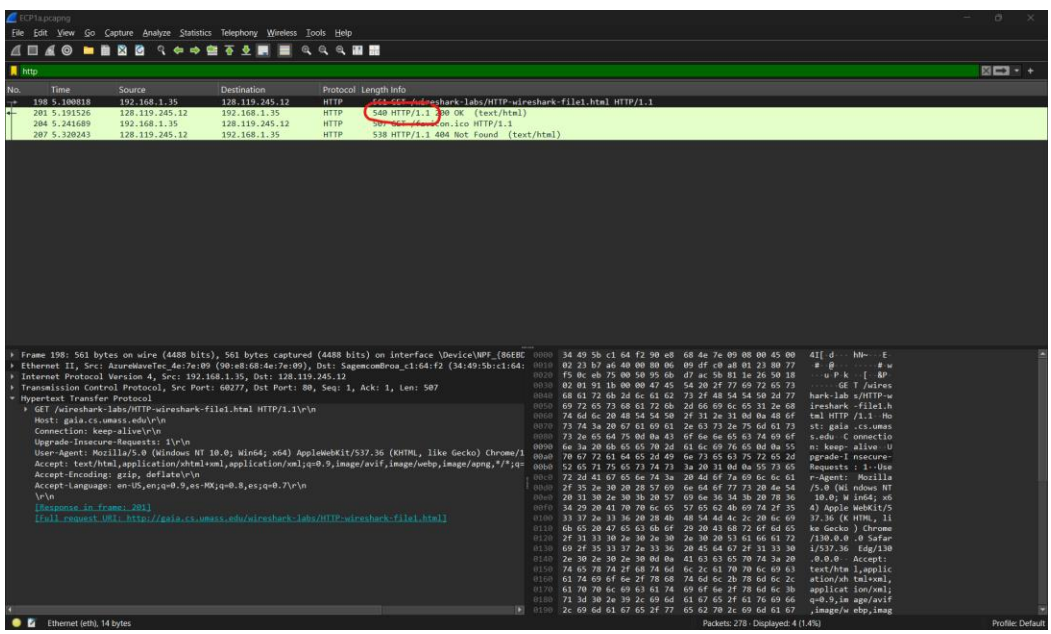
Part 1: Analyzing HTTP Messages using Wireshark

Part 1a – Basic HTTP GET/response interaction

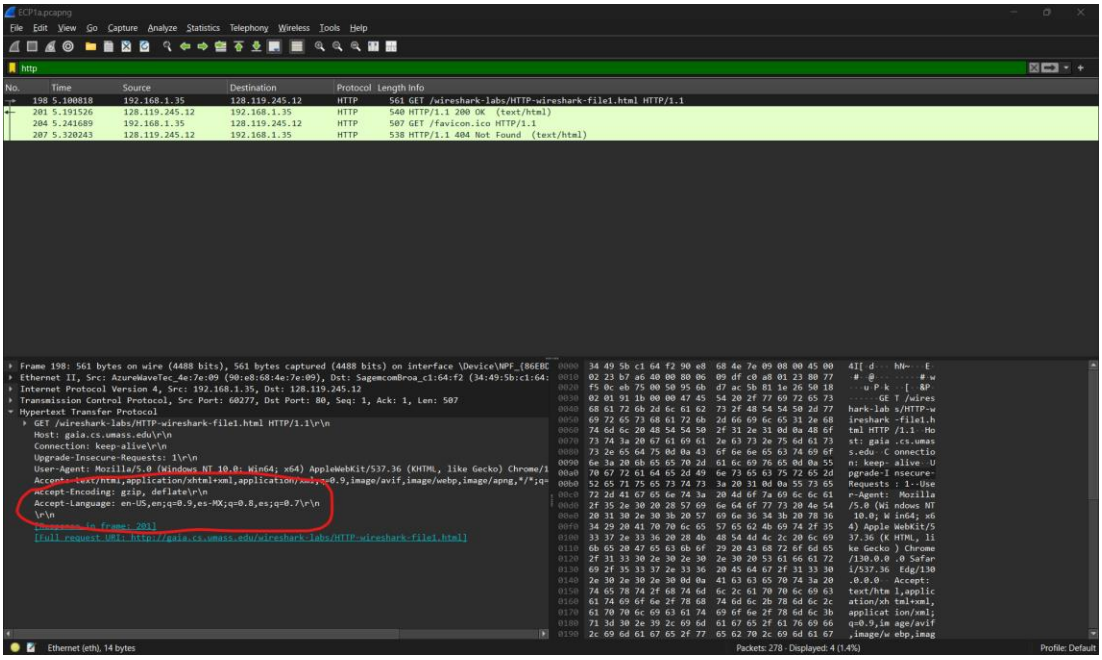
1. Is your browser running HTTP version 1.0 or 1.1? What version of HTTP is the server running?

   My browser is running HTTP version 1.1



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

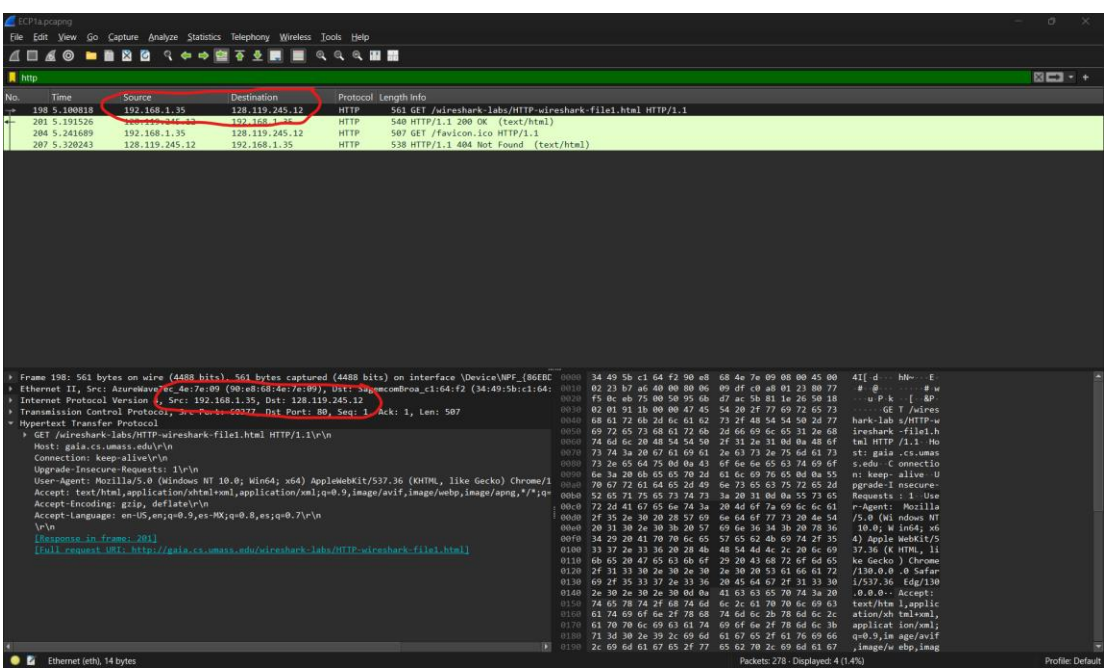
   My browser indicates that it can accept English (US) and Spanish (MX).



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

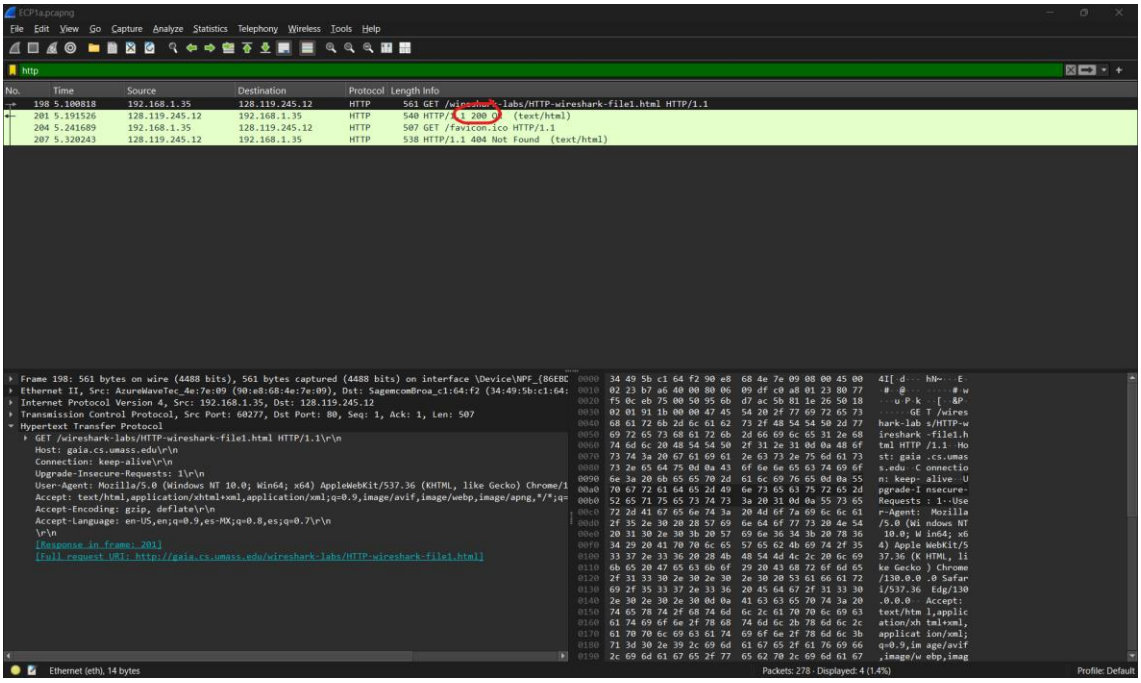
   Computer IP address: 192.168.1.35

   Gaia.cs.umass.edu IP address: 128.119.245.12



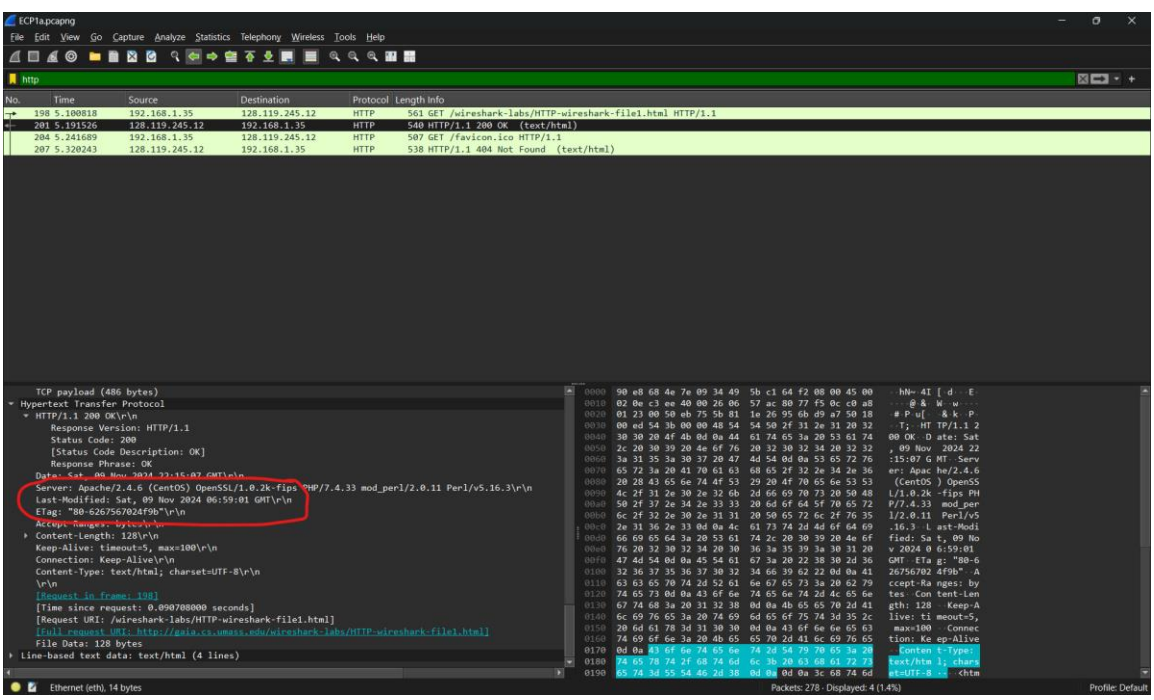
4. What is the status code returned from the server to your browser?

   Status Code: 200



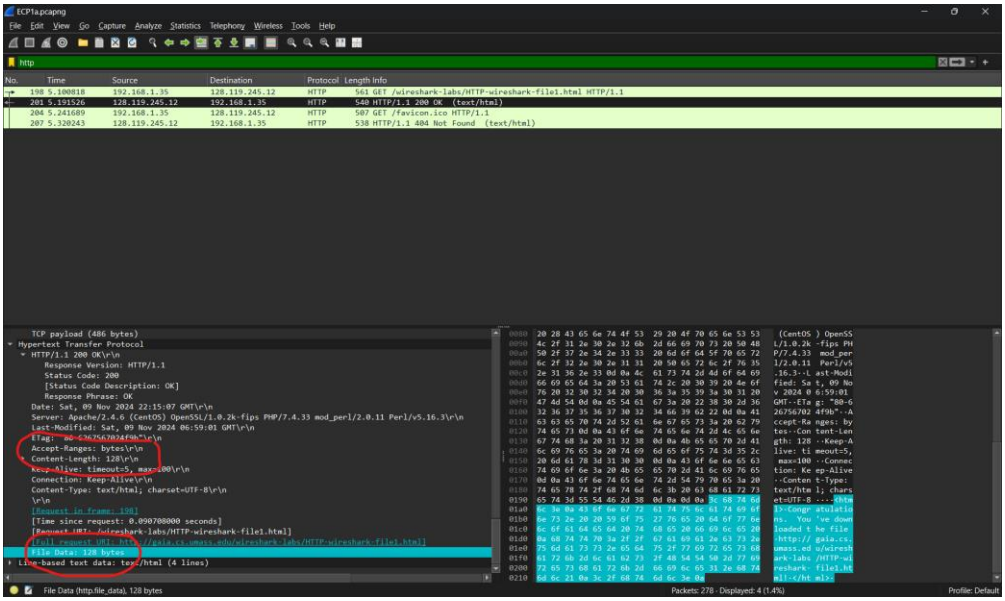
5. When was the HTML file that you are retrieving last modified at the server?

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



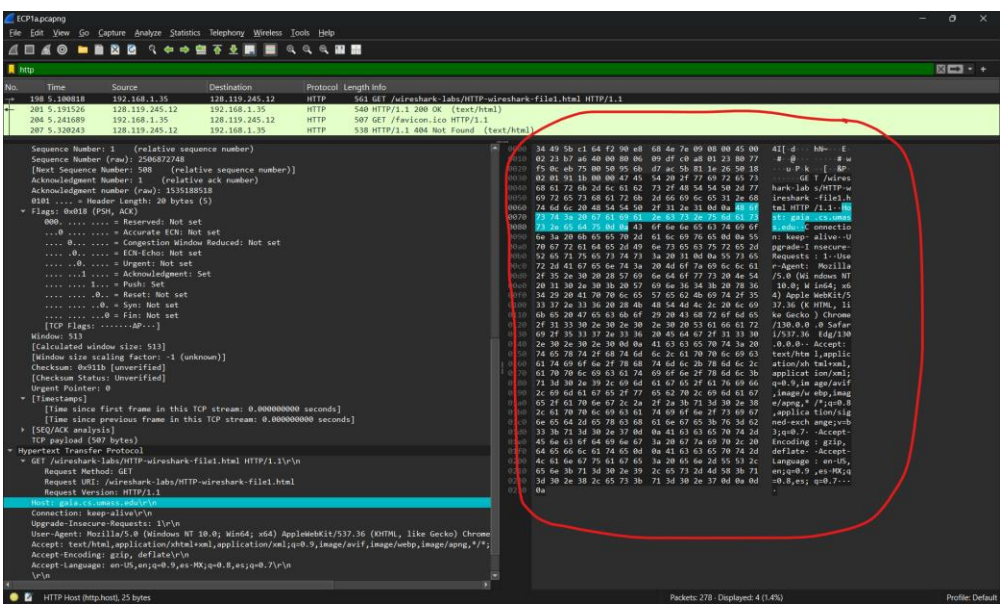
6. How many bytes of content are being returned to your browser?

   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.

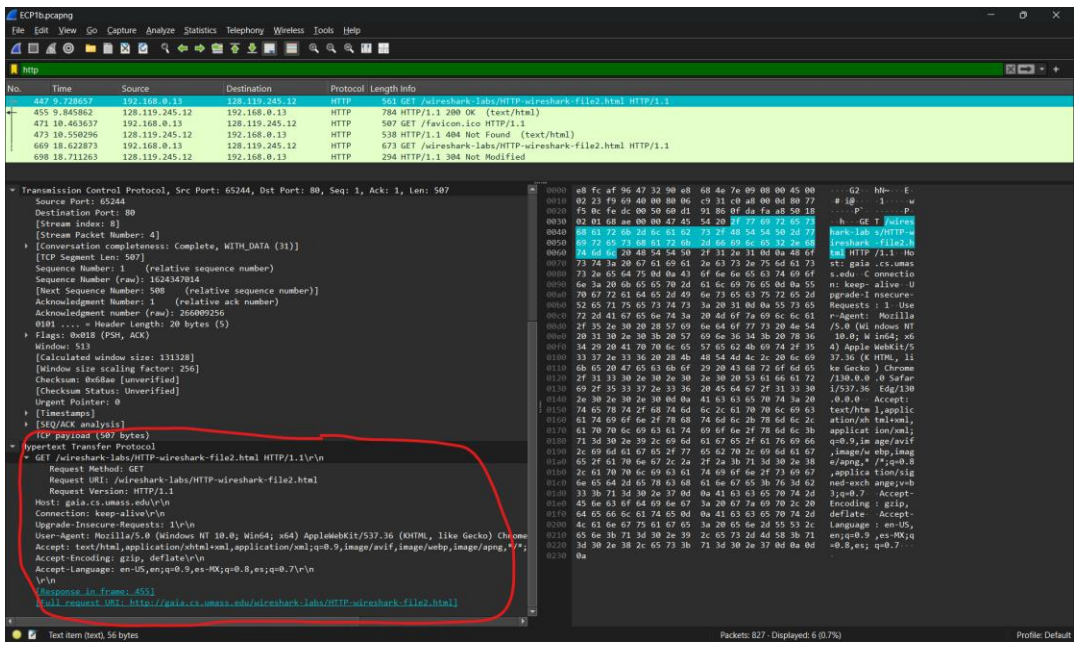
   I do not see any headers within the data that are not displayed in the packet-listing window.



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

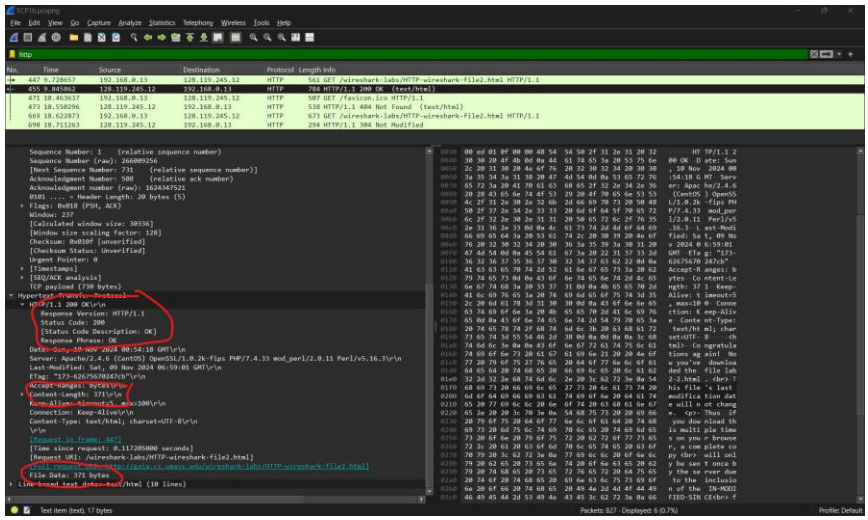
1. Inspect the contents of the first HTTP GET request from your browser to the server. Do you see an “IF-MODIFIED-SINCE” line in the HTTP GET?

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



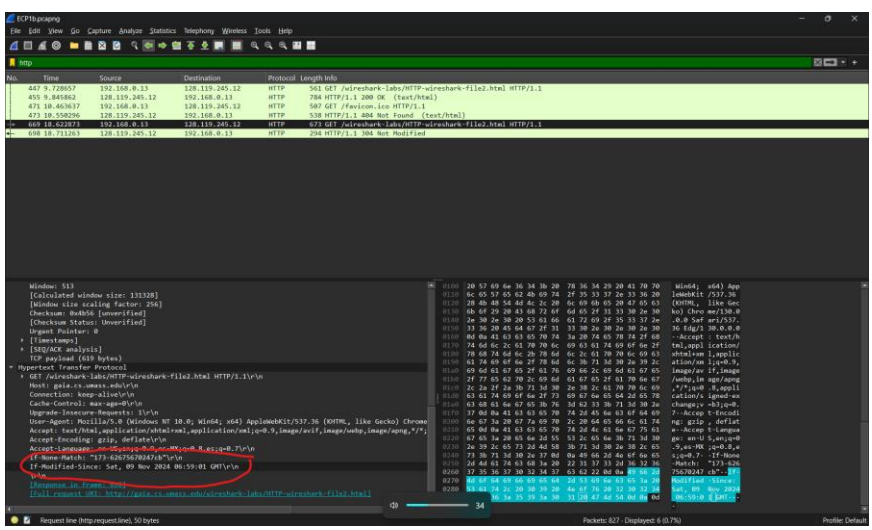
2. Inspect the contents of the server response. Did the server explicitly return the contents of the file? How can you tell?

   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.



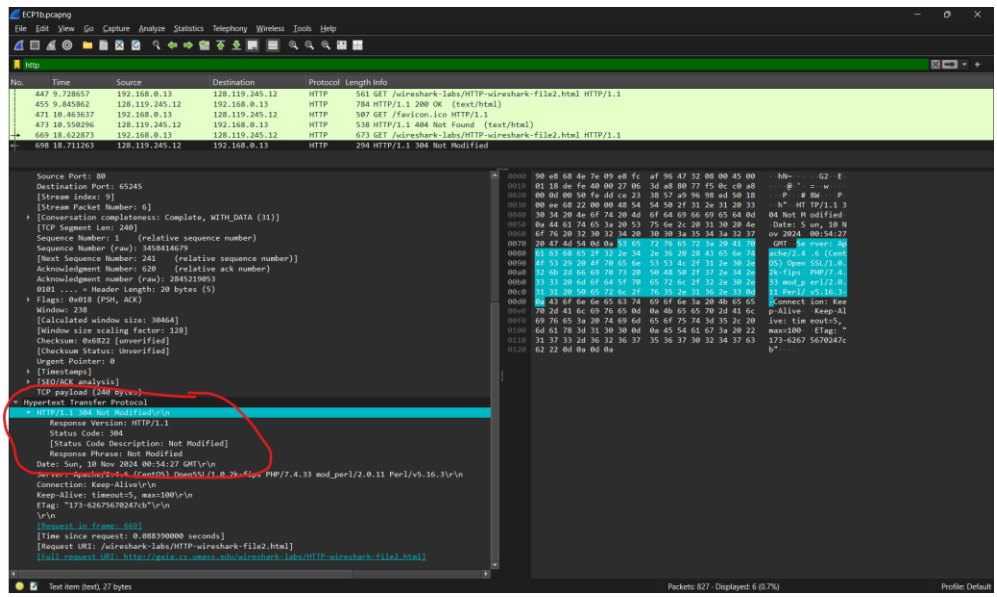
3. Now inspect the contents of the second HTTP GET request from your browser to the server. Do you see an “IF-MODIFIED-SINCE:” line in the HTTP GET? If so, what infor mation follows the “IF-MODIFIED-SINCE:” header?

   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.

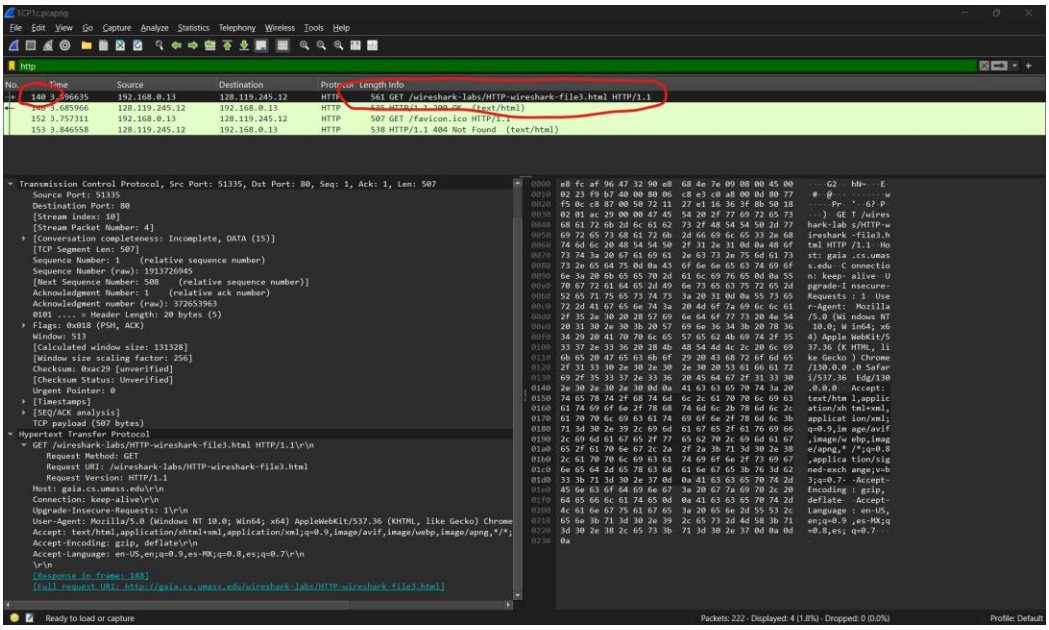
It did not return the contents of the file. Underneath Hypertext Transfer Protocol > HTTP/1.1 304 Not Modified\r\n the Response Phrase is Not Modified. When looked up, HTTP 304 Not Modified signifies that the resource cached by the client is still valid.



Part 1c: Retrieving Long Documents

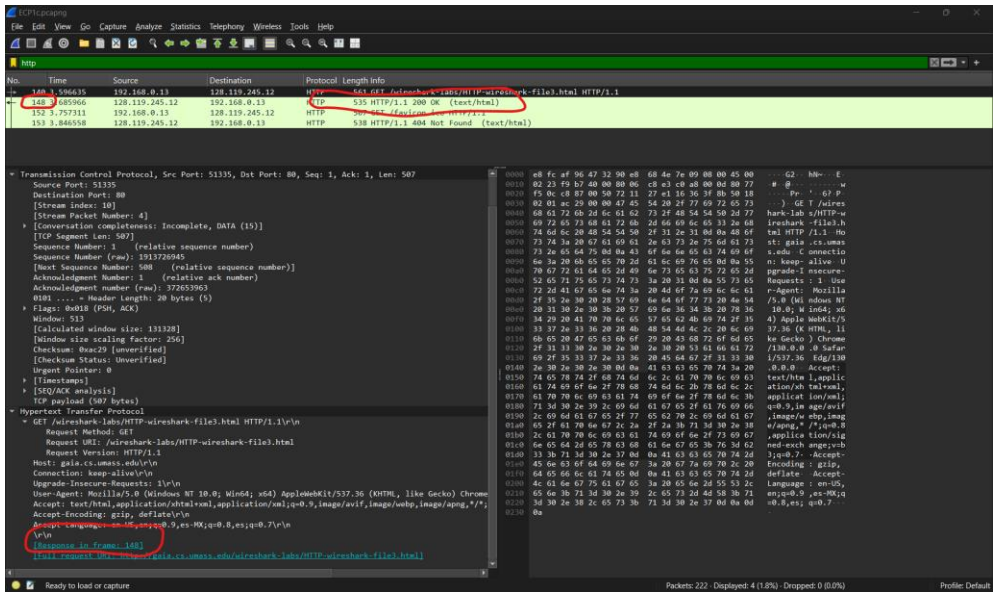
1. How many HTTP GET request messages did your browser send? Which packet number in the trace contains the GET message for the Bill of Rights?

   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.



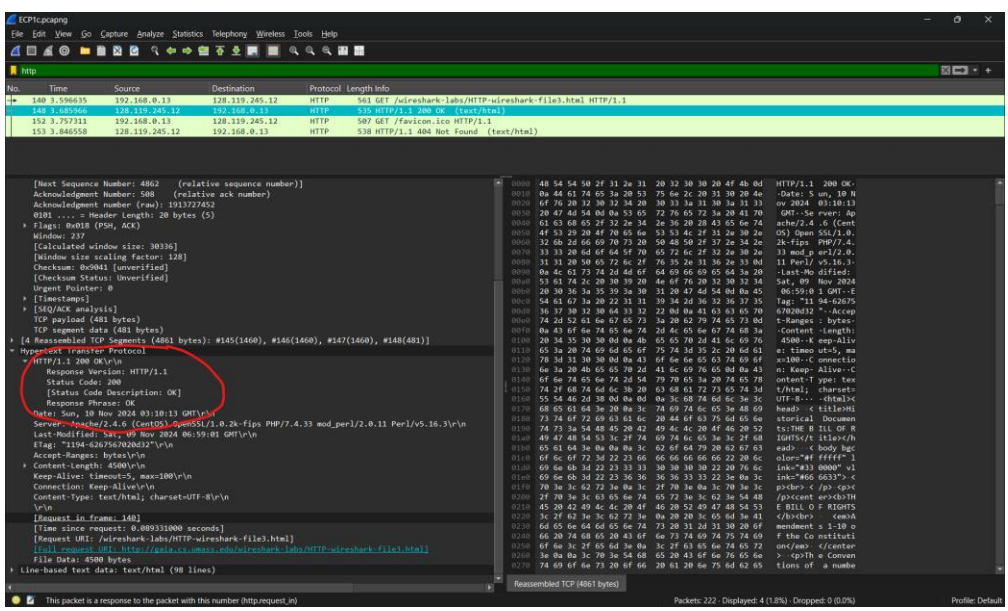
2. Which packet number in the trace contains the status code and phrase associated with the response to the HTTP GET request?

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



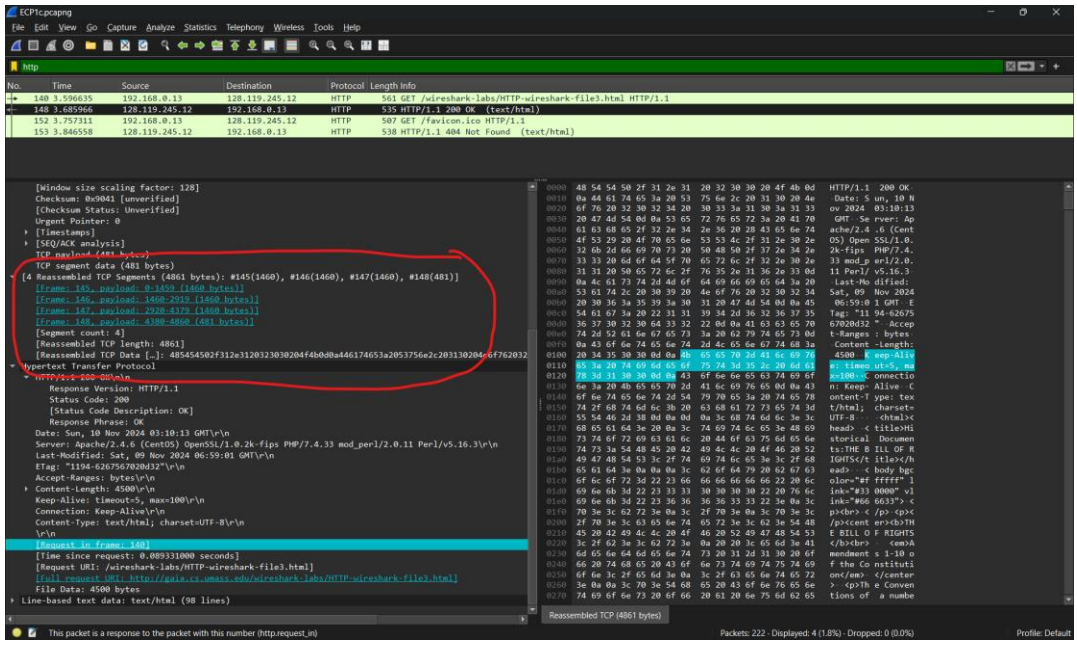
3. What is the status code and phrase in the response?

   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?

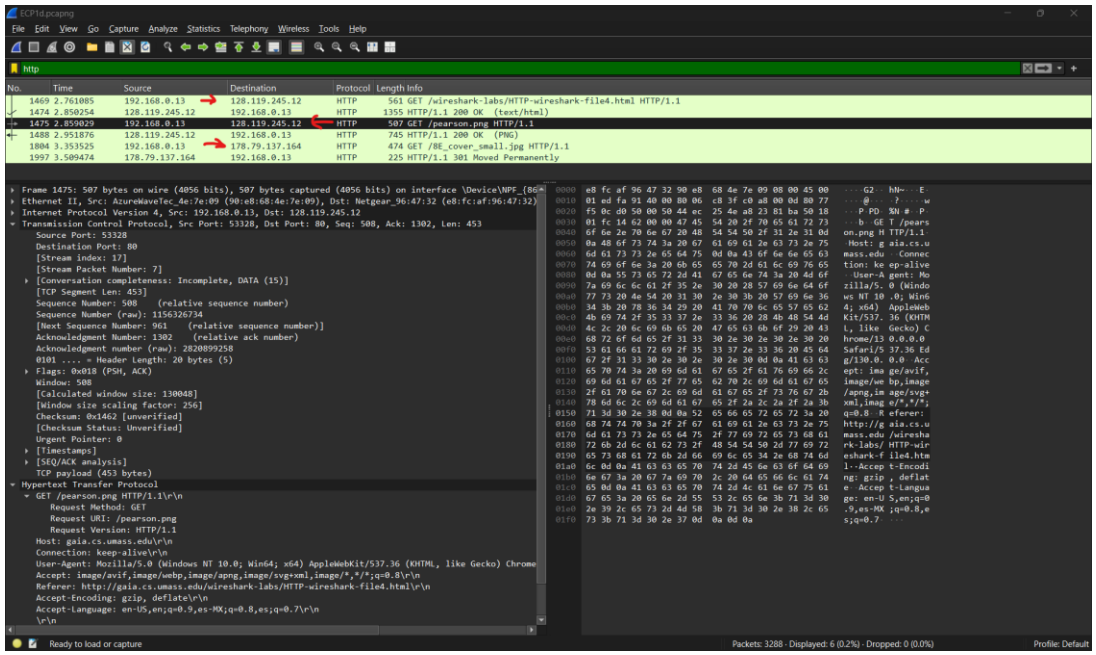
   Four data-containing TCP segments were needed to carry the single HTTP response and the text of the Bill of Rights. Those being packets 145, 146, 147, and 148.



Part 1d: HTML Documents with Embedded Objects

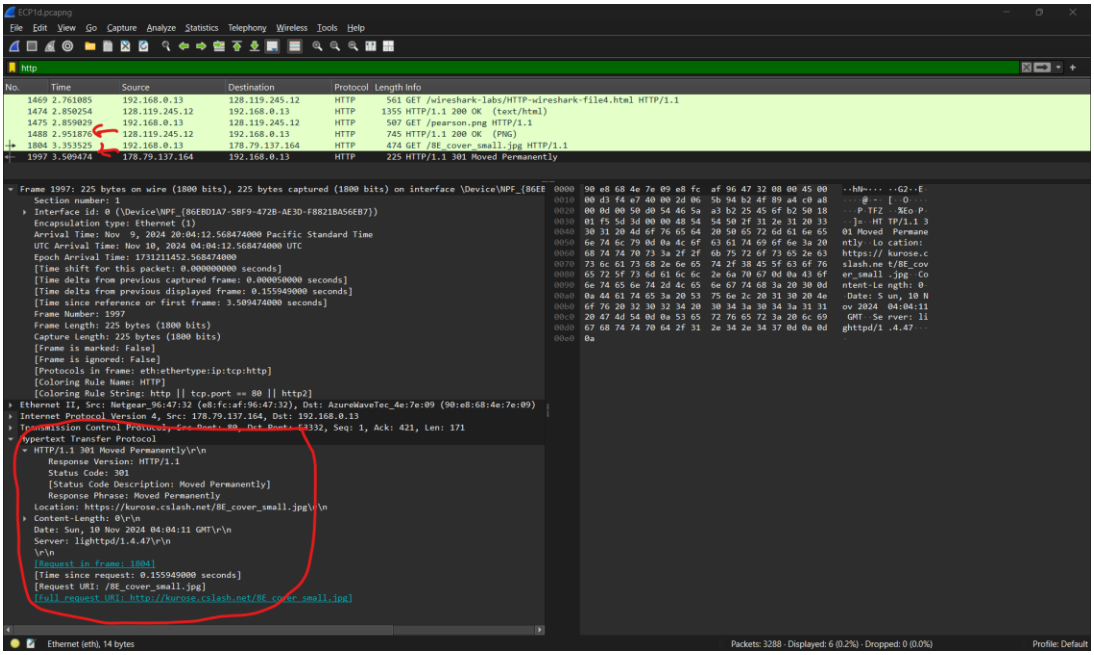
1. How many HTTP GET request messages did your browser send? To which Internet addresses were these GET requests sent?

   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.

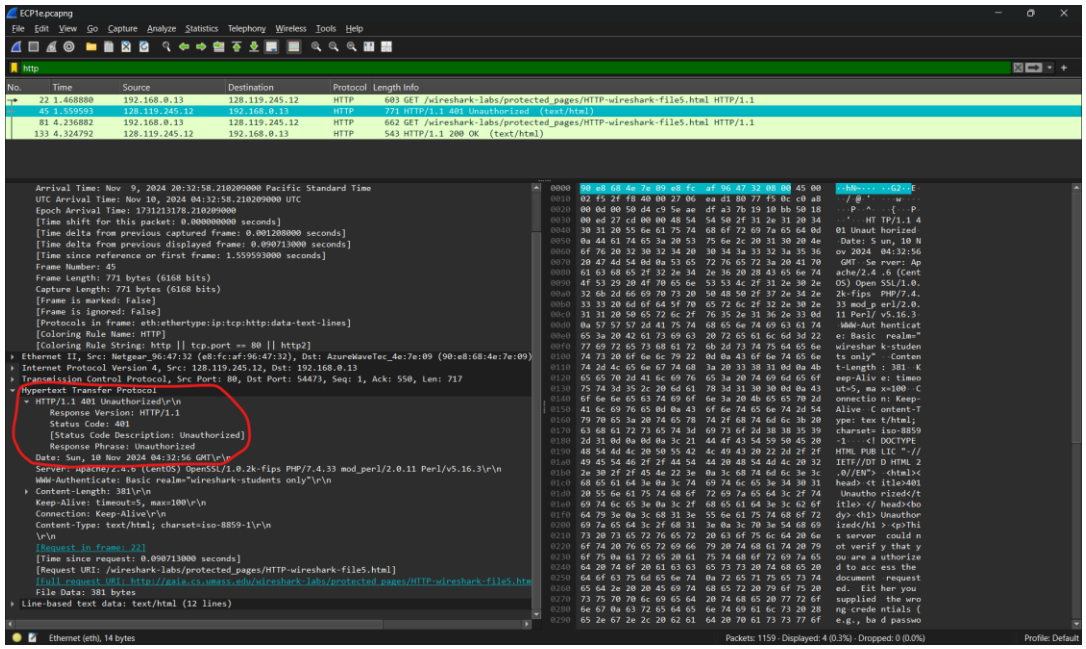
   The browser downloaded the two images serially. You can tell because there is a gap in the timestamps of packets 1488 and 1997 which are the responses to the two latter GET request messages. Additionally, the second image required a redirection and would have downloaded only after the first image.



Part 1e: HTTP Authentication

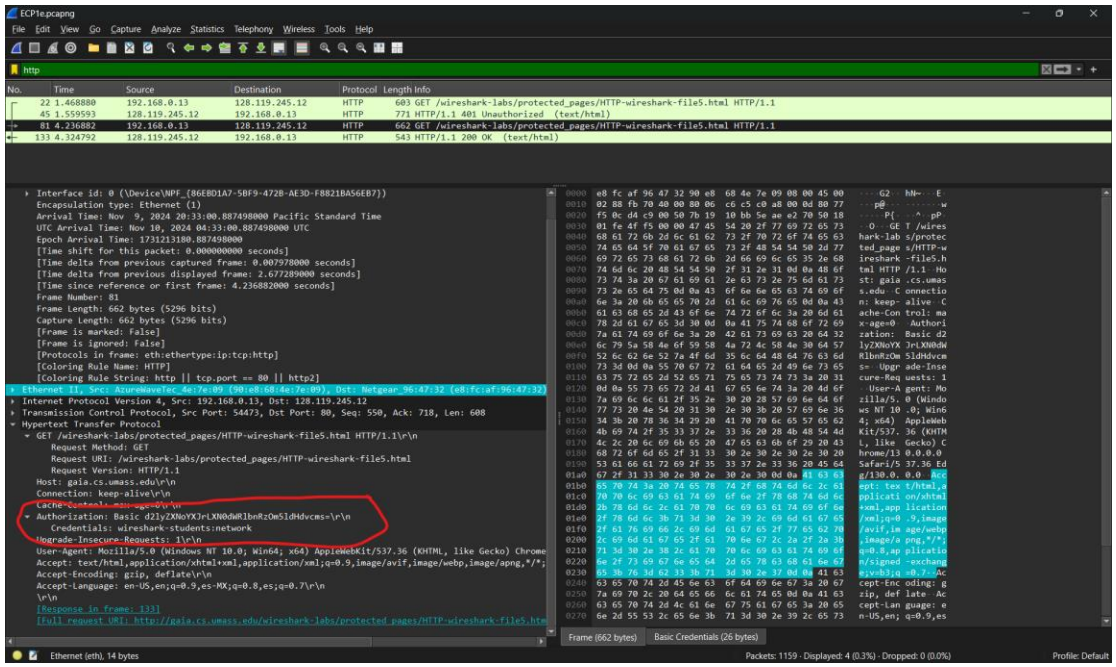
1. What is the server's response (status code and phrase) in response to the initial HTTP GET message from your browser?

   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:\U
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

```
# from socket import * didnt work here
import socket
# needed for time
import time
# needed for date time in response message
from datetime import datetime

# Server details
serverAddress = ('localhost', 12000)  # Server IP and port
totalPings = 10                       # Total number of pings to send, change this num to 50 for ECP2c
bufferSize = 1024                     # Size of buffer for receiving messages

# Initialize statistics
rttVals = []
packetLossCt = 0

# My name used in pings
myName = "Edith"

# Create UDP socket
clientSocket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
clientSocket.settimeout(1)  # Setting timeout to 1s

for pingNum in range(1, totalPings + 1):
    # format it
    # %a = abb. weekday name; %b = abb. month name
    timeStamp = datetime.now().strftime("%a %b %d %H:%M:%S %Y")
    message = f"{myName} {pingNum} {timeStamp}"

    try:
        # Record the send time
        timeStart = time.time()

        # Send the message to the server
        clientSocket.sendto(message.encode(), serverAddress)

        # Wait for response from server
        # Recall to run udppingserver_no_loss.py first!
        # Or you'll get an error
        response, _ = clientSocket.recvfrom(bufferSize)

        # Record the end time and calculate RTT
```

```python
        endTime = time.time()
        rtt = (endTime - timeStart) * 1000  # Converting RTT to milliseconds
        rttVals.append(rtt)

        # Print server's response and RTT
        print(f"{message}: server reply: {response.decode()}, RTT = {rtt:.2f} ms")

    except socket.timeout:
        # Handle case where the request timed out
        print(f"{myName} {pingNum}: timed out, message was lost")
        packetLossCt += 1

# MATH YAY
if rttVals:
    minVal = min(rttVals)
    maxVal = max(rttVals)
    avgVal = sum(rttVals) / len(rttVals)
else:
    minVal = maxVal = avgVal = 0.0

# equation taken from labs
packetLossRt = (packetLossCt / totalPings) * 100

# Check online for proper formatting of integers with decimals and percentages
# Ping Summary
print(f"Min RTT: {minVal:.2f} ms")
print(f"Max RTT: {maxVal:.2f} ms")
print(f"Avg RTT: {avgVal:.2f} ms")
print(f"Packet lost: {packetLossRt:.2f}%")

```

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\App
' 'c:\Users\cruze\.vscode\extensions\ms-python.debugpy-2024.12.0-win32-x64\bundled\libs\debugpy\adapte
s\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

```python
# udppingserver_no_loss.py
# first Mod
from socket import *
import time
import random

# Create a UDP socket
serverSocket = socket(AF_INET, SOCK_DGRAM)
# Assign IP address and port number to socket
serverSocket.bind(('', 12000))
while True:
    # Receive the client packet along with the address it is coming from
    message, address = serverSocket.recvfrom(1024)

    delay = random.randint(10, 20) / 1000  # Convert ms to s
    time.sleep(delay)  # Sleep only accepts seconds

    # The server responds
    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 be tween 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\P
led\libs\debugpy\adapter/../..\debugpy\launcher' '60677' '--' '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)
```