Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

CSU34031 Advanced Telecommunications Web Proxy Server

Kamil Przepiórowski
17327895

February 25, 2020

Contents

1 Requirements **1**

2 Browser Setup **2**

3 Overview of design **3**

4 GUI **3**

5 Blocking **4**
5.1 Blocking 4
5.2 Unblocking 4

6 HTTP vs HTTPS **5**

7 Websockets **5**

8 Caching **6**
8.1 Cache Miss 6
8.2 Cache Hit 6

9 Code **7**

1 Requirements

• Respond to HTTP HTTPS requests and should display each request on a management console. It should forward the request to the Web server and relay the response to the browser

• Handle Websocket connections

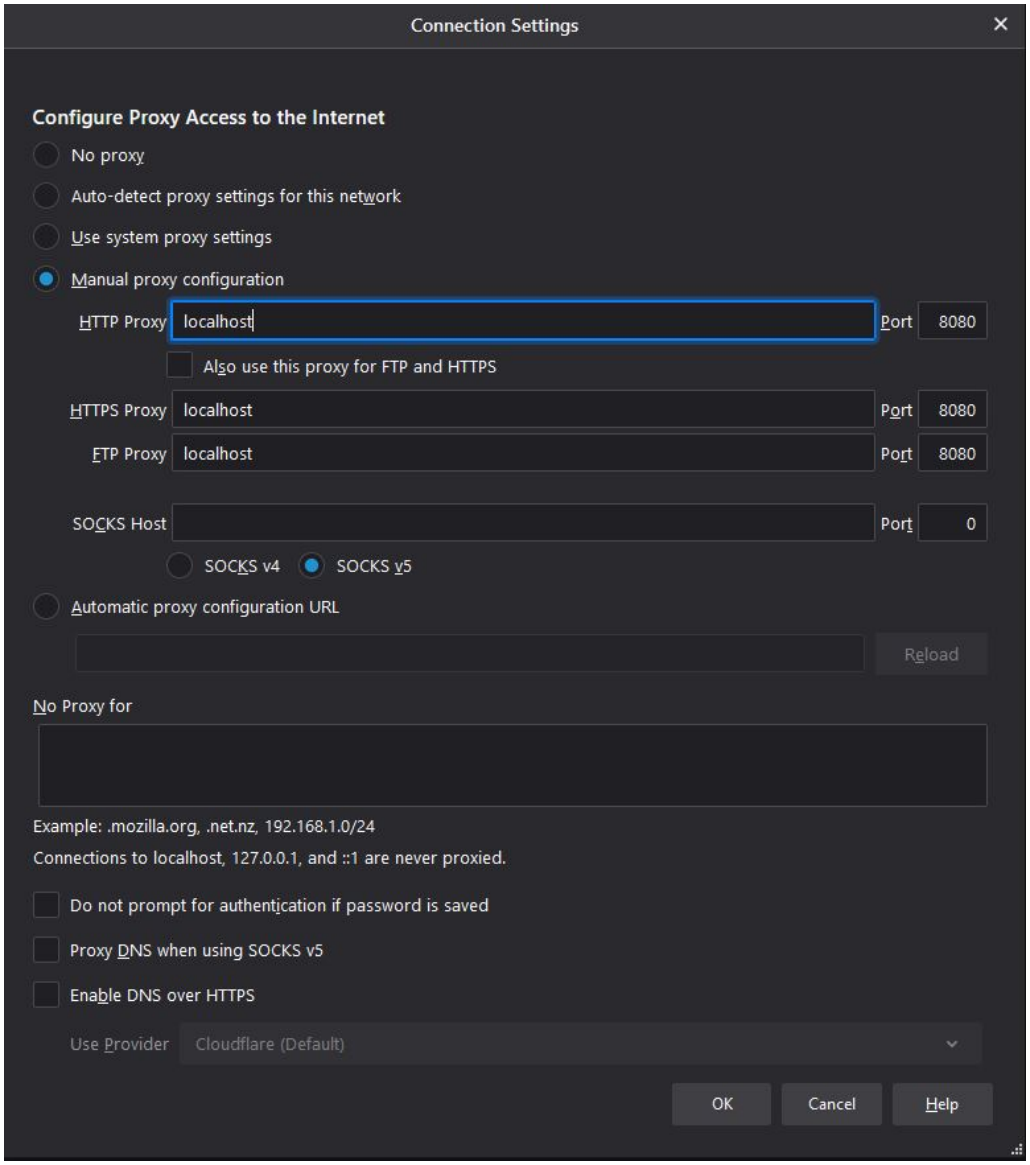
• Dynamically block selected URLs via the management console

• Efficiently cache requests locally and thus save bandwidth. You must gather timing and bandwidth data to prove the efficiency of your proxy

• Handle multiple requests simultaneously by implementing a threaded server

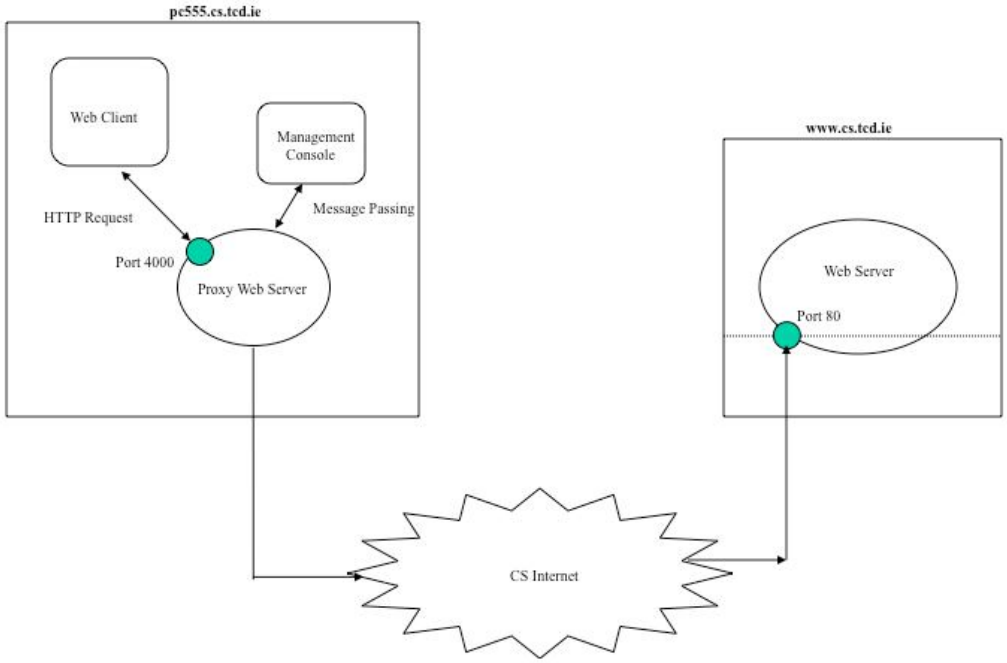
2 Browser Setup

I used Mozilla Firefox to test my proxy server. I had to configure the following Network Settings to enable the proxy. The port here has to be the same as the one the proxy is listening to.



3 Overview of design

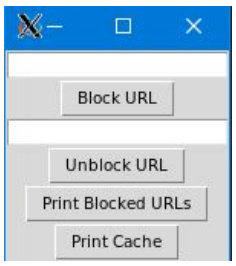
The purpose of this project was to design a web proxy that will be able to handle http/https requests as well as websockets, and allow for dynamic blocking of URLs. The was required to implement a cache as well as multiple threads to handle many connections at once. The overview of the design looks as follows:



The high level implementation is as follows: The proxy listens to a certain port for any requests from the client, if a request is received, a new thread is created and the connection is then handled on that thread. This allows for multiple connections at once. First the request line is parsed to gather the method and URL. Once these are known, we can then determine whether the request is HTTP or HTTPS and handle them accordingly, if the URL is not a blocked one. A cache is implemented for HTTP.

4 GUI

I decided to use tkinter to develop a simple user interface rather than having everything done through command line. The GUI looks as follows:

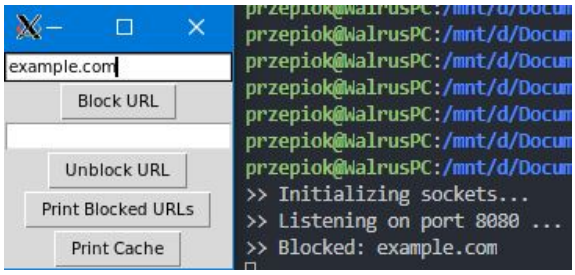


You can enter a URL and block/unblock it, as well as print the blocked URLs / current state of the cache in the console.

5 Blocking

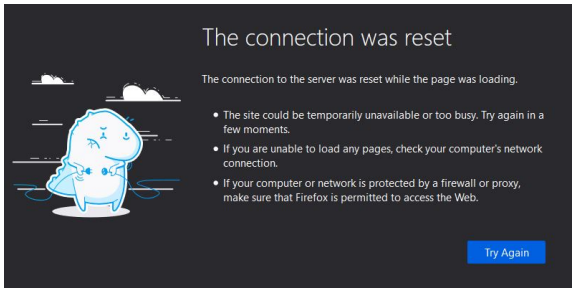
5.1 Blocking

As mentioned, you can enter a URL and block it. Once the "Block URL" button is pressed, a message is printed in console that the URL has in fact been blocked. Now if you try to access a blocked page, you get the following result and message in console.



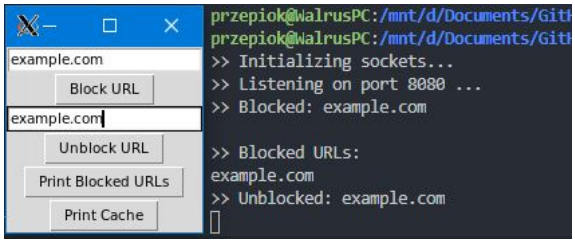
When you try to access a blocked URL, you get the following message in console, and the site isn't loaded.

```
>> New connection. Number of active connections: 4
http://example.com/ is blocked.
>> New connection. Number of active connections: 4
http://example.com/ is blocked.
>> New connection. Number of active connections: 4
http://example.com/ is blocked.
>> New connection. Number of active connections: 4
http://example.com/ is blocked.
http://example.com/ is blocked.
>> New connection. Number of active connections: 4
>> New connection. Number of active connections: 4
http://example.com/ is blocked.
```

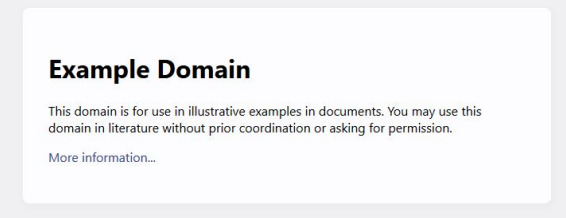


5.2 Unblocking

You are also able to unblock websites if they are blocked. A message is printed in the console that the URL is now unblocked.



When you try to access the URL again, the site now loads.



Blocking a blocked site / unblocking an unblocked site results in nothing other than a message in the console.

6 HTTP vs HTTPS

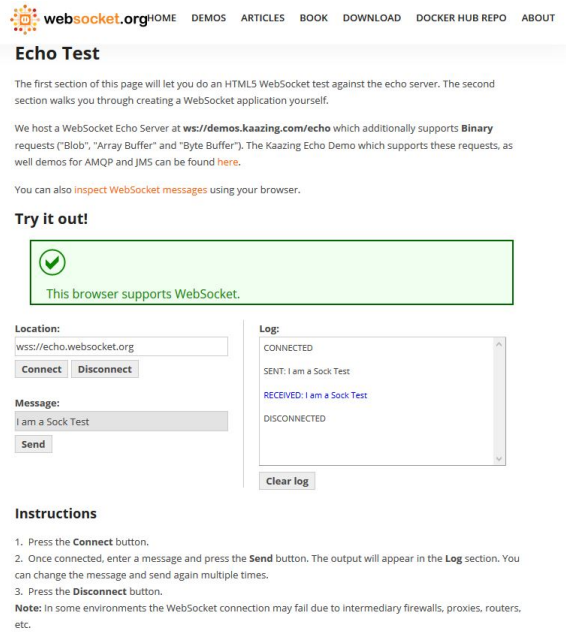
Once a request comes in, the request line is parsed for the method and URL. HTTP uses the GET method whereas HTTPS uses CONNECT. Once we distinguish the method, we can handle the request accordingly.

For HTTP we create a connection and if data (from server) is not empty, send it to the browser. Connection is stopped when a zero length chunk is received. HTTP responses are cached to improve efficiency of future requests.

For HTTPS we need to establish a connection (3-way handshake) as it is a secure protocol and then similarly to HTTP we forward data from the server to the client (browser). We do not cache anything for HTTPS.

7 Websockets

To test websockets I went to https://www.websocket.org/echo.html



8 Caching

8.1 Cache Miss

Before we handle requests, we check if the URL is already in cache. The first time a URL is visited, it will result in a cache miss. The time taken to complete the request is logged in console, and the server response (with URL as key) is cached for future use.

```
>> Current Cache:
>> Request: GET http://example.com/ HTTP/1.1
>> New connection. Number of active connections: 1
>> Connected to example.com on port 80
>> Request took: 2.211134672164917s
>> Added to cache: example.com
```

8.2 Cache Hit

If a URL is already in the cache then the server response is simply taken from cache rather than requesting from the server again. The time taken to complete the request is printed to show the speed up of the cache.

```
>> Request: GET http://example.com/ HTTP/1.1
>> New connection. Number of active connections: 1
>> Connected to example.com on port 80
>> Request took: 2.2142601013183594s
>> Added to cache: example.com
>> New connection. Number of active connections: 1
>> Request: GET http://example.com/ HTTP/1.1
>> Connected to example.com on port 80
>> Sending cached response to user
>> Request took: 0.010995626449584961s with cache.
>> Request took: 2.2142601013183594s without cache.
```

9 Code

The code here is very badly formatted, please see my GitHub repo for a good version : https://github.com/kamilprz/WebProxy

```
import os, sys, threading, socket, time, select
import tkinter as tk
from tkinter import*

# dict for blocked URLs
blocked = set([])
# dict for cache
cache = {}
# dict for time of response before caching.
response_times = {}
HTTP_BUFFER = 4096
HTTPS_BUFFER = 8192
MAX_ACTIVE_CONNECTIONS = 60
PORT = 8080
active_connections = 0

#tkinter - GUI used to dynamically block and unlock URLs
def tkinter():
        console = tk.Tk()

        def block_url():
                url = block.get()
                if url not in blocked:
                        blocked.add(url)
                        print(">> Blocked: " + url)
                else:
                        print(">> Already blocked")

        block = Entry(console)
        block.pack()
        block_button = Button(console, text = "Block URL", command = block_url
        block_button.pack()

        def unblock_url():
                url = unblock.get()
                if url not in blocked:
                        print(">> " + url + " is not blocked")
                else:
                        blocked.discard(url)
                        print(">> Unblocked: " + url)

        unblock = Entry(console)
        unblock.pack()
        unblock_button = Button(console, text = "Unblock URL", command = unblo
```

```
        unblock_button.pack()

        # prints all blocked urls
        def print_blocked():
                print("\n>> Blocked URLs: ")
                for x in blocked:
                        print(x)

        print_blocked = Button(console, text = "Print Blocked URLs", command =
        print_blocked.pack()

        # prints all cached urls
        def print_cache():
                print("\n>> Current Cache: ")
                for x in cache.keys():
                        print(x)

        print_cache = Button(console, text = "Print Cache", command = print_ca
        print_cache.pack()

        mainloop()


# MAIN PROGRAM
def main():
        # boot up the tkinter gui
        thread = threading.Thread(target = tkinter)
        thread.setDaemon(True)
        thread.start()

        try:
                # Ininitiate socket
                sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
                # bind socket to port
                sock.bind(('', PORT))
                sock.listen(MAX_ACTIVE_CONNECTIONS)
                print(">> Initializing sockets...")
                print(">> Listening on port {0} ...".format(PORT))
                # print(">> Blocked Sites:")
        except Exception:
                print(">> Error")
                sys.exit(2)

        global active_connections
        while active_connections <= MAX_ACTIVE_CONNECTIONS:
                try:
                        # accept connection from browser
                        conn, client_address = sock.accept()
                        active_connections += 1
```

```
                        # create thread for the connection
                        thread = threading.Thread(name = client_address, targe
                        thread.setDaemon(True)
                        thread.start()
                        print(">> New connection. Number of active connections
                except KeyboardInterrupt:
                        sock.close()
                        sys.exit(1)
        sock.close()


# 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'

                                if isBlocked(url):
                                        active_connections -= 1
                                        conn.close()
                                        return

                                else:
                                        # need to parse url for webserver and
                                        print(">> Request: " + request_line)
                                        webserver = ""
                                        port = -1
                                        tmp = parseURL(url, type)
                                        if len(tmp) > 0:
                                                webserver, port = tmp
                                                # print(webserver)
                                                # print(port)
                                        else:
                                                return

                                        print(">> Connected to " + webserver +
```

```
                # check cache for response
                start = time.time()
                x = cache.get(webserver)
                if x is not None:
                        # if in cache - don't bother s
                        print(">> Sending cached respo
                        conn.sendall(x)
                        finish = time.time()
                        print(">> Request took: " + st
                        print(">> Request took: " + st

                else:
                        # connect to web server socket
                        sock = socket.socket(socket.AF
                        # sock.connect((webserver, por

                        # handle http requests
                        if type == 'http':
                                # print("im a http req
                                # string builder to bu
                                start = time.time()
                                string_builder = bytea
                                sock.connect((webserve

                                # send client request
                                sock.send(data)
                                sock.settimeout(2)

                                try:
                                        while True:
                                                # try
                                                webser
                                                # if d
                                                if len



                                                # comm
                                                else:

                                except socket.error:
                                        pass

                                # communication is ove
                                finish = time.time()
                                print(">> Request took
                                response_times[webserv
                                cache[webserver] = str
                                print(">> Added to cac
```

```
                                                        active_connections -=
                                                        sock.close()
                                                        conn.close()

                                                # handle https requests
                                                elif type == 'https':
                                                        sock.connect((webserve
                                                        # print("im a https re
                                                        conn.send(bytes("HTTP/

                                                        connections = [conn, s
                                                        keep_connection = True

                                                        while keep_connection:
                                                                ready_sockets,

                                                                if error_socke
                                                                        break

                                                                for ready_sock
                                                                        # look
                                                                        other

                                                                try:
                                                                        data =
                                                                except socket.
                                                                        print(
                                                                        ready_

                                                                if data:
                                                                        other.
                                                                        keep_c
                                                                else:
                                                                        keep_c
                                except IndexError:
                                        pass
                        except UnicodeDecodeError:
                                pass
                else:
                        pass

                # active_connections -= 1
                # print(">> Closing client connection...")
                # conn.close()
                # return

def isBlocked(url):
        for x in blocked:
                if x in url:
```

```
                        print(url + " is blocked.")
                        return True
        return False


def parseURL(url, type):
        # isolate url from ://
        http_pos = url.find("://")
        if (http_pos == -1):
                temp = url
        else:
                temp = url[(http_pos+3):]

        # find pos of port if there is one
        port_pos = temp.find(":")

        # find end of webserver
        webserver_pos = temp.find("/")
        if webserver_pos == -1:
                webserver_pos = len(temp)

        webserver = ""
        port = -1
        # default port
        if (port_pos == -1 or webserver_pos < port_pos):
                if type == "https":
                        # https
                        port = 443
                else:
                        # http
                        port = 80
                webserver = temp[:webserver_pos]
        # defined port
        else:
                port = int((temp[(port_pos+1):])[:webserver_pos-port_pos-1])
                webserver = temp[:port_pos]

        return [webserver, int(port)]

if __name__ == '__main__':
        main()
```