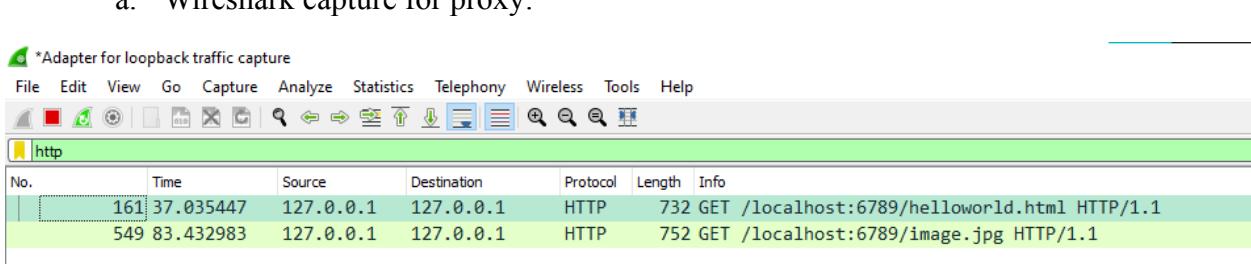


Lab 2: Building a Web Server

1. A brief description of the basic logic of your program.
 - a. For our web server, the logic is pretty simple. First, we create a socket and bind to it. After this step, we start listening for any incoming requests. The rest of this program is pretty straightforward in that it parses the packet and sends the contents of the request to the connection socket. In addition to this, there is error handling if the connection socket is able to receive and send the packets.
 - b. For our proxy server, the logic is slightly more complicated. It follows the same logic as the web server in creating a socket and binding to it, listening for requests, and sending/receiving packets. However, the difference is that there is a middle man between the client and the web server. This is called the proxy server. In order to do this, we create another socket and bind the existing web server (6789) to the new port (8888). After all of this is done, if the file is successfully read, then we create a file and store it in cache.
2. Sample runs and screenshots as instructed to demonstrate the correctness of your program.
 - a. Wireshark capture for proxy:



- b. Google colab photos:

A screenshot of a Google Colab terminal session. The session is titled "[30]".

- The first part of the session shows a file upload command: "Choose Files helloworld.html". It lists a file named "helloworld.html(text/html)" and shows the command "curl http://localhost:6789/helloworld.html -o my_helloworld.html".
- The second part of the session shows the file being saved: "Saving helloworld.html to helloworld.html" and "helloworld.html uploaded".
- The third part of the session shows a note: "*Step 4: Run the browser curl to fetch helloworld.html. *".
- The fourth part shows a curl command being run: "curl http://localhost:6789/helloworld.html -o my_helloworld.html". The output shows the progress of the download: "% Total % Received % Xferd Average Speed Time Dload Upload Total" and values "100 92 0 92 0 0 92000 0 ---:--- ---:---".
- The fifth part shows a note: "Step 5: Based on the web server software in step 1, you can develop a simple web server in C/C++".

```

8s [30] filename = next(iter(uploaded))
      print(filename, "uploaded")

      Choose Files helloworld.html
      • helloworld.html(text/html) - 90 bytes, last modified: 3/6/2023 - 100% done
      Saving helloworld.html to helloworld.html
      helloworld.html uploaded

*Step 4: Run the browser curl to fetch helloworld.html. *

0s [ ] !curl http://localhost:6789/image.jpg -o my_image.jpg
      % Total    % Received % Xferd  Average Speed   Time   Time   Current
                                         Dload  Upload Total   Spent   Left  Speed
      100 8087k    0 8087k     0       0  717M      0 --:--:-- --:--:-- 92000

```

Step 5: Based on the web server software in step 1, you can develop a similar software that runs on localhost but at a different port, say 8888. Similar to steps 1 and 2, upload your proxy server python file here and run it in the background.

```

[ ] from google.colab import files
      print("Upload your proxy server file here")
      uploaded = files.upload()
      proxyname = next(iter(uploaded))
      print(proxyname, "uploaded")

[ ] !nohup python -u $proxyname >& proxy.log &

```

```

1s [ ] !curl http://localhost:6789/helloworld.html -o my_helloworld.html
      % Total    % Received % Xferd  Average Speed   Time   Time   Time  Current
                                         Dload  Upload Total   Spent   Left  Speed
      100     92    0     92     0       0  92000      0 --:--:-- --:--:-- 92000

```

*Step 4: Run the browser curl to fetch helloworld.html. *

```

1s [ ] !curl http://localhost:6789/image.jpg -o my_image.jpg
      % Total    % Received % Xferd  Average Speed   Time   Time   Time  Current
                                         Dload  Upload Total   Spent   Left  Speed
      100 8087k    0 8087k     0       0  717M      0 --:--:-- --:--:-- 717M

```

```

1s [ ] !curl http://localhost:8888/localhost:6789/helloworld.html
      <html>
      <body>

      <h1>Network Programming</h1>

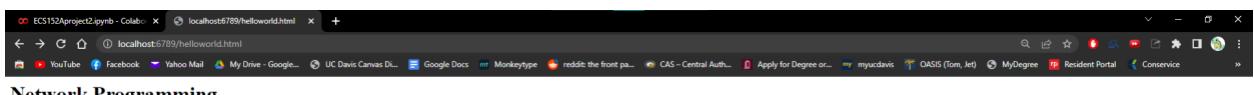
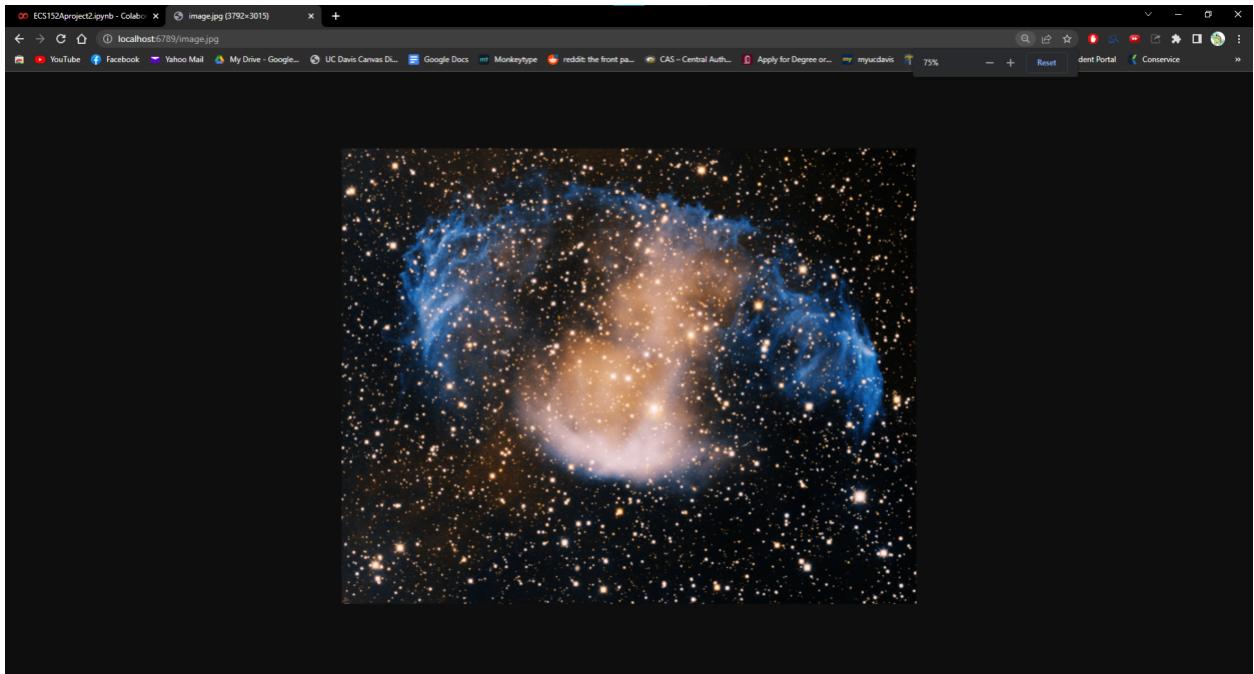
      <p>Hello World.</p>

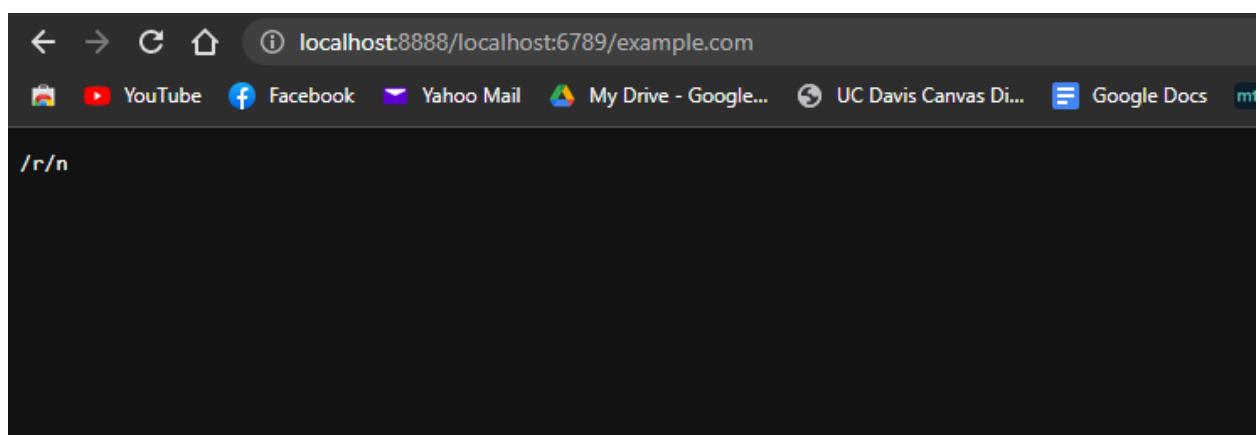
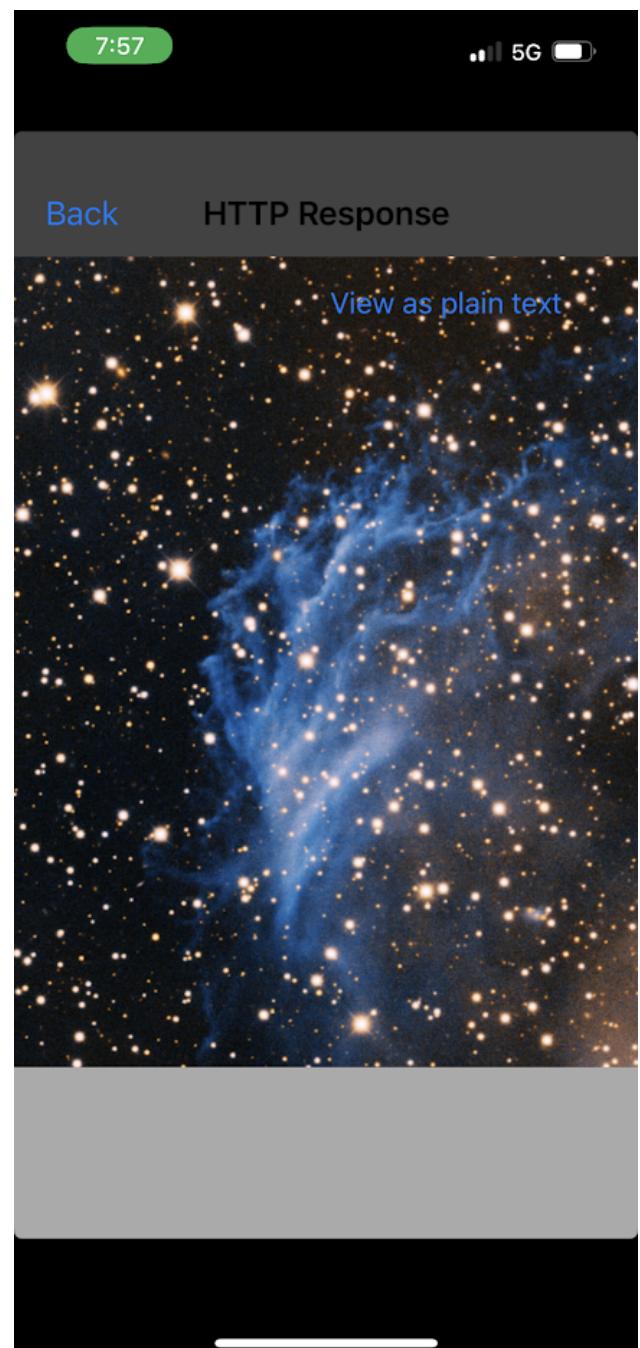
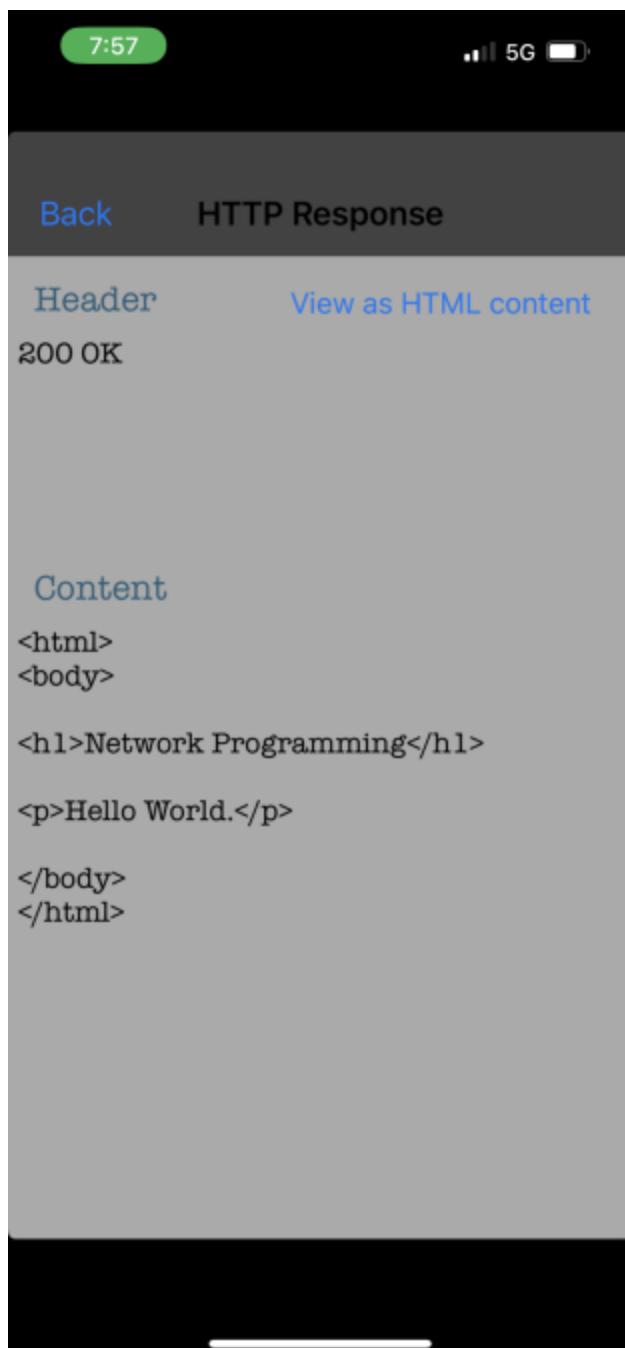
      </body>
      </html>

```

```
[83] !curl http://localhost:8888/localhost:6789/image.jpg -o myimage.jpg
```

% Total	% Received	% Xferd	Average Speed	Time	Time	Time
Dload	Upload	Total	Spent	Left		
100	8087k	0	394M	0	--:--:--	--:--:--





http						
No.	Time	Source	Destination	Protocol	Length	Info
162	42.573047	127.0.0.1	127.0.0.1	HTTP	728	GET /localhost:6789/example.com HTTP/1.1

c. Terminal output:

```
(base) kevinmooney@Kevins-MacBook-Pro-68 proxy % python3 ProxyServerPROB.py |(base) kevinmooney@Kevins-MacBook-Pro-68 project2 % python3 WebServerPROB_1.py
Ready to serve...
helloworld.html
Ready to serve...
Ready to serve...
Ready to serve...
image.jpg
Ready to serve...
Ready to serve...
Ready to serve... |Ready to serve...
b'GET /helloworld.html HTTP/1.1\r\n'
Ready to serve...
'^[[A[[Bb'GET /helloworld.html HTTP/1.1\r\n'
Ready to serve...
b'GET /image.jpg HTTP/1.1\r\n'
Ready to serve...
```

3. Answers to questions raised in this assignment.

- a. The performance of our proxy server is faster than the web server because it acts as a caching mechanism. For previously requested content, it recognizes it and speeds up the transaction since it has known info already.
- b. The time required for the proxy server was less than the time required for the web server. This is because the proxy server already had the file in the cache and was able to pull it up faster.

4. Web Server Program

```
# Import socket module
from socket import *
serverPort = 6789

# Create a TCP server socket
#(AF_INET is used for IPv4 protocols)
#(SOCK_STREAM is used for TCP)

serverSocket = socket(AF_INET, SOCK_STREAM)

# Fill in start
serverSocket.bind(("", serverPort))
serverSocket.listen(1)
# Fill in end

# Server should be up and running and listening to the incoming
connections
```

```
while True:
    print('Ready to serve...')

    # Set up a new connection from the client
    connectionSocket, addr = serverSocket.accept() #Fill in start
#Fill in end

    # If an exception occurs during the execution of try clause
    # the rest of the clause is skipped
    # If the exception type matches the word after except
    # the except clause is executed
    try:
        # Receives the request message from the client
        message = connectionSocket.recv(1024).decode() #Fill in start
#Fill in end

        # Extract the path of the requested object from the message
        # The path is the second part of HTTP header, identified by
[1]
        filename = message.split()[1]

        # Because the extracted path of the HTTP request includes
        # a character '\', we read the path from the second
character
        f = open(filename[1:], 'rb')

        # Store the entire content of the requested file in a
temporary buffer
        outputdata = f.read() #Fill in start           #Fill in end

        # Send the HTTP response header line to the connection
socket
        # Fill in start
        connectionSocket.send("HTTP/1.1 200 OK\r\n\r\n".encode())
# Fill in end

        # Send the content of the requested file to the connection
socket
        connectionSocket.sendall(outputdata)
        connectionSocket.send("\r\n".encode()))
```

```
# Close the client connection socket
connectionSocket.close()

except IOError:

    # Send HTTP response message for file not found
    # Fill in start
    connectionSocket.send("HTTP/1.1 404 Not
Found\r\n\r\n".encode())
    # Fill in end

    # Close the client connection socket
    # Fill in start
    connectionSocket.close()
    # Fill in end

serverSocket.close()
```

Proxy Server Program

```
5. # Import socket module
6. from socket import *
7.
8. serverPort = 8888
9. # Create a TCP server socket
10.# (AF_INET is used for IPv4 protocols)
11.# (SOCK_STREAM is used for TCP)
12.
13.serverSocket = socket(AF_INET, SOCK_STREAM)
14.webServer = socket(AF_INET, SOCK_STREAM)
15.
16.# Fill in start
17.serverSocket.bind(("" , serverPort))
18.serverSocket.listen(1)
19.# Fill in end
20.
```

```
21. # Server should be up and running and listening to the incoming
   connections
22. while True:
23.     print('Ready to serve...')
24.
25.     # Set up a new connection from the client
26.     connectionSocket, clientAddr = serverSocket.accept() #Fill in
      start           #Fill in end
27.     print (clientAddr)
28.     message = connectionSocket.recv(1024)
29.     filename_whost = message.split()[1]
30.     filename_b = filename_whost.split(b"/")[2]
31.     filename = filename_b.decode()
32.     # If an exception occurs during the execution of try clause
33.     # the rest of the clause is skipped
34.     # If the exception type matches the word after except
35.     # the except clause is executed
36.     try:
37.         #if file is in cache
38.         # Receives the request message from the client
39.         f = open(filename, 'rb')
40.
41.         # Store the entire contenet of the requested file in a
   temporary buffer
42.         outputdata = f.read()#Fill in start           #Fill in end
43.
44.         # Send the HTTP response header line to the connection
   socket
45.         # Fill in start
46.         connectionSocket.send("HTTP/1.1 200 OK\r\n\r\n".encode())
47.         # Fill in end
48.
49.         # Send the content of the requested file to the connection
   socket
```

```
50.         #for i in range(0, len(outputdata)):
51.         connectionSocket.sendall(outputdata)
52.         connectionSocket.send("\r\n".encode())
53.
54.     # Close the client connection socket
55.     connectionSocket.close()
56.
57. except IOError:
58.     #if file is not in cache
59.     print(filename)
60.     try:
61.         webServer.connect(('127.0.0.1', 6789))
62.         tmpstr ="GET " + "/" + filename + " HTTP/1.1\r\n"
63.         webServer.send(tmpstr.encode())
64.         newFile = webServer.makefile('b',0)
65.         #newFile.write("GET " + "/" + filename + " HTTP/1.1\r\n")
66.         tempBuffer = newFile.readlines()
67.
68.         tempFile= open("./" + filename,"wb")
69.         for line in tempBuffer:
70.             tempFile.write(line)
71.             connectionSocket.send(line)
72.             connectionSocket.send("/r/n".encode())
73.     except:
74.
75.         # Send HTTP response message for file not found
76.         # Fill in start
77.             connectionSocket.send("HTTP/1.1 404 Not
    Found\r\n\r\n".encode())
78.         # Fill in end
79.
80.         # Close the client connection socket
81.         # Fill in start
```

```
82.         connectionSocket.close()
83.         # Fill in end
84.         serverSocket.close()
```