CS3031 Advanced Telecommunications
Project I

Séamus Woods
15317173

19/02/2019

0.1 Specification

The objective of the excercise 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 HTTP & HTTPS Requests

0.3 Websocket Connections

0.4 Dynamically Blockings URLs

0.5 Caching

0.6 Threading

0.7 Code

```
1 #! /usr/bin/env python
2 import os, sys, thread, socket, time
```

```
3  import Tkinter as tk
4  from Tkinter import *
5
6  # CONSTANTS
7  # How many pending connection will the queue hold?
8  BACKLOG = 200
9  # Max number of bytes to receive at once?
10 MAX_DATA_RECV = 4096
11 # Set true if you want to see debug messages.
12 DEBUG = True
13 # Dict to store the blocked URLs
14 blocked = {}
15 # Dict to act as a cache, stores responses.
16 cache = {}
17 # Dict to store time of response before caching.
18 timings = {}
19
20 # Tkinter function.. Used to dynamicall block URLs.
21 # Also used to display the current blocked URLs and the cache.
22 def tkinter():
23         # Create block and unblock entries..
24         console = tk.Tk()
25         block = Entry(console)
26         block.grid(row=0,column=0)
27         unblock = Entry(console)
28         unblock.grid(row=1, column=0)
29
30         # Function for blocking urls.. basically take whats in
               the entry cell and put it into
31         # the dict..
32         def block_url():
33                 ret = block.get()
34                 temp = blocked.get(ret)
35                 if temp is None:
36                         blocked[ret] = 1
37                         print("[*] Successfully blocked: " +
                               ret)
38                 else:
39                         print("[*] This website is already
                               blocked..")
40         # Creating a button to call the block_url function..
41         block_button = Button(console, text="Block URL",
               command=block_url)
42         block_button.grid(row=0, column=1)
43
```

```
44          # Function for unblocking urls.. basically tkaes whats
                in the entry cell and removes it
45          # from the blocked dict if it exists..
46          def unblock_url():
47                  ret = unblock.get()
48                  temp = blocked.get(ret)
49                  if temp is None:
50                          print("[*] Url is not blocked: " + ret)
51                  else:
52                          blocked.pop(ret)
53                          print("[*] Successfully unblocked: " +
                                ret)
54          # Creating a button to call the unblock_url function..
55          unblock_button = Button(console, text="Unlock URL",
                command=unblock_url)
56          unblock_button.grid(row=1, column=1)
57
58          # Function to print all currently blocked urls..
59          def print_blocked():
60                  print(blocked)
61          print_blocked = Button(console, text="Print Blocked
                URLs", command=print_blocked)
62          print_blocked.grid(row=3, column=0)
63
64          # Function to print all currently cached pages..
65          def print_cache():
66                  for key, value in cache.iteritems():
67                          print key
68          print_blocked = Button(console, text="Print Cache",
                command=print_cache)
69          print_blocked.grid(row=3, column=1)
70
71          # Could add other functionality here :D
72
73          mainloop()
74
75  # MAIN PROGRAM
76  def main():
77          # Run a thread of our tkinter function..
78          thread.start_new_thread(tkinter,())
79
80          try:
81                  # Ask user what port they'd like to run the
                        proxy on..
82                  listening_port = int(raw_input("[*] Enter
```

```
                    Listening Port Number: "))
83          except KeyboardInterrupt:
84                  # Handling keyboard interrupt.. looks nicer..
85                  print("\n[*] User Requested An Interrupt")
86                  print("[*] Application Exiting...")
87                  sys.exit()
88          try:
89                  # Ininitiate socket
90                  s = socket.socket(socket.AF_INET,
                        socket.SOCK_STREAM)
91                  # Bind socket for listen
92                  s.bind(('', listening_port))
93                  # Start listening for incoming connections
94                  s.listen(BACKLOG)
95                  print("[*] Initializing sockets... done")
96                  print("[*] Sockets binded successfully...")
97                  print("[*] Server started successfully [ %d
                        ]\n" % (listening_port))
98          except Exception, e:
99                  print("[*] Unable to initalize socket...")
100                 sys.exit(2)
101
102         while True:
103                 try:
104                         # Accept connection from client browser
105                         conn, client_addr = s.accept()
106                         # Receive client data
107                         data = conn.recv(MAX_DATA_RECV)
108                         # Start a thread
109                         thread.start_new_thread(proxy_thread,
                                (conn, data, client_addr))
110                 except KeyboardInterrupt:
111                         s.close()
112                         print("[*] Proxy server shutting
                                down...")
113                         sys.exit(1)
114         s.close()
115
116
117
118 def proxy_thread(conn, data, client_addr):
119         print("")
120         print("[*] Starting new thread...")
121         try:
122                 # Parsing the request..
```

```
123                 first_line = data.split('\n')[0]
124                 url = first_line.split(' ')[1]
125                 method = first_line.split(' ')[0]
126                 print("[*] Connecting to url " + url)
127                 print("[*] Method: " + method)
128                 if (DEBUG):
129                         print("[*] URL: " + url)
130
131                 # Find pos of ://
132                 http_pos = url.find("://")
133                 if (http_pos == -1):
134                         temp = url
135                 else:
136                         # Rest of url..
137                         temp = url[(http_pos+3):]
138                 # Finding port position if there is one..
139                 port_pos = temp.find(":")
140
141                 # Find end of web server
142                 webserver_pos = temp.find("/")
143                 if webserver_pos == -1:
144                         webserver_pos = len(temp)
145
146                 webserver = ""
147                 port = -1
148                 # Default port..
149                 if (port_pos == -1 or webserver_pos < port_pos):
150                         port = 80
151                         webserver = temp[:webserver_pos]
152                 # Specific port..
153                 else:
154                         port =
                            int((temp[(port_pos+1):])[:webserver_pos-port_pos-1])
155                         webserver = temp[:port_pos]
156
157                 # Checking if we already have the response in
                    our cache..
158                 t0 = time.time()
159                 x = cache.get(webserver)
160                 if x is not None:
161                         # If we do, don't bother with
                            proxy_server function and send the
                            response on..
162                         print("[*] Found in Cache!")
163                         print("[*] Sending cached response to
```

```
                                user..")
164                             conn.sendall(x)
165                             t1 = time.time()
166                             print("[*] Request took: " + str(t1-t0)
                                    + "s with cache.")
167                             print("[*] Request took: " +
                                    str(timings[webserver]) + "s before
                                    it was cached..")
168                             print("[*] That's " +
                                    str(timings[webserver]-(t1-t0)) + "s
                                    slower!")
169                     else:
170                             # If we don't, continue..
171                             proxy_server(webserver, port, conn,
                                    client_addr, data, method)
172             except Exception, e:
173                     pass
174
175
176     def proxy_server(webserver, port, conn, client_addr, data,
            method):
177             s = socket.socket(socket.AF_INET, socket.SOCK_STREAM) #
                    Initiating socket..
178
179             # Checking our blocked dict to check if the URL the
                    user is trying to connect to
180             # is blocked..
181             for key, value in blocked.iteritems():
182                     if key in webserver and value is 1:
183                             print("That url is blocked!")
184                             conn.close()
185                             return
186
187             # If the method is CONNECT, we know this is HTTPS.
188             if method == "CONNECT":
189                     try:
190                             # Connect to the webserver..
191                             s.connect((webserver, port))
192                             reply = "HTTP/1.0 200 Connection
                                    established\r\n"
193                             reply += "Proxy-agent: Pyx\r\n"
194                             reply += "\r\n"
195                             print("[*] Sending connection
                                    established to server..")
196                             conn.sendall(reply.encode())
```

```
197                 except socket.error as err:
198                         print(err)
199                         return
200                 conn.setblocking(0)
201                 s.setblocking(0)
202                 # Bidirectional messages here.. (Websocket
                        connection)
203                 print("[*] Websocket connection set up..")
204                 while True:
205                         try:
206                                 #print("[*] Receiving request
                                        from client..")
207                                 request =
                                        conn.recv(MAX_DATA_RECV)
208                                 #print("[*] Sending request to
                                        server..")
209                                 s.sendall(request)
210                         except socket.error as err:
211                                 pass
212                         try:
213                                 #print("[*] Receiving reply
                                        from server..")
214                                 reply = s.recv(MAX_DATA_RECV)
215                                 #print("[*] Sending reply to
                                        client..")
216                                 conn.sendall(reply)
217                         except socket.error as err:
218                                 pass
219                 print("[*] Sending response to client..")
220         # Else we know this is HTTP.
221         else:
222                 # String builder to build response for our
                        cache.
223                 t0 = time.time()
224                 string_builder = bytearray("", 'utf-8')
225                 s.connect((webserver, port))
226                 print("[*] Sending request to server..")
227                 s.send(data)
228                 s.settimeout(2)
229                 try:
230                         while True:
231                                 #print("[*] Receiving response
                                        from server..")
232                                 reply = s.recv(MAX_DATA_RECV)
233                                 if (len(reply) > 0):
```

```
234                                         #print("[*] Sending
                                                response to
                                                client..")
235                                         # Send reply back to
                                                client
236                                         conn.send(reply)
237                                         string_builder.extend(reply)
238                                 else:
239                                         break
240                     except socket.error:
241                             pass
242                     print("[*] Sending response to client..")
243                     t1 = time.time()
244                     print("[*] Request took: " + str(t1-t0) + "s")
245                     timings[webserver] = t1-t0
246                     # After response is complete, we can store this
                            in cache.
247                     cache[webserver] = string_builder
248                     print("[*] Added to cache: " + webserver)
249                     # Close server socket
250                     s.close()
251                     # Close client socket
252                     conn.close()
253
254
255 if __name__ == '__main__':
256         main()
```