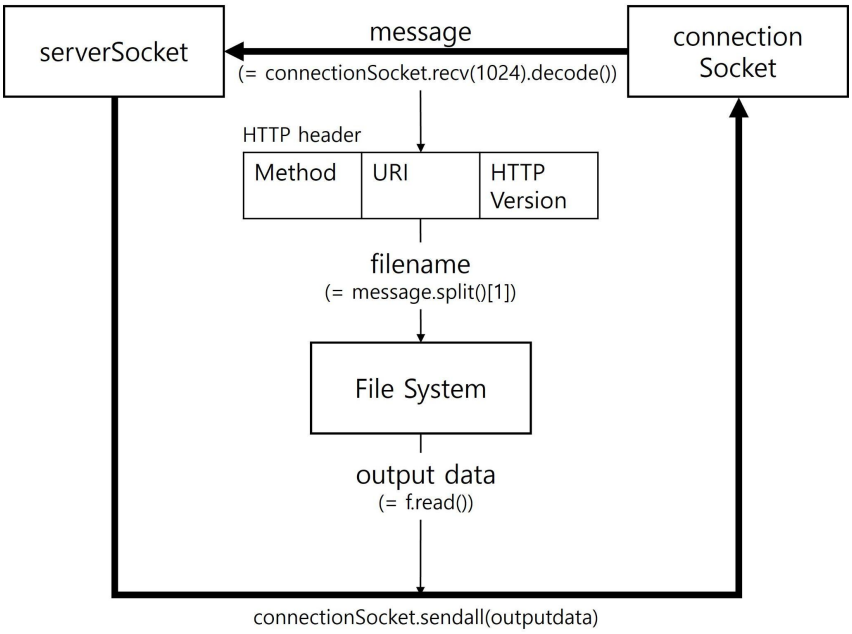
Building a Web Server
ECS 152A

Winter 2023

Erik Young
Sohyun Yoo
March 9th, 2023

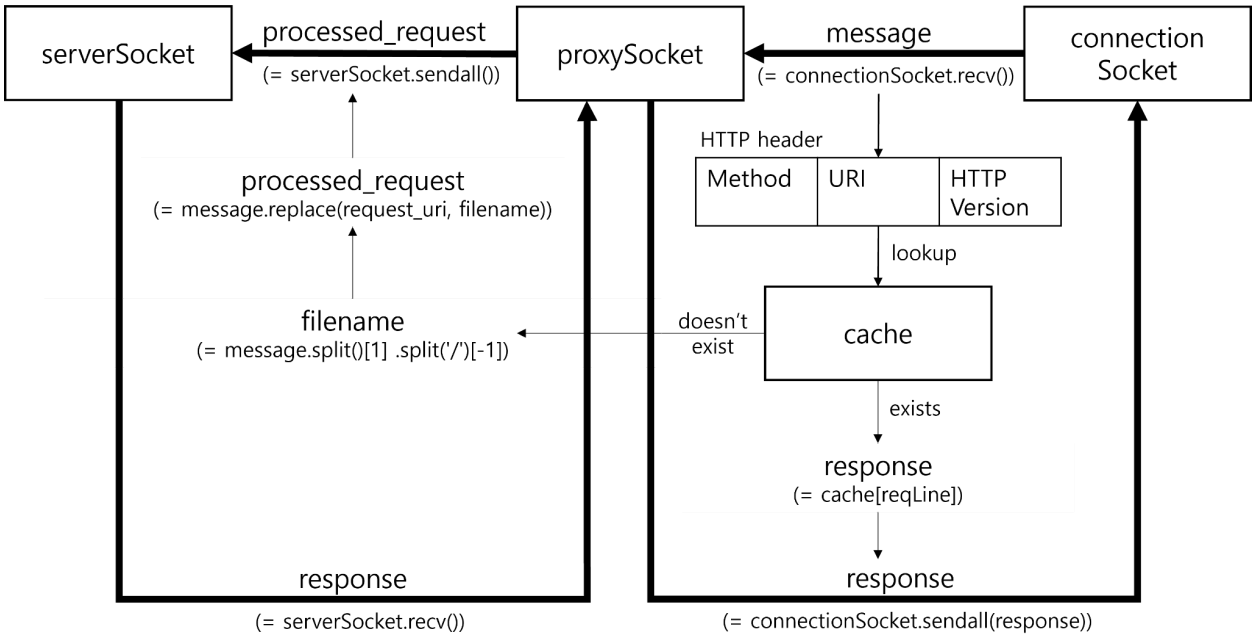
1 Methodology

1.1 Web Server Program



For this program, we initially set up the serverSocket by binding it to the local host address and given port number(6789). Then we ran the server to repeatedly listen to response and set up a new connection from the client through serverSocket's accept function. When a request message was received, we extracted the path of the requested object from the message by splitting it and taking the second part according to HTTP header format. Then we accessed the file system using the file path, retrieved the file in binary format and stored the content in a temporary buffer by reading it. To acknowledge the client about response status, we sent HTTP response header line with status 200 first and sent requested data afterwards. During this process, if the server faces IOError, we made it send a 404 Not Found error message. At the end, we closed the connectionSocket and serverSocket. The process of retrieving data using request can be described as the diagram above.

1.2 Web Proxy Server Program

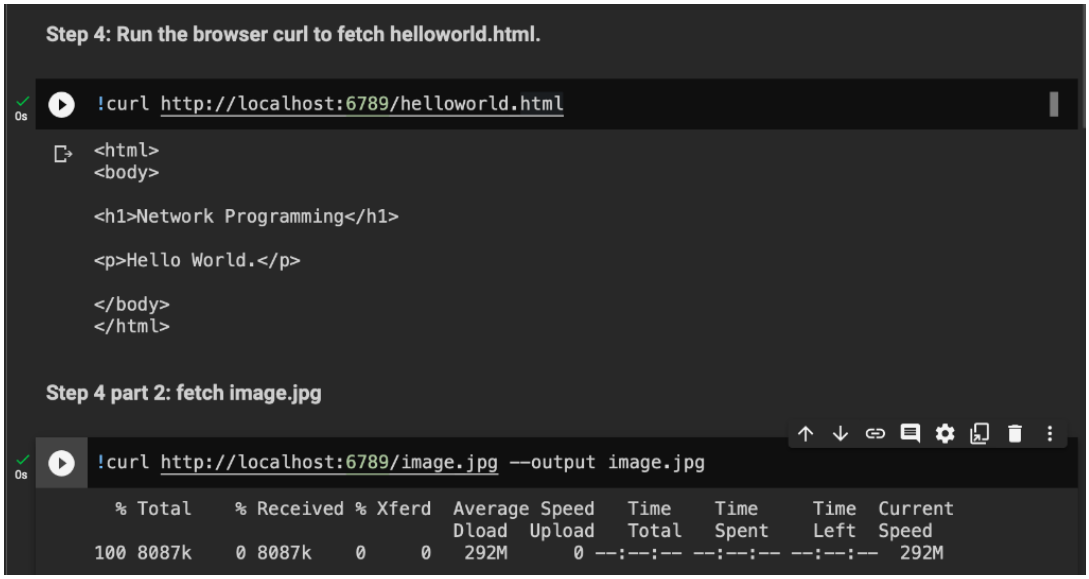


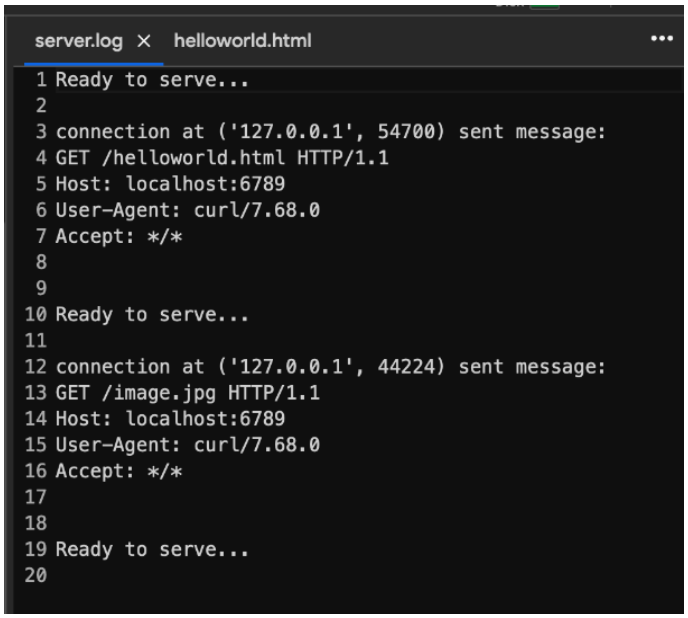
For this program, we initially set up the proxySocket by binding it to the local host address and given port number(8888). Then we ran the proxy server to repeatedly listen to response and set up a new connection from the client through proxySocket's accept function. When a request message is received, the proxy server initially looks up the cache and sends the response right away if the request exists. Otherwise, it processes the request using HTTP format again, removing server information by editing the uri part of the request. Then we set up a new serverSocket and connect it to the host and port number that were requested. As the proxy server receives the response from the server, it forwards the response to the client until no more data is received in its buffer. During this process, if the server faces IOError, we made it send a 404 Not Found error message. At the end, we closed the connectionSocket, serverSocket and proxySocket. The process of forwarding data using request can be described as the diagram above.

2 Results

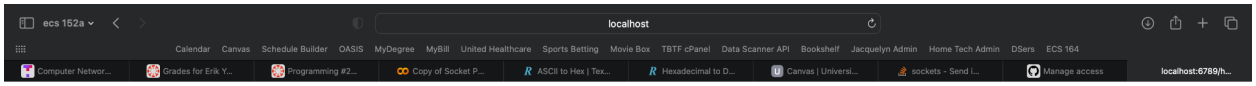
2.1 Web Server Program

2.1.1. Task1: Colab Outputs



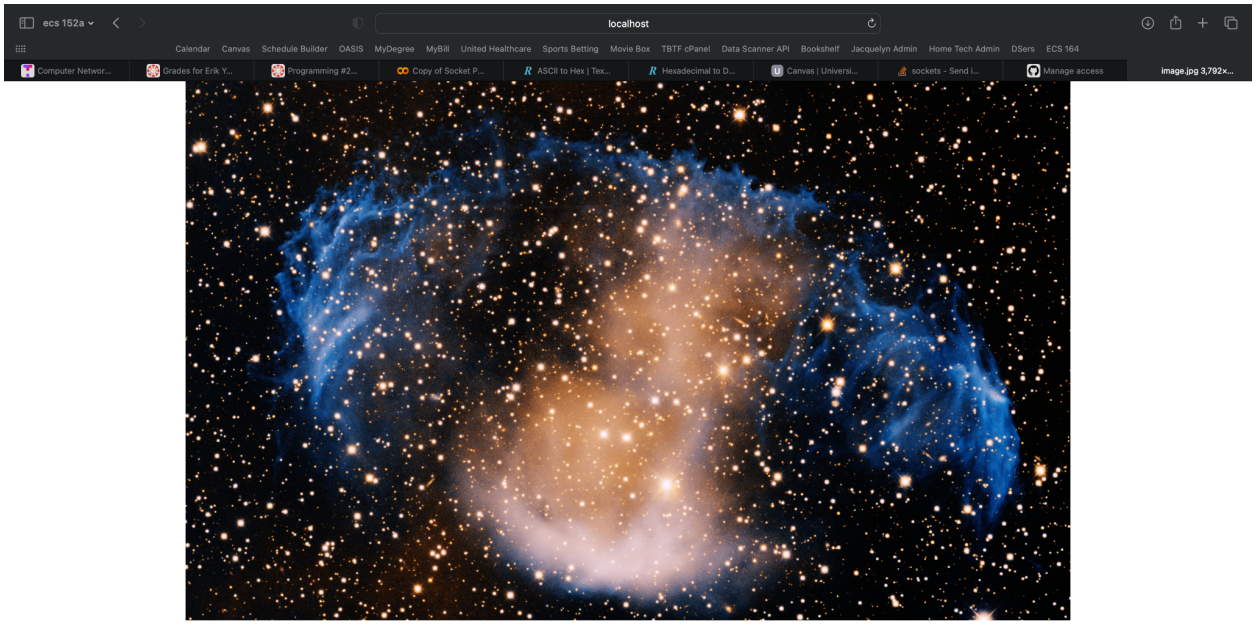


2.1.2. Task2: Local environment outputs



Network Programming

Hello World.




```
erikyoung85@Eriks-MacBook-Pro programming2 % python3 WebServerPROB.py
Ready to serve...

connection at ('127.0.0.1', 49968) sent message:
GET /helloworld.html HTTP/1.1
Host: localhost:6789
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Upgrade-Insecure-Requests: 1
Cookie: csrftoken=3PRgoWCjXHLQkfVOQ08EQUbxqoxtQNQasiAY3lFQhKcIcgJRVgsQL5WfNlyTOOH3
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/16.3 Safari/605.1.15
Accept-Language: en-US,en;q=0.9
Accept-Encoding: gzip, deflate
Connection: keep-alive


Ready to serve...

connection at ('127.0.0.1', 49970) sent message:
GET /image.jpg HTTP/1.1
Host: localhost:6789
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Upgrade-Insecure-Requests: 1
Cookie: csrftoken=3PRgoWCjXHLQkfVOQ08EQUbxqoxtQNQasiAY3lFQhKcIcgJRVgsQL5WfNlyTOOH3
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/16.3 Safari/605.1.15
Accept-Language: en-US,en;q=0.9
Accept-Encoding: gzip, deflate
Connection: keep-alive


Ready to serve...
```

2.2 Web Proxy Server Program

2.2.1. Task3: Colab Proxy Server Outputs (First fetch)

Step 6: Run the browser but this time through the proxy server

```
!curl http://localhost:8888/localhost:6789/helloworld.html
```

```
<html>
<body>

<h1>Network Programming</h1>

<p>Hello World.</p>

</body>
</html>
```

Step 6 part 2: Fetch the image through the proxy server

```
!curl http://localhost:8888/localhost:6789/image.jpg --output image.jpg
```

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 8087k    0 8087k    0     0   607M      0 --:--:-- --:--:-- --:--:--  607M
```

proxy.log

```
1 Ready to serve...
2
3 connection at ('127.0.0.1', 56086) sent message:
4 GET /localhost:6789/helloworld.html HTTP/1.1
5 Host: localhost:8888
6 User-Agent: curl/7.68.0
7 Accept: */*
8
9
10 forwarding to localhost:6789
11
12 Ready to serve...
13
14 connection at ('127.0.0.1', 34186) sent message:
15 GET /localhost:6789/image.jpg HTTP/1.1
16 Host: localhost:8888
17 User-Agent: curl/7.68.0
18 Accept: */*
19
20
21 forwarding to localhost:6789
22
23 Ready to serve...
24
```

2.2.2. Task3: Colab Proxy Server Outputs (Second fetch)

Step 6: Run the browser but this time through the proxy server

```
[8]  !curl http://localhost:8888/localhost:6789/helloworld.html

<html>
<body>

<h1>Network Programming</h1>

<p>Hello World.</p>

</body>
</html>
```

Step 6 part 2: Fetch the image through the proxy server

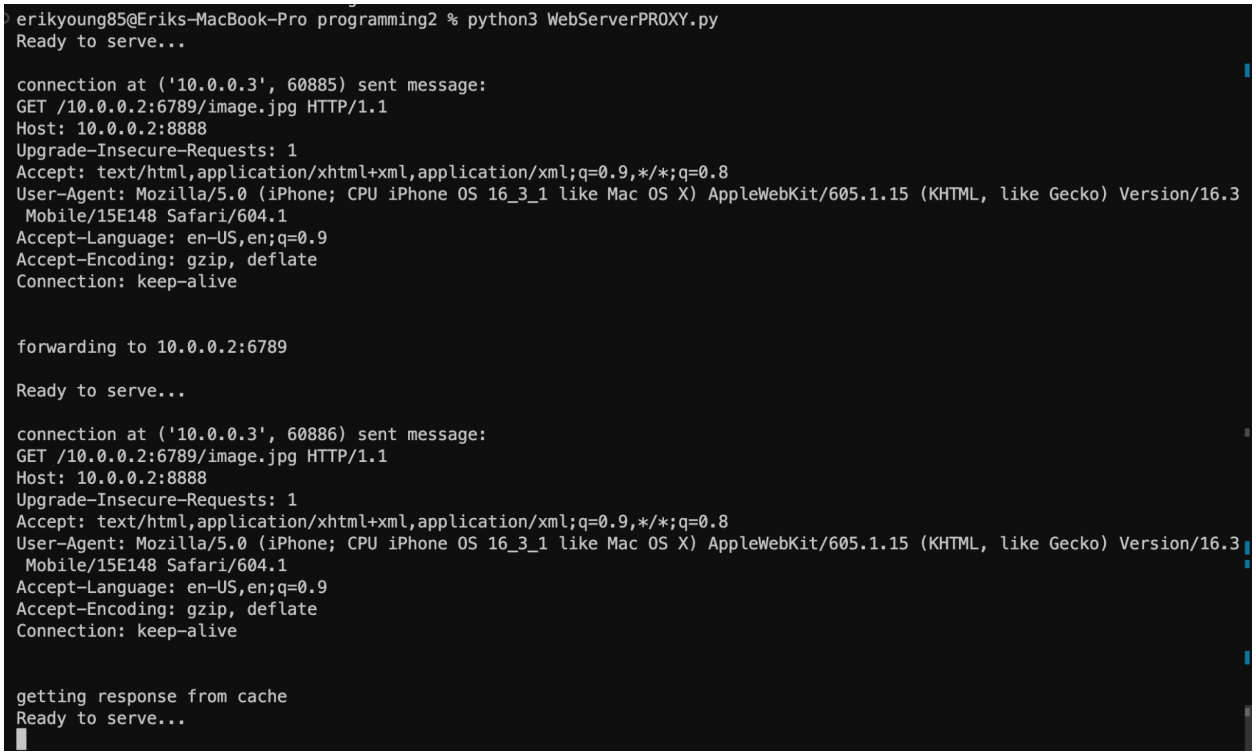
```
!curl http://localhost:8888/localhost:6789/image.jpg --output image.jpg

  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 8087k    0 8087k    0     0  1789k      0 --:--:--  0:00:04 --:--:-- 1873k
```

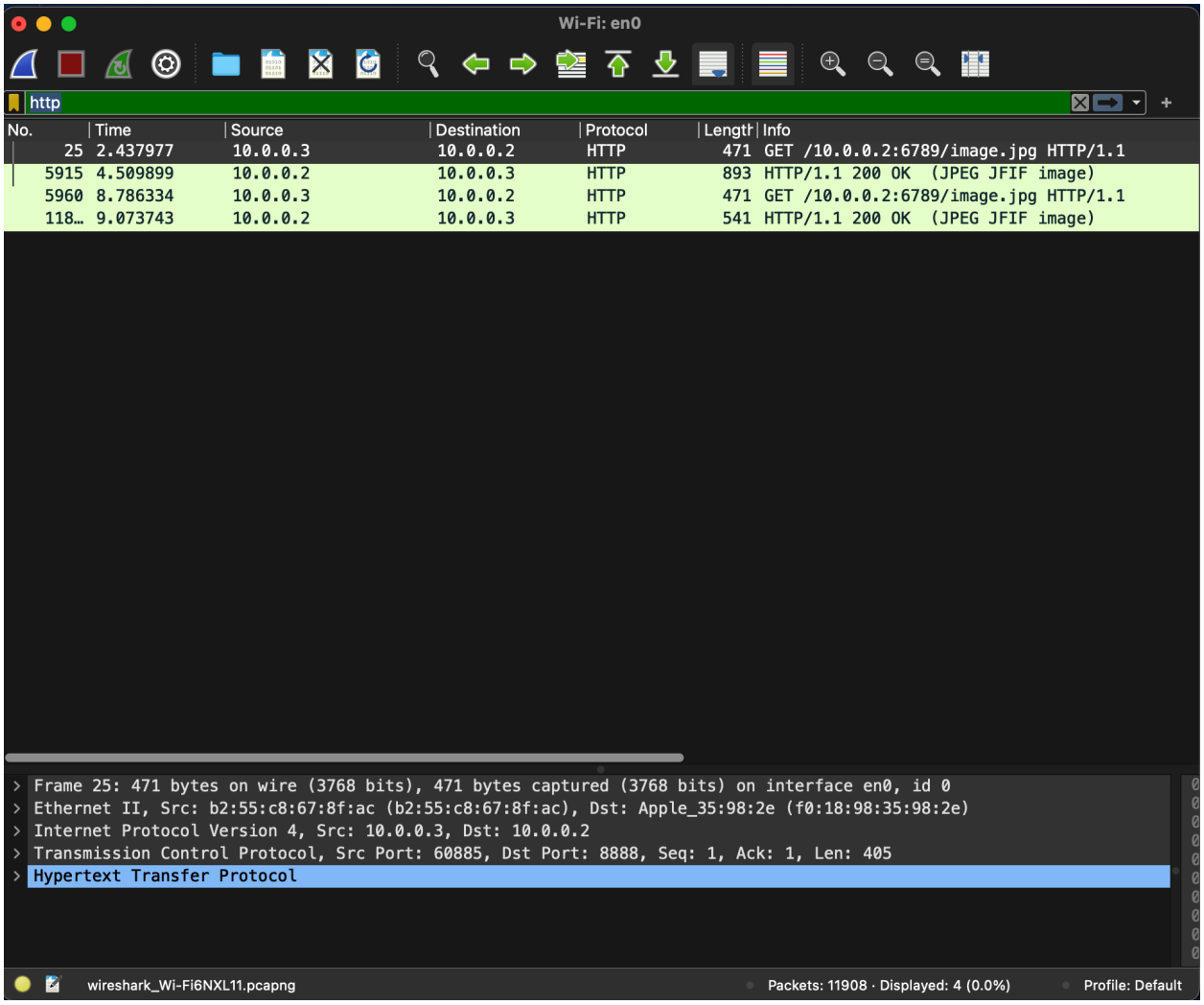
proxy.log

```
1 Ready to serve...
2
3 connection at ('127.0.0.1', 56086) sent message:
4 GET /localhost:6789/helloworld.html HTTP/1.1
5 Host: localhost:8888
6 User-Agent: curl/7.68.0
7 Accept: */*
8
9
10 forwarding to localhost:6789
11
12 Ready to serve...
13
14 connection at ('127.0.0.1', 34186) sent message:
15 GET /localhost:6789/image.jpg HTTP/1.1
16 Host: localhost:8888
17 User-Agent: curl/7.68.0
18 Accept: */*
19
20
21 forwarding to localhost:6789
22
23 Ready to serve...
24
```

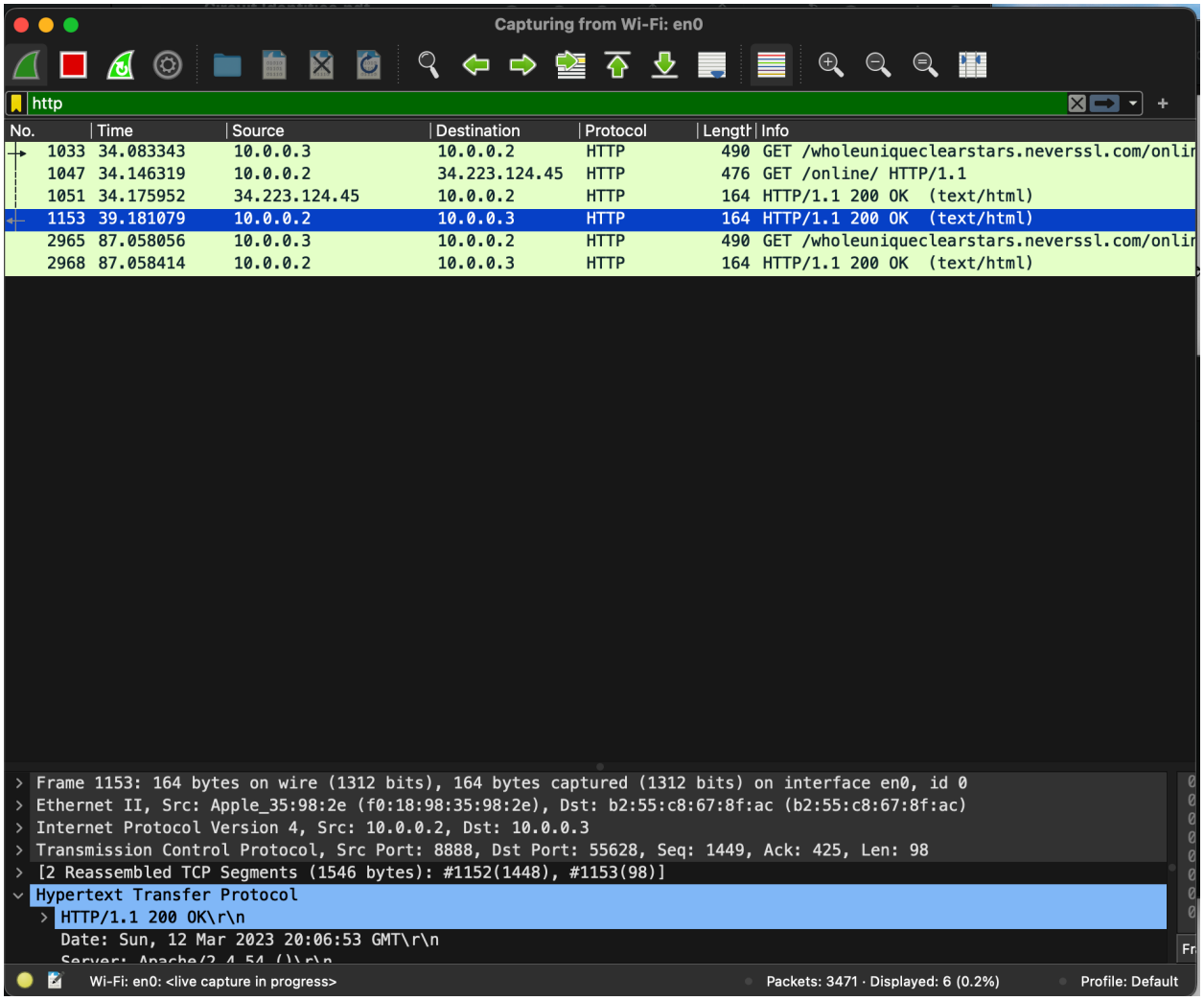
2.2.3. Task4: Local Environment Proxy Server Outputs



2.2.4. Task5: WireShark Output



2.2.4. Task6: External Web Server Output



3 Discussion

3.1 Processing URI

For the Proxy Server, we initially tried forwarding the request from the client right away to the server without any processing step, which resulted in 404 File Not Found error. The reason behind this problem was that the server tries to access the file system with a file path with the local host address attached. Thus we could successfully solve the problem by adding processing which removed the host address from the request uri.

3.3 Buffer Size Problem

When testing the Proxy Server with the given image, we faced Broken pipe error. The cause of the error was that for the recv() method, the buffer size we set was not enough for large messages. In other words, the server failed to send the message because the message size was much larger than the buffer size. To solve this problem, we used sendall() method which sends the data to the proxy server until everything has been sent, instead of send() method which sends less bytes than sendall(). We also changed how the proxy server receives the data using a while loop, so that the socket receives a chunk of data continuously until there is no more data received.

4 Appendix

4.1 Web Server Program Source Code

```
# Import socket module
from socket import *

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

serverSocket = socket(AF_INET, SOCK_STREAM)

serverSocket.bind(('', 6789))
serverSocket.listen(1)

# 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()

    # 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
        print(f"\nconnection at {addr} sent message:")
        print(message)

        # 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()

        # Send the HTTP response header line to the connection socket
        connectionSocket.sendall("HTTP/1.1 200 OK\r\n".encode())

        # required blank line
        connectionSocket.sendall("\r\n".encode())

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

        # Close the client connection socket
        connectionSocket.close()

    except IOError:
        # Send HTTP response message for file not found
        connectionSocket.sendall("HTTP/1.1 404 Not Found\r\nFile Not
Found".encode())
        connectionSocket.sendall("\r\n".encode())

        # Close the client connection socket
        connectionSocket.close()

serverSocket.close()
```

4.2 Web Proxy Server Program Source Code

```
# Import socket module
from socket import *

# Create a TCP server socket
#(AF_INET is used for IPv4 protocols)
#(SOCK_STREAM is used for TCP)
cache = {}
proxySocket = socket(AF_INET, SOCK_STREAM)

proxySocket.bind(('', 8888))
proxySocket.listen(1)

# 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 = proxySocket.accept()

    # 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()
        print(f"\nconnection at {addr} sent message:")
        print(message)
        response = b""

        # if in cache
        reqLine = message.split('\n')[0]
        if reqLine in cache:
            print("getting response from cache")
            response = cache[reqLine]
```

```
        # if not in cache
        else:
            #process request to forward
            request_uri = message.split()[1]
            request_uri_list = request_uri.split('/')
            # grab target server host and port
            targetServer = request_uri_list[1]
            # delete proxy host and port
            del request_uri_list[1]
            # make new uri for target server
            new_uri = '/'.join(request_uri_list)
            if new_uri == '':
                new_uri = '/'

            # change request line
            processed_request = message.replace(request_uri, new_uri)

            # change host line
            currHost = message.split('\n')[1].split()[1]
            processed_request = processed_request.replace(currHost,
targetServer)

            # get hostname and port from serverDetails
            targetServer = targetServer.split(':')
            hostname = targetServer[0]
            port = int(targetServer[1]) if len(targetServer) > 1 else 80
            print(f"forwarding to {hostname}:{port}\n")

            #forward request
            serverSocket = socket(AF_INET, SOCK_STREAM)
            serverSocket.connect((hostname, port))
            serverSocket.sendall(processed_request.encode())

            # get message from server and send to client
            chunkNum = 1
            while True:
                chunk = serverSocket.recv(4096)
                if len(chunk) == 0:     # No more data received, quitting
                    break
                response += chunk
                print(f"sent {chunkNum} chunks")
                chunkNum += 1
```

```
            # add to cache
            cache[reqLine] = response

        # send response to client
        connectionSocket.sendall(response)

        serverSocket.close()
        connectionSocket.close()

    except IOError:
        # Send HTTP response message for file not found
        print("\nerror occurred\n")
        connectionSocket.sendall("HTTP/1.1 404 Not Found\r\nFile Not
Found".encode())
        connectionSocket.sendall("\r\n".encode())

        # Close the client connection socket
        connectionSocket.close()

proxySocket.close()
```