

CS3031 Advanced Telecommunications

Web Proxy Server

Anton Yamkovoy
17331565

February 24, 2020

0.1 Specification

The objective of the exercise is to implement a Web Proxy Server. A Web proxy is a local server, which fetches items from the Web on behalf of a Web client instead of the client fetching them directly. This allows for caching of pages and access control.

The program should be able to:

1. 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.
2. Handle websocket connections.
3. Dynamically block selected URLs via the management console.
4. Efficiently cache requests locally and thus save bandwidth. You must gather timing and bandwidth data to prove the efficiency of your proxy.
5. Handle multiple requests simultaneously by implementing a threaded server.

0.2 Implementation

To explain the implementation of the web proxy, I will go through a few typical threads of execution of the program, I have all of the features implemented that are required in the specification.

In any case when the program is run, a list of blocked sites is constructed from a file, these are stored in the list for later usage, and for interaction with the gui.

Before running the program you need to set up the proxy settings on your OS, as shown in the image below. I run the proxy on localhost, and port '8080', these settings are set at the start of the program.

Use a proxy server

☒ On

Address	Port
<input type="text" value="localhost"/>	<input type="text" value="8080"/>

Use the proxy server except for addresses that start with the following entries. Use semicolons (;) to separate entries.

☐ Don't use the proxy server for local (intranet) addresses

When the required settings are set up, the program is run, firstly the socket is initialised on ipv4, and a tcp connection, the socket is bound to the port, specified at the start, and the socket starts to listen for connections from the browser on the given port, which is why the settings are necessary for operation, If any of these steps fail the error is handled and the program is terminated.

```
PS C:\Users\Anton> cd .\Documents\GitHub\advanced_telecomms\  
PS C:\Users\Anton\Documents\GitHub\advanced_telecomms> python proxy.py  
$ Initialising socket...  
$ Bind successful  
$ Server listening on port [ 8080 ]  
  
Blocked Sites @ Init:  
  
learnyouahaskell.com  
www.un.org  
  
_
```

After socket initiation, we begin to loop around trying to accept connections from the browser, when a connection is found, we extract data, calculate

bandwidth, and start a initial timer, before entering a new thread which services this request, the thread starts a function proxy, that will decode, and enter a http or https handler based on the decoding and breakdown of the request.

When we enter, the proxy function, a few error checking cases are done, such as checking if the data is null, after this the data is passed into the parsing function, this returns a 5-tuple, a boolean to notify if it is a https request, webserver, port, url and method

After this the cache is checked, if the cache already contains a cached response based on this 'url' that was returned from the parsing function, the cached response is sent to the client, and some statistics are printed out to show the time savings and bandwidth usage previously to caching

```
Windows PowerShell
learnyouahaskell.com has been unblocked

$ Decoding request
$ Method : GET
$ URL : http://learnyouahaskell.com/chapters
$ Webserver: learnyouahaskell.com
$ starting new thread for : http://learnyouahaskell.com/chapters
$ sending HTTP request url: http://learnyouahaskell.com/chapters
$ added to cache (URL) : http://learnyouahaskell.com/chapters
$ time elapsed : 3.8497421741485596 seconds

$ total bandwidth : 3358 bytes

$ Decoding request
$ Method : GET
$ URL : http://learnyouahaskell.com/chapters

$ Webserver: learnyouahaskell.com

$ Decoding request
$ starting new thread for : http://learnyouahaskell.com/chapters
$ Method : CONNECT
$ Cache hit on url: http://learnyouahaskell.com/chapters
$ Request took: 0.0 seconds with cache.
$ Request took: 3.811753988265991 seconds previously$ URL : clients4.google.com:443

$ Webserver: clients4.google.com
$ Decoding request
$ starting new thread for : https://clients4.google.com/
$ Method : CONNECT
$ sending HTTPS
$ URL : clients4.google.com:443
$ Webserver: clients4.google.com
$ starting new thread for : https://clients4.google.com/
$ sending HTTPS

$ Decoding request
$ Method : GET
$ URL : http://learnyouahaskell.com/favicon.png
$ Webserver: learnyouahaskell.com
$ starting new thread for : http://learnyouahaskell.com/favicon.png
$ sending HTTP request url: http://learnyouahaskell.com/favicon.png
$ added to cache (URL) : http://learnyouahaskell.com/favicon.png
$ time elapsed : 3.371859550476074 seconds

$ total bandwidth : 980 bytes
```

If the request has not been previously cached, and it is a http request, it enters the http function, where a new socket is initiated, a timer is started to measure the time of the non cached request, using our client connection and the socket, we construct a http reply from the webserver, and send it out to the client, acting as a middleman receiving the data from the server and sending it on to the client.

The different types of requests are identified using the methods in the headers of the request. In the case where the request is identified as a https

request after the decoding stage in the proxy function, it cannot be cached as it is https, so we jump straight to the https handler function, here we connect to the webserver, and establish a non blocking bidirectional socket connection, following this we receive the request from the client, pass it onto the server, and reply back to the client with the server response.

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\Anton> cd .\Documents\GitHub\advanced_telecomms\
PS C:\Users\Anton\Documents\GitHub\advanced_telecomms> python proxy.py
$ Initialising socket...
$ Bind successful
$ Server listening on port [ 8080 ]

Blocked Sites @ Init:
learnyouahaskell.com
www.un.org

$ Decoding request
$ Method : CONNECT
$ URL : www.scss.tcd.ie:443
$ Webserver: www.scss.tcd.ie
$ starting new thread for : https://www.scss.tcd.ie/
$ sending HTTPS

$ Decoding request
$ Method : CONNECT
$ URL : www.scss.tcd.ie:443
$ Webserver: www.scss.tcd.ie
$ starting new thread for : https://www.scss.tcd.ie/
$ sending HTTPS

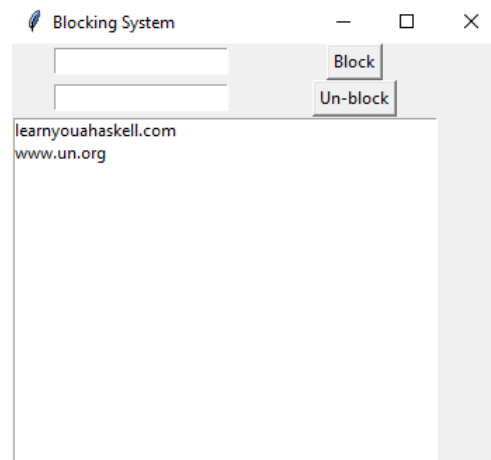
$ Decoding request
$ Method : CONNECT
$ URL : www.google-analytics.com:443
$ Webserver: www.google-analytics.com
$ starting new thread for : https://www.google-analytics.com/
$ sending HTTPS

$ Decoding request
$ Method : CONNECT
$ URL : www.google-analytics.com:443
$ Webserver: www.google-analytics.com
$ starting new thread for : https://www.google-analytics.com/
$ sending HTTPS
```

Another feature that I have implemented is a interface for blocking sites based on the webserver, a user can enter input to block or unblock certain sites,

from the being accessed, this is done dynamically using a separate thread launched at the start of the program that doesn't interfere with the rest of the program.

The initially blocked sites are found from a file at the start of the program, and printed in the console at launch, but they can be removed easily using the gui, which was implemented using a python gui library 'tkinter'



When a page is blocked the user will see this message if trying to access the blocked site.



This page isn't working

learnyouahaskell.com didn't send any data.

ERR_EMPTY_RESPONSE

Reload

0.3 Code

```
1 import socket
2 import tkinter as tk
3 from tkinter import *
4 import sys, _thread
5 import time
6 import ssl
7
8 #port number & buffer size
9 #CONSTANTS
10 global list
11 MAXDATA = 4096 # 4k default buffer
12 DEFAULT_PORT = 8080
13 BACKLOG = 1000
```



```

14 blocked_sites = [line.rstrip('\n') for line in
    open('blocked.txt')] # loading
15
16 #blocked_sites.append("scratchpads.eu") ## http blocking
17 #blocked_sites.append("www.un.org") ## https example blocking
18
19 cache = {} # dict as a cache for http responses
20 timeData = {}
21
22
23
24
25
26
27
28
29
30
31 def main():
32
33
34     try:
35         _thread.start_new_thread(gui,())
36         my_socket = socket.socket(socket.AF_INET,
            socket.SOCK_STREAM)# af_inet = ipv4, sock_stream =
            tcp
37         print("$ Initialising socket...")
38         my_socket.bind((' ',DEFAULTPORT))# all interfaces port
            : default_port
39         print("$ Bind successful")
40         my_socket.listen(BACKLOG)# will accept #|backlog|
            values until it wont accept new connections
41         print("$ Server listening on port [ %d ]\n" %
            (DEFAULTPORT))
42         printBlocked()
43     except Exception as e0:
44         print("$ socket initiation failed")
45         sys.exit(0)
46
47     ## loop while waiting for new connections from browser
48
49     while True:
50         try:
51             my_connection, my_address = my_socket.accept() #
                setting up connection, addr using socket

```

```

52         data = my_connection.recv(MAXDATA)
53         bandwidth = len(data)
54         starting_time = time.time()
55         _thread.start_new_thread(proxy,(my_connection,
56                                         data, my_address, starting_time, bandwidth))
56         time.sleep(0.0001)
57     except ConnectionResetError:
58         pass
59     except KeyboardInterrupt:
60         #my_connection.close()
61         my_socket.close()
62         print("$ socket force closed")
63         sys.exit(1)
64
65     my_socket.close()
66     return
67
68
69
70 def printBlocked():
71     print("Blocked Sites @ Init:\n")
72     for i in blocked_sites:
73         print(i)
74     print("\n")
75
76 def gui():
77
78     def block_entry():
79         e = block.get()
80         if e not in blocked_sites:
81             blocked_sites.append(e)
82             list.insert(END,e)
83             print(e + " has been blocked")
84         else:
85             print(e," has already been blocked")
86
87     def unblock_entry():
88         e = unblock.get()
89         if e in blocked_sites:
90             blocked_sites.remove(e)
91             for i, listbox_entry in enumerate(list.get(0, END)):
92                 if listbox_entry == e:
93                     list.delete(i)
94             print(e," has been unblocked")
95     else:

```

```

96         print(e," is not blocked")
97
98
99     gui = tk.Tk()
100     gui.geometry("350x300")
101     gui.title('Blocking System')
102
103
104     block = Entry(gui)
105     block.grid(row=0, column=0)
106     unblock = Entry(gui)
107     unblock.grid(row=1, column=0)
108
109     block_button = Button(gui, text = "Block",
110                           command=block_entry)
111     block_button.grid(row=0, column=1)
112     unblock_button = Button(gui, text = "Un-block",
113                             command=unblock_entry)
114     unblock_button.grid(row=1, column=1)
115
116     list = Listbox(gui)
117     list.grid(row=2, columnspan=2)
118     list.config(width=50, height=50)
119
120     for elem in blocked_sites:
121         list.insert(END,elem)
122
123     mainloop()
124
125
126 def proxy(my_connection, data,
127           my_address, starting_time, bandwidth):
128
129     if data is None:
130         return
131
132     https, web_srv, port, url, method = parse_req(data)
133
134     if web_srv in blocked_sites:
135         print("$ That host is restricted! (" ,web_srv, ")")
136         my_connection.close()
137         return

```

```

138     if method != "GET" and method != "CONNECT":
139         my_connection.close()
140         print("$ Invalid method :",method, " connection closed")
141         return
142
143
144     print("$ starting new thread for : ",url)
145
146     t0 = time.time()
147     cached_response = cache.get(url)
148     if cached_response != None:
149         my_connection.sendall(cached_response)
150         t1 = time.time()
151         print("$ Cache hit on url: ",url,"\n$ Request took: " +
            str(t1-t0) + " seconds with cache.\n $ Request took:
            " + str(timeData[url]) + " seconds previously")
152         #my_connection.close()
153         return
154
155
156     else:
157         if https == False:
158             bw =
                proxy_srv_http(web_srv,port,my_connection,data,my_address,url,bandwidth)
159         else:
160             proxy_srv_https(web_srv,port,my_connection,my_address,url)
161
162
163         end = time.time()
164         if https == False:
165             print("$ time elapsed :
                ",str(end-starting_time), " seconds\n")
166             if(bw == None):
167                 print("$ No bandwidth usage (retrieved from
                    cache) \n")
168             else:
169                 print("$ total bandwidth : ",str(bw), "
                    bytes")
170         my_connection.close()
171         return
172
173
174 def proxy_srv_http(web_srv, port, my_connection, data,
175     my_address, url,bandwidth):

```

```

176
177     t0 = time.time()
178
179     print("$ sending HTTP request url:",url)
180     my_socket2 = socket.socket(socket.AF_INET,
181                                socket.SOCK_STREAM)
182     my_socket2.connect((web_srv , port))
183     my_socket2.settimeout(3)
184     my_socket2.setblocking(0)
185
186     my_socket2.send(data)
187     b = len(data)
188     my_socket2.settimeout(3)
189
190     if url in cache:
191         print("$ Found request in cache")
192         t1 = time.time()
193         print("$ Request took: " + str(t1-t0) + "s with cache.")
194
195     else:
196         http_reply = bytearray("", 'utf-8')
197         try:
198             while True:
199                 rcv = my_socket2.recv(MAXDATA)
200                 if (len(rcv) > 0):
201                     my_connection.send(rcv)
202                     http_reply.extend(rcv)
203                 else:
204                     break
205             except socket.error:
206                 #print("$ HTTP socket error:", str(socket.error))
207             pass
208             t1 = time.time()
209             cache[url] = http_reply
210
211             timeData[url] = t1-t0
212             print("$ added to cache (URL) : ",url)
213             my_socket2.close()
214             #print("$ HTTP Request completed (URL): ",url)
215             b += len(http_reply)
216             return b
217     print("$ HTTP Request completed (URL): ",url)
218     my_socket2.close()
219     #print("$ HTTP socket closed ")

```

```

220
221
222
223
224
225 def proxy_srv_https(web_srv, port, my_connection, my_address, url):
226
227
228     print("$ sending HTTPS")
229     try:
230         my_socket2 = socket.socket(socket.AF_INET,
231                                     socket.SOCK_STREAM)
232         my_socket2.connect((web_srv, port))
233         reply = "HTTP/1.0 200 Connection
234                 established\r\nProxy-agent: Pyx\r\n\r\n"
235         my_connection.sendall(reply.encode())
236
237     except socket.error as err:
238         print(err)
239         return
240
241     my_connection.setblocking(0)
242     my_socket2.setblocking(0)
243
244     while True:
245         try:
246             request = my_connection.recv(MAX_DATA)
247             my_socket2.sendall(request)
248         except socket.error as err:
249             pass
250         try:
251             reply = my_socket2.recv(MAX_DATA)
252             my_connection.sendall(reply)
253         except socket.error as err:
254             pass
255
256     print("$ HTTPS request completed (URL) : ", url)
257
258 def get_host(line_data):
259     host = ""
260     for line in line_data:
261         h = line.find("Host")
262         if h != -1:
263             host = line

```

```

263         break
264     return host
265
266
267 def get_websrv(host):
268     web_srv = host.split(": ")[1]
269     port_pos = web_srv.find(":")
270     web_srv2 = ""
271     i = 0
272     if port_pos != -1:
273         while i < port_pos:
274             web_srv2 += web_srv[i]
275             i += 1
276     else:
277         web_srv2 = web_srv
278     return web_srv2
279
280
281 def parse_req(data):
282     try:
283         https = False
284         print("")
285         print("$ Decoding request")
286
287         line_data = data.decode().split("\r\n")
288
289         get = line_data[0].find("GET")
290         method = line_data[0].split(' ')[0]
291         if get == -1:
292             https = True
293
294         url_http = line_data[0].split(' ')[1]
295
296         url = "https://" + url_http.split(':')[0] + "/"
297
298         host = get_host(line_data)
299         webserver = get_websrv(host)
300
301         print("$ Method :",method)
302         print("$ URL :", url_http)
303         print("$ Webserver:",webserver)
304
305         if https is True:
306             port = 443
307     else:

```

```

308         port = 80
309     if https:
310         return https, webserver, port, url, method
311     else:
312         return https, webserver, port, url_http, method
313 except Exception:
314     #print("decode exception")
315     pass
316     return True, 0, 0, "", ""
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335 main()

```