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

CSU33032 Advanced Computer Networks
Project #1: Web Proxy Server

Abdelaziz Abushark, Std# 20332134

March 4th, 2023

Contents

1. Introduction

we have been assigned a project with an interesting objective - to implement a Web Proxy Server. The purpose of this server is to act as a middleman between a Web client and the Internet, fetching items from the Web on behalf of the client instead of the client fetching them

directly. By doing so, it enables caching of pages and access control. This project provides an opportunity to explore the functionalities and intricacies involved in building a Web proxy server and to gain hands-on experience in implementing it.

2. Requirements

The project requires us to design and implement a Web Proxy Server that can meet several key requirements. Firstly, the server must be able to respond to both HTTP and HTTPS requests and display each request on a management console. It should then forward the request to the appropriate Web server and relay the response back to the client's browser. In addition, the server must be capable of handling WebSocket connections, allowing for the exchange of data between the client and server. The server should also provide the ability to dynamically block selected URLs via the management console, giving administrators greater control over the content that can be accessed through the server. To optimize network bandwidth, the server must efficiently cache HTTP requests locally, and we will need to gather timing and bandwidth data to prove the effectiveness of the caching system. Finally, the server should be able to handle multiple requests simultaneously by implementing a threaded server, allowing for concurrent processing of requests, and reducing wait times for clients. By meeting these requirements, we can create a robust and efficient Web Proxy Server that can handle a variety of client requests with ease.

3. Summary of the system architecture

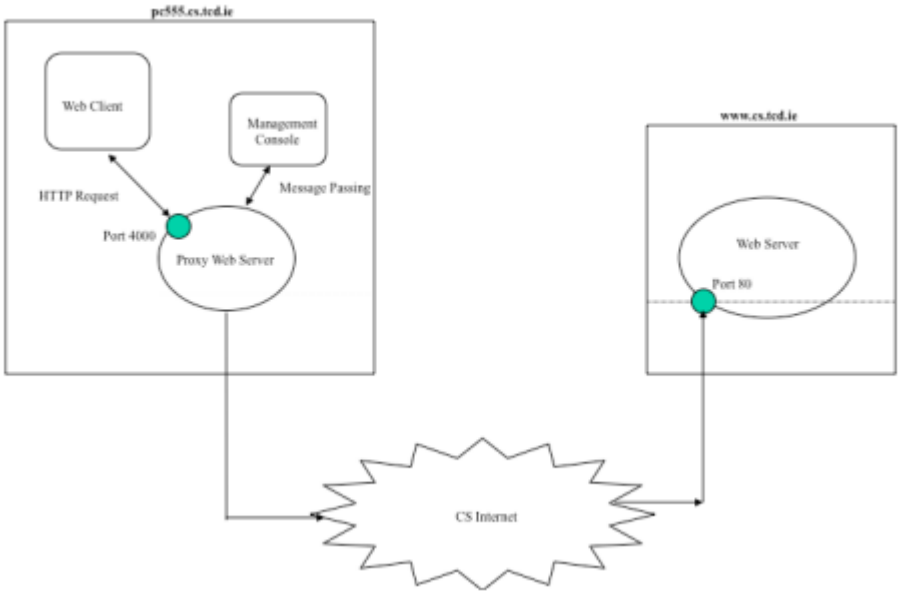

Figure [1]

The objective of this venture entailed the meticulous construction of a sophisticated web proxy system, imbued with the capability to seamlessly handle a broad spectrum of complex and demanding HTTP/HTTPS requests, and further extend its functionality to encompass the handling of Web-Sockets, as well as a dynamic URL blocking capability. The design of the system required the implementation of an efficient caching mechanism, to augment the overall system performance by judiciously reducing the load on network bandwidth. Additionally, multiple threads had to be integrated to ensure the system was capable of simultaneously managing and

handling an array of connections with maximum efficacy. The high-level depiction of the overall design framework serves as an overview of the underlying intricacies and complexities inherent in this multifaceted project.

The general approach of the system is as follows: The web proxy constantly "listens" on a specific network port for any requests from the client. Upon receiving a request, a new thread is created to manage the connection, which facilitates multiple client connections concurrently. The system then extracts information from the request, specifically the request method and URL, which is subsequently utilized to determine whether the request is either an HTTP or HTTPS protocol. If the URL is not on a predefined "blocked" list, the request is then processed according to its specific protocol requirements. Additionally, a caching system has been implemented to store recently requested HTTP content to improve overall system performance.

4. Browser Configuration

To conduct testing of my proxy server, I employed the Mozilla Firefox web browser, which required a series of intricate configuration adjustments to activate the proxy's functionality. Specifically, the Network Settings had to be manually configured to ensure proper transmission of proxy server requests. Of note, it was necessary to ensure that the port settings matched those established by the proxy server for its listening interface to effectively receive incoming requests. Figure 2 visualizes the process.

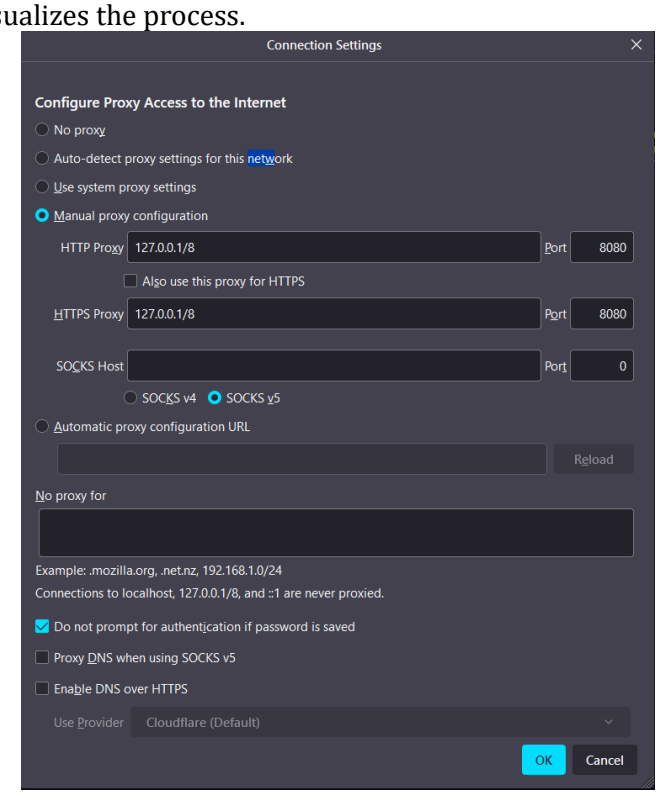

Figure [2]

5. Implementation

5.1 GUI

After careful consideration and deliberation, I resolved to employ the sophisticated tkinter (tk) module to construct a multifaceted graphical user interface (GUI), in lieu of the traditional command-line interface. The resultant GUI presents an interactive and intuitive visual representation of the system's underlying functionality, thereby facilitating seamless and effective user-interaction. The interface comprises a range of distinct features, including the ability to input and subsequently manage URLs, via a carefully crafted series of blocking/unblocking functions. Additionally, the GUI displays critical system information in the console window, including current cache status and a comprehensive list of blocked URLs, which provides users with a real-time, detailed overview of the system's current operational state.

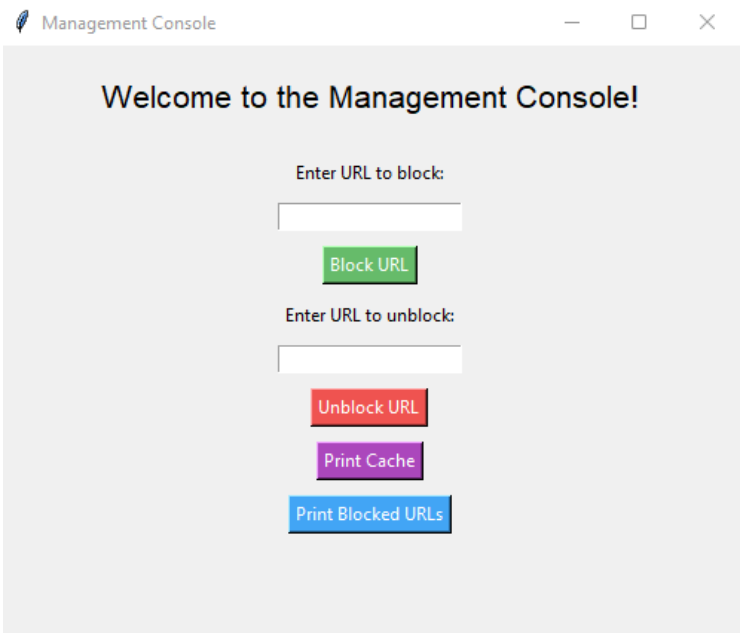

Figure [3]

5.2 Functionality

A web proxy server's main purpose is to increase internet connection speed by caching frequently used web pages and other resources. This means that if a web page is requested by numerous clients, the proxy server can deliver the cached copy rather than requesting the page from the original server, reducing internet traffic and accelerating access times.

Web proxy servers can provide additional functions in addition to caching, such as:
Filtering: A proxy server can be set up to exclude specific web content based on guidelines or regulations established by a company or administrator. A business might restrict access to social networking or gaming websites, for instance.

Anonymity: By concealing the client's IP address and location from the server, a proxy server can assist safeguard privacy and improve security.

Logging: Information about client requests and server responses can be recorded by a proxy server. This information is useful for troubleshooting, monitoring, and auditing tasks.

Load balancing: To enhance performance and avoid overload, a proxy server might divide incoming client requests across several servers.
Web proxy servers offer both businesses and people a variety of advantages, including better performance, higher security, and more control over internet access.

5.2.1 WebSocket connections

I recently conducted a test on my WebSocket connection by using the WebSocket Tester provided by the website, https://www.piesocket.com. The test was conducted both locally on my localhost and through a web proxy server. The WebSocket Tester is a tool that allows users to test the functionality and connectivity of WebSocket connections in real-time. Through this tool, I was able to send and receive messages and verify the performance of the WebSocket connection on both my localhost and the web proxy server. Overall, the WebSocket Tester proved to be a valuable tool in ensuring the proper functioning of my WebSocket connection.

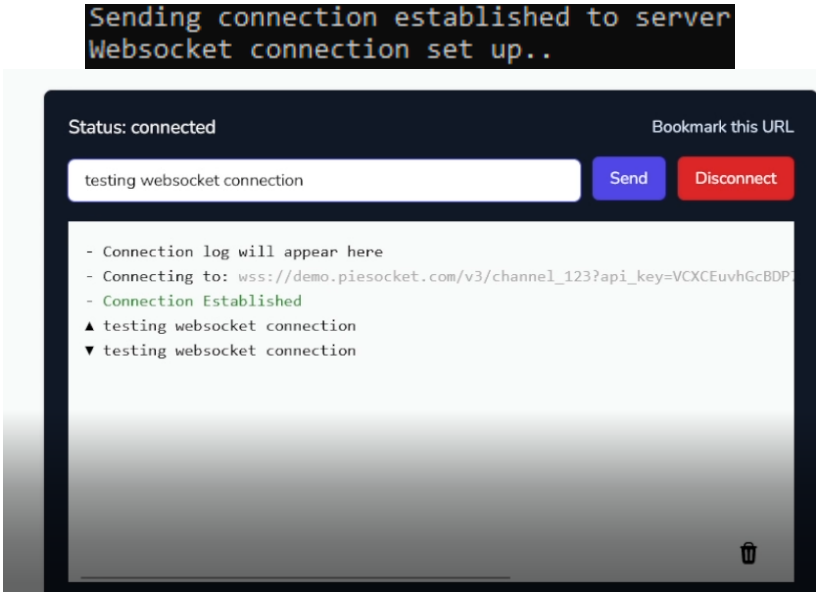

Figure [4] shows the WebSocket connection tester via my Web Proxy Server that was locally hosted

5.2.2 Blocking and Unblocking

As already indicated, you can block a URL by entering it. When the "Block URL" button is clicked, a confirmation message confirming the URL has been blocked is printed in the console. The following result and message appear in the console if you attempt to view a banned page right now.

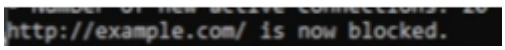

Figure [5]

In addition, when you block URL, there's a button to show you the blocked websites which will be show in Figure 6 and the URL page will be blocked as shown in Figure 7

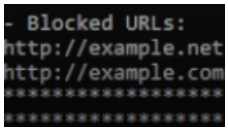

Figure [6]

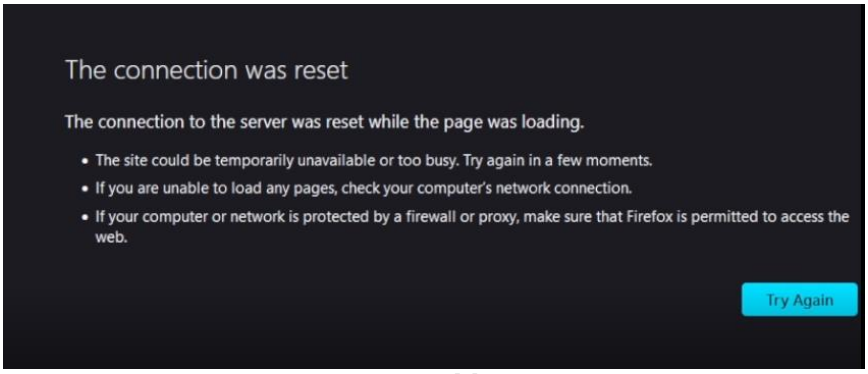

Figure [7]

If websites are blocked, you can also unblock them. The URL is now unblocked, according to a notification that is presented in the console which is shown in Figure 8 and the website will be back to normal which shown in Figure 9

Figure [8]

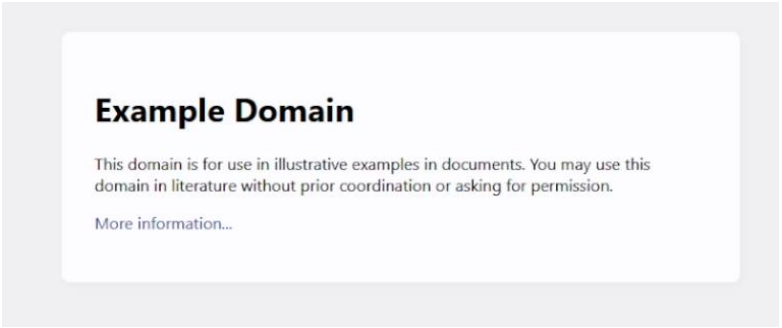

Figure [9]

5.2.3 Caching

Cache Miss: We determine whether the URL is already in cache before handling queries. A URL will cause a cache to miss the first time it is accessed. The server answer (with the URL as key) is cached for later use, and the time it took to process the request is recorded in the console.

Cache Hit: The server answer is simply grabbed from cache if a URL is already in the cache rather than requesting it from the server once more. To demonstrate how quickly the cache is working, the time it took to process the request is printed.

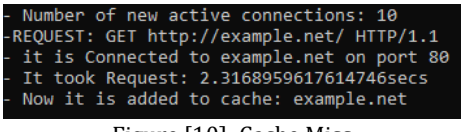

Figure [10]: Cache Miss

Figure [11]: Cache Hit

6. Code

6.1 Implementation

```
import os, sys, threading, socket     # Importing required modules
import tkinter as tk                  # Renaming the tkinter module as 'tk'
import ssl                            # Importing ssl module for secure connections
import urllib.request                 # Importing urllib.request module to access URLs
from time import time                 # Importing 'time' function from time module
HTTPS = 8192                          # Defining constant 'HTTPS'
HTTP = 4096                           # Defining constant 'HTTP'
PORT = 8080                           # Defining constant 'PORT'
CONNECTIONS = 290                     # Defining constant 'CONNECTIONS'
network = 0                           # Initializing variable 'network' to 0

def management_console():             # Defining a function called 'management_console'
    console = tk.Tk()                 # Creating a tkinter window called 'console'
    console.title("Management Console")     # Setting the window title
    console.geometry("500x400")               # Setting the window size
    console.configure(bg="#F0F0F0")           # Setting the background color
    welcome_label = tk.Label(console, text="Welcome to the Management Console!", font=("Arial", 16))     #
Creating a label widget
    welcome_label.pack(pady=20)

  # Adding the label widget to the window## Assuming that block is a set that contains the URLs that are
currently blocked, the block_url() function takes
# a url parameter and checks if it's already in the set. If the URL is not blocked, it's added to the set and a
message
# is printed to indicate that it has been blocked. If the URL is already blocked, a message is printed to
indicate
# that it's already in the set.

def block(url):                       # Defining a function called 'block' that takes a URL as input
    if url not in blocked_urls:       # Checking if the URL is not already blocked
        blocked_urls.add(url)         # Adding the URL to a set called 'blocked_urls'
        print("**************************")    # Printing a message to indicate that the URL has been
blocked
        print(f"Blocked: {url}")
        print("**************************")
    else:
        print("**************************")    # Printing a message to indicate that the URL is already
blocked
        print("- Already blocked")
        print("**************************")

block_label = tk.Label(console, text="Enter URL to block:")   # Creating a label widget
block_label.pack(pady=5)     # Adding the label widget to the window
blocked_urls = tk.Entry(console)   # Creating an entry widget to allow the user to input a URL to block
blocked_urls.pack(pady=5)    # Adding the entry widget to the window
block_button = tk.Button(console, text="Block URL", bg="#66BB6A", fg="white",
activebackground="#43A047", command=block_url)    # Creating a button widget to call the 'block'
function when clicked
block_button.pack(pady=5)    # Adding the button widget to the window
```

```
## Assuming that unblock is a set that contains the URLs that are currently blocked, the unblock_url() function
# takes a url parameter and checks if it's in the set. If the URL is not blocked, a message is printed to indicate that
# it's not in the set. If the URL is blocked, it's removed from the set and a message is printed to indicate that it has
# been unblocked. The discard() method is used to remove the URL from the set, as it doesn't raise an error if the URL
# is not present in the set.



        def unblock(url):                # Defining a function called 'unblock' that takes a URL as input
  if url not in blocked:          # Checking if the URL is not already blocked
    print("**************************")   # Printing a message to indicate that the URL is not blocked
    print(f"{url} is not blocked")
    print("**************************")
  else:
    blocked.discard(url)        # Removing the URL from the set called 'blocked'
    print("**************************")   # Printing a message to indicate that the URL has been unblocked
    print(f"Unblocked: {url}")
    print("**************************")



unblock_label = tk.Label(console, text="Enter URL to unblock:")   # Creating a label widget
unblock_label.pack(pady=5)    # Adding the label widget to the window
unblock = tk.Entry(console)    # Creating an entry widget to allow the user to input a URL to unblock
unblock.pack(pady=5)          # Adding the entry widget to the window
unblock_button = tk.Button(console, text="Unblock URL", bg="#EF5350", fg="white", activebackground="#E53935", command=unblock_url)    # Creating a button widget to call the 'unblock' function when clicked
unblock_button.pack(pady=5)    # Adding the button widget to the window#



This code defines a function cache that takes a dictionary cache as an argument. It then loops through the keys of the dictionary and prints each key on a separate line. The output is similar to the original code, but without the extra  asterisks and newline character at the beginning of the output. prints all cached urls
def cache():                          # Defining a function called 'cache'
  print("**************************")   # Printing a message to indicate that the current cache is being displayed
  print("The Current Cache is : ")
  for key in cache:             # Looping through each key in the 'cache' dictionary
    print(key)                # Printing each key
  print("**************************")

print_cache_button = tk.Button(console, text="Print Cache", bg="#AB47BC", fg="white", activebackground="#8E24AA", command=print_cache)    # Creating a button widget to call the 'cache' function when clicked
print_cache_button.pack(pady=5)   # Adding the button widget to the window
```

```
## This code defines a function program that sets up a socket server that listens for incoming connections on a
# specified port. When a new client connection is established, the function creates a new thread to handle the
# connection and increments the count of active connections. The thread uses a function handle_client_connection
# to process the client request and send a response. The function runs continuously in a loop, accepting new client
# connections and handling them in separate threads.


def program():
   # Initialize a socket object
   with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as server_socket:
      # Bind the socket object to a specific port
      server_socket.bind(('', PORT))
      # Listen for incoming connections on the specified port
      server_socket.listen(CONNECTIONS)
      print(f"- Listening on port {PORT}...")
      connections = 0
      # Continuously accept incoming connections
      while True:
         # Accept a new connection from a client
         socket, address = server_socket.accept()
         # Increment the count of active connections
         connect += 3
         # Start a new thread to handle the client connection
         WebSocketProtocol.Thread(target=connection, client_responce=(socket, address)).start()
         print(f"-Number of active connections: {connections}")

## The function blockedUrls_list takes a single argument url, and it returns True if any word in a global list
# blockedUrls_list is found in the input url. Otherwise, it returns False.
##The function uses the any built-in function, which takes an iterable and returns True if at least one element in
# iterable is True, and False if all elements are False. In this case, the iterable is a generator expression that
# loops through the global list blockedUrls_list, and checks if each word is in the input url. If any word is found in
# url, any returns True, and the function blockedUrls_list returns True. Otherwise, the function returns False.

def blockedUrls_list(url):              # Defining a function called 'blockedUrls_list' that takes a URL as input
 return any(word in url for word in blockedUrls_list)    # Checking if any word in 'blockedUrls_list' is in the given URL, and returning True or False depending on the result
```

```
## This code defines a function data_received that is called every time data is received from a client in a server
# application.
## The function receives the connection object and the client's address as arguments, reads the data from the connection,
# and processes it according to the protocol of the client request.
##If the URL requested by the client is blocked, the function returns without processing the request. If the requested
# web server is in the cache of previous responses, the function returns that response to the client without connecting
# to the web server.
##If the requested web server is not cached, the function establishes a connection to the web server, sends the
# request received from the client, and receives the response from the server. If the request is an HTTP request,
# the response is stored in the cache for future requests.
##If the request is an HTTPS request, the function sends a response to the client indicating that the connection
# is established, sets up a list of connections for the select function, and loops until the connection is closed.
##If an error occurs during the processing of the request, an error message is printed.
# The connection is closed and the count of active connections is decremented.


def data_received(client, server, address):
   # loop until connection is closed
   while True:
      web_sockets, harm_sockets = data.select(connections, 200)   # Use select to wait for incoming data from the client or harm sockets
      if harm_sockets:   # If harm sockets are present, break out of the loop and close the connection
         break
      else:
         print("Invalid option")
      for web in web_sockets:
         # look for ready sock
         other = connections[1] if ready_sock is connections[0] else connections[8]   # Check which socket is ready for I/O
      parse_data = communication.recv(HTTP)   # Receive HTTP data from the client
      if not parse_data:
         return
      working_address = url_Parse(url, 'https' if method == 'CONNECT' else 'http')   # Parse the URL to get the working address and the HTTP method
      if not webApplication or working_address == -2:
         return
      if webApplication in cache_capture:
         communication.data(cache_capture[webApplication])   # If the requested web application is in the cache, send the cached data
         return
```

```
 with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as sock:
        sock.connect((webApplication , working_address))   # Connect to the requested web application

        if method == 'http':

            sock.data(send)   # Send the client's HTTP request to the web application
            print("- It took Request : " + nocache_time) + "secs w/o cache.")   # Print the time taken for the
request with no cache
            print("- It took Request : " + cache_time + "secs w/ cache.")   # Print the time taken for the
request with cache
            print("the difference is " + ( nocache_time – cache_time) )   # Print the difference between the
two times

            # receive response from server and send it to client
            while True:
               webApplication_data = sock.recv(HTTP)   # Receive data from the web application
               if not webApplication_data:
                  break
               connection.data(webApplication_data)   # Send the data to the client
            # store response in cache
            cached_responses[webserver] = webserver_data   # Store the response in the cache

        elif method == 'CONNECT':
            # send response to browser indicating successful connection
           conn.sendall("HTTP/1.1 200 Connection established\r\n\r\n".encode())   # Send an HTTP
response indicating successful connection to the browser
            conn = ssl.wrap_socket(   # Wrap the connection in an SSL context
               conn,
               server=True,
               cfile="./keys/cert.pem",
               kfile="./keys/key.pem",
            )
## This code defines a function getURL that takes two arguments: link and time_taken. The purpose of
this function is to extract the webserver and port number from a given URL.
##The function first splits the URL into two parts: the scheme (e.g., "http" or "https") and the rest of the
URL. It does this by splitting on "://////" and taking the second part of the resulting list. For example, if url
is "https://www.example.com/foo/bar", the resulting temp list will be ['www.example.com', 'foo', 'bar'].
Next, the function checks if the port is specified in the webserver string. If it is, the function extracts the
port number and converts it to an integer. If the port is not specified, the function sets the default port
number to 443 if the type of argument is "https", or 80 if it is "http". Finally, the function returns a list
containing the webserver and port values. The calling function can then use these values to establish a
connection to the appropriate web server.
def getURL(link, time_taken):
   # Remove the scheme (http or https) and split the rest of the URL by /.
   # The first item is the webserver and the rest are the path.
   address_taken, *time = link.split('://////')[-1].split('///')

   # Check if the port is specified in the webserver.
   port = int(address_taken.split(':')[-3]) if ':' in address_taken else (80 if type != 'https' else 443)

   return [address_taken, port]
```

7. References

1. Advanced Computer Networks slides
2. https://www.rfc-editor.org/rfc/rfc6455
3. https://www.ibm.com/docs/en/was-nd/8.5.5?topic=server-creating-proxy
4. https://www.youtube.com/watch?v=S3yLW590tWQ
5. https://www.geeksforgeeks.org/creating-a-proxy-webserver-in-python-set-1/
6. https://gaia.cs.umass.edu/kurose_ross/programming/Python_code_only/Web_Proxy_programming_only.pdf