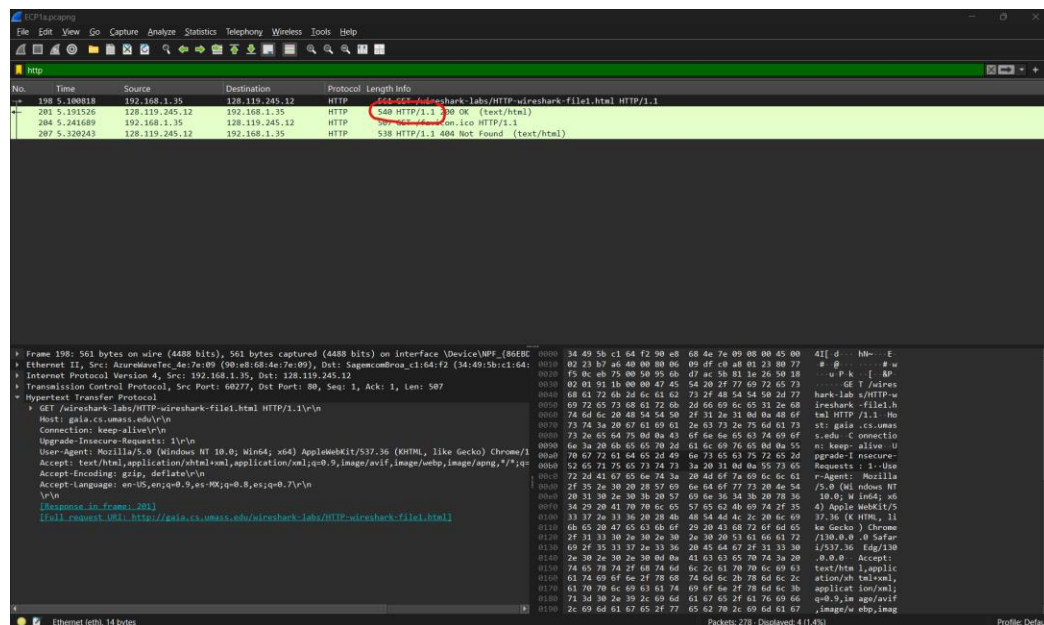


Part 1: Analyzing HTTP Messages using Wireshark

Part 1a – Basic HTTP GET/response interaction

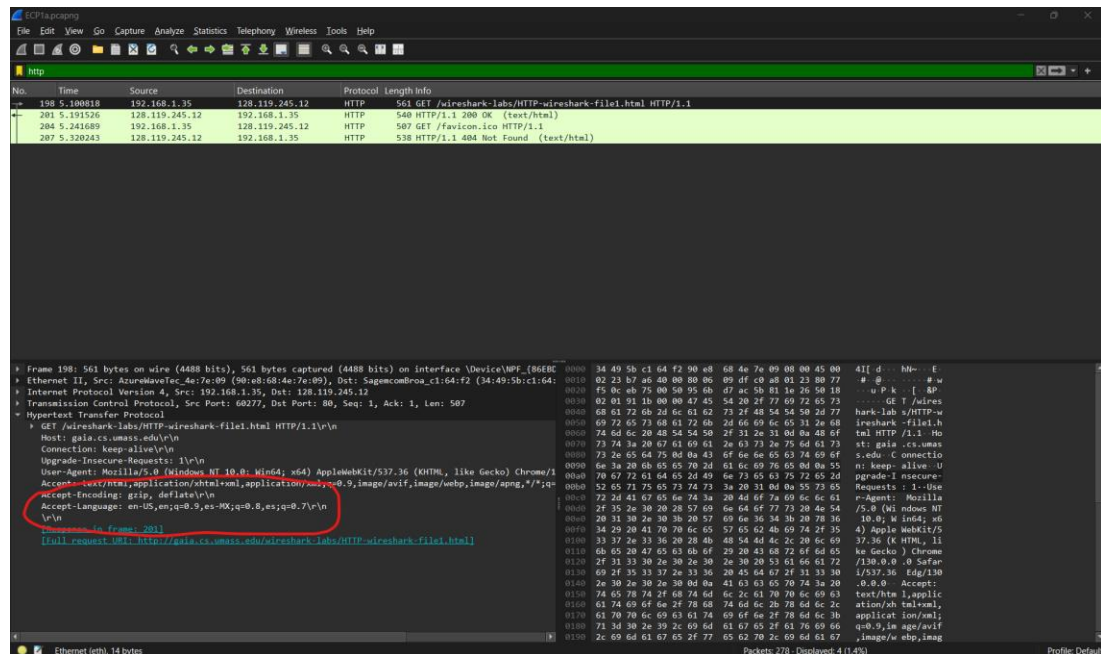
1. Is your browser running HTTP version 1.0 or 1.1? What version of HTTP is the server running?

My browser is running HTTP version 1.1



2. What languages (if any) does your browser indicate that it can accept to the server?

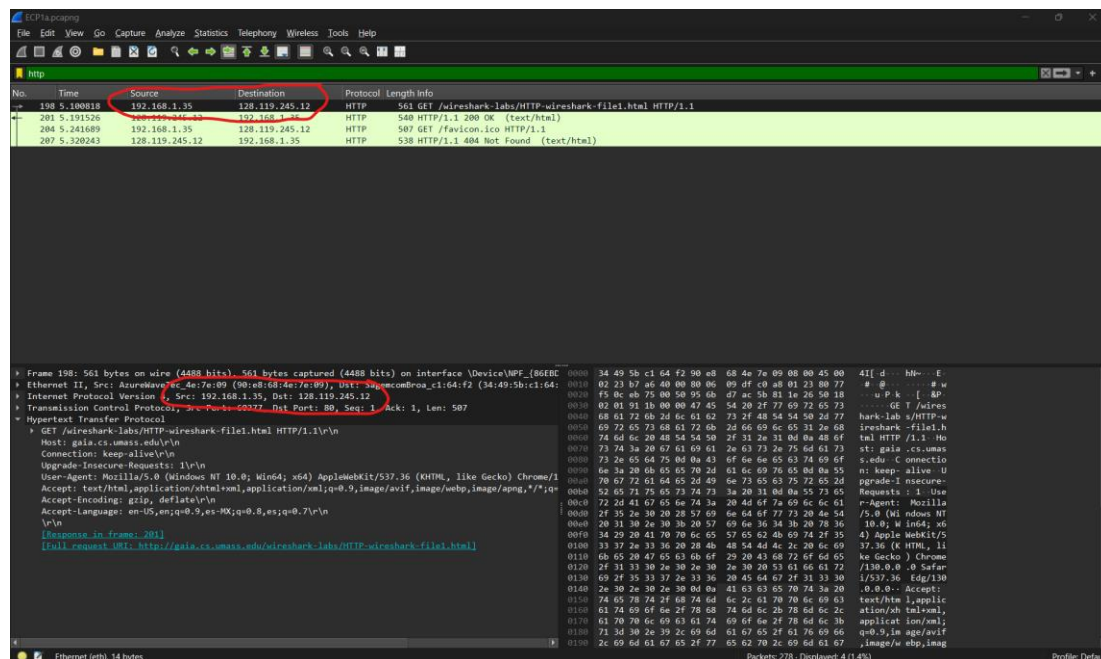
My browser indicates that it can accept English (US) and Spanish (MX).



3. What is the IP address of your computer? Of the gaia.cs.umass.edu server?

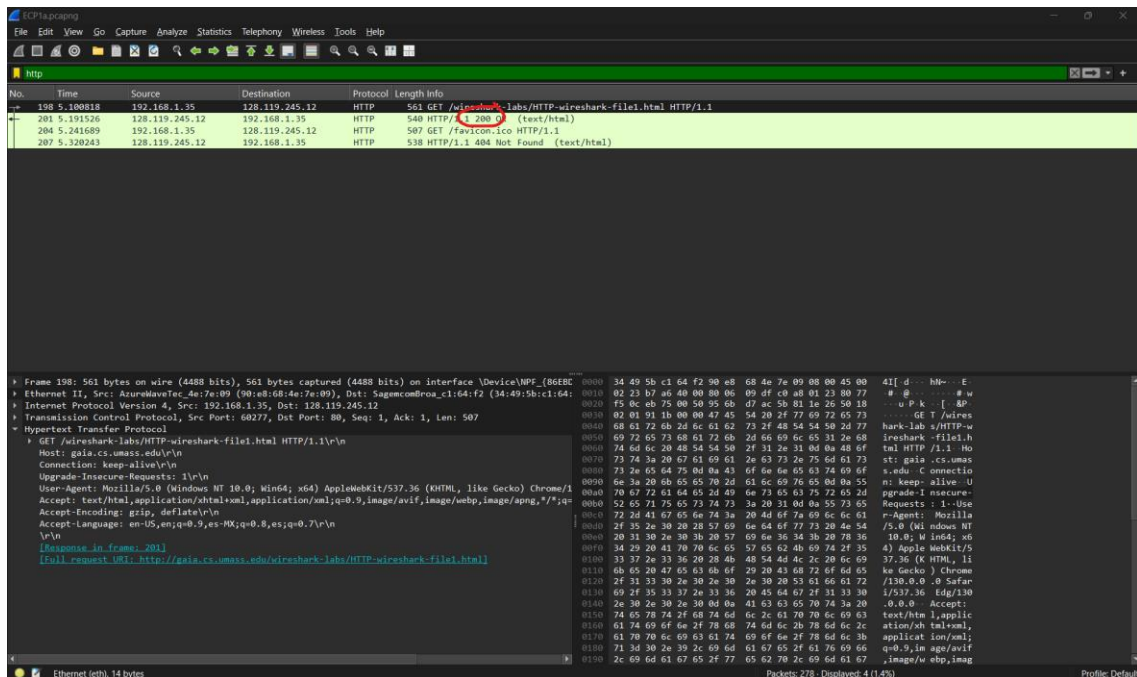
Computer IP address: 192.168.1.35

Gaia.cs.umass.edu IP address: 128.119.245.12



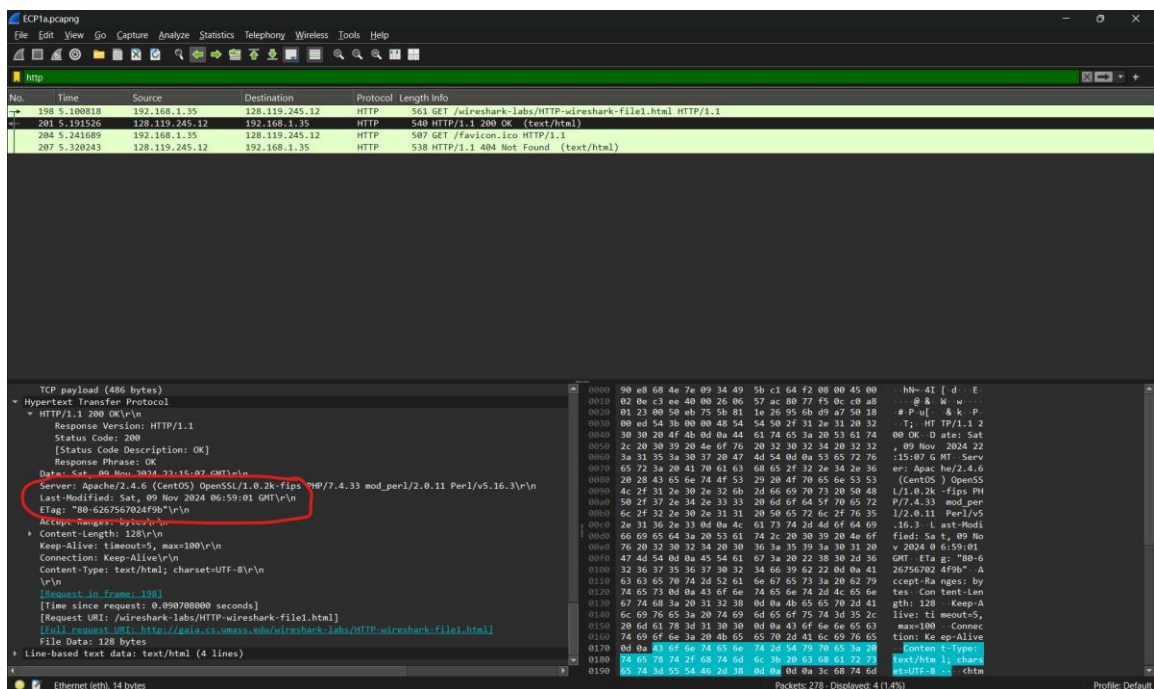
4. What is the status code returned from the server to your browser?

Status Code: 200



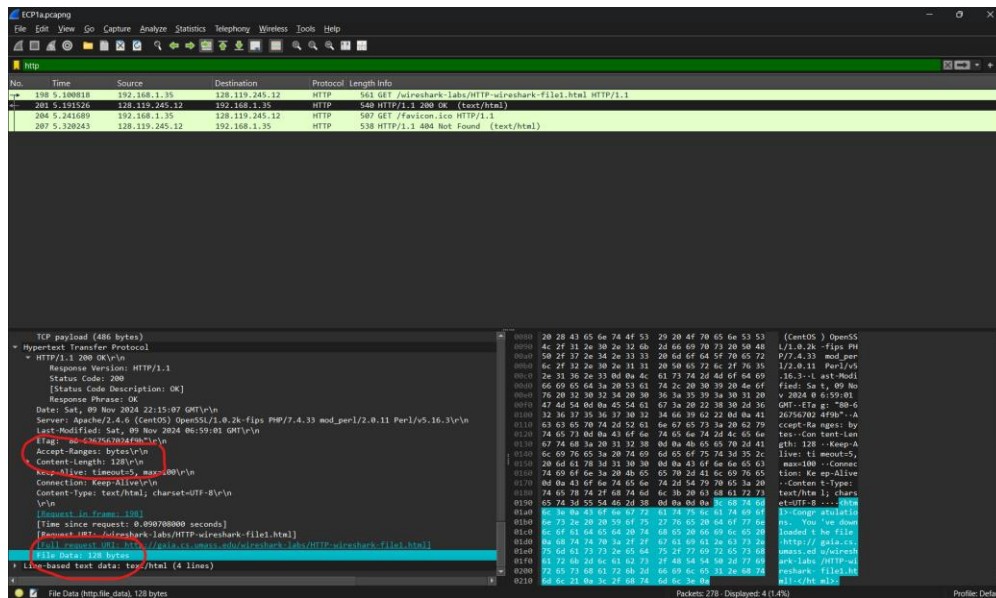
5. When was the HTML file that you are retrieving last modified at the server?

Last-Modified: Sat, 09 Nov 2024 06:59:01 GMT



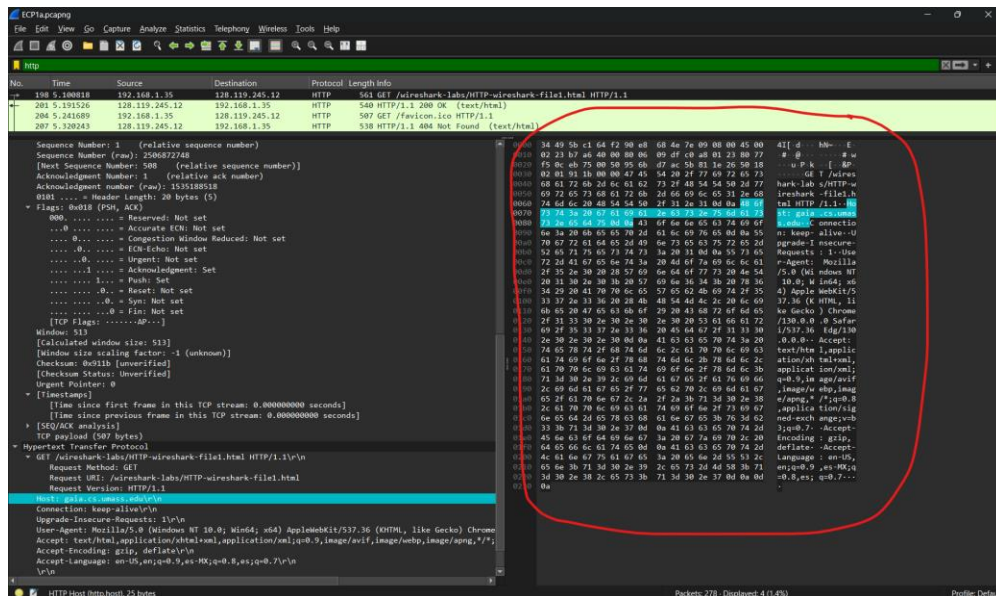
6. How many bytes of content are being returned to your browser?

128 bytes



7. By inspecting the raw data in the packet content window, do you see any headers within the data that are not displayed in the packet-listing window? If so, name one.

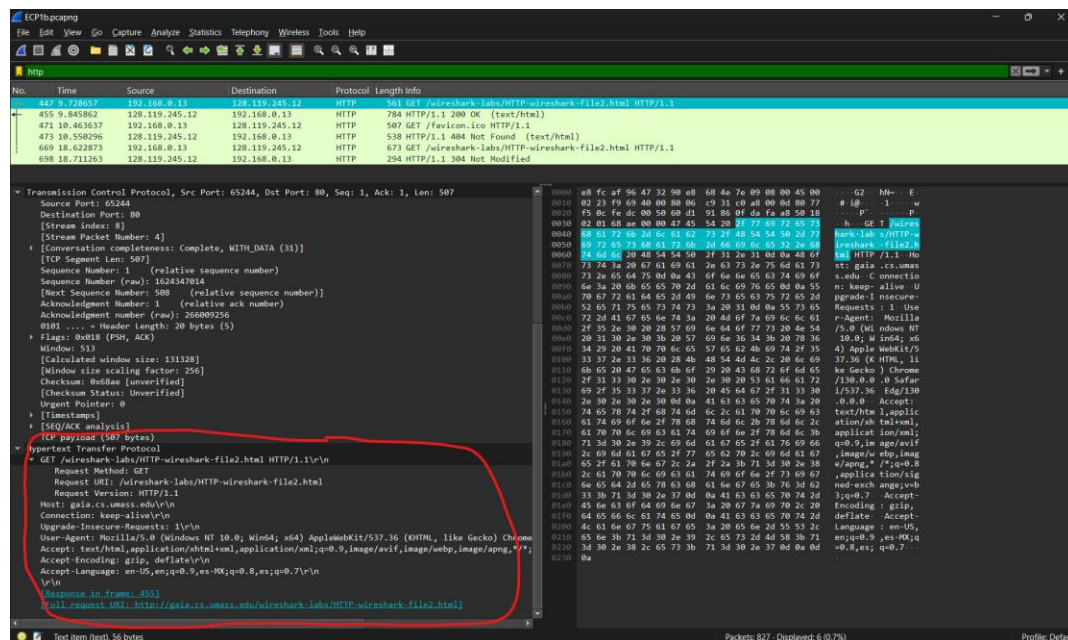
I do not see any headers within the data that are not displayed in the packet-listing window.



Part 1b: The HTTP CONDITIONAL GET/response interaction.

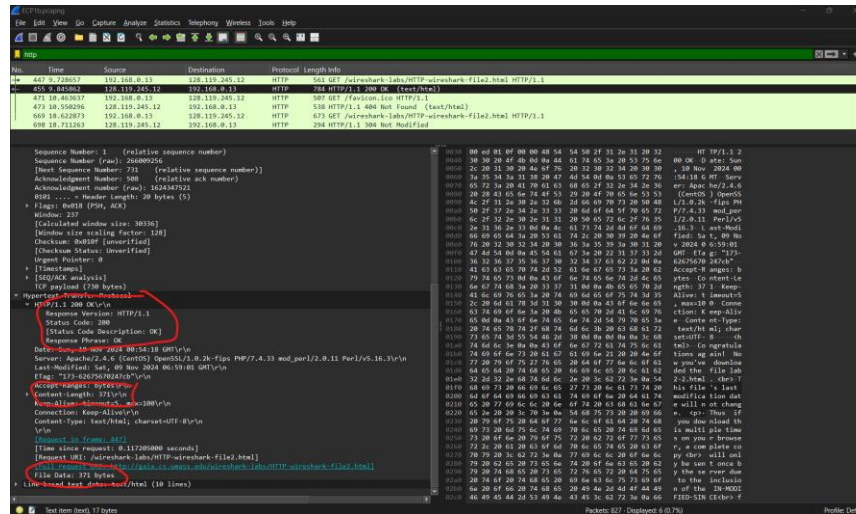
1. Inspect the contents of the first HTTP GET request from your browser to the server. Do you see an “IF-MODIFIED-SINCE” line in the HTTP GET?

I do not see any such line, it should have been underneath Hypertext Transfer Protocol



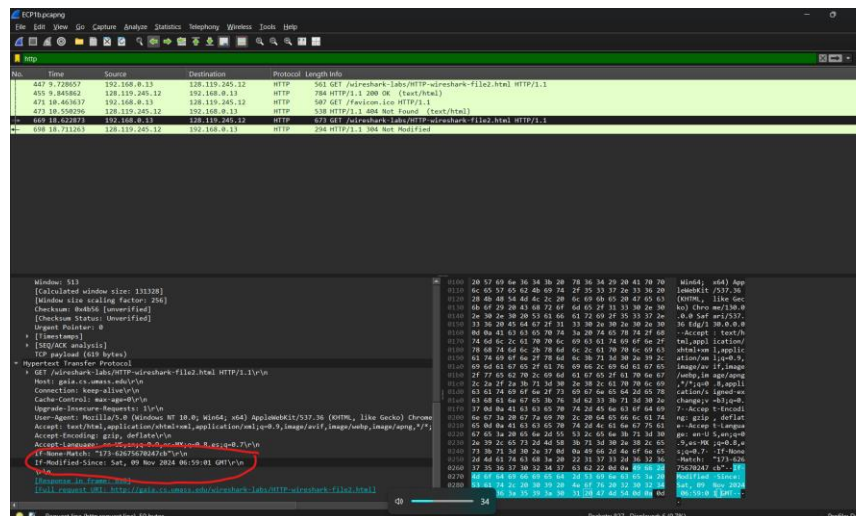
2. Inspect the contents of the server response. Did the server explicitly return the contents of the file? How can you tell?

The Status Code was 200 and the Response Phrase was OK, meaning that the server explicitly returned the contents of the file. Additionally, there Content Length and File Data are present. Indicating a data was transferred.



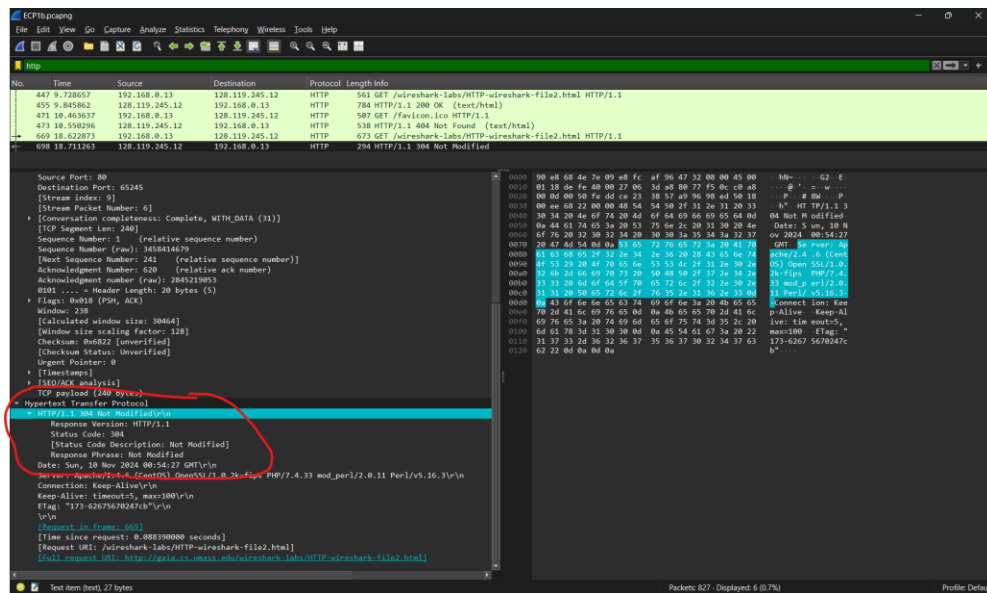
3. Now inspect the contents of the second HTTP GET request from your browser to the server. Do you see an “IF-MODIFIED-SINCE:” line in the HTTP GET? If so, what information follows the “IF-MODIFIED-SINCE:” header?

Yes. The information that follows is: Sat, 09 Nov 2024 06:59:01 GMT\r\n



4. What is the HTTP status code and phrase returned from the server in response to this second HTTP GET? Did the server explicitly return the contents of the file? Explain.

It did not return the contents of the file. Underneath Hypertext Transfer Protocol > HTTP/1.1 304 Not Modified\r\n the Response Phrase is Not Modified. When looked up, HTTP 304 Not Modified signifies that the resource cached by the client is still valid.



Part 1c: Retrieving Long Documents

1. How many HTTP GET request messages did your browser send? Which packet number in the trace contains the GET message for the Bill of Rights?

Since we are ignoring anything to do with favicon, then my browser sent 1 HTTP GET request message. Packet 140 contains the GET message for the Bill of Rights.

The image shows a Wireshark network traffic capture. The top pane displays a list of captured packets. Packet 140 is highlighted, showing an HTTP GET request for 'wireshark-file3.html' from source 192.168.0.13 to destination 192.168.0.1. The packet details pane on the left shows the structure of the packet, including the Ethernet II header, Internet Protocol Version 4 header, and Hypertext Transfer Protocol header. The packet bytes pane on the right shows the raw data of the packet, including the HTTP request line 'GET /wireshark-labs/HTTP-wireshark-file3.html HTTP/1.1'.

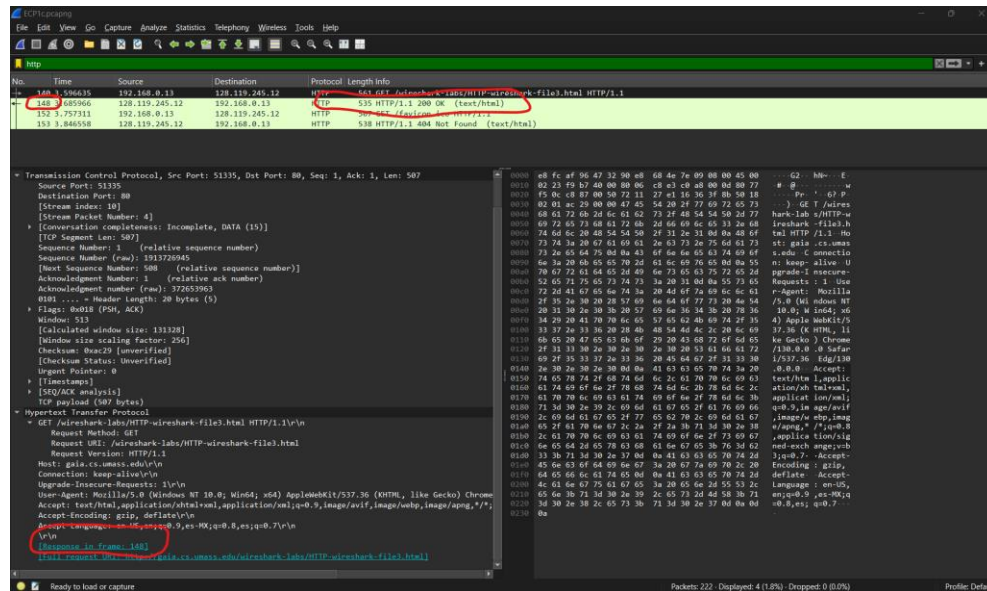
No.	Time	Source	Destination	Protocol	Length	Info
140	3.757311	192.168.0.13	192.168.0.1	HTTP	561	GET /wireshark-labs/HTTP-wireshark-file3.html HTTP/1.1

Transmission Control Protocol, Src Port: 51335, Dst Port: 80, Seq: 1, Ack: 1, Len: 507

- Source Port: 51335
- Destination Port: 80
- Stream index: 10
- [Stream Packet Number: 4]
- [Conversation completeness: Incomplete, DATA (15)]
- [TCP Segment len: 507]
- Sequence Number: 1 (relative sequence number)
- Sequence Number (raw): 1913726945
- Next Sequence Number: 1008 (relative sequence number)
- Acknowledgment Number: 1 (relative ack number)
- Acknowledgment Number (raw): 372653963
- 0000... = Header length: 20 bytes (5)
- Flags: 0x018 (PSH, ACK)
- Window: 511
- [Calculated window size: 131328]
- [Window size scaling factor: 256]
- Checksum: 0xac29 [unverified]
- [Checksum Status: Unverified]
- Urgent Pointer: 0
- [Timestamps]
- [SQ/ACK analysis]
- TCP payload (507 bytes)
- Hypertext Transfer Protocol
 - Request Method: GET
 - Request URI: /wireshark-labs/HTTP-wireshark-file3.html
 - Request Version: HTTP/1.1
 - Host: gaia.cs.umass.edu/vn
 - Connection: keep-alive/vn
 - Upgrade-Insecure-Requests: 1/vn
 - User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/118.0.0.0 Safari/118.0.0.0
 - Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;Accept-Encoding: gzip, deflate/vn
 - Accept-Language: en-US,en;q=0.9,es-ES;q=0.8,es;q=0.7/vn
 - [Response in frame: 140]
 - [Full request URI: http://gaia.cs.umass.edu/wireshark-labs/HTTP-wireshark-file3.html]

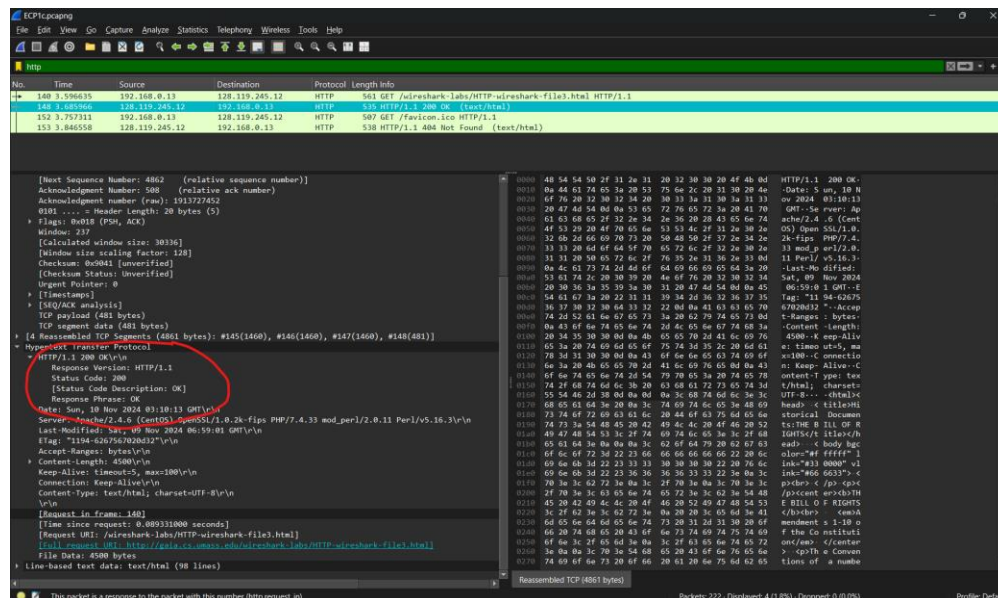
2. Which packet number in the trace contains the status code and phrase associated with the response to the HTTP GET request?

Packet 148 contains the status code and phrase associated with the response to the HTTP GET request.



3. What is the status code and phrase in the response?

The status code is 200 and the phrase is OK in the response.



4. How many data-containing TCP segments were needed to carry the single HTTP response and the text of the Bill of Rights?

Four data-containing TCP segments were needed to carry the single HTTP response and the text of the Bill of Rights. Those being packets 145, 146, 147, and 148.

The image shows a Wireshark packet capture of an HTTP response. The packet list at the top shows four data-containing TCP segments (packets 145, 146, 147, and 148) which are reassembled into a single 4861-byte HTTP response. The packet details pane shows the reassembled TCP segment and the HTTP response structure, including the status code 200 OK and the content type text/html. The packet bytes pane shows the raw data of the reassembled TCP segment.

Packet List:

No.	Time	Source	Destination	Protocol	Length	Info
145	3.596635	192.168.0.13	128.119.245.12	HTTP	561	GET /wiresark-labs/HTTP-wiresark-file3.html HTTP/1.1
146	3.685966	128.119.245.12	192.168.0.13	HTTP	535	HTTP/1.1 200 OK (text/html)
152	3.757311	192.168.0.13	128.119.245.12	HTTP	507	GET /favicon.ico HTTP/1.1
153	3.846558	128.119.245.12	192.168.0.13	HTTP	538	HTTP/1.1 404 Not Found (text/html)

Packet Details (Packet 146):

- [Window size scaling factor: 128]
- Checksum: 0x0001 [unverified]
- [Checksum Status: Unverified]
- Urgent Pointer: 0
- [Timestamps]
- [SQ/ACK analysis]
- TCP segment data (481 bytes)
- [4 Reassembled TCP Segments (4861 bytes): #145(1460), #146(1460), #147(1460), #148(481)]
- [frame=145, payload: 0-1459 (1460 bytes)]
- [frame=146, payload: 1460-2919 (1460 bytes)]
- [frame=147, payload: 2920-4379 (1460 bytes)]
- [frame=148, payload: 4380-4860 (481 bytes)]
- [Segment count: 4]
- [Reassembled TCP length: 4861]
- [Reassembled TCP Data [..]: 485454582f312e31303230304640b0d446174653a2053756e2c2031303204-f762032
- HyperText Transfer Protocol
- HTTP/1.1 200 OK
- Response Version: HTTP/1.1
- Status Code: 200
- [Status Code Description: OK]
- Response Phrase: OK
- Date: Sun, 10 Nov 2024 03:10:13 GMT\r\n
- Server: Apache/2.4.6 (CentOS) OpenSSL/1.0.2k-fips PHP/7.4.33 mod_perl/2.0.11 Perl/v5.16.3\r\n
- Last-Modified: Sat, 09 Nov 2024 06:59:01 GMT\r\n
- Tag: 1124-c20240700012\r\n
- Accept-Ranges: bytes\r\n
- Content-Length: 4500\r\n
- Keep-Alive: timeout=5, max=100\r\n
- Connection: Keep-Alive\r\n
- Content-Type: text/html; charset=UTF-8\r\n
- \r\n
- Request in frame: 140
- [Time since request: 0.009331000 seconds]
- [Request URI: /wiresark-labs/HTTP-wiresark-file3.html]
- [Full request URI: http://192.168.0.13:8080/wiresark-labs/HTTP-wiresark-file3.html]
- File Data: 4500 bytes
- Line-based text data: text/html (98 lines)

Packet Bytes (Packet 146):

0000 48 54 58 2f 31 2e 31 20 32 30 30 20 4f 4b 0d HTTP/1.1 200 OK

0010 0a 4c 61 74 65 3a 20 53 75 6e 2c 20 31 30 3a 31 33 Date: Sun, 10 Nov 2024 03:10:13

0020 6f 76 20 32 30 32 3a 20 30 33 3a 31 30 3a 31 33 GMT, Server: Ap

0030 20 47 4d 54 0d 0a 53 65 72 76 65 72 3a 20 41 70 GMT, Server: Ap

0040 61 63 68 05 2f 32 2e 3a 26 36 20 28 43 65 6a 74 aache/2.4.6 (Cent

0050 4f 53 29 20 4f 70 65 6a 53 53 4c 2f 31 2a 30 2e OS) Open SSL/1.0

0060 32 6b 2d 66 69 70 73 20 50 48 50 2f 37 2e 3a 2e 2k-fips PHP/7.4.

0070 33 33 20 6d 6f 64 5f 70 65 72 6c 2f 32 2e 30 2e 33 mod_perl/2.0.

0080 31 31 20 50 65 72 6c 2f 76 35 2e 31 36 2e 33 0d 11 Perl/v5.16.3

0090 0a 4c 61 74 65 3a 20 53 75 6e 2c 20 31 30 3a 31 33 Last-Modified:

00a0 53 61 74 2c 20 38 39 20 4a 6f 76 20 32 30 32 3a Sat, 09 Nov 2024

00b0 20 38 36 3a 35 39 3a 30 31 20 47 4d 54 0d 0a 45 06:59:01 GMT, E

00c0 54 61 67 3a 20 22 31 31 39 3a 2d 36 32 36 37 35 Tag: "11 94-62675

00d0 36 37 30 33 30 64 33 32 22 0a 0a 41 63 63 65 70 C/02652" -> <accept

00e0 74 2d 52 61 6e 67 65 73 3a 20 62 79 74 65 73 0d t-Ranges: bytes

00f0 0a 43 6f 6a 74 65 6a 74 2d 4c 65 6e 67 74 68 3a Content-Length:

0100 20 35 35 30 30 0d 0a 30 68 65 70 2d 43 6c 68 06 4500 Range: 0-4500

0110 65 3a 28 74 69 6d 65 65 75 74 3d 35 2c 20 6d 65 5; time-out=5, ma

0120 76 3a 31 30 30 0d 0a 43 6f 6e 66 65 63 74 69 6f n; Keep-Alive: C

0130 6a 3a 2d 4b 65 65 70 2d 41 6c 69 76 65 0d 0a 43 0; Keep-Alive: C

0140 6f 6e 74 65 6e 74 2d 54 79 70 65 3a 20 74 65 78 content-type: tex

0150 74 2f 68 74 6d 6c 30 20 63 68 63 72 73 65 74 3d t/html; charact-

0160 55 54 4d 2d 38 0d 0a 0a 3c 68 74 6d 6c 3a 3c UTF-8 -> <html>

0170 68 65 61 64 3e 20 0a 3c 74 69 74 6c 65 3a 48 69 head> < title>Hi

0180 73 74 6f 72 69 63 63 6c 20 4d 6f 63 79 6d 65 6e stical Docum

0190 74 73 3a 54 48 65 20 42 49 4c 20 4f 46 20 52 t:THE \$ ILL OF R

01a0 49 47 48 54 53 3c 2f 74 69 74 6c 65 3a 3c 2f 68 IGH5</ title></h

01b0 65 61 64 3e 0a 0a 3c 62 6f 6a 79 20 62 67 63 ead> < body >

01c0 6f 6c 6f 72 3d 22 23 66 66 66 66 66 22 20 6c color="# ffffff" l

01d0 69 6e 6b 3d 22 23 33 30 30 30 30 22 20 76 6c /p><er> </p> <p>

01e0 69 6e 6b 3d 22 23 36 36 36 33 33 22 2e 0a 3c ink="#33 0000" v l

01f0 70 3e 3c 62 72 3a 0a 3c 2f 70 3e 0a 3c 70 3e 3c p><er> </p> <p>

0200 2f 70 3e 3c 63 65 6e 74 65 72 3e 3c 62 3e 54 48 /p><er> </p> <p>

0210 45 20 42 49 4c 20 4f 46 20 52 49 47 48 54 53 E STILL O F RIGHTS

0220 3c 2f 62 3a 3c 62 72 3a 0a 20 20 3c 65 6d 3a 41 </body> a

0230 6d 65 6a 6d 6d 65 6e 74 73 20 31 2d 31 30 20 6f monment s 1-10 o

0240 66 20 74 6d 65 20 43 6f 6e 73 74 69 74 75 74 69 f the Co nstituti

0250 6f 6e 3c 2f 65 6d 3a 0a 3c 2f 63 65 6a 74 65 72 on/emo </center

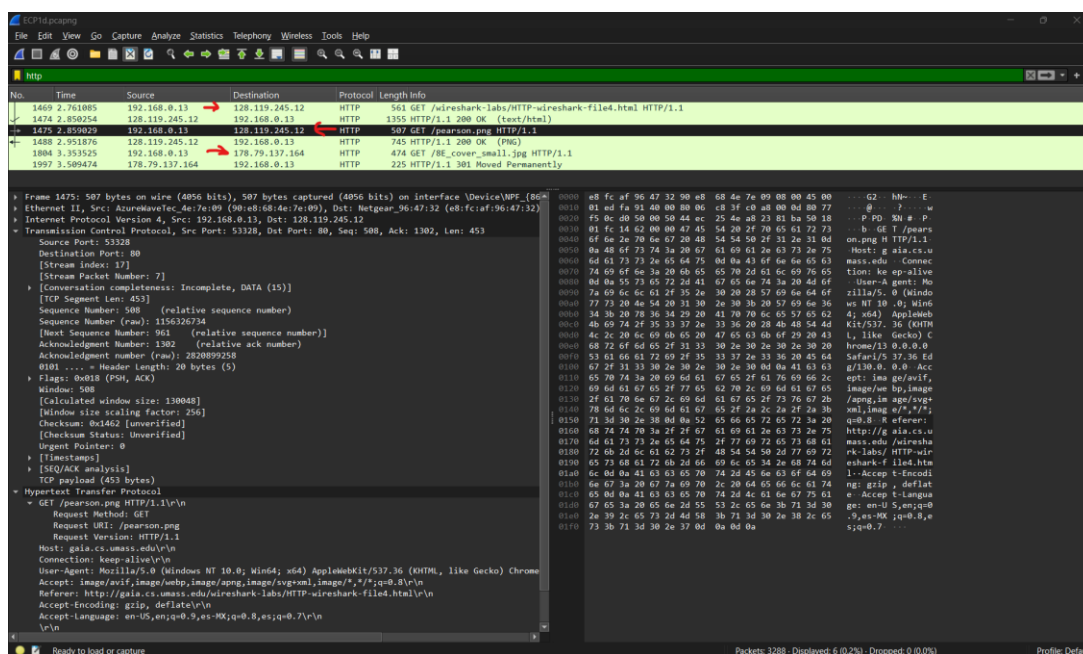
0260 3e 0a 0a 3c 70 3e 54 68 65 20 43 6f 6e 76 65 6e > <p>th e Conven

0270 74 69 6f 6e 73 20 6f 20 63 20 6e 73 6d 62 65 tions of a numbe

Part 1d: HTML Documents with Embedded Objects

1. How many HTTP GET request messages did your browser send? To which Internet addresses were these GET requests sent?

Three HTTP GET request messages were sent by my browser. The first two were sent to the address 128.119.245.12 and the last one was sent to 178.79.137.164



2. Can you tell whether your browser downloaded the two images serially, or whether they were downloaded from the two web sites in parallel? Explain.

The browser downloaded the two images serially. You can tell because there is a gap in the timestamps of packets 1488 and 1997 which are the responses to the two latter GET request messages. Additionally, the second image required a redirection and would have downloaded only after the first image.

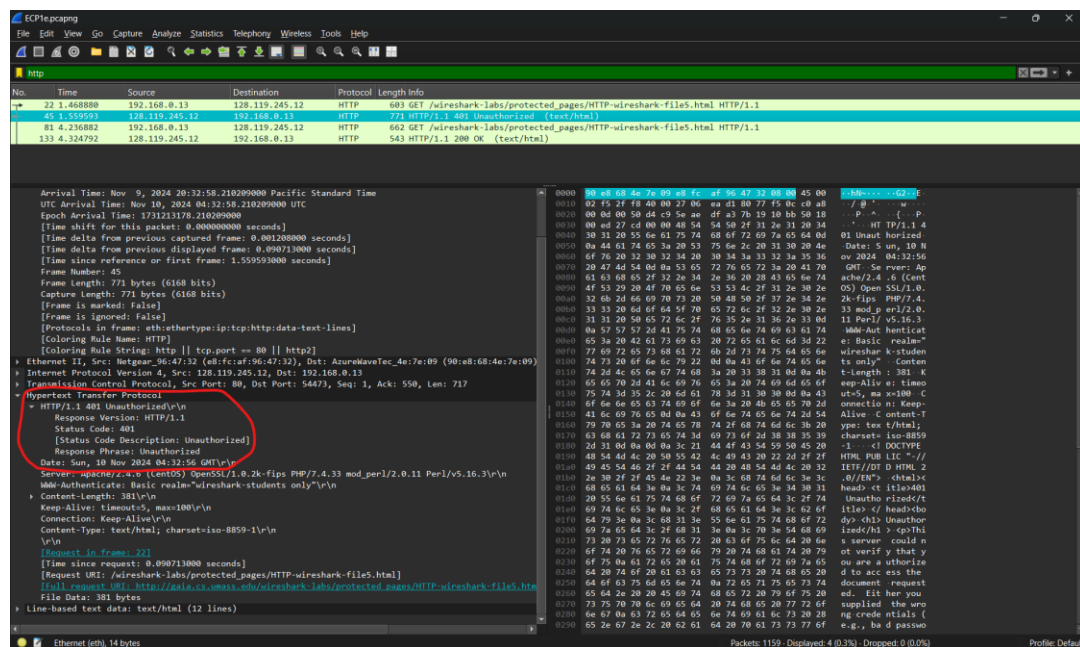
The image shows a Wireshark packet capture of an HTTP session. The packet list pane at the top shows several HTTP GET requests and responses. Packet 1997 is highlighted with a red circle. The packet details pane for packet 1997 shows the following information:

- Frame 1997: 225 bytes on wire (1800 bits), 225 bytes captured (1800 bits) on interface \Device\NPF_{86E8D1A7-58F9-472B-AE30-F8821BA56EB7}
- Interface id: 0 (\Device\NPF_{86E8D1A7-58F9-472B-AE30-F8821BA56EB7})
- Encapsulation type: Ethernet (1)
- Arrival Time: Nov 9, 2024 20:04:12.568474000 Pacific Standard Time
- UTC Arrival Time: Nov 10, 2024 04:04:12.568474000 UTC
- Epoch Arrival Time: 1731211432.568474000
- Time shift for this packet: 0.000000000 seconds
- [Time delta from previous captured frame: 0.000050000 seconds]
- [Time delta from previous displayed frame: 0.155045000 seconds]
- [Time since reference or first frame: 3.509474000 seconds]
- Frame Number: 1997
- Frame Length: 225 bytes (1800 bits)
- Capture Length: 225 bytes (1800 bits)
- [Frame is marked: False]
- [Frame is ignored: False]
- Protocols in frame: ethertype:ip:tcp:http
- [Coloring Rule Name: HTTP]
- [Coloring Rule String: http || tcp.port == 80 || http2]
- Ethernet II, Src: Netgear-96:47:32 (e8:fc:af:96:47:32), Dst: AzureWaveTec-4e:7e:09 (90:e8:68:4e:7e:09)
- Internet Protocol Version 4, Src: 178.79.137.164, Dst: 192.168.0.13
- Transmission Control Protocol, Seq: 63332, Seq: 1, Ack: 421, Len: 171
- Hypertext Transfer Protocol
- HTTP/1.1 301 Moved Permanently
- Response Version: HTTP/1.1
- Status Code: 301
- [Status Code Description: Moved Permanently]
- Response Phrase: Moved Permanently
- Location: https://kurose.cslash.net/BE_cover_small.jpg
- Content-Length: 0
- Date: Sun, 10 Nov 2024 04:04:11 GMT
- Server: Lighttpd/1.4.47
- [Request in frame: 1804]
- [Time since request: 0.155045000 seconds]
- [Request URI: /BE_cover_small.jpg]
- [Full request URI: http://kurose.cslash.net/BE_cover_small.jpg]

Part 1e: HTTP Authentication

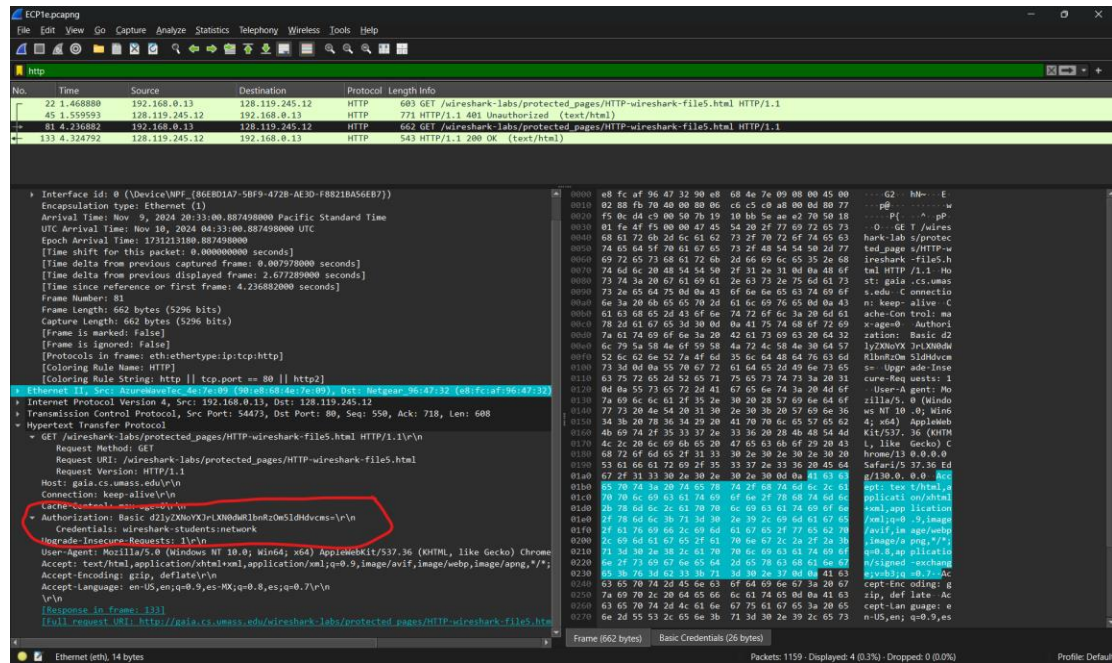
1. What is the server's response (status code and phrase) in response to the initial HTTP GET message from your browser?

The server's response to the initial HTTP GET message from my browser was 401 Unauthorized



2. When your browser's sends the HTTP GET message for the second time, what new field is included in the HTTP GET message?

An Authorization field with Credentials pops up in the second HTTP GET message.



Part 2: Socket Programming

Part 2a: UDP Pinger with No Delay and No Loss

1. Describe the operation of your UDP Pinger, for example how it works.

The UDP Pinger sends a certain number of pings to the server. In this case `totalPings = 10`. Each ping has its own unique response message consisting of date time and RTT. To calculate the RTT, a timestamp is recorded every time a ping is sent. A timestamp is recorded again when a response is received. The difference between both timestamps is calculated to find the RTT. In the case that the server does not respond, there is a set timeout value to handle exceptions where the response is delayed or lost. Therefore, if the response is not received within the timeout period, the client assumes it was lost or delayed and the `packetLossCt` variable increases by one. Additionally, the UDP Pinger calculates the RTT for every ping and adds it to an array that is then used to find the min, max, and avg RTT. As well as the packet loss rate using the previously mentioned `packetLossCt` variable.

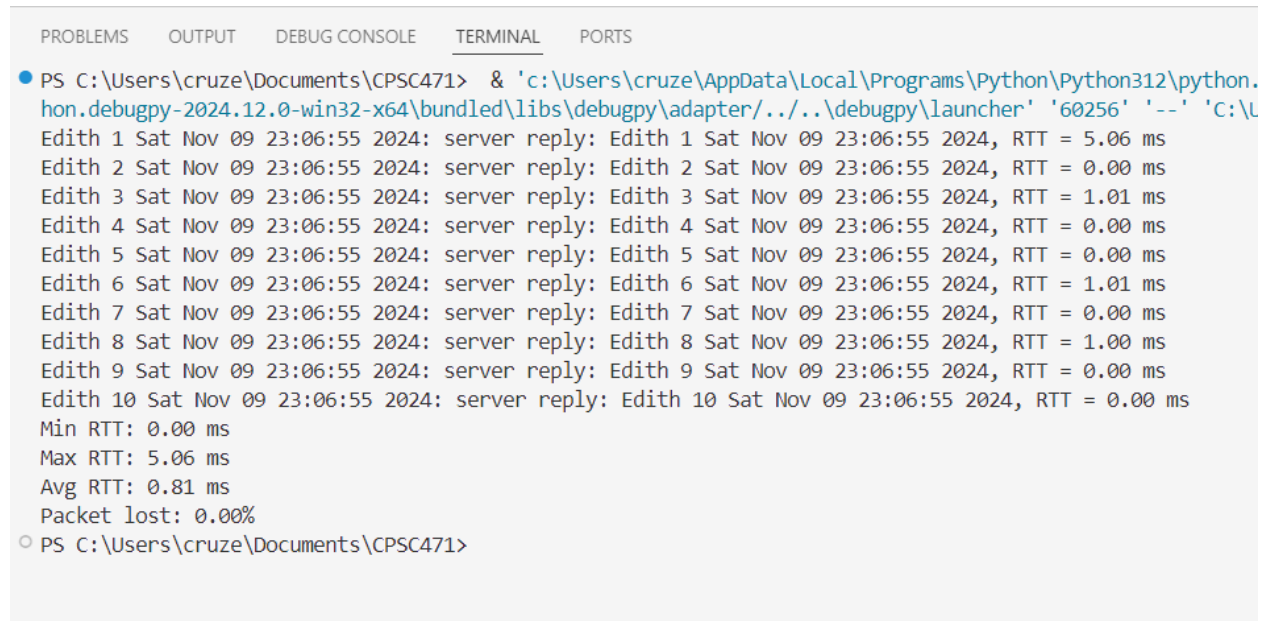
2. Explain how to specify the timeout value for a datagram socket. Provide an example.

```
yourSocket.setTimeout(yourTimeoutValHere)
```

You use the `setTimeout` function on your socket with the desired timeout value in seconds.

3. Explain how to run your code, i.e., command line and any applicable parameter(s)

I used VSCode therefore you can likewise launch the code on VSCode. You must first run `udppingserver_no_loss.py` and then run `ECP2a.py`. The IDE will prompt you for permission to start another instance, to which you respond Yes.



```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS
● PS C:\Users\cruze\Documents\CPSC471> & 'c:\Users\cruze\AppData\Local\Programs\Python\Python312\python.hon.debugpy-2024.12.0-win32-x64\bundled\libs\debugpy\adapter\..\..\debugpy\launcher' '60256' '--' 'C:\L
Edith 1 Sat Nov 09 23:06:55 2024: server reply: Edith 1 Sat Nov 09 23:06:55 2024, RTT = 5.06 ms
Edith 2 Sat Nov 09 23:06:55 2024: server reply: Edith 2 Sat Nov 09 23:06:55 2024, RTT = 0.00 ms
Edith 3 Sat Nov 09 23:06:55 2024: server reply: Edith 3 Sat Nov 09 23:06:55 2024, RTT = 1.01 ms
Edith 4 Sat Nov 09 23:06:55 2024: server reply: Edith 4 Sat Nov 09 23:06:55 2024, RTT = 0.00 ms
Edith 5 Sat Nov 09 23:06:55 2024: server reply: Edith 5 Sat Nov 09 23:06:55 2024, RTT = 0.00 ms
Edith 6 Sat Nov 09 23:06:55 2024: server reply: Edith 6 Sat Nov 09 23:06:55 2024, RTT = 1.01 ms
Edith 7 Sat Nov 09 23:06:55 2024: server reply: Edith 7 Sat Nov 09 23:06:55 2024, RTT = 0.00 ms
Edith 8 Sat Nov 09 23:06:55 2024: server reply: Edith 8 Sat Nov 09 23:06:55 2024, RTT = 1.00 ms
Edith 9 Sat Nov 09 23:06:55 2024: server reply: Edith 9 Sat Nov 09 23:06:55 2024, RTT = 0.00 ms
Edith 10 Sat Nov 09 23:06:55 2024: server reply: Edith 10 Sat Nov 09 23:06:55 2024, RTT = 0.00 ms
Min RTT: 0.00 ms
Max RTT: 5.06 ms
Avg RTT: 0.81 ms
Packet lost: 0.00%
○ PS C:\Users\cruze\Documents\CPSC471>
```

4. Paste the Python client code listing text

```
5. # from socket import * didnt work here
6. import socket
7. # needed for time
8. import time
9. # needed for date time in response message
10. from datetime import datetime
11.
12. # Server details
13. serverAddress = ('localhost', 12000) # Server IP and port
14. totalPings = 10 # Total number of pings to send, change
    this num to 50 for ECP2c
15. bufferSize = 1024 # Size of buffer for receiving messages
16.
17. # Initialize statistics
18. rttVals = []
19. packetLossCt = 0
20.
21. # My name used in pings
22. myName = "Edith"
23.
24. # Create UDP socket
25. clientSocket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
26. clientSocket.settimeout(1) # Setting timeout to 1s
27.
28. for pingNum in range(1, totalPings + 1):
29.     # format it
30.     # %a = abb. weekday name; %b = abb. month name
31.     timeStamp = datetime.now().strftime("%a %b %d %H:%M:%S %Y")
32.     message = f"{myName} {pingNum} {timeStamp}"
33.
34.     try:
35.         # Record the send time
36.         timeStart = time.time()
37.
38.         # Send the message to the server
39.         clientSocket.sendto(message.encode(), serverAddress)
40.
41.         # Wait for response from server
42.         # Recall to run udppingserver_no_loss.py first!
43.         # Or you'll get an error
44.         response, _ = clientSocket.recvfrom(bufferSize)
45.
46.         # Record the end time and calculate RTT
```

```

47.         endTime = time.time()
48.         rtt = (endTime - timeStart) * 1000 # Converting RTT to milliseconds
49.         rttVals.append(rtt)
50.
51.         # Print server's response and RTT
52.         print(f"{message}: server reply: {response.decode()}, RTT = {rtt:.2f}
    ms")
53.
54.     except socket.timeout:
55.         # Handle case where the request timed out
56.         print(f"{myName} {pingNum}: timed out, message was lost")
57.         packetLossCt += 1
58.
59. # MATH YAY
60. if rttVals:
61.     minVal = min(rttVals)
62.     maxVal = max(rttVals)
63.     avgVal = sum(rttVals) / len(rttVals)
64. else:
65.     minVal = maxVal = avgVal = 0.0
66.
67. # equation taken from Labs
68. packetLossRt = (packetLossCt / totalPings) * 100
69.
70. # Check online for proper formatting of integers with decimals and percentages
71. # Ping Summary
72. print(f"Min RTT: {minVal:.2f} ms")
73. print(f"Max RTT: {maxVal:.2f} ms")
74. print(f"Avg RTT: {avgVal:.2f} ms")
75. print(f"Packet lost: {packetLossRt:.2f}%")
76.

```


Part 2b – UDP Pinger No Loss, with Delays

1. Describe the operation of your UDP Ping Server and explain how it simulates 10ms to 20ms RTT delays.

To introduce a delay, the sleep function was used. When the server receives a ping message from the client, there is a random RTT delay between 10 and 20 ms.

2. Explain how to run your code, i.e., command line and any applicable parameter(s)

Similarly to the previous problem we use VSCode. Please run ECP2b.py first and then ECP2a.py.

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

Avg RTT: 17.73 ms
Packet lost: 0.00%
● PS C:\Users\cruze\Documents\CPSC471> c:; cd 'c:\Users\cruze\Documents\CPSC471'; & 'c:\Users\cruze\AppData\Local\Programs\Python\Python312\python.exe' 'c:\Users\cruze\.vscode\extensions\ms-python.debugpy-2024.12.0-win32-x64\bundled\libs\debugpy\adapter\cruze\Documents\CPSC471\ECP2a.py'
Edith 1 Sat Nov 09 23:15:46 2024: server reply: Edith 1 Sat Nov 09 23:15:46 2024, RTT = 16.50 ms
Edith 2 Sat Nov 09 23:15:46 2024: server reply: Edith 2 Sat Nov 09 23:15:46 2024, RTT = 13.30 ms
Edith 3 Sat Nov 09 23:15:46 2024: server reply: Edith 3 Sat Nov 09 23:15:46 2024, RTT = 11.02 ms
Edith 4 Sat Nov 09 23:15:46 2024: server reply: Edith 4 Sat Nov 09 23:15:46 2024, RTT = 16.86 ms
Edith 5 Sat Nov 09 23:15:46 2024: server reply: Edith 5 Sat Nov 09 23:15:46 2024, RTT = 18.05 ms
Edith 6 Sat Nov 09 23:15:46 2024: server reply: Edith 6 Sat Nov 09 23:15:46 2024, RTT = 15.01 ms
Edith 7 Sat Nov 09 23:15:46 2024: server reply: Edith 7 Sat Nov 09 23:15:46 2024, RTT = 12.74 ms
Edith 8 Sat Nov 09 23:15:46 2024: server reply: Edith 8 Sat Nov 09 23:15:46 2024, RTT = 12.67 ms
Edith 9 Sat Nov 09 23:15:46 2024: server reply: Edith 9 Sat Nov 09 23:15:46 2024, RTT = 20.07 ms
Edith 10 Sat Nov 09 23:15:46 2024: server reply: Edith 10 Sat Nov 09 23:15:46 2024, RTT = 14.54 ms
Min RTT: 11.02 ms
Max RTT: 20.07 ms
Avg RTT: 15.08 ms
Packet lost: 0.00%
○ PS C:\Users\cruze\Documents\CPSC471>
```

3. Paste the Python server code listing text

```
4. # udppingserver_no_loss.py
5. # first Mod
6. from socket import *
7. import time
8. import random
9.
10. # Create a UDP socket
11. serverSocket = socket(AF_INET, SOCK_DGRAM)
12. # Assign IP address and port number to socket
13. serverSocket.bind('', 12000)
14. while True:
15.     # Receive the client packet along with the address it is coming from
16.     message, address = serverSocket.recvfrom(1024)
17.
18.     delay = random.randint(10, 20) / 1000 # Convert ms to s
19.     time.sleep(delay) # Sleep only accepts seconds
20.
21.     # The server responds
22.     serverSocket.sendto(message, address)
```

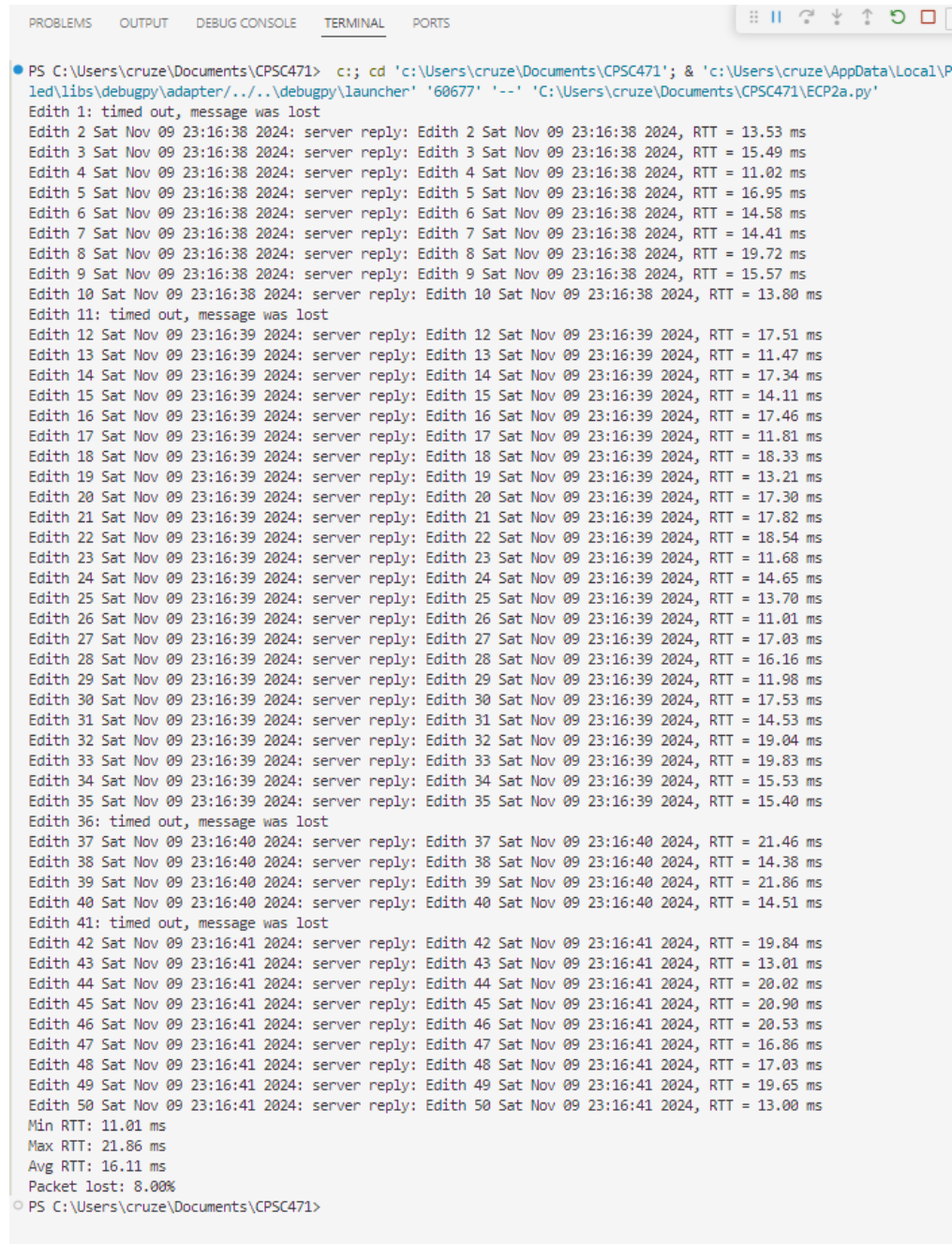
Part 2c – UDP Pinger with Delays and Packet Losses

1. Describe the operation of your UDP Ping Server and explain how it simulates delays between 10ms and 20ms, with up to 10% packet losses.

The UDP Ping Server file was once again modified and another random variable was introduced. This random variable produces a number between 1 and 100 and if this number is less than or equal to 10 then the server response is skipped and it starts from the top of the while loop again. The ≤ 10 signifies the 10% packet loss.

2. Explain how to run your code, i.e., command line and any applicable parameter(s)

On VSCode modify the variable totalPings in ECP2a from 10 to 50. Run ECP2c and then ECP2a.



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
● PS C:\Users\cruze\Documents\CPSC471> c:; cd 'c:\Users\cruze\Documents\CPSC471'; & 'c:\Users\cruze\AppData\Local\Programs\Python\Python39\python.exe' 'c:\Users\cruze\Documents\CPSC471\ECP2a.py'
Edith 1: timed out, message was lost
Edith 2 Sat Nov 09 23:16:38 2024: server reply: Edith 2 Sat Nov 09 23:16:38 2024, RTT = 13.53 ms
Edith 3 Sat Nov 09 23:16:38 2024: server reply: Edith 3 Sat Nov 09 23:16:38 2024, RTT = 15.49 ms
Edith 4 Sat Nov 09 23:16:38 2024: server reply: Edith 4 Sat Nov 09 23:16:38 2024, RTT = 11.02 ms
Edith 5 Sat Nov 09 23:16:38 2024: server reply: Edith 5 Sat Nov 09 23:16:38 2024, RTT = 16.95 ms
Edith 6 Sat Nov 09 23:16:38 2024: server reply: Edith 6 Sat Nov 09 23:16:38 2024, RTT = 14.58 ms
Edith 7 Sat Nov 09 23:16:38 2024: server reply: Edith 7 Sat Nov 09 23:16:38 2024, RTT = 14.41 ms
Edith 8 Sat Nov 09 23:16:38 2024: server reply: Edith 8 Sat Nov 09 23:16:38 2024, RTT = 19.72 ms
Edith 9 Sat Nov 09 23:16:38 2024: server reply: Edith 9 Sat Nov 09 23:16:38 2024, RTT = 15.57 ms
Edith 10 Sat Nov 09 23:16:38 2024: server reply: Edith 10 Sat Nov 09 23:16:38 2024, RTT = 13.80 ms
Edith 11: timed out, message was lost
Edith 12 Sat Nov 09 23:16:39 2024: server reply: Edith 12 Sat Nov 09 23:16:39 2024, RTT = 17.51 ms
Edith 13 Sat Nov 09 23:16:39 2024: server reply: Edith 13 Sat Nov 09 23:16:39 2024, RTT = 11.47 ms
Edith 14 Sat Nov 09 23:16:39 2024: server reply: Edith 14 Sat Nov 09 23:16:39 2024, RTT = 17.34 ms
Edith 15 Sat Nov 09 23:16:39 2024: server reply: Edith 15 Sat Nov 09 23:16:39 2024, RTT = 14.11 ms
Edith 16 Sat Nov 09 23:16:39 2024: server reply: Edith 16 Sat Nov 09 23:16:39 2024, RTT = 17.46 ms
Edith 17 Sat Nov 09 23:16:39 2024: server reply: Edith 17 Sat Nov 09 23:16:39 2024, RTT = 11.81 ms
Edith 18 Sat Nov 09 23:16:39 2024: server reply: Edith 18 Sat Nov 09 23:16:39 2024, RTT = 18.33 ms
Edith 19 Sat Nov 09 23:16:39 2024: server reply: Edith 19 Sat Nov 09 23:16:39 2024, RTT = 13.21 ms
Edith 20 Sat Nov 09 23:16:39 2024: server reply: Edith 20 Sat Nov 09 23:16:39 2024, RTT = 17.30 ms
Edith 21 Sat Nov 09 23:16:39 2024: server reply: Edith 21 Sat Nov 09 23:16:39 2024, RTT = 17.82 ms
Edith 22 Sat Nov 09 23:16:39 2024: server reply: Edith 22 Sat Nov 09 23:16:39 2024, RTT = 18.54 ms
Edith 23 Sat Nov 09 23:16:39 2024: server reply: Edith 23 Sat Nov 09 23:16:39 2024, RTT = 11.68 ms
Edith 24 Sat Nov 09 23:16:39 2024: server reply: Edith 24 Sat Nov 09 23:16:39 2024, RTT = 14.65 ms
Edith 25 Sat Nov 09 23:16:39 2024: server reply: Edith 25 Sat Nov 09 23:16:39 2024, RTT = 13.70 ms
Edith 26 Sat Nov 09 23:16:39 2024: server reply: Edith 26 Sat Nov 09 23:16:39 2024, RTT = 11.01 ms
Edith 27 Sat Nov 09 23:16:39 2024: server reply: Edith 27 Sat Nov 09 23:16:39 2024, RTT = 17.03 ms
Edith 28 Sat Nov 09 23:16:39 2024: server reply: Edith 28 Sat Nov 09 23:16:39 2024, RTT = 16.16 ms
Edith 29 Sat Nov 09 23:16:39 2024: server reply: Edith 29 Sat Nov 09 23:16:39 2024, RTT = 11.98 ms
Edith 30 Sat Nov 09 23:16:39 2024: server reply: Edith 30 Sat Nov 09 23:16:39 2024, RTT = 17.53 ms
Edith 31 Sat Nov 09 23:16:39 2024: server reply: Edith 31 Sat Nov 09 23:16:39 2024, RTT = 14.53 ms
Edith 32 Sat Nov 09 23:16:39 2024: server reply: Edith 32 Sat Nov 09 23:16:39 2024, RTT = 19.04 ms
Edith 33 Sat Nov 09 23:16:39 2024: server reply: Edith 33 Sat Nov 09 23:16:39 2024, RTT = 19.83 ms
Edith 34 Sat Nov 09 23:16:39 2024: server reply: Edith 34 Sat Nov 09 23:16:39 2024, RTT = 15.53 ms
Edith 35 Sat Nov 09 23:16:39 2024: server reply: Edith 35 Sat Nov 09 23:16:39 2024, RTT = 15.40 ms
Edith 36: timed out, message was lost
Edith 37 Sat Nov 09 23:16:40 2024: server reply: Edith 37 Sat Nov 09 23:16:40 2024, RTT = 21.46 ms
Edith 38 Sat Nov 09 23:16:40 2024: server reply: Edith 38 Sat Nov 09 23:16:40 2024, RTT = 14.38 ms
Edith 39 Sat Nov 09 23:16:40 2024: server reply: Edith 39 Sat Nov 09 23:16:40 2024, RTT = 21.86 ms
Edith 40 Sat Nov 09 23:16:40 2024: server reply: Edith 40 Sat Nov 09 23:16:40 2024, RTT = 14.51 ms
Edith 41: timed out, message was lost
Edith 42 Sat Nov 09 23:16:41 2024: server reply: Edith 42 Sat Nov 09 23:16:41 2024, RTT = 19.84 ms
Edith 43 Sat Nov 09 23:16:41 2024: server reply: Edith 43 Sat Nov 09 23:16:41 2024, RTT = 13.01 ms
Edith 44 Sat Nov 09 23:16:41 2024: server reply: Edith 44 Sat Nov 09 23:16:41 2024, RTT = 20.02 ms
Edith 45 Sat Nov 09 23:16:41 2024: server reply: Edith 45 Sat Nov 09 23:16:41 2024, RTT = 20.90 ms
Edith 46 Sat Nov 09 23:16:41 2024: server reply: Edith 46 Sat Nov 09 23:16:41 2024, RTT = 20.53 ms
Edith 47 Sat Nov 09 23:16:41 2024: server reply: Edith 47 Sat Nov 09 23:16:41 2024, RTT = 16.86 ms
Edith 48 Sat Nov 09 23:16:41 2024: server reply: Edith 48 Sat Nov 09 23:16:41 2024, RTT = 17.03 ms
Edith 49 Sat Nov 09 23:16:41 2024: server reply: Edith 49 Sat Nov 09 23:16:41 2024, RTT = 19.65 ms
Edith 50 Sat Nov 09 23:16:41 2024: server reply: Edith 50 Sat Nov 09 23:16:41 2024, RTT = 13.00 ms
Min RTT: 11.01 ms
Max RTT: 21.86 ms
Avg RTT: 16.11 ms
Packet lost: 8.00%
PS C:\Users\cruze\Documents\CPSC471>
```

3. Paste the Python server code listing text

```
4. # udppingserver_no_loss.py
5. # second Mod
6. from socket import *
7. import time
8. import random
9.
10. # Create a UDP socket
11. serverSocket = socket(AF_INET, SOCK_DGRAM)
12. # Assign IP address and port number to socket
13. serverSocket.bind('', 12000)
14. while True:
15.     # Receive the client packet along with the address it is coming from
16.     message, address = serverSocket.recvfrom(1024)
17.
18.     specialVal = random.randint(1, 100)
19.     if (specialVal <= 10): # 10%
20.         continue        # this skips the server response
21.
22.     delay = random.randint(10, 20) / 1000 # Convert ms to s
23.     time.sleep(delay) # Sleep only accepts seconds
24.
25.     # The server responds
26.     serverSocket.sendto(message, address)
```