Advanced Telecommunications - Proxy

Owen Burke, 15316452

February 27, 2019

High-level description of the protocol design and implementation

The following implementation of a Proxy server is written in python 3.
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.

High-level description :
The first step was to establish a connection to the web browser so that the proxy could then forward on the requests to the end host.
This was done by creating a socket that is bound to port 50000 on the localhost(127.0.0.1). Within the web browser's proxy settings, one can redirect all traffic through any proxy of your choice, so these values must be provided.
I then accept incoming connections on this socket, and for each connection is spool up a new daemon thread, starting with the handle_browserConnection function.
I also have the threads listening for a keyboard exception, in order for the user to provide input for blocking/unblocking hosts, etc.
This "handle_browserConnection" function does many of the essential operations such as determining whether the site is http/https, pulling out the information from the request about the end server, printing to the management console, determining whether the cached data is valid, etc.

The design was then to have two different functions. One that deals with http traffic and another that deals with https.
The http function is "forwardData" and this essentially just makes another socket that is connected to the end server, which we know from handle_browserConnection. Then sends the request to the server and then sends the response back to the connection we have made with the browser.
The https function is "httpsForward" and this establishes a secure connection with the end server. In this function, when it receives a http connect request, it sends a http 200 back to the browser connection, as a signal that we are ready to transmit (see https://en.wikipedia.org/wiki/HTTP_tunnel).
We then forward on the encrypted data from the browser to the server and vice

versa. I believe this establishes a duplex stream allowing the proxy to handle web socket connections (as the proxy correctly displays https://www.websocket.org/echo.html) (See figure 4).
In terms of blocking hosts, when the user triggers the keyboard exception, by ctrl+c, they can either add or remove blocked hosts. Given that the proxy is threaded as mentioned before , host can be blocked/unblocked dynamically. This is implemented using a set as it has O(1) operations. Whenever the browser wants to communicate, the "handle_browserConnection" function checks if the host is currently in the set. If it is not, the connection is performed as described above. If it is in the set, for http sites, a simple piece of html is returned to the browser saying the host is blocked. For https sites, the handshake isn't performed and the browser displays that the connection couldn't be established securely **(See figure 1 & 2)**. In terms of design, I decided to block the host rather than the specific URL, as this made more sense to me (to block all of www.nytimes.com rather than blocking www.nytimes.com/section/opinion and not blocking www.nytimes.com/section/business).

For caching, whenever a request is made from the browser (as long as it is http, because https traffic cannot be cached due to encryption etc), the response is stored in a dictionary/hashmap where the key is the url from the request and the response is the associated value. If that request is made again, the proxy sends of the request to the server (with an If-Modified-Since header with the time stamp of the current cached value) and if the server responds with a 304, we know the value in the cache is up to date, and this data is sent back to the browser. If it is not a 304, the new data is read from the server into the cache, replacing the old data and this is then sent back to the browser (the timestamp is also updated).

Unfortunately, the caching doesn't work fully but you can see from the code that the foundation is there. You can see from the comments that certain aspects of the code relates to caching. I am unsure as to why the time for the cache is greater than the time for the end server (as you can see from figure 3). To see the current state of the caching, you must uncomment these sections. Otherwise, comment them out to see the proxy work as expected just without caching. (See figure 3 for caching)
Also, some times an connection refused exception will be thrown. I'm unsure as to why this is the case, however everything renders in the browser fine and all the links, etc work.

Code :

```
1 import socket, sys, ssl, time
2 from threading import Thread
3 from wsgiref.handlers import format_date_time
```

```
4  from datetime import datetime
5
6
7  blockedList = set()
8
9  BUFFSIZE = 1024
10 socketList = []
11 s = None
12
13 cache = {}                #the actual cache
14 dates = {}                #associated dates for url
15 cacheTiming = {}          #for url, record times for return
       from cache
16 hostTiming = {}           #for url, record times for return
       from end server
17
18
19
20 def closeSockets(*sockets):                    # close
       sockets
21     for curSocket in sockets:
22         if curSocket:
23             curSocket.close()
24     if s:
25         s.close()
26     print("Shutting down")
27     sys.exit(-1)
28
29
30 def userInputs(*sockets):                       #handle user
       input for blocking/unblocking of hosts
31     userIn = input("\nWould you like to exit the proxy (
           type \"exit\") or block a URL (give a host) or
           unblock a host (\"r\")? (No input = show blocked
           list): ").lower()
32     if userIn == 'exit':
33         closeSockets(sockets)
34     elif userIn == "":
35         print ("Blocked = " + str(blockedList))
36     elif userIn == "r":
37         unblock = input("Provide a host to unblock : ")
38         try:
39             blockedList.remove(unblock)
40             print("removed " + unblock)
41         except KeyError:
42             print("Host not in blocked list")
```

```
43      else:
44          blockedList.add(userIn.lower())
45          print(userIn + " added to list")
46
47
48
49  def cacher(destHost, destPort, key, data):
                        #Caching function
50      start = time.time()
                                                    #start
            timer
51      responseToBrowser = b""
                                                #response to
            send to browser
52
53      if key in cache:
                                                        #check
            key is in cache
54          forwardSocket = socket.socket()
55          forwardSocket.connect((destHost, destPort))
56
57          responseToBrowser = cache[key]
                                        #get the response
                from the cache
58
59          details = str(data.decode()).split("\r\n")
60
61          i = 0
62          for string in details:
63              if string.startswith("If-Modified-Since:"):
64                  details[i] = "If-Modified-Since: " + str(
                        dates[key]) + "\r\n"
65                  break
66              i = i + 1
67
68          newData = b""
69
70          for string in details:
71              if string is not '':
72                  new = string + "\r\n"
73                  newData += str.encode(new)
74
75          forwardSocket.send(newData)
                                        #send of request to
                see if we get a 304
76
```

```
77
78          rep = forwardSocket.recv(BUFFSIZE)
79
80          if b"304" in rep:
                                                    #If is
                304
81              end = time.time()
                                                    #end the
                     timer
82              cacheTiming[key] = (end-start)
                                        #time it took to send
                     data to browser from cache
83              return responseToBrowser
                                                #return to
                    browser
84          else:
85              toCache = rep
                                                           #
                    cache new response from server
86              while 1:
87                  rep = forwardSocket.recv(BUFFSIZE)
88                  if len(rep) > 0:
89                      toCache += rep
90                  else:
91                      break
92              now = datetime.now()
93              timeStamp = time.mktime(now.timetuple())
94              dates[key] = format_date_time(timeStamp)
                            #update timestamp
95              cache[key] = toCache
                                                  #update
                    data in cache
96              print(str(key) + " updated")
97              end = time.time()
98              cacheTiming[key] = (end-start)
99              return toCache
100
101     else:
102         return None
103
104
105
106
107
108 #
    ##########################################################################
```

```
      CACHING - http function to cache data - uncomment for
      caching
109
110 # def forwardData(destHost, destPort, browserConnection,
    data ,address):
111 #     try:
112 #         details = str(data.decode()).split("\r\n")
113 #         key = details[0]

    #obtain the url/key
114
115 #         starttime = time.time()

    #start timer
116
117 #         forwardSocket = socket.socket(socket.AF_INET,
    socket.SOCK_STREAM)
118 #         forwardSocket.connect((destHost, int(destPort)))
                                    #connect socket to
    server
119 #         forwardSocket.sendall(data)
                                                          #
    send to server
120
121
122 #         toCache = b""

    #data to be cached
123
124 #         forwardSocket.setblocking(0)
125
126 #         while 1:
127 #             try:
128 #                 rep = forwardSocket.recv(BUFFSIZE)
                                          #take in reply
129 #                 toCache = toCache + rep
                                                      #append
      to data to be cached
130 #                 print ("\n")
131 #                 browserConnection.send(rep)
132 #             except BlockingIOError:
133 #                 break
134
135 #         now = datetime.now()
136 #         stamp = time.mktime(now.timetuple())
137 #         dates[key] = format_date_time(stamp)
```

```
                                                    #obtain
    date
138 #         print("New entry : " + str(key))
139 #         cache[key] = toCache

    #add data to cache
140 #         endtime = time.time()

    #end timer
141 #         hostTiming[key] = (endtime-starttime)
                                                    #how long it
     took when asking server
142
143 #         forwardSocket.close()
144 #         browserConnection.close()
145 #     except KeyboardInterrupt:
146 #         userInputs(browserConnection, forwardSocket)
147
148 #
    ##########################################################################################

149
150 # Non caching forwardData - comment out when caching
151
152 def forwardData(destHost, destPort, browserConnection,
    data ,address):
153     try:
154         forwardSocket = socket.socket(socket.AF_INET,
              socket.SOCK_STREAM)           #create socket
              that is connected to end server
155         forwardSocket.connect((destHost, int(destPort)))
156         forwardSocket.sendall(data)
157         while 1:
158             rep = forwardSocket.recv(BUFFSIZE)
                                                    #take
                  in response from server
159             if(len(rep) <= 0):
160                 break
161             print ("\n")
162             browserConnection.sendall(rep)
                                                    #
                send response to browser
163         forwardSocket.close()

            #close connections
164         browserConnection.close()
```

```
            #close connections
165     except KeyboardInterrupt:
166         userInputs(browserConnection, forwardSocket)
167
168 # ##########################################################################
169
170
171
172 def httpsForward(destHost, destPort, browserConnection,
        data ,address):
173     try:
174         httpsSocket = socket.socket(socket.AF_INET,
              socket.SOCK_STREAM)
175         response = b"HTTP/1.0 200 Connection Established\
              r\nConnection: close\r\nProxy-agent: Pyx\r\n\r
              \n\r\n"  #Relay 200 ok to browser to signal
              ready for transmission
176         browserConnection.send(response)
177         httpsSocket.connect((destHost, int(destPort)))
                                          #connect to server
178
179         browserConnection.setblocking(0)
                                    #set connections as non
              -blocking (see https://docs.python.org/2/howto
              /sockets.html)
180         httpsSocket.setblocking(0)
181         while True:
182             try:

                  #multiple try/excepts are so if client
                  gives error, then try with server (with
                  one big except = would have situation
                  where
183                 req = browserConnection.recv(BUFFSIZE)
                                #   client gives error then
                      try again immediately with the same
                      client).
184                 if len(req) <= 0:
185                     break
186                 httpsSocket.send(req)
                                                  #send data
                      to/from browser to server
187             except Exception:
```

```
188                 pass
189             try:
190                 rep = httpsSocket.recv(BUFFSIZE)
191                 if len(rep) <= 0:
192                     break
193                 browserConnection.send(rep)
194             except Exception:
195                 pass
196     except Exception:
197         print(Exception)
198
199
200
201
202
203
204 def handle_browserConnection(data, browserConnection,
        address):
205     try:
206         isHttps = False
207         details = str(data.decode()).split("\r\n")
208         if details[0].startswith("CONNECT"):
                                            #Determine whether
                http/https
209             isHttps = True
210         hostIndex = 0
211         i = 0
212         for string in details:
213             if(string[ : 4] == "Host"):
214                 hostIndex = i
215             i = i+1
216         print ("\n")
217         destHost = details[hostIndex][6:]
                                           #Find the host
            name and port in the incoming request
218
219         if destHost.split(":")[0] not in blockedList:
220             for string in details:
221                 print('\x1b[0;32;40m' + string + '\x1b[0m
                    ')              #Print request to
                        management console
222             destPortTemp = destHost.find(":")
223             destPort = 0
224             if(destPortTemp == -1 and isHttps == False):
225                 destPort = 80
```

```
                    #Default port for http = 80
226             elif(destPortTemp == -1 and isHttps == True):
227                 destPort = 443

                    #Default port for https = 443
228             else:
229                 destPort = destHost[destPortTemp+1 :]
230                 destHost = destHost[ : destPortTemp]
231
232 #   ##########################################################################
     CACHING - uncomment for caching
233             # key = details[0]
234             # response = cacher(destHost, int(destPort),
                    key, data)
235             # time.sleep(1)
236             # if response is not None:
237             #     browserConnection.sendall(response)
238             #     print("Found in cache : " + str(
                    cacheTiming[key]))
239             #     print("From host : " + str(hostTiming[
                    key]))
240             #     browserConnection.close()
241             #     return
242 #   ##########################################################################


243
244
245             if(isHttps == False):
246                 forwardData(destHost, destPort,
                        browserConnection, data ,address)   #
                        Begin http transmission
247             else:
248                 httpsForward(destHost, destPort,
                        browserConnection, data ,address)  #
                        Begin https transmission
249
250         else:
251             print("This host is blocked\n")
                                                            #
                    Return blocked host html
252             browserConnection.sendall(b"HTTP/1.0 200 OK\r
                    \nContent-Type: text/html\r\n\r\n<html><
                    body>Blocked Host.</body></html>\r\n\r\n")
253             browserConnection.close()
```

```
254         except KeyboardInterrupt:
255             userInputs(browserConnection)
256 
257 
258 #   ##########################################################################################
259 
260 
261 
262 def run():
263     try:
264         s = socket.socket(socket.AF_INET, socket.
                SOCK_STREAM)        #Sets up socket.
265         s.setsockopt(socket.SOL_SOCKET, socket.
                SO_REUSEADDR, 1)
266         s.bind(('127.0.0.1', 50000))
                                                #binds socket
                to localhost port xxxxxx.
267         s.listen(15)
                
                #Listen for connections made to the socket. 15
                 = max number of queued connections.
268         print("Listening...")
269     except Exception:
270         print("Failure")
271 
272 
273     while 1:
274         try:
275             (browserConnection, address) = s.accept()
                            #Accept incoming
                browserConnection. Returns connection
                details and addr.
276             data = browserConnection.recv(BUFFSIZE)
                            #Receive max of 8192 bytes,
                at once, of incoming data. Returns string
                representation of data.
277             thread = Thread(target =
                handle_browserConnection, args = (data,
                browserConnection, address))
278             thread.daemon = True
279             thread.start()
                                                        #
                startNewthread to print and forward.
280         except KeyboardInterrupt:
```

```
                                                    #Allows for
                shutting down of proxy.
281             userInputs()
282
283
284
285
286 run() #start proxy
```

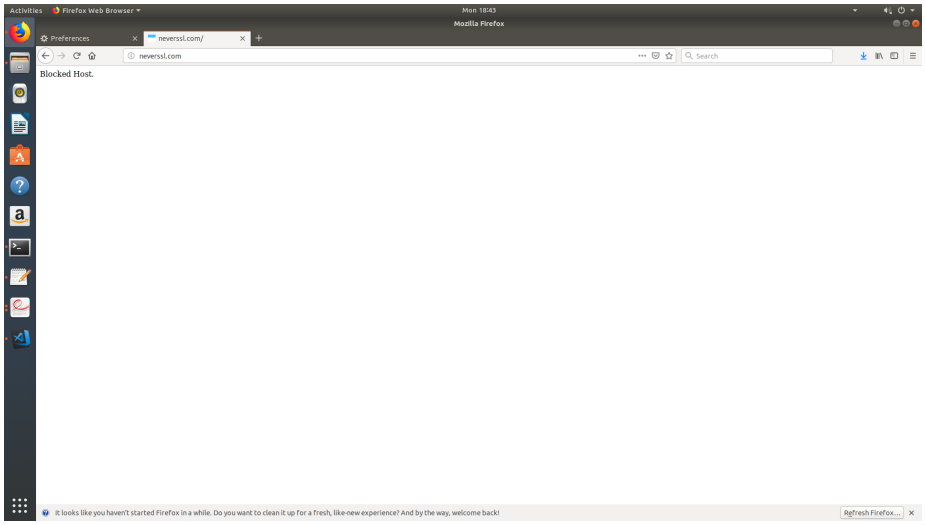

Figure 1: http blocking

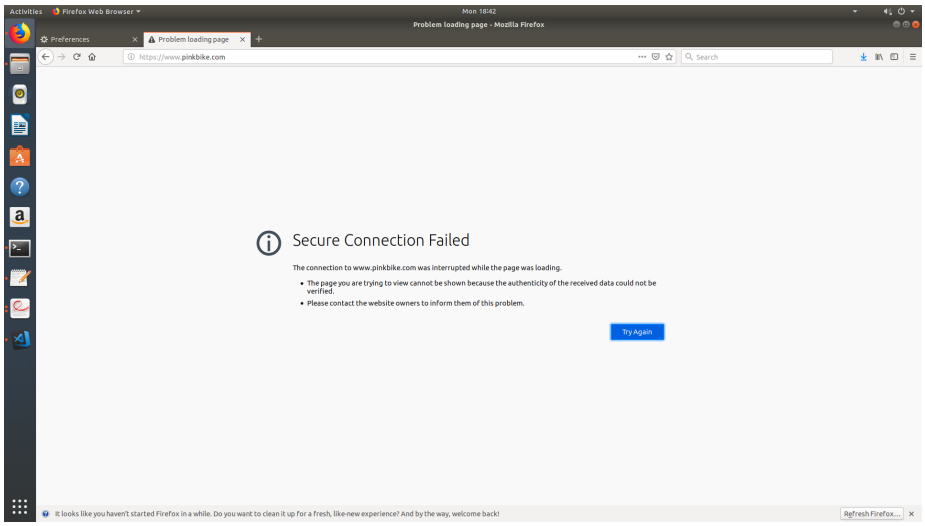

Figure 2: https blocking

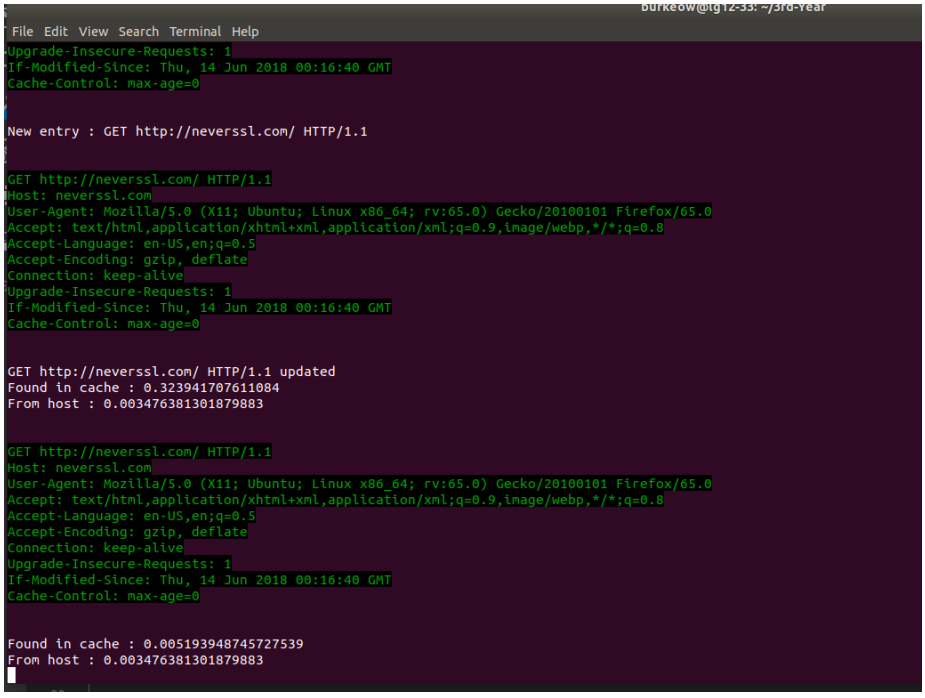

Figure 3: caching

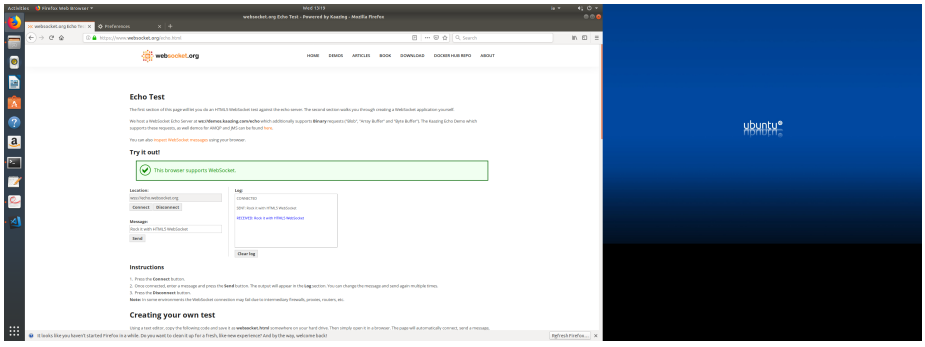

Figure 4: websockets