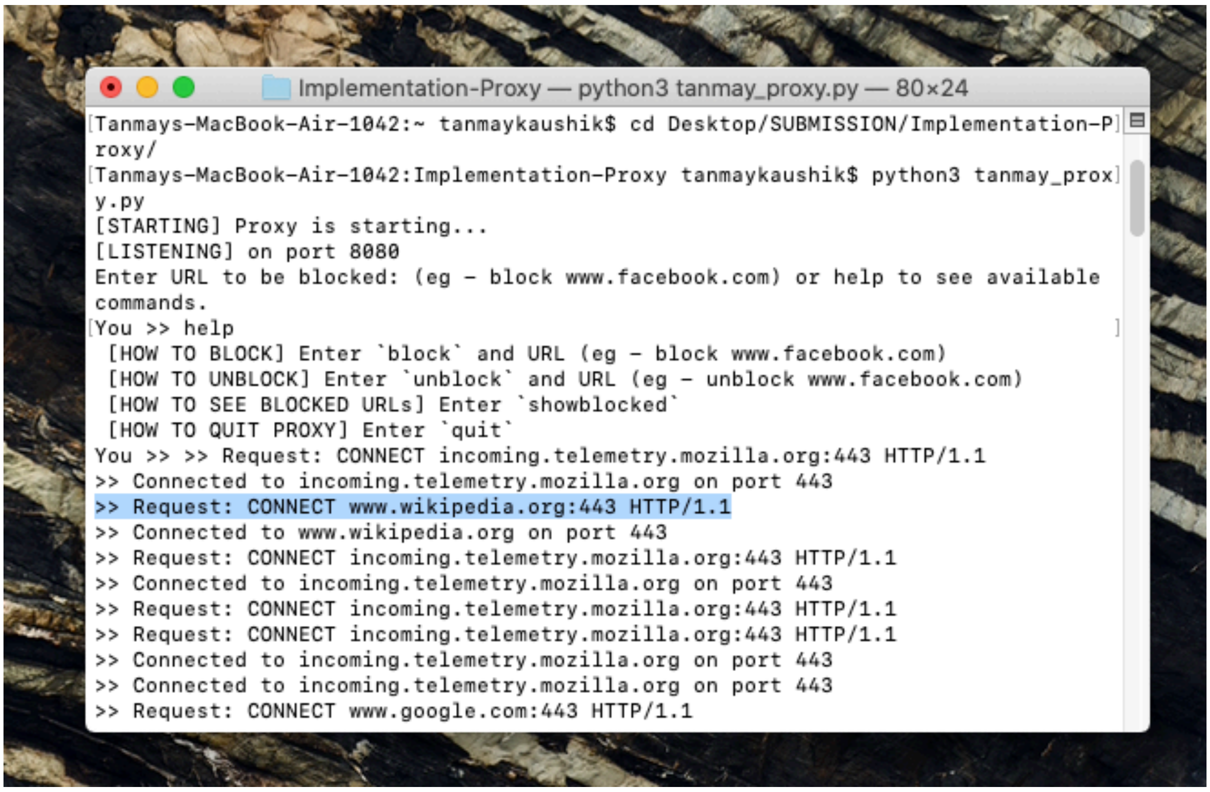


Implementation-Proxy — python3 tanmay_proxy.py — 115×24

```
Last login: Sun Mar 14 12:29:36 on ttys000
-bash: /Users/tanmaykaushik/.profile: No such file or directory

The default interactive shell is now zsh.
To update your account to use zsh, please run `chsh -s /bin/zsh`.
For more details, please visit https://support.apple.com/kb/HT208050.
Tanmays-MacBook-Air-1042:~ tanmaykaushik$ cd Desktop/SUBMISSION/Implementation-Proxy/
Tanmays-MacBook-Air-1042:Implementation-Proxy tanmaykaushik$ python3 tanmay_proxy.py
[STARTING] Proxy is starting...
[LISTENING] on port 8080
Enter URL to be blocked: (eg - block www.facebook.com) or help to see available commands.
You >> help
 [HOW TO BLOCK] Enter `block` and URL (eg - block www.facebook.com)
 [HOW TO UNBLOCK] Enter `unblock` and URL (eg - unblock www.facebook.com)
 [HOW TO SEE BLOCKED URLs] Enter `showblocked`
 [HOW TO QUIT PROXY] Enter `quit`
You >> 
```

When the proxy is started, the program is listening on a specified port - 8080 in my case. Later, when the user opens the browser, the requests are shown on the management unit. In the example below, I opened the browser and searched the Wikipedia page (www.wikipedia.org)



```
Tanmays-MacBook-Air-1042:~ tanmaykaushik$ cd Desktop/SUBMISSION/Implementation-P
roxy/
Tanmays-MacBook-Air-1042:Implementation-Proxy tanmaykaushik$ python3 tanmay_prox
y.py
[STARTING] Proxy is starting...
[LISTENING] on port 8080
Enter URL to be blocked: (eg - block www.facebook.com) or help to see available
commands.
You >> help
 [HOW TO BLOCK] Enter `block` and URL (eg - block www.facebook.com)
 [HOW TO UNBLOCK] Enter `unblock` and URL (eg - unblock www.facebook.com)
 [HOW TO SEE BLOCKED URLs] Enter `showblocked`
 [HOW TO QUIT PROXY] Enter `quit`
You >> >> Request: CONNECT incoming.telemetry.mozilla.org:443 HTTP/1.1
>> Connected to incoming.telemetry.mozilla.org on port 443
>> Request: CONNECT www.wikipedia.org:443 HTTP/1.1
>> Connected to www.wikipedia.org on port 443
>> Request: CONNECT incoming.telemetry.mozilla.org:443 HTTP/1.1
>> Connected to incoming.telemetry.mozilla.org on port 443
>> Request: CONNECT incoming.telemetry.mozilla.org:443 HTTP/1.1
>> Request: CONNECT incoming.telemetry.mozilla.org:443 HTTP/1.1
>> Connected to incoming.telemetry.mozilla.org on port 443
>> Connected to incoming.telemetry.mozilla.org on port 443
>> Request: CONNECT www.google.com:443 HTTP/1.1
```

When a request is received from the Web-Proxy server, a new thread is created and started. The request line is decoded/parsed and the URL is decoded. Along with obtaining the URL, the type of request is parsed which tells if the request is HTTP or HTTPS.

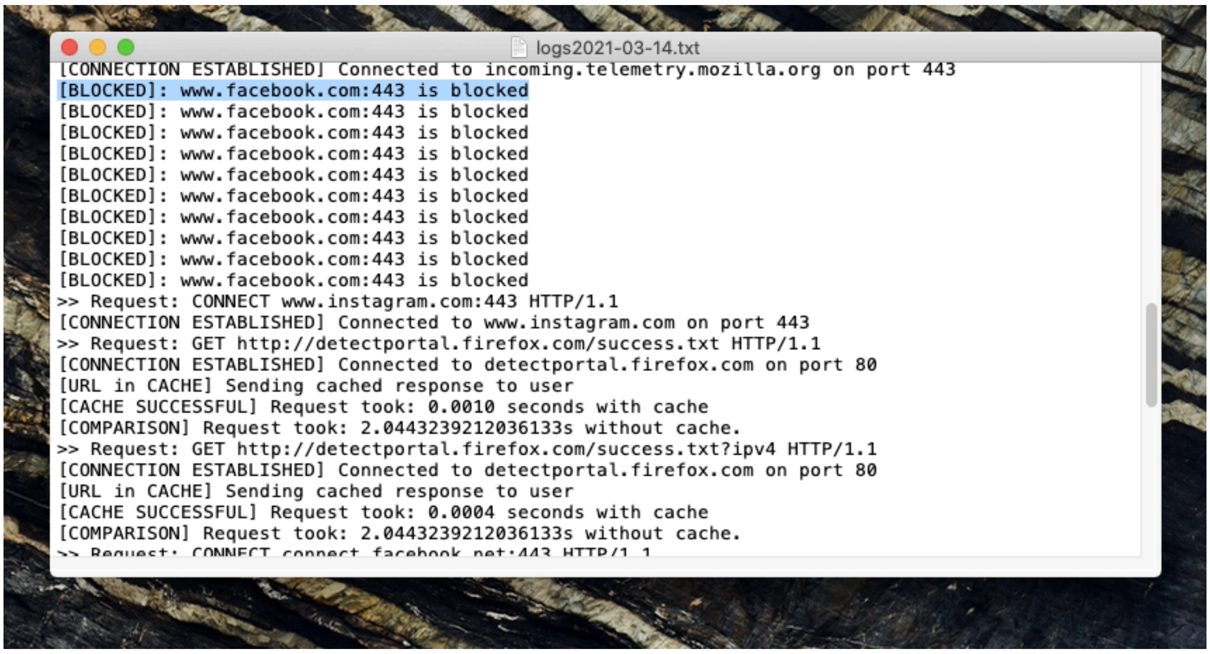
Depending on the type of method, the request is handled differently which is explained in the section below

BLOCKING

Before handling the request, it is checked if the URL is blocked or not. If the URL is blocked, then the connection is stopped and closed. To check if a URL is blocked, a different method is called which takes the URL as an argument.

In the method, we go through the set of blocked URLs. If URL is found, True is returned, else False.

This process is clearly shown below. I have blocked `www.facebook.com` and then I tried to access it:



```
[CONNECTION ESTABLISHED] Connected to incoming.telemetry.mozilla.org on port 443
[BLOCKED]: www.facebook.com:443 is blocked
[BLOCKED]: www.facebook.com:443 is blocked
[BLOCKED]: www.facebook.com:443 is blocked
[BLOCKED]: www.facebook.com:443 is blocked
[BLOCKED]: www.facebook.com:443 is blocked
[BLOCKED]: www.facebook.com:443 is blocked
[BLOCKED]: www.facebook.com:443 is blocked
[BLOCKED]: www.facebook.com:443 is blocked
[BLOCKED]: www.facebook.com:443 is blocked
[BLOCKED]: www.facebook.com:443 is blocked
>> Request: CONNECT www.instagram.com:443 HTTP/1.1
[CONNECTION ESTABLISHED] Connected to www.instagram.com on port 443
>> Request: GET http://detectportal.firefox.com/success.txt HTTP/1.1
[CONNECTION ESTABLISHED] Connected to detectportal.firefox.com on port 80
[URL in CACHE] Sending cached response to user
[CACHE SUCCESSFUL] Request took: 0.0010 seconds with cache
[COMPARISON] Request took: 2.0443239212036133s without cache.
>> Request: GET http://detectportal.firefox.com/success.txt?ipv4 HTTP/1.1
[CONNECTION ESTABLISHED] Connected to detectportal.firefox.com on port 80
[URL in CACHE] Sending cached response to user
[CACHE SUCCESSFUL] Request took: 0.0004 seconds with cache
[COMPARISON] Request took: 2.0443239212036133s without cache.
>> Request: CONNECT connect.facebook.net:443 HTTP/1.1
```

CACHING

After checking if the URL is blocked, we determine if the request has been previously cached or not. At this point we start a clock. If the request is found in the cache, we retrieve the response directly from the cache instead of retrieving if from the server.

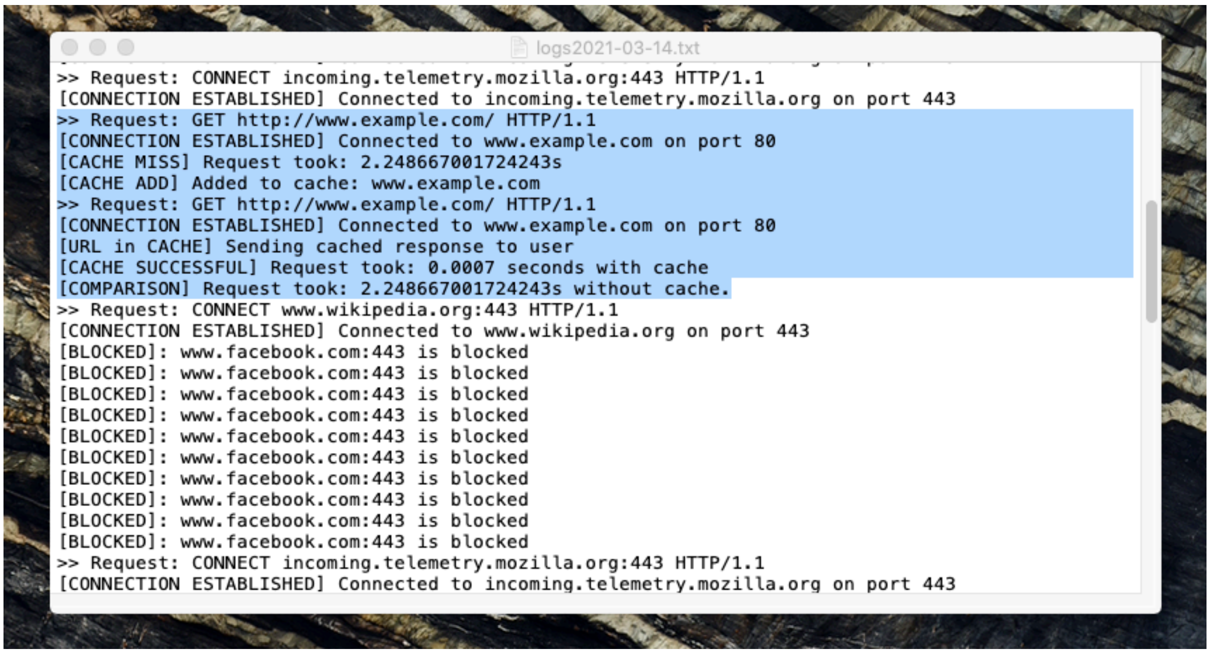
When the response is taken, we stop the clock and record the time taken by the cache. Time taken by cache is printed and Time taken by server is also printed for comparison. This can be seen below in which cached copy of `www.example.com` was retrieved.

The first time `www.example.com` was requested, we encountered a [CACHE MISS] because this was the first time we requested this url. So the response was retrieved from the server which took 2.2486 seconds. The response was stored locally in the cache which can be seen [CACHE ADD].

Next time when the page was requested, we found that the copy of that response was stored in cache - [URL in CACHE].

The response is taken from the cache which takes 0.0007 seconds. The time taken to retrieve the request from server is also shown for reference which was 2.2486 seconds.

This proves that using cache is much more efficient.



```
>> Request: CONNECT incoming.telemetry.mozilla.org:443 HTTP/1.1
[CONNECTION ESTABLISHED] Connected to incoming.telemetry.mozilla.org on port 443
>> Request: GET http://www.example.com/ HTTP/1.1
[CONNECTION ESTABLISHED] Connected to www.example.com on port 80
[CACHE MISS] Request took: 2.248667001724243s
[CACHE ADD] Added to cache: www.example.com
>> Request: GET http://www.example.com/ HTTP/1.1
[CONNECTION ESTABLISHED] Connected to www.example.com on port 80
[URL in CACHE] Sending cached response to user
[CACHE SUCCESSFUL] Request took: 0.0007 seconds with cache
[COMPARISON] Request took: 2.248667001724243s without cache.
>> Request: CONNECT www.wikipedia.org:443 HTTP/1.1
[CONNECTION ESTABLISHED] Connected to www.wikipedia.org on port 443
[BLOCKED]: www.facebook.com:443 is blocked
[BLOCKED]: www.facebook.com:443 is blocked
[BLOCKED]: www.facebook.com:443 is blocked
[BLOCKED]: www.facebook.com:443 is blocked
[BLOCKED]: www.facebook.com:443 is blocked
[BLOCKED]: www.facebook.com:443 is blocked
[BLOCKED]: www.facebook.com:443 is blocked
[BLOCKED]: www.facebook.com:443 is blocked
[BLOCKED]: www.facebook.com:443 is blocked
[BLOCKED]: www.facebook.com:443 is blocked
>> Request: CONNECT incoming.telemetry.mozilla.org:443 HTTP/1.1
[CONNECTION ESTABLISHED] Connected to incoming.telemetry.mozilla.org on port 443
```

HTTP Request vs HTTPS Request

When the request line is decoded, the URL and Method is retrieved. If the method is GET, that means the request is an HTTP request and if the method is CONNECT, that means the request is an HTTPS request.

For HTTP request, a connection is created. The data received from the server is checked and if it is not empty, the data is sent to the browser. When zero length chunk is received by the server, connection is closed. Along with this, HTTP requests are cached locally which is described below.

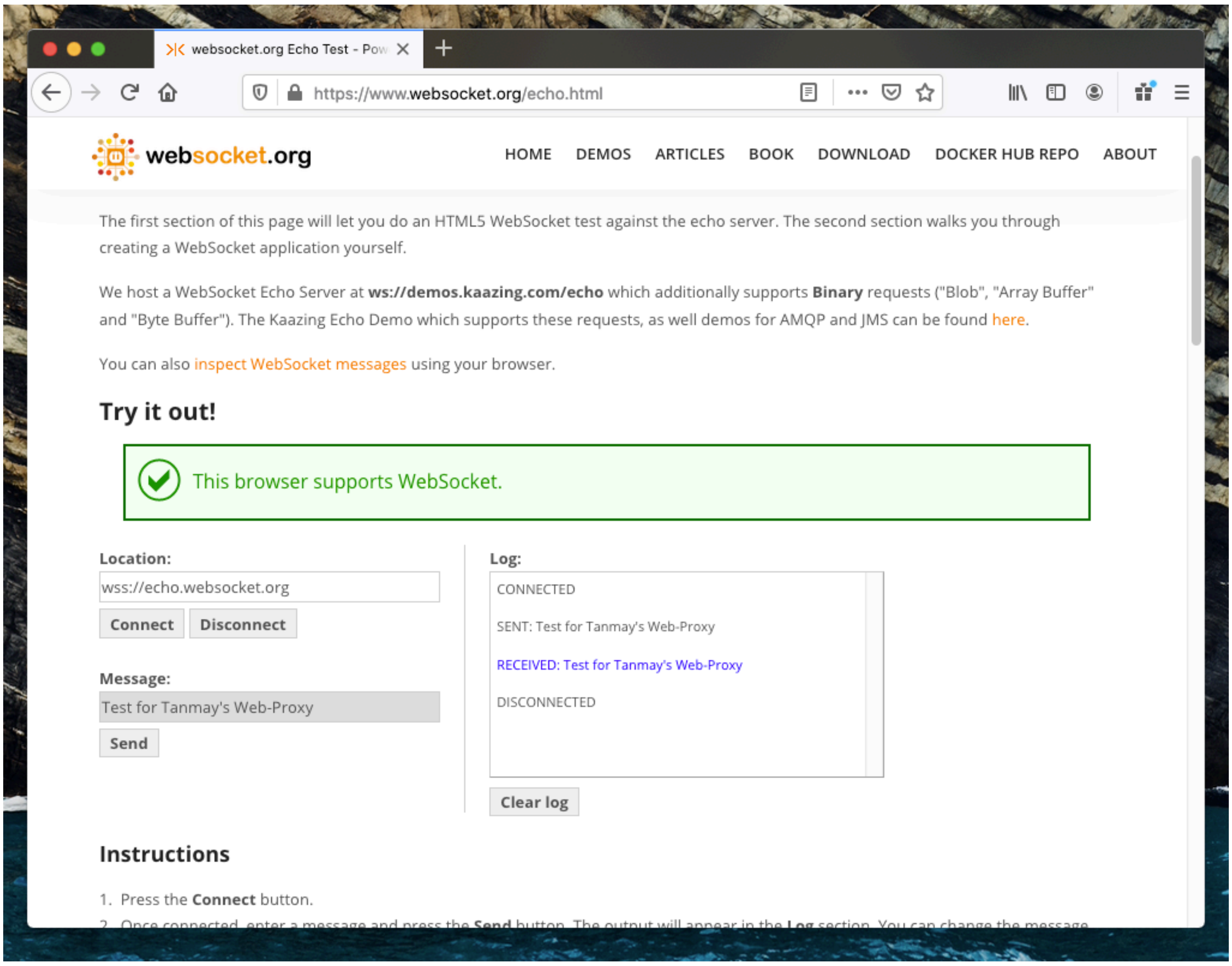
In case of HTTPS request, we have to establish a 3-way connection between the server and the client. The reason behind the 3-way handshake is that HTTPS is a secure connection.

After the connection is established, we do the same procedure to HTTP requests which was described above.

WebSockets

After implementing all the requirements, I tested for WebSocket connections.

I tested WebSockets functionality using the website https://www.websocket.org/echo.html This website lets one do an HTML5 WebSocket test against the echo server :


websocket.org Echo Test - Pow
https://www.websocket.org/echo.html
websocket.org
HOME
DEMOS
ARTICLES
BOOK
DOWNLOAD
DOCKER HUB REPO
ABOUT

The first section of this page will let you do an HTML5 WebSocket test against the echo server. The second section walks you through creating a WebSocket application yourself.

We host a WebSocket Echo Server at **ws://demos.kaazing.com/echo** which additionally supports **Binary** requests ("Blob", "Array Buffer" and "Byte Buffer"). The Kaazing Echo Demo which supports these requests, as well demos for AMQP and JMS can be found here.

You can also inspect WebSocket messages using your browser.

Try it out!

This browser supports WebSocket.

Location: wss://echo.websocket.org

Connect | Disconnect

Message: Test for Tanmay's Web-Proxy

Send

Log:

```
CONNECTED

SENT: Test for Tanmay's Web-Proxy

RECEIVED: Test for Tanmay's Web-Proxy

DISCONNECTED
```

Clear log

Instructions

1. Press the **Connect** button.
2. Once connected, enter a message and press the **Send** button. The output will appear in the **Log** section. You can change the message

CODE LISTING

For easy instructions to run and test code, kindly access my GitHub page :

https://github.com/taaanmay/Web-Proxy-Server.git

(Link will be available after the submission date)

```
import socket
import _thread
import threading
import sys
#import requests
import datetime
import time
import os
from cmd import Cmd
from urllib.request import Request, urlopen, HTTPError
import select


#Constants
PORT = 8080
SOCKET_IP = socket.gethostbyname(socket.gethostname())
#Get IP Address
ADDR = (SOCKET_IP, PORT)
HTTPS_BUFFER = 8192
HTTP_BUFFER = 4096

MAX_CONNECTIONS = 60
active_connections = 0

cache = {}
blocked = set([])
response_times = {}

#cache_list = {}     # List of Cached Files
#blocked_list = []   # List of Blocked URLs
prev_blocked_list = [] #Previously blokcked URLS
```

```
class input_cmd(Cmd):
    prompt = "You >> "

    def do_help(self, args):
        print(" [HOW TO BLOCK] Enter `block` and URL (eg -
block www.facebook.com)")
        print(" [HOW TO UNBLOCK] Enter `unblock` and URL (eg -
unblock www.facebook.com)")
        print(" [HOW TO SEE BLOCKED URLs] Enter `showblocked`
")
        print(" [HOW TO QUIT PROXY] Enter `quit`")

    def do_block(self, args):
        try:
            arg_URL = args.rsplit(" ", 1)
            arg_URL = arg_URL[0]
        except Exception as error:
            print("Please enter the URL")
        #Adding 'www.' to the URL if not present
        if not "www." in arg_URL:
            arg_URL = "www." + arg_URL
        #Add URL to the blocking list
        blocked.add(arg_URL)
        #Maintaining a Print List of blocked URLs
        if arg_URL not in blocked and len(prev_blocked_list) <
10 :
            prev_blocked_list.append(arg_URL)

        #Printing the blocked URL
        print("[BLOCKED] : ", arg_URL)


    def do_showblocked(self, args):
        #If there are no URLs blocked, provide suggestions
based on the previously blocked URLs that were unblocked
        if blocked == []:
            print("There are no blocked URLs. Some suggestions
based on previous activities: ", prev_blocked_list)
        else:
            print(f"LIST OF BLOCKED URLs : {blocked}")

    def do_unblockall(self, args):
        blocked.clear()
        print("All blocked URLs are unblocked")

    def do_unblock(self, args):
        arg_URL = args.rsplit(" ", 1)
        arg_URL = arg_URL[0]

        #Adding 'www.' to the URL if not present
        if not "www." in arg_URL:
            arg_URL = "www." + arg_URL

        if arg_URL in blocked:
            blocked.remove(arg_URL)
            print("[UNBLOCKED] : ", arg_URL)
        else:
            print("This URL was never blocked")

    def do_quit(self, args):
        print("[QUITTING]: Bye!")
        raise KeyboardInterrupt()
```

```
def user_help_method(console, irr):
    console.cmdloop("Enter URL to be blocked: (eg - block
www.facebook.com) or help to see available commands.")
```

```
def start():
    #Initialise CMD Prompt
    console = input_cmd()
    #Multi-Threaded System
    t = threading.Thread(target= user_help_method , args=
(console, None))
#_thread.start_new_thread(user_help_method, (console, None))
    t.start()

    try:
        sock = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
        sock.bind(ADDR)
#Socket is bound to a port
        sock.listen(MAX_CONNECTIONS)
        print(f"[LISTENING] on port {PORT} ")
        log(f"[LISTENING] on port {PORT} ")
    except Exception as error:
        print(f"Error in start method {error}")
        sys.exit(2)


    #_thread.start_new_thread(listen_method, (sock,PORT))
    # while(1):
    #     one = 1

    log("[STARTING] Proxy is starting...")


    global active_connections
    while(active_connections <= MAX_CONNECTIONS):
        try:
            # accept connection from browser
            conn, client_address = sock.accept()

            active_connections = active_connections + 1

            #Data is Received from the browser
            ##data = conn.recv(BUFFER)

            # create thread for the connection
            ##t = threading.Thread(target= requestMethod , 
args= (conn, data, PORT))
#_thread.start_new_thread(requestMethod, (conn, data, PORT))
            ##t.start()
```

```
            #Create New Thread and call proxy_connection 
method
            thread = threading.Thread(name = client_address, 
target= proxy_connection , args= (conn, client_address))
#_thread.start_new_thread(requestMethod, (conn, data, PORT))
            thread.setDaemon(True)
            thread.start()
        #except Exception as error:
        except KeyboardInterrupt:
            sock.close()
            ##print(f"[ERROR while listening] : {error}")
            sys.exit(1)
```

```
# receive data and parse it, check http vs https
def proxy_connection(conn, client_address):
    global active_connections
```

```
    # receive data from browser
    data = conn.recv(HTTP_BUFFER)
    # print(data)
    if len(data) > 0:
        try:
            # get first line of request
            request_line = data.decode().split('\n')[0]
            try:
                method = request_line.split(' ')[0]
                url = request_line.split(' ')[1]
                if method == 'CONNECT':
                    type = 'https'
                else:
                    type = 'http'
```

```
                #Check if URL is blocked
                if check_blocked(url):
                    active_connections -= 1
                    #Connection is closed
                    conn.close()
                    return

                #URL Not Blocked
                else:
                    # need to parse url for webserver and port
                    print(">> Request: " + request_line)
                    log(">> Request: " + request_line)
                    webserver = ""
                    port = -1
                    tmp = parseURL(url, type)
                    if len(tmp) > 0:
                        webserver, port = tmp

                    else:
                        return

                    print("[CONNECTION ESTABLISHED] Connected 
to " + webserver + " on port " + str(port))
                    log("[CONNECTION ESTABLISHED] Connected to 
" + webserver + " on port " + str(port))

                    # check cache for response
                    start = time.time()
                    x = cache.get(webserver)
                    if x is not None:
                        # Response in cache.
                        print("[URL in CACHE] Sending cached 
response to user")
                        log("[URL in CACHE] Sending cached 
response to user")
                        #Send the response without setting up 
socket
                        conn.sendall(x)
                        finish = time.time()
                        time_taken = finish-start
```

```
            print(f"[CACHE SUCCESSFUL] Request 
took: {finish - start:0.4f} seconds with cache")
            log(f"[CACHE SUCCESSFUL] Request took: 
{finish - start:0.4f} seconds with cache")
            #print(f">> Request took: " + {finish-
start:0.4f} + "s with cache.")
            print("[COMPARISON] Request took: " + 
str(response_times[webserver]) + "s without cache.")
            log("[COMPARISON] Request took: " + 
str(response_times[webserver]) + "s without cache.")

          #Response not in socket.
          #Setup a socket
          #Check if HTTP or HTTPS and handle request 
accordingly
          else:
            # connect to web server socket
            sock = socket.socket(socket.AF_INET, 
socket.SOCK_STREAM)
            # sock.connect((webserver, port))

            # handle http requests
            if type == 'http':
              if handle_HTTP_request(sock, 
webserver, port, data, conn) is True:
                active_connections -= 1

            # handle https requests
            elif type == 'https':
              handle_HTTPS_request(sock, 
webserver, conn,data, port)
      except IndexError:
        pass
    except UnicodeDecodeError:
      pass
  else:
    pass
```

```python

def parseURL(url, type):
    http_position = url.find("://")

    # Type of request
    if (http_position == -1):
        temp = url
    else:
        temp = url[(http_position+3):]

    # Obtaining Port position and Base from the Request
    port_position = temp.find(":")
    webserver_position = temp.find("/")

    if webserver_position == -1:
        webserver_position = len(temp)

    webserver = ""
    port = -1

    #DEFAULT PORT
    if port_position == -1 or webserver_position <
port_position:
        if type == "https":
            port = 443
        else:
            port = 80

        webserver = temp[:webserver_position]
    # defined port
    else:
        port = int((temp[(port_position+1):])
[:webserver_position-port_position-1])
        webserver = temp[:port_position]

    return [webserver, int(port)]
```

```
def check_blocked(url):
    for temp_url in blocked:
        if temp_url in url:
            print(f"[BLOCKED]: {url} is blocked")
            log(f"[BLOCKED]: {url} is blocked")
            return True
    return False
```

```
# Logs all the information that would be too much to send to
the commandline.
def log(input):
    
    currTime = datetime.datetime.now().strftime("%Y-%m-%d")
    newFile = "/Users/tanmaykaushik/Desktop/SUBMISSION/
Implementation-Proxy/logs" + str(currTime) + ".txt"
    newFile = newFile.replace(' ', '_')
    newFile = newFile.replace(':', '_')
    newFile = "" + newFile
    file = open(newFile, "a")
    file.write("\n" + input)
    file.flush()
```

```
def handle_HTTP_request(sock, webserver, port, data, conn):
    # print("im a http request")
                            # string builder to build response
for cache.
                            start = time.time()
                            string_builder = bytearray("",
'utf-8')
                            sock.connect((webserver, port))

                            # send client request to server
                            sock.send(data)
```

```
              sock.settimeout(2)

              try:
                while True:
                  # try to receive data from
the server
                  webserver_data =
sock.recv(HTTP_BUFFER)
                  # if data is not empty,
send it to the browser
                  if len(webserver_data) >
0:

conn.send(webserver_data)

string_builder.extend(webserver_data)
                  # communication is stopped
when a zero length of chunk is received
                  else:
                    break
              except socket.error:
                pass

              # communication is over so can now
store the response_time and response which was built
              finish = time.time()
              print("[CACHE MISS] Request took:
" + str(finish-start) + "s")
              log("[CACHE MISS] Request took: "
+ str(finish-start) + "s")
              response_times[webserver] = finish
- start
              cache[webserver] = string_builder
              print("[CACHE ADD] Added to cache:
" + webserver)
              log("[CACHE ADD] Added to cache: "
+ webserver)

              sock.close()
```

```
                            conn.close()
                            return True

#Method to Handle HTTPS request
def handle_HTTPS_request(sock, webserver, conn,data, port):
    sock.connect((webserver, port))
                            # print("im a https request")
    #Send message - Connection Established
    conn.send(bytes("HTTP/1.1 200 Connection 
Established\r\n\r\n", "utf8"))


    connections = [conn, sock]
    keep_connection = True

    while keep_connection:
        ready_sockets, sockets_for_writing, error_sockets = 
select.select(connections, [], connections, 100)

        if error_sockets:
            break

        for ready_sock in ready_sockets:
            # look for ready sock
            other = connections[1] if ready_sock is 
connections[0] else connections[0]

            try:
                data = ready_sock.recv(HTTPS_BUFFER)
            except socket.error:
                print(">> Connection timeout...")
                ready_sock.close()

            if data:
                other.sendall(data)
                keep_connection = True
            else:
                keep_connection = False
```

```
print("[STARTING] Proxy is starting...")
start()
```