BHARATIYA VIDYA BHAVAN'S
SARDAR PATEL INSTITUTE OF TECHNOLOGY
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai – 400058-India
Department of Computer Engineering

Name	**Shivam Santosh Kadam**
UID no.	**2023300099**
Experiment No.	**3**

AIM:	**To explore and analyze the HTTP protocol using Wireshark to capture and inspect HTTP interactions, including GET requests, responses, and authentication.**

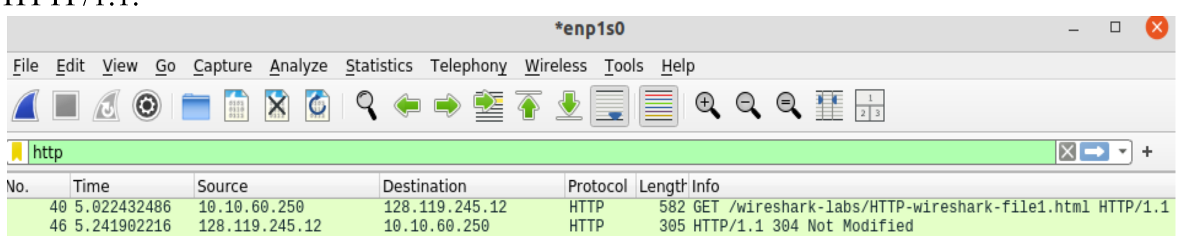
SOLUTION:

Task A: 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?**
   Answer:

   Browser (Client) HTTP Version: My browser is using **HTTP/1.1**, as indicated by the "GET" request line:GET /wireshark-labs/HTTP-wireshark-file1.html HTTP/1.1. This shows that the browser is requesting the resource using HTTP/1.1.

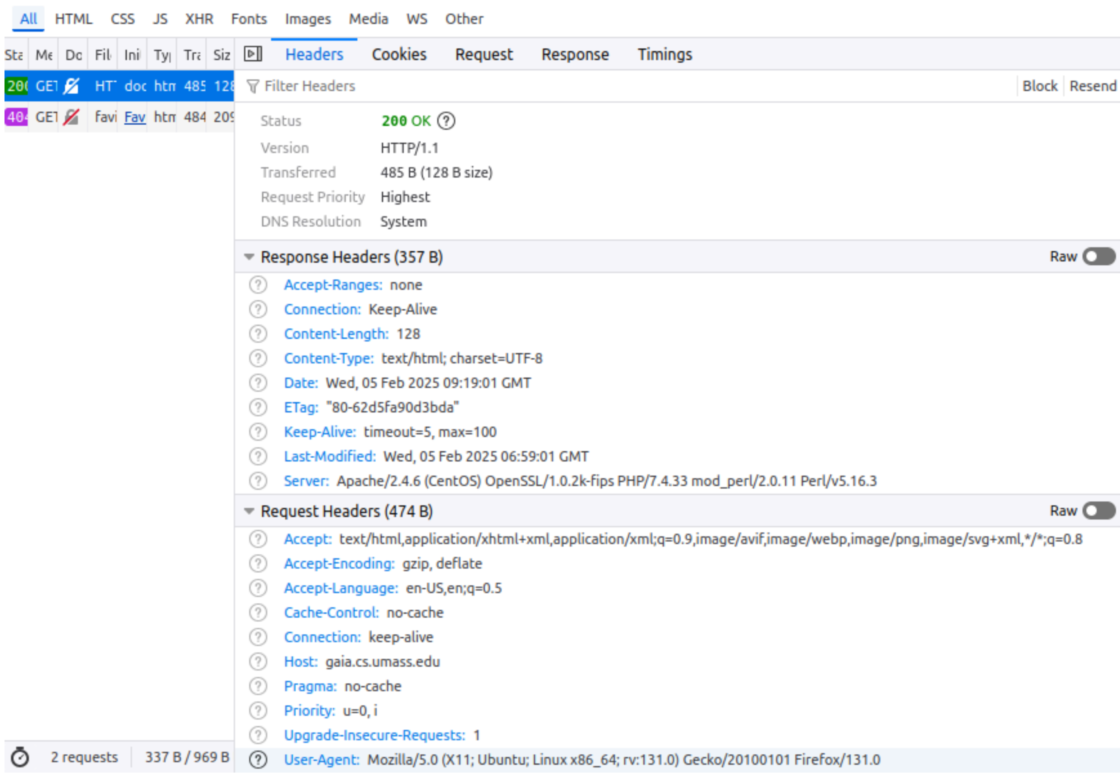
   Server HTTP Version: The server is also using **HTTP/1.1**, as seen in the response: HTTP/1.1 304 Not Modified . This indicates that the server's response is also using HTTP/1.1.



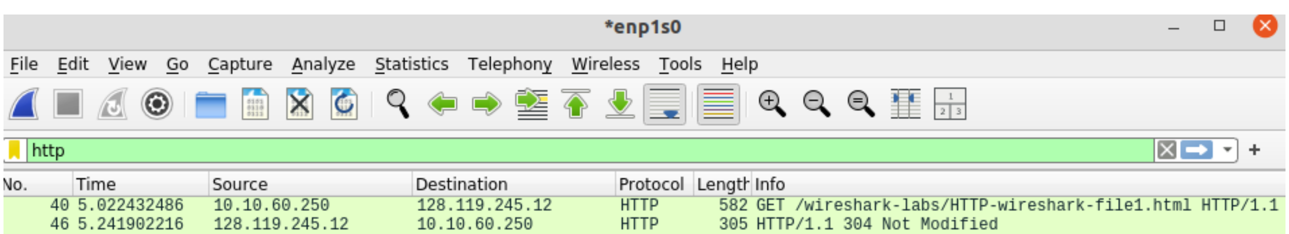
2. **What languages (if any) does your browser indicate that it can accept to the server? In the captured session, what other information (if any) does the browser provide the server with regarding the user/browser?**
   Answer: In the captured session, the browser indicates the following details it provides to the server:

- **Languages Accepted:** The browser indicates a preference for English (US) with the `Accept-Language` header, specifically `en-US`. It also indicates that it can accept general English (`en`), but with a lower preference (`q=0.5`).

- **User-Agent:** The browser identifies itself as Mozilla Firefox version 131.0, running on an Ubuntu Linux x86_64 system, through the `User-Agent` header. This provides information about the browser and operating system.
- **Accept:** The browser specifies the types of content it is willing to accept from the server. This includes HTML, XHTML, XML, and various image formats like AVIF, WebP, PNG, and SVG, with a fallback option for any type (`*/*`) at a lower preference (`q=0.8`).
- **Accept-Encoding:** The browser supports compressed content and indicates it can accept gzip and deflate encoding methods, optimizing data transfer.
- **Cache-Control:** The browser requests that the content should not be cached by the server (`no-cache`), ensuring that the latest version of the resource is always fetched.
- **Connection:** The browser prefers to keep the connection alive with the server, as indicated by the `Connection: keep-alive` header.
- **Host:** The browser specifies the target domain for the request with the `Host` header, which in this case is `gaia.cs.umass.edu`.
- **Pragma:** Similar to the `Cache-Control` header, the `Pragma` header further emphasizes the request for no caching (`no-cache`).
- **Upgrade-Insecure-Requests:** The browser indicates a preference for upgrading any insecure HTTP requests to HTTPS, ensuring a more secure connection.
- **Priority:** The request is marked with a high priority (`u=0, i`), indicating urgency in processing the request.



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



The IP address of my computer(**10.10.60.250**) can be found in the source IP field of the IP header in the GET request. The IP address of the server is shown in the destination IP field(**128.119.245.12**) of the same GET request.The IP addresses will be part of the packet details. You can identify your machine's IP by checking your network connection and comparing it to the packet.

4. **What is the status code returned from the server to your browser?**
   Answer: The status code can be found in the response message from the server. Typically, a 200 status code means the request was successful, while other codes like 404 indicate errors. The response will also include a phrase corresponding to the code, e.g., "200 OK" for

success.

In my case, however, the server returned a **304 status code**, which means "Not Modified." This indicates that the requested resource has not been modified since the last request, and the browser can use the cached version of the content.

```
129 10.980447615  128.119.245.12      10.10.60.250       HTTP          305 HTTP/1.1 304 Not Modified
```

5. **When was the HTML file that you are retrieving last modified at the server?**
   Answer: Look for the `Last-Modified` header in the server's response. It will provide the date and time the file was last modified.This timestamp helps the browser to decide if the file is still valid or if it needs to request a fresh copy.

   Last-Modified: Wed, 05 Feb 2025 06:59:01 GMT

6. **How many bytes of content are being returned to your browser?**
   Answer: The `Content-Length` header in the response will indicate the number of bytes of content.This field tells the browser how many bytes it should expect in the body of the response.

   Content-Length: 128
   Content-Type: text/html; charset=UTF-8

7. **By inspecting the raw data in the "packet bytes" pane, do you see any HTTP headers within the data that are not displayed in the "packet details" pane? If so, name one.**
   Answer: Some headers, such as `Set-Cookie`, may not always be visible in the "packet details" pane. You might need to inspect the raw byte data for all headers.
   Some headers are not always parsed by Wireshark's dissectors or might be embedded within the payload, requiring a deeper look at the raw data.

```
▸ Frame 129: 305 bytes on wire (2440 bits), 305 bytes captured (2440 bits) on interface enp1s0, id 0
▸ Ethernet II, Src: CiscoMeraki_4f:00:01 (00:18:0a:4f:00:01), Dst: HonHaiPrecis_1d:9a:76 (a4:ae:11:1d:9a:76)
▸ Internet Protocol Version 4, Src: 128.119.245.12, Dst: 10.10.60.250
▸ Transmission Control Protocol, Src Port: 80, Dst Port: 35504, Seq: 1, Ack: 517, Len: 239
▾ Hypertext Transfer Protocol
  ▾ HTTP/1.1 304 Not Modified\r\n
       Response Version: HTTP/1.1
       Status Code: 304
       [Status Code Description: Not Modified]
       Response Phrase: Not Modified
    Date: Wed, 05 Feb 2025 09:10:33 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
    Connection: Keep-Alive\r\n
    Keep-Alive: timeout=5, max=100\r\n
    ETag: "80-62d5fa90d3bda"\r\n
    \r\n
    [Request in frame: 123]
    [Time since request: 0.228922510 seconds]
    [Request URI: /wireshark-labs/HTTP-wireshark-file1.html]
    [Full request URI: http://gaia.cs.umass.edu/wireshark-labs/HTTP-wireshark-file1.html]

0000  a4 ae 11 1d 9a 76 00 18  0a 4f 00 01 08 00 45 00   ·····v·· ·O····E·
0010  01 23 7e b5 40 00 27 06  17 98 80 77 f5 0c 0a 0a   ·#~·@·'· ···w····
0020  3c fa 00 50 8a b0 85 87  f3 06 63 ad aa f8 80 18   <··P···· ··c·····
0030  00 eb 2a cc 00 00 01 01  08 0a eb 81 e8 c4 84 70   ··*····· ·······p
0040  6e a7 48 54 54 50 2f 31  2e 31 20 33 30 34 20 4e   n·HTTP/1 .1 304 N
0050  6f 74 20 4d 6f 64 69 66  69 65 64 0d 0a 44 61 74   ot Modif ied··Dat
0060  65 3a 20 57 65 64 2c 20  30 35 20 46 65 62 20 32   e: Wed,  05 Feb 2
0070  30 32 35 20 30 39 3a 31  30 3a 33 33 20 47 4d 54   025 09:1 0:33 GMT
0080  0d 0a 53 65 72 76 65 72  3a 20 41 70 61 63 68 65   ··Server : Apache
0090  2f 32 2e 34 2e 36 20 28  43 65 6e 74 4f 53 29 20   /2.4.6 ( CentOS)
00a0  4f 70 65 6e 53 53 4c 2f  31 2e 30 2e 32 6b 2d 66   OpenSSL/ 1.0.2k-f
00b0  69 70 73 20 50 48 50 2f  37 2e 34 2e 33 33 20 6d   ips PHP/ 7.4.33 m
00c0  6f 64 5f 70 65 72 6c 2f  32 2e 30 2e 31 31 20 50   od_perl/ 2.0.11 P
00d0  65 72 6c 2f 76 35 2e 31  36 2e 33 0d 0a 43 6f 6e   erl/v5.1 6.3··Con
00e0  6e 65 63 74 69 6f 6e 3a  20 4b 65 65 70 2d 41 6c   nection:  Keep-Al
00f0  69 76 65 0d 0a 4b 65 65  70 2d 41 6c 69 76 65 3a   ive··Kee p-Alive:
0100  20 74 69 6d 65 6f 75 74  3d 35 2c 20 6d 61 78 3d    timeout =5, max=
0110  31 30 30 0d 0a 45 54 61  67 3a 20 22 38 30 2d 36   100··ETa g: "80-6
0120  32 64 35 66 61 39 30 64  33 62 64 61 22 0d 0a 0d   2d5fa90d 3bda"···

No.: 129 · Time: 10.980447615 · Source: 128.119.245.12 · Destination: 10.10.60.250 · Protocol: HTTP · Length: 305 · Info: HTTP/1.1 304 Not Modified
☑ Show packet bytes     Layout: Vertical (Stacked)
```

Task B: HTTP CONDITIONAL GET/response interaction

8. **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?**
   Answer: The "If-Modified-Since" header does not appear in the first request because it is sent in subsequent requests when the browser checks for updates.
   The browser only sends this header when it has cached the resource and wants to check whether the server has a newer version.

No.	Time	Source	Destination	Protocol	Length	Info
22	4.682162393	10.10.60.250	128.119.245.12	HTTP	497	GET /wireshark-labs/HTTP-wireshark-file2.html HTTP/1.1
27	4.919873917	128.119.245.12	10.10.60.250	HTTP	796	HTTP/1.1 200 OK (text/html)
30	4.996440575	10.10.60.250	128.119.245.12	HTTP	471	GET /favicon.ico HTTP/1.1
31	5.232132123	128.119.245.12	10.10.60.250	HTTP	550	HTTP/1.1 404 Not Found (text/html)

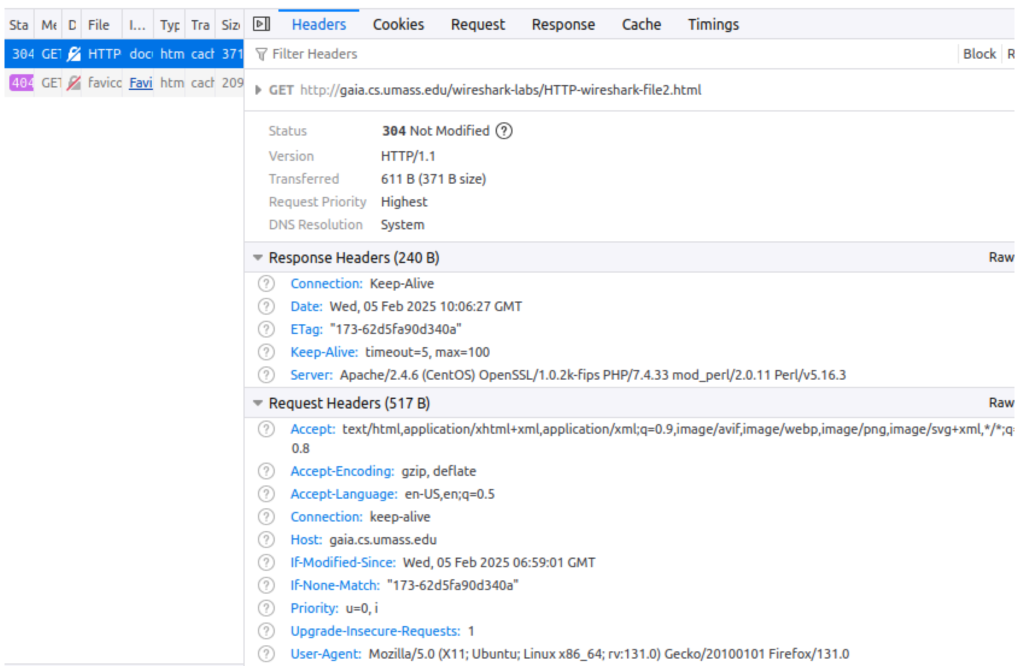
9. **Inspect the contents of the server response. Did the server explicitly return the contents of the file? How can you tell?**
   Answer: The server sends the contents, I saw a `200 OK` status along with the file in the response body. The absence of the `200 OK` status (for example, a `304 Not Modified` response) would indicate that the server did not send the file because it hasn't changed.

27	4.919873917	128.119.245.12	10.10.60.250	HTTP	796	HTTP/1.1 200 OK (text/html)

10. **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?**

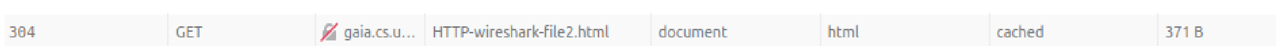
Answer: The second request should include an `If-Modified-Since` header with the date and time from the first request's `Last-Modified` header.
This indicates the browser's cached version of the resource, and it allows the server to decide if the content has changed.

`If-Modified-Since:` header with the same date and time to indicate it has a cached version of the resource. The server can then use this header to determine whether the resource has been modified since that timestamp and respond accordingly.



11. **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.**

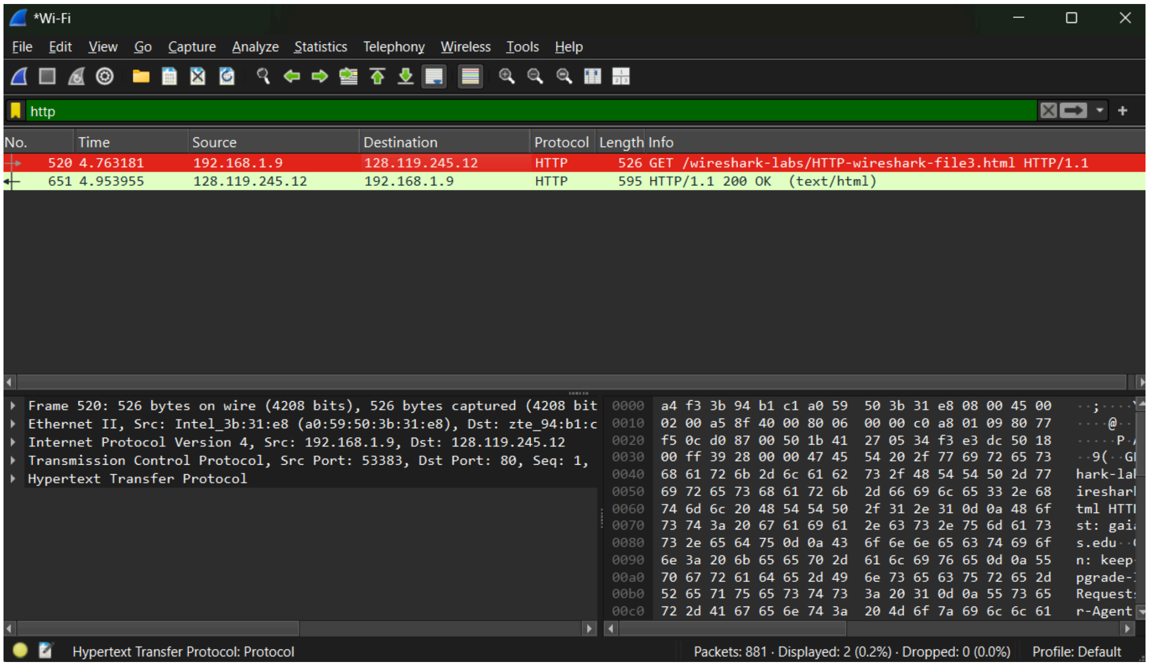
Answer: The response is `304 Not Modified` if the content hasn't changed, indicating that the file was not re-sent. The server would return a `304 Not Modified` if the file hasn't changed since the last request.



Task C: Retrieving Long Documents

12. **How many HTTP GET request messages were sent by your browser?**

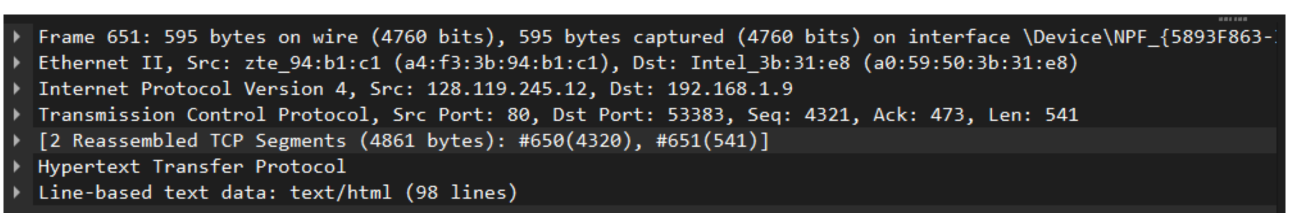
Answer: There is only one HTTP GET request for the long document, which requests the entire HTML file. The GET request is sent for the HTML file, and then the server responds by sending the content in multiple TCP segments.



13. **How many data-containing TCP segments were needed to carry the single HTTP response?**

    Answer: To determine how many TCP segments were needed to carry the HTTP response, I looked at the packets labeled as "TCP segment of a reassembled PDU" in Wireshark. Each of these segments carries a part of the HTTP response, and the total number of such segments tells me how many were needed to transfer the entire response. This happens when the response is large and is split into multiple TCP segments.

    In this case, I found that **2 TCP segments** were required to carry the HTTP response. This typically happens when the server sends large files or a large amount of data, which gets broken down into multiple segments for transmission. The client then reassembles these segments to recreate the full HTTP response.



```
864 9.959045   128.119.245.12   192.168.1.9   TCP    54 80 → 53383 [FIN, ACK] Seq=4862 Ack=473 Win=30336 Len=0
651 4.953955   128.119.245.12   192.168.1.9   HTTP   595 HTTP/1.1 200 OK  (text/html)
650 4.953955   128.119.245.12   192.168.1.9   TCP    4374 80 → 53383 [ACK] Seq=1 Ack=473 Win=30336 Len=4320 [TCP PDU reassembled in 651]
649 4.953195   128.119.245.12   192.168.1.9   TCP    54 80 → 53383 [ACK] Seq=1 Ack=473 Win=30336 Len=0
521 4.763414   128.119.245.12   192.168.1.9   TCP    66 80 → 53382 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1440 SACK_PERM WS=128
516 4.761371   128.119.245.12   192.168.1.9   TCP    66 80 → 53383 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1440 SACK_PERM WS=128
```

14. **What is the status code and phrase associated with the response to the HTTP GET request?**

    Answer: The status code is `200 OK` since the request was successful. This indicates that the server is sending the full content of the HTML file.

```
651 4.953955   128.119.245.12   192.168.1.9   HTTP   595 HTTP/1.1 200 OK  (text/html)
```

15. **Is there any HTTP header information in the transmitted data associated with TCP segmentation?**

    Answer: The HTTP headers are typically only present in the first segment, while the following segments only contain the body of the response. After the initial headers are sent, subsequent segments only contain the entity-body of the response.

    Yes, there is HTTP header information in the transmitted data associated with TCP segmentation. **Wireshark reassembles these segments** and shows the complete HTTP response in Frame 651. The HTTP headers were likely in **Segment #650**, but since Wireshark reconstructs the full response, it displays them in Frame 651.

```
Frame 651: 595 bytes on wire (4760 bits), 595 bytes captured (4760 bits) on interface \Device\NPF_{5893F863-32BE-47F7-ABFD-F8C514C73656}, id 0
Ethernet II, Src: zte_94:b1:c1 (a4:f3:3b:94:b1:c1), Dst: Intel_3b:31:e8 (a0:59:50:3b:31:e8)
Internet Protocol Version 4, Src: 128.119.245.12, Dst: 192.168.1.9
Transmission Control Protocol, Src Port: 80, Dst Port: 53383, Seq: 4321, Ack: 473, Len: 541
[2 Reassembled TCP Segments (4861 bytes): #650(4320), #651(541)]
Hypertext Transfer Protocol
  HTTP/1.1 200 OK\r\n
  Date: Wed, 05 Feb 2025 15:55:28 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: Wed, 05 Feb 2025 06:59:01 GMT\r\n
  ETag: "1194-62d5fa90cf972"\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: 520]
  [Time since request: 0.190774000 seconds]
  [Request URI: /wireshark-labs/HTTP-wireshark-file3.html]
  [Full request URI: http://gaia.cs.umass.edu/wireshark-labs/HTTP-wireshark-file3.html]
  File Data: 4500 bytes
Line-based text data: text/html (98 lines)
```

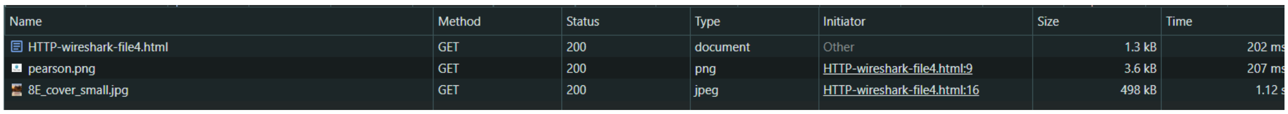
Task D: HTML Documents with Embedded Objects

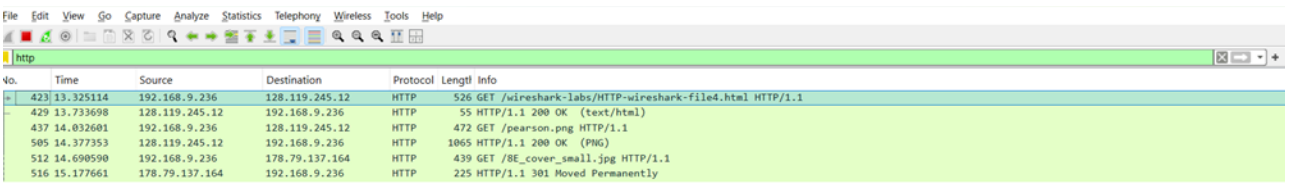
16. **How many HTTP GET request messages were sent by your browser? To which Internet addresses were these GET requests sent?**

    Answer: I observed that my browser sent **3 HTTP GET requests**:

1. One for the base HTML file to `gaia.cs.umass.edu`.

2. The other two for the images: `pearson.png` from `gaia.cs.umass.edu` and `8E_cover_small.jpg` from `manic.cs.umass.edu`.





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

Answer: From my observation, I noticed two separate GET requests being sent at roughly the same time but with different IP addresses. This suggests that the images were likely downloaded in parallel. Modern web browsers typically download objects in parallel to speed up page loading. This is achieved by opening multiple connections to the servers hosting the embedded objects.

By analyzing the timestamps of the HTTP GET requests and their corresponding responses, I could tell that the requests for the images overlapped or were close together, indicating parallel downloads. Additionally, since the images were hosted on different servers (gaia.cs.umass.edu and manic.cs.umass.edu), it was likely that the browser utilized multiple connections to fetch the images in parallel, as is typical with modern web browsers.

- If the timestamps for the image GET requests overlap or are close together, the browser downloaded the images **in parallel**.
- If the timestamps indicate that one image request and response was completed before the next image request was sent, the downloads occurred **serially**.

Wireshark · Follow TCP Stream (tcp.stream eq 28) · Wi-Fi

```
GET /wireshark-labs/HTTP-wireshark-file4.html HTTP/1.1
Host: gaia.cs.umass.edu
Connection: keep-alive
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/132.0.0.0 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9


HTTP/1.1 200 OK
Date: Wed, 05 Feb 2025 15:51:45 GMT
Server: Apache/2.4.6 (CentOS) OpenSSL/1.0.2k-fips PHP/7.4.33 mod_perl/2.0.11 Perl/v5.16.3
Last-Modified: Wed, 05 Feb 2025 06:59:01 GMT
ETag: "3ae-62d5fa90d2c3a"
Accept-Ranges: bytes
Content-Length: 942
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html; charset=UTF-8

<html>
<head>
<title>Lab2-4 file: Embedded URLs</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
</head>

<body bgcolor="#FFFFFF" text="#000000">

<p>
<img src="http://gaia.cs.umass.edu/pearson.png" WIDTH="140" HEIGHT="82" > </p>
<p>This little HTML file is being served by gaia.cs.umass.edu.
It contains two embedded images. The image above, also served from the
gaia.cs.umass.edu web site, is the logo of our publisher, Pearson.
The image of our 8th edition book cover below is stored at, and served from,
  a  WWW server kurose.cslash.net in France:</p>
<p align="left"><img src="http://kurose.cslash.net/8E_cover_small.jpg"
                            width="168" height="220"></p>
And while we have your attention, you might want to take time to check out the
                            available open resources for this book at
                            <a href="http://gaia.cs.umass.edu/kurose_ross"> http://gaia.cs.umass.edu/kurose_ross</a>.
```

Task E: HTTP Authentication

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

    Answer: When I first enter the URL in my browser, it sends an initial HTTP GET request to the server. The server will respond with a `401 Unauthorized` status code, indicating that authentication is required. The server includes the `WWW-Authenticate` header in its response, which specifies the authentication method (e.g., Basic Authentication) and a realm (a string that defines the protected area).

HTTP-wireshark-file5.html	GET	401	document	Other	716 B	195 ms

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

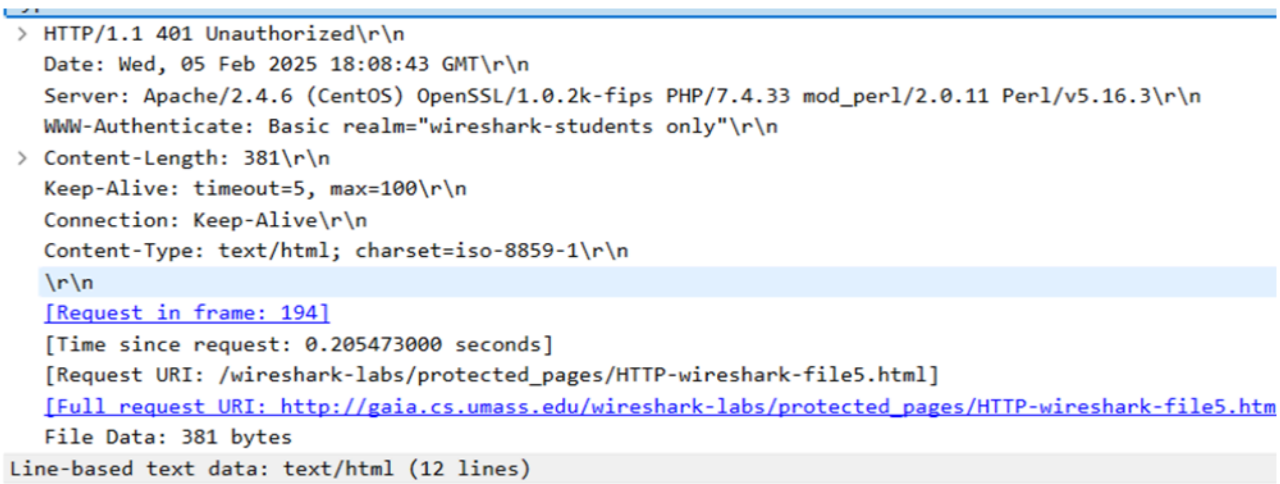
http

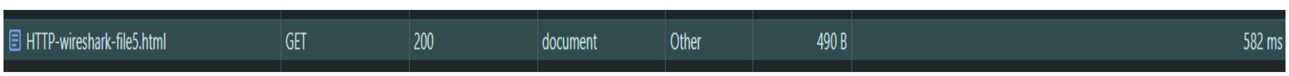
No.	Time	Source	Destination	Protocol	Length	Info
190	19.593561	192.168.9.236	128.119.245.12	HTTP	542	GET /wireshark-labs/protected_pages/HTTP-wireshark-file5.html HTTP/1.1
234	19.925613	128.119.245.12	192.168.9.236	HTTP	771	HTTP/1.1 401 Unauthorized (text/html)
391	43.047641	192.168.9.236	128.119.245.12	HTTP	627	GET /wireshark-labs/protected_pages/HTTP-wireshark-file5.html HTTP/1.1
392	43.757213	128.119.245.12	192.168.9.236	HTTP	544	HTTP/1.1 200 OK (text/html)

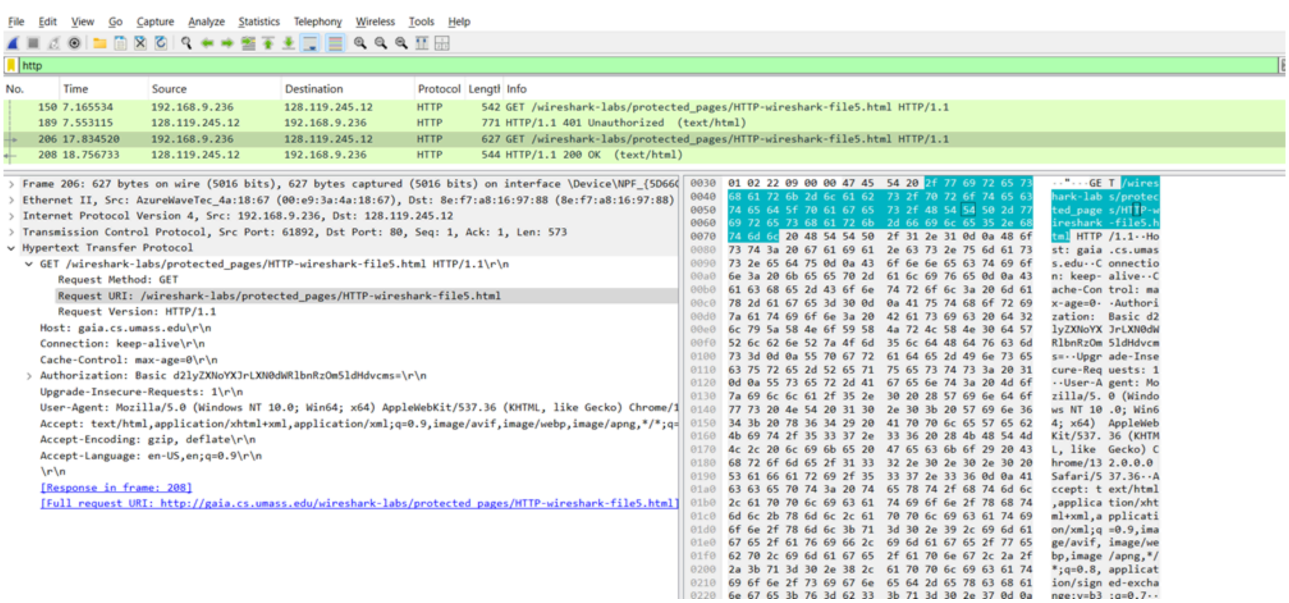
    The server challenges the client to provide credentials to access the resource.

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

Answer: After receiving the `401 Unauthorized` response, the browser prompts the user to enter a username and password via a pop-up dialog box.Once the credentials are entered, the browser re-sends the HTTP GET request, but this time it includes the `Authorization` header with the credentials encoded in Base64 format.This header provides the credentials for HTTP basic authentication.







Reasons:

- The server verifies the credentials by decoding the Base64 string.
- If the credentials are valid, the server responds with a **200 OK** status code and sends the requested content (e.g., the HTML page).
- If the credentials are invalid, the server would again respond with a `401 Unauthorized` status code, and the browser would prompt the user to re-enter their credentials.

Observations on Security:

- The **username and password are not encrypted**, only encoded in Base64. Base64 is an encoding mechanism, not an encryption algorithm, meaning anyone with access to the encoded string can easily decode it to retrieve the credentials.
- To demonstrate this, you can use any online Base64 decoder to decode the string `d2lyZXNoYXJrLXN0dWRlbnRzOm5ldHdvcms=`, which translates to:
  - Username: `wireshark-students`
  - Password: `network`

Security Concerns and Best Practices:

Risk of Eavesdropping: Since HTTP is an unencrypted protocol, the credentials can be intercepted by anyone monitoring the network traffic using tools like Wireshark. This makes HTTP Basic Authentication insecure on its own.

Mitigating Measures:

- Use HTTPS: Encrypting the communication channel using HTTPS ensures that even if the traffic is intercepted, the credentials remain secure.
- Use stronger authentication mechanisms: Modern websites often rely on token-based authentication (e.g., OAuth) or other secure protocols to protect user credentials.

20. **What does the "Connection: close" and "Connection: Keep-alive" header field imply in HTTP protocol? When should one be used over the other?**

    Answer: The Connection header field in HTTP is used to control the behavior of network connections between a client (browser) and a server. This header helps define whether the connection should be terminated after a single HTTP request/response or kept open for additional requests.

    ### 1. Connection: close

- **Definition:** The Connection: close header indicates that the connection between the client and server will be **closed** immediately after the current HTTP request/response is completed.

- **Key Characteristics:**
  - After the server sends the requested data to the client, it closes the TCP connection.
  - The client must establish a new connection to the server for every subsequent HTTP request.
  - This was the default behavior in **HTTP/1.0**, where each HTTP transaction (request/response pair) occurred over a separate TCP connection.
- **Advantages:**
  - Simple to implement: Each request is isolated, so there's no need to manage persistent connections.
  - Suitable for environments with low traffic or for single-request operations.
- **Disadvantages:**
  - High overhead: Establishing and tearing down a new TCP connection for each request is resource-intensive and adds latency.
  - Inefficient for modern web pages, which often require multiple resources (e.g., CSS, JavaScript, images).
- **Use Cases:**
  - Applications or services where only a single request is expected, and maintaining a persistent connection is unnecessary.
  - Situations where the server or client has limited resources and cannot handle the overhead of managing many open connections.

2. Connection: Keep-Alive

- **Definition:** The Connection: Keep-Alive header is used to keep the connection between the client and server **open** after the initial request/response transaction. This allows multiple HTTP requests to be sent over the same connection without repeatedly establishing and tearing it down.
- **Key Characteristics:**
  - The client and server reuse the same TCP connection for multiple requests/responses.
  - This behavior is the default in **HTTP/1.1**, which assumes persistent connections unless explicitly told otherwise via the Connection: close header.
  - Keep-Alive can include additional parameters, such as:
    - timeout: Specifies how long the connection should remain open.
    - max: Specifies the maximum number of requests allowed over the connection.
- **Advantages:**
  - **Reduced latency:** Reusing the same TCP connection eliminates the need to re-establish new connections for each request.
  - **Lower resource usage:** Less overhead in setting up and tearing down TCP connections.
  - **Improved performance:** Essential for modern web pages, which often require many resources (e.g., CSS, JavaScript, images) to be fetched in quick succession.

- **Disadvantages:**
    - Requires the server and client to manage open connections, which could consume resources if too many are kept alive unnecessarily.
    - Can lead to resource exhaustion if connections are left open for too long without proper management.
- **Use Cases:**
    - High-traffic websites or applications where multiple resources (e.g., web assets) are requested in a short time.
    - Real-time applications or APIs that require frequent communication between the client and server.

When to Use Connection: close:

1. **Single-request transactions:** A client or application only sends one HTTP request to the server (e.g., API ping).
2. **Resource-limited servers:** Servers with limited memory and processing power might prefer to close connections immediately to free resources.
3. **Security concerns:** If persistent connections pose risks (e.g., long-lived connections from unauthenticated users), closing them immediately may be preferred.

When to Use Connection: Keep-Alive:

1. **Multi-resource web pages:** Modern websites often require multiple resources like images, CSS files, and JavaScript files, which can all be fetched over the same connection.
2. **High-performance applications:** Persistent connections reduce latency and are better suited for high-traffic applications or APIs.
3. **Real-time communication:** Scenarios like live chat or streaming applications benefit from maintaining open connections for quick data exchanges.

CONCLUSION:	In this experiment, I learned how to use Wireshark to capture and analyze HTTP traffic, including GET requests and responses. I explored the process of retrieving simple and large HTML files, handling conditional GET requests, and examining HTTP headers and status codes. Additionally, I investigated how web browsers interact with servers for content retrieval and authentication. Overall, this experiment enhanced my understanding of the HTTP protocol and how data is exchanged over the web.