Author: Santiago Bermudez

CSC 138

Section 4

1. The Basic HTTP GET/response interaction

Let’s begin our exploration of HTTP by downloading a very simple HTML file - one that is very short, and contains no embedded objects. Do the following:

1. Start up your web browser.

2. Start up the Wireshark packet sniffer, as described in the Introductory lab (but don’t yet begin packet capture). Enter “http” (just the letters, not the quotation marks) in the display-filter-specification window, so that only captured HTTP messages will be displayed later in the packet-listing window. (We’re only interested in the HTTP protocol here, and don’t want to see the clutter of all captured packets).

3. Wait a bit more than one minute (we’ll see why shortly), and then begin Wireshark packet capture.

4. Enter the following to your browser

http://gaia.cs.umass.edu/wireshark-labs/HTTP-wireshark-file1.html

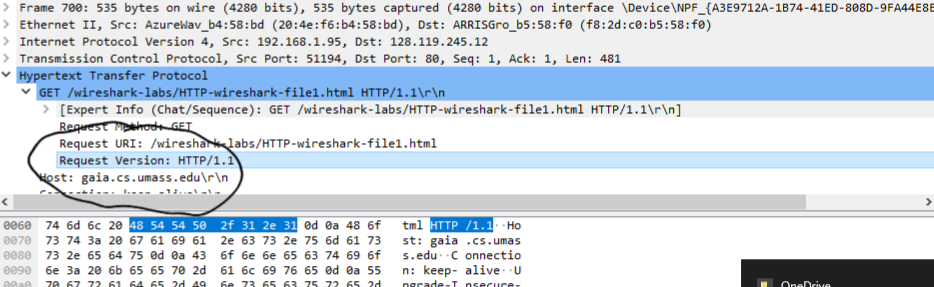
Your browser should display the very simple, one-line HTML file.

5. Stop Wireshark packet capture.

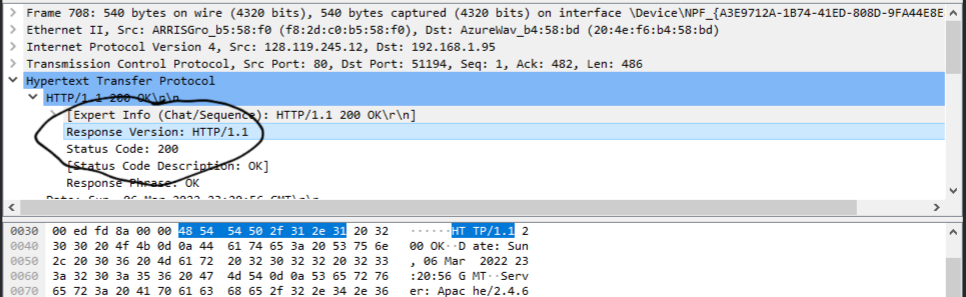
By looking at the information in the HTTP GET and response messages, answer the following questions. When answering the following questions, you should print out the GET and response messages (see the introductory Wireshark lab for an explanation of how to do this) and indicate where in the message you’ve found the information that answers the following questions. When you hand in your assignment, annotate the output so that it’s clear where in the output you’re getting the information for your answer (e.g., for our classes, we ask that students markup paper copies with a pen, or annotate electronic copies with text in a colored font).

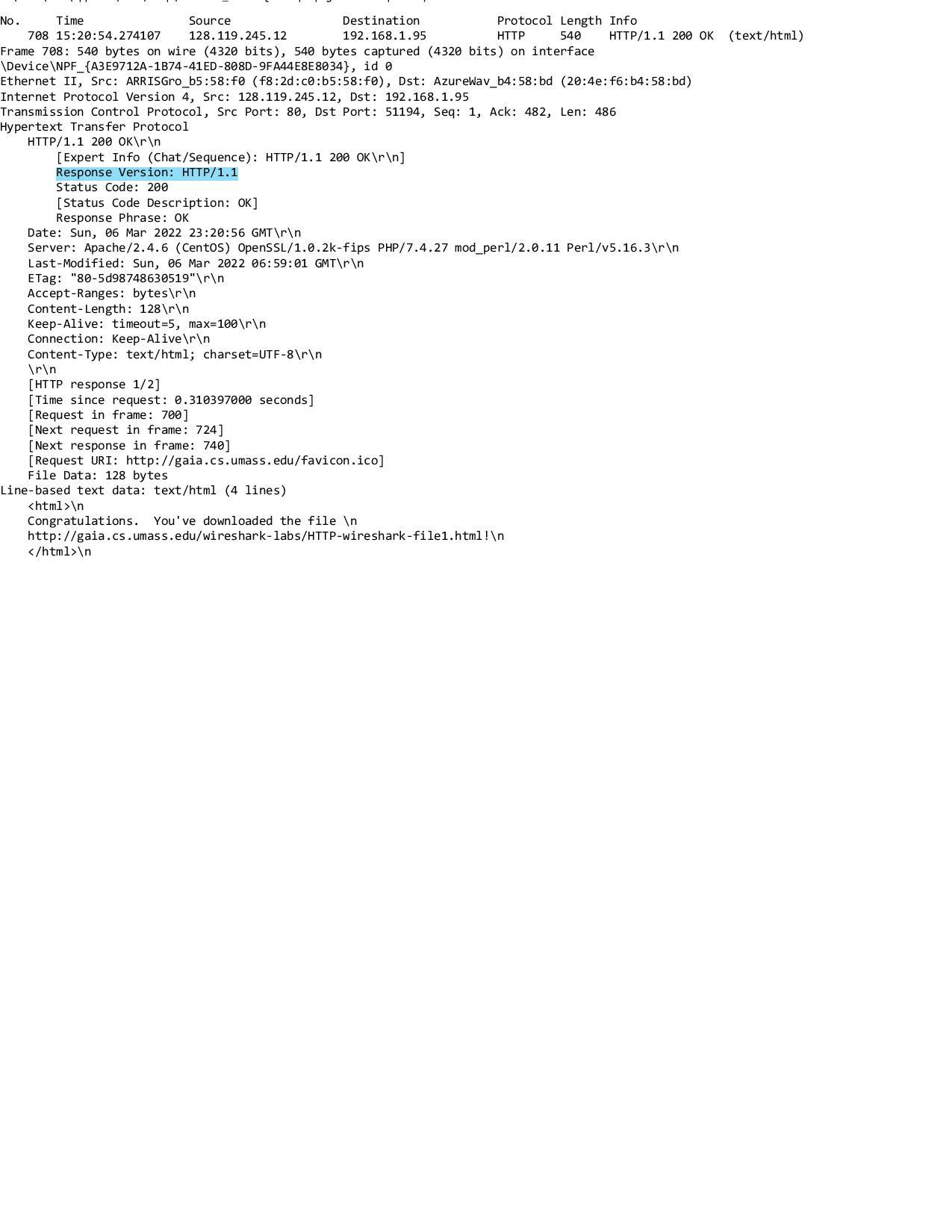
1. Is your browser running HTTP version 1.0 or 1.1? What version of HTTP is the server running?

My browser appears to be running HTTP version 1.1. The server is running HTTP version 1.1. I got this answer by looking under the first subtab of GET and response under the Hypertext Transfer Protocol information display in the packet-header details window. Please note that for all the questions, you will see the proof for my answers circled in a screenshot or highlighted in the printed page (\*Possibly even both). Please note that drawing circles on a touchscreen is hard, so they may not be perfect, but you get the point.





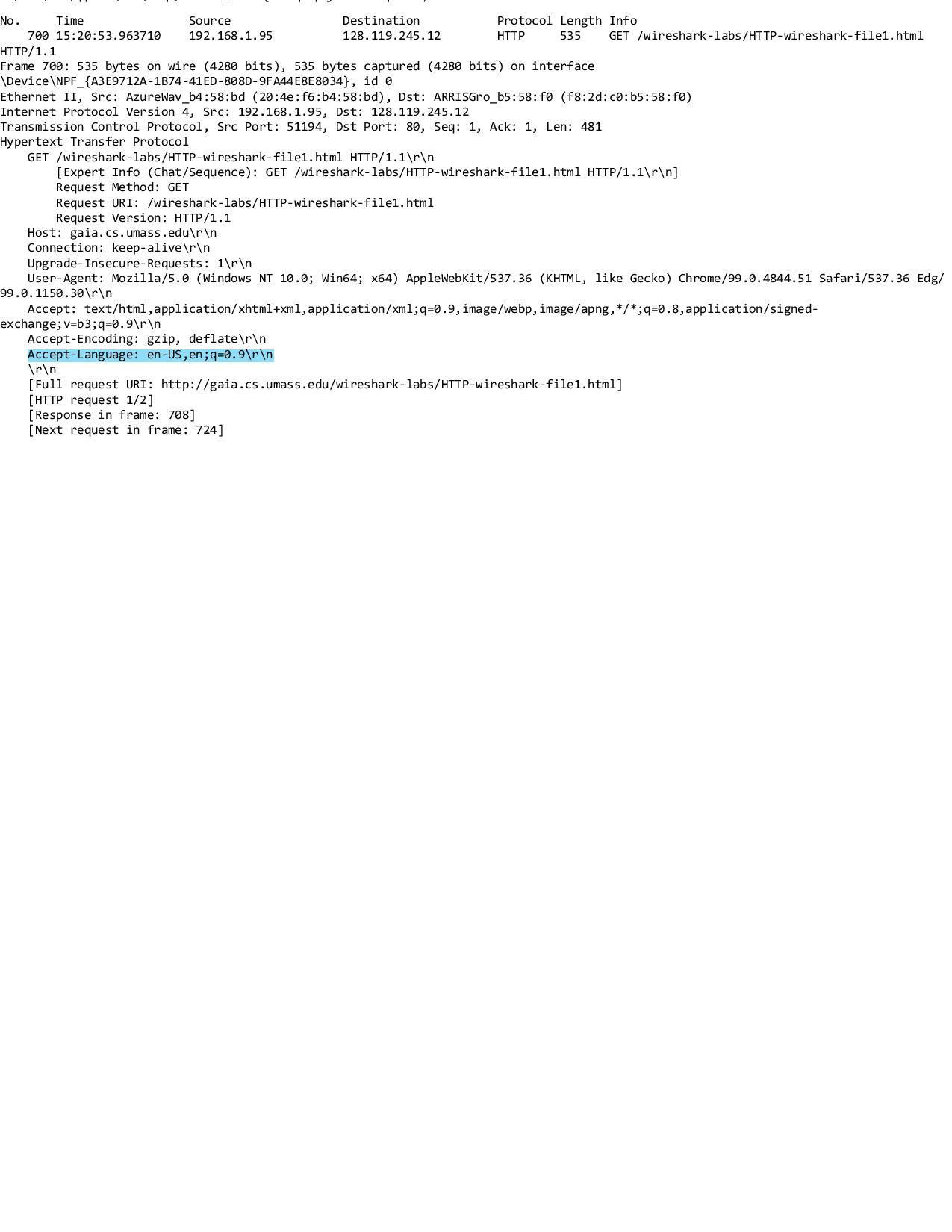




2. What languages (if any) does your browser indicate that it can accept to the server?

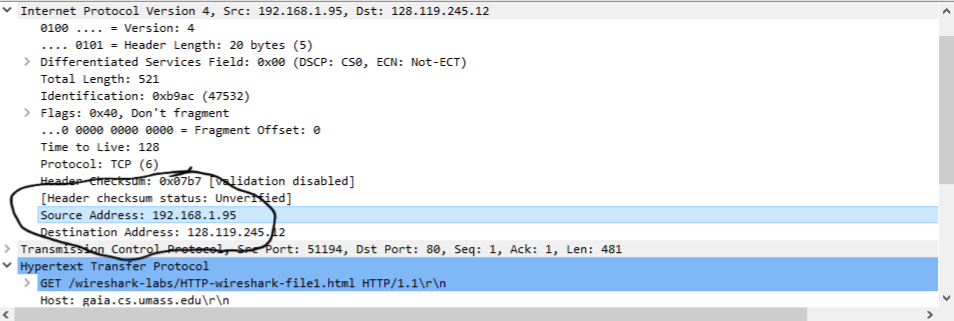
After looking in the GET message to find out what my computer can accept to the server, it seems that the languages that my browser accepts are en-us, en.

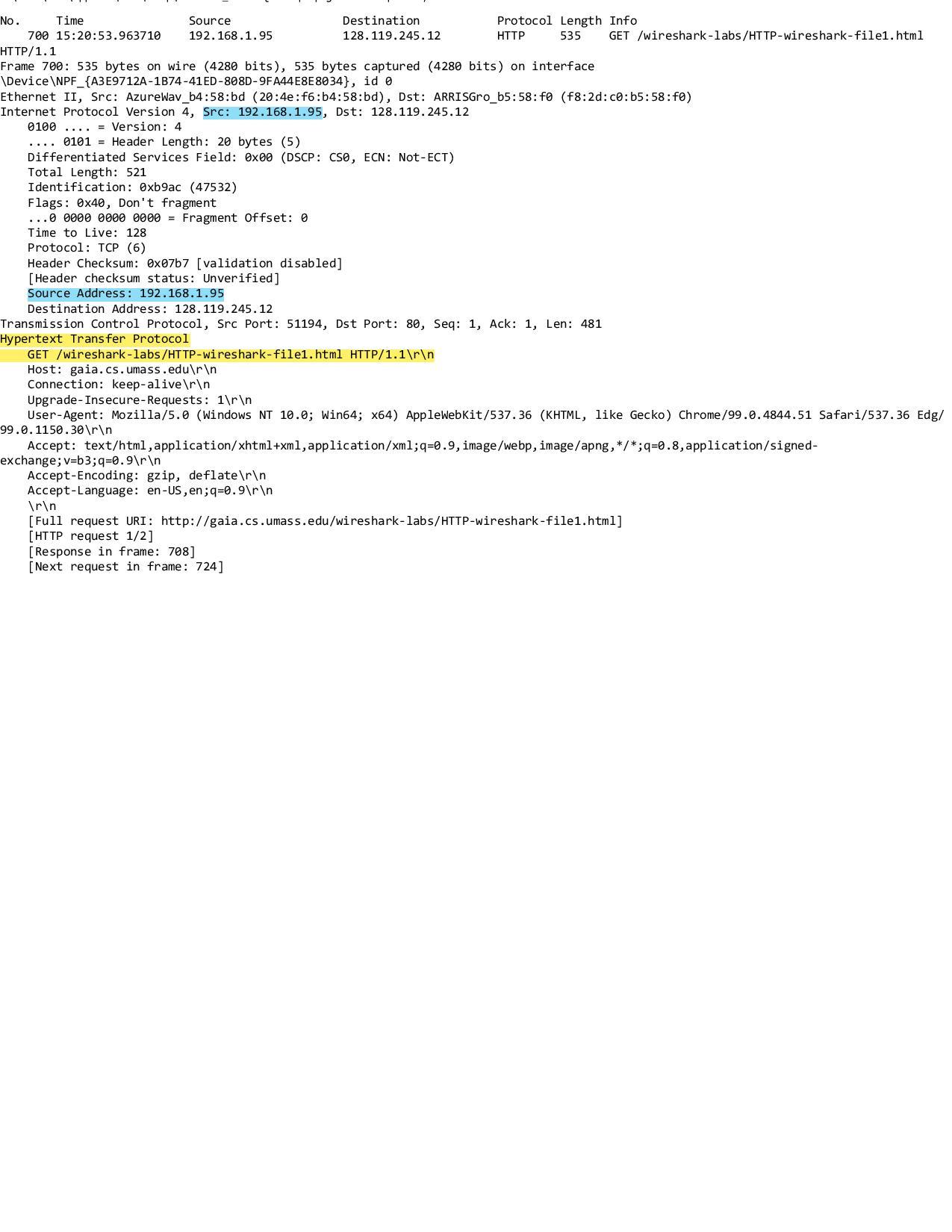


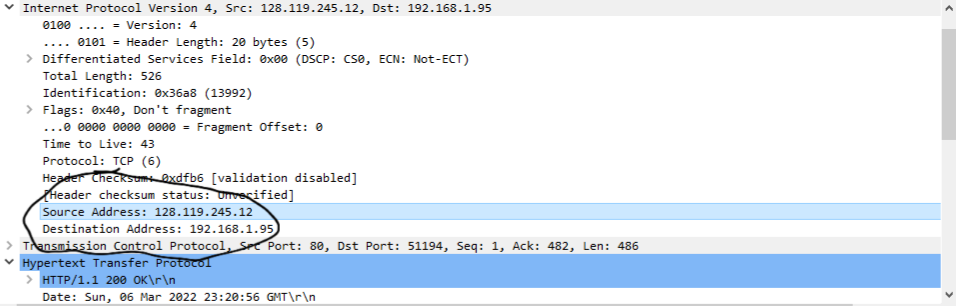


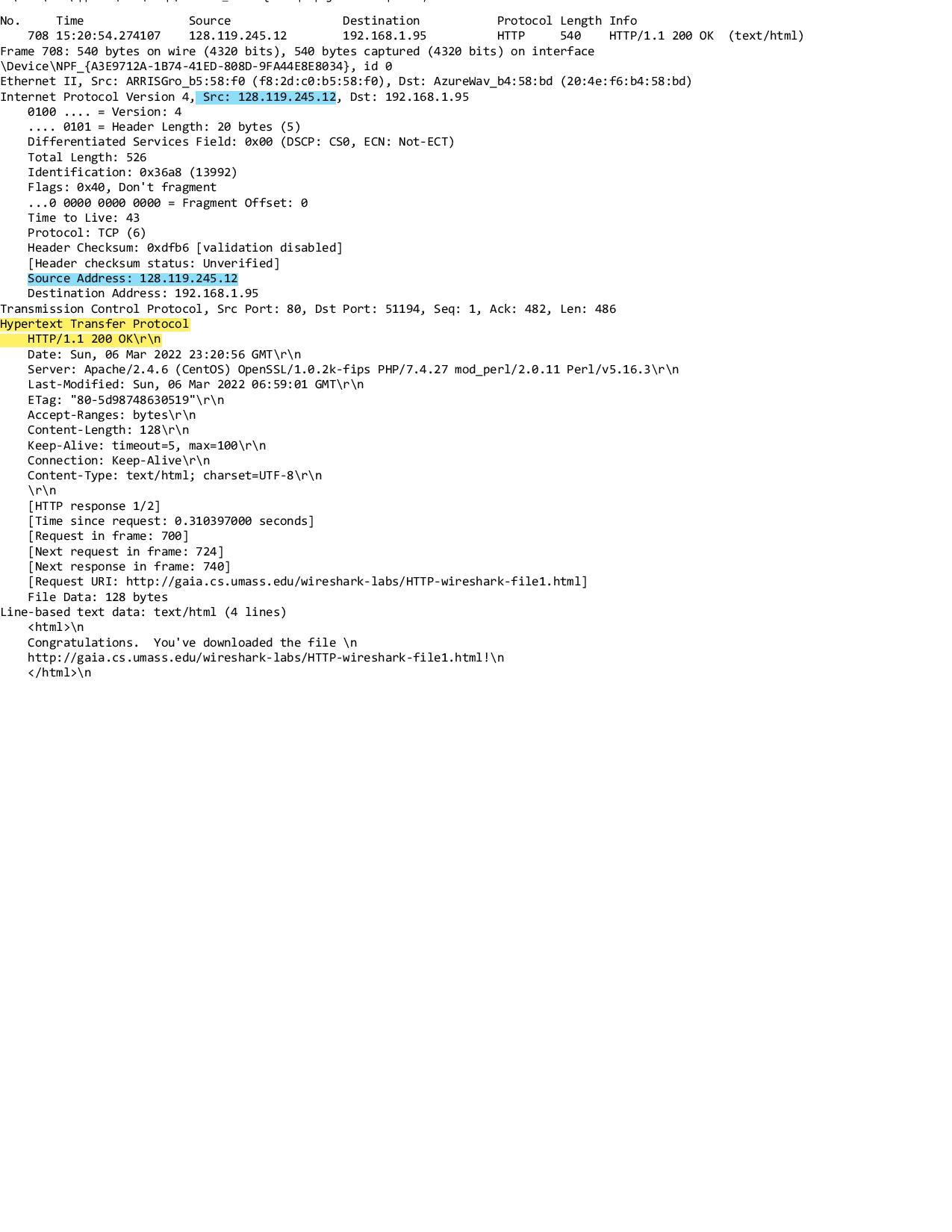
3. What is the IP address of your computer? Of the gaia.cs.umass.edu server?

The IP address of my computer is 192.168.1.95. The IP address of the gaia.cs.umass.edu server is 128.119.245.12.



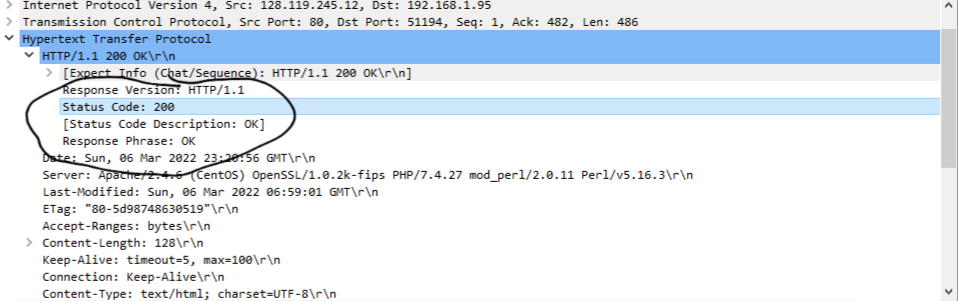






4. What is the status code returned from the server to your browser?

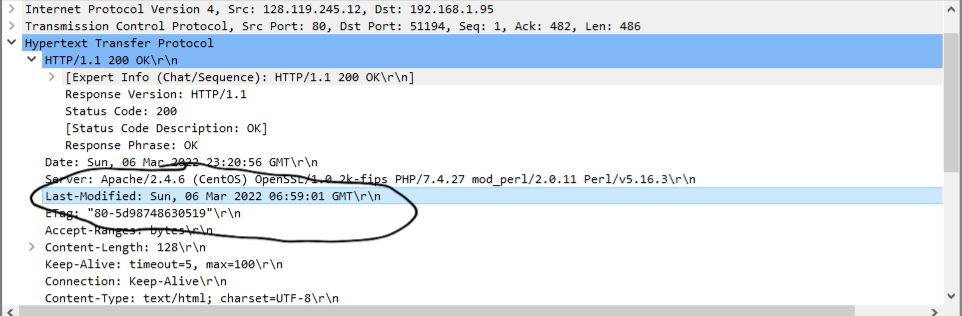
The status code returned from the server to my browser is a 200 OK status code.





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

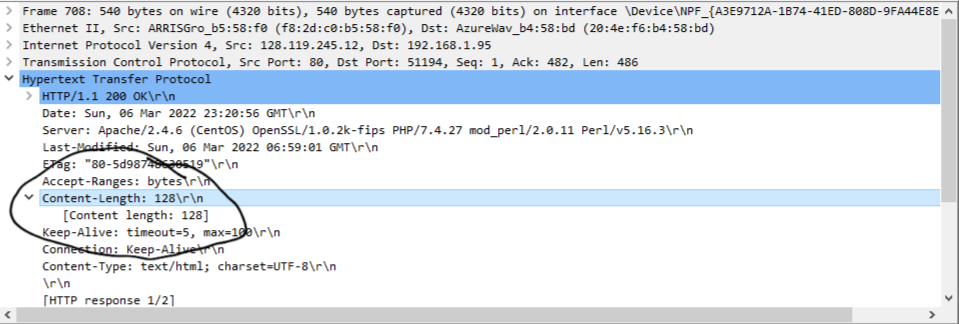
The HTML file was last modified at Sun, 06 Mar 2022 06:59:01 GMT.

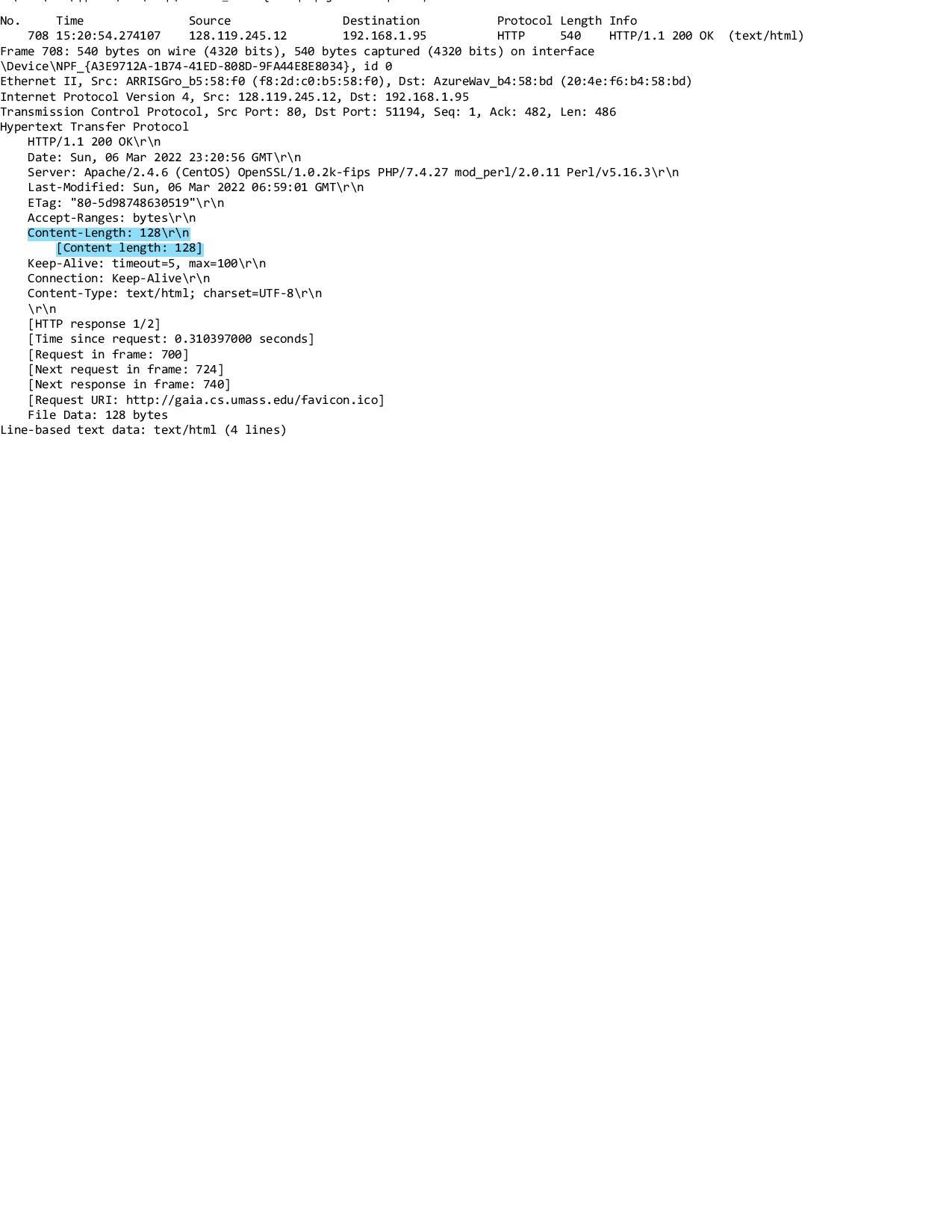




6. How many bytes of content are being returned to your browser?

From the response message information, it says that the content-length is 128. This means that 128 bytes of content are being returned to my browser.



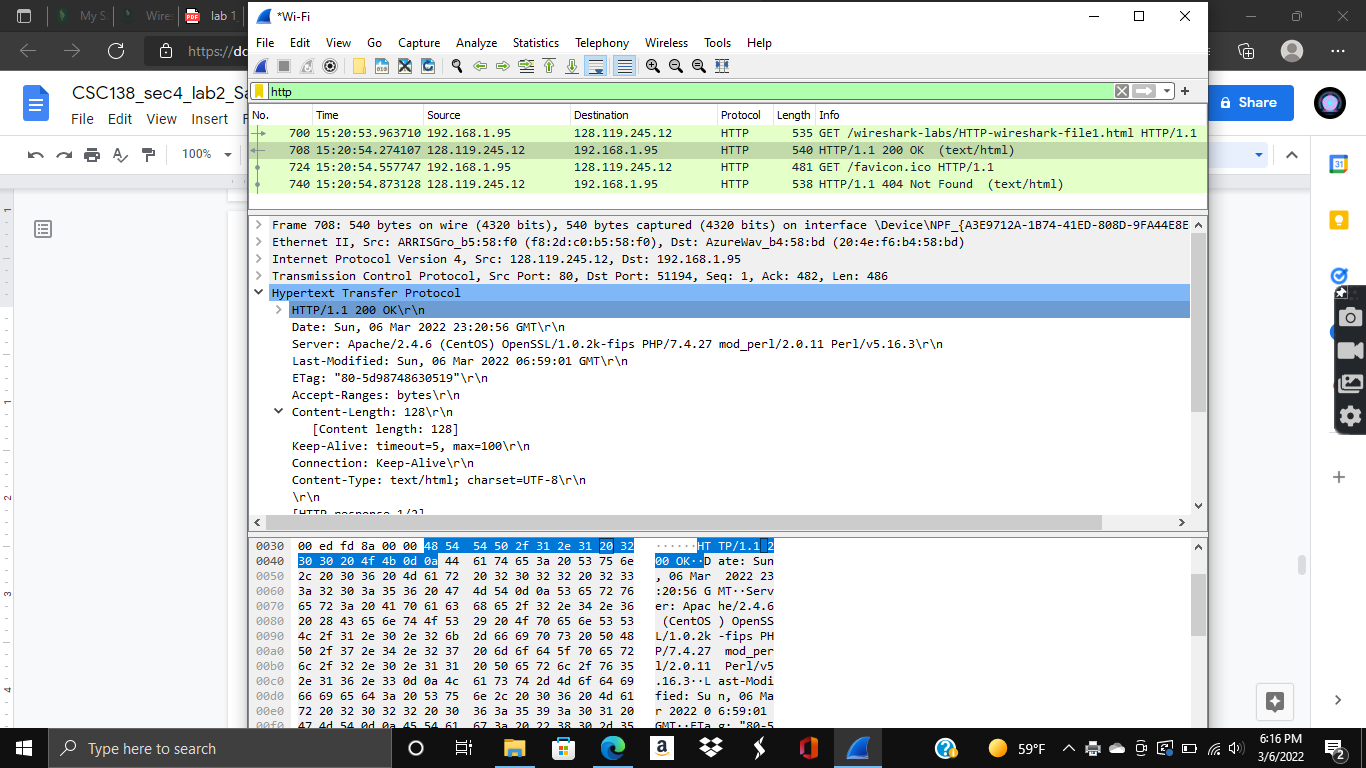


7. By inspecting the raw data in the packet content window, do you see any headers

within the data that are not displayed in the packet-listing window? If so, name one.

No, I do not see any headers within the data that are not displayed in the packet-listing window.

\*Normally, there would be an annotated screenshot or highlight on a printed page, but in this case there is nothing to show, in that nothing is found. There is still a screenshot and printed page, but no annotations.





In your answer to question 5 above, you might have been surprised to find that the document you just retrieved was last modified within a minute before you downloaded the document. That’s because (for this particular file), the gaia.cs.umass.edu server is setting the file’s last-modified time to be the current time, and is doing so once per minute. Thus, if you wait a minute between accesses, the file will appear to have been recently modified, and hence your browser will download a “new” copy of the document.

2. The HTTP CONDITIONAL GET/response interaction

Recall from Section 2.2.6 of the text, that most web browsers perform object caching and thus perform a conditional GET when retrieving an HTTP object. Before performing the steps below, make sure your browser’s cache is empty. (To do this under Firefox, select *Tools->Clear Recent History* and check the Cache box, or for Internet Explorer, select *Tools->Internet Options->Delete File;* these actions will remove cached files from your browser’s cache.) Now do the following:

• Start up your web browser, and make sure your browser’s cache is cleared, as discussed above.

• Start up the Wireshark packet sniffer

• Enter the following URL into your browser

http://gaia.cs.umass.edu/wireshark-labs/HTTP-wireshark-file2.html Your browser should display a very simple five-line HTML file.

• Quickly enter the same URL into your browser again (or simply select the refresh button on your browser)

• Stop Wireshark packet capture, and enter “http” in the display-filter-specification window, so that only captured HTTP messages will be displayed later in the packet-listing window.

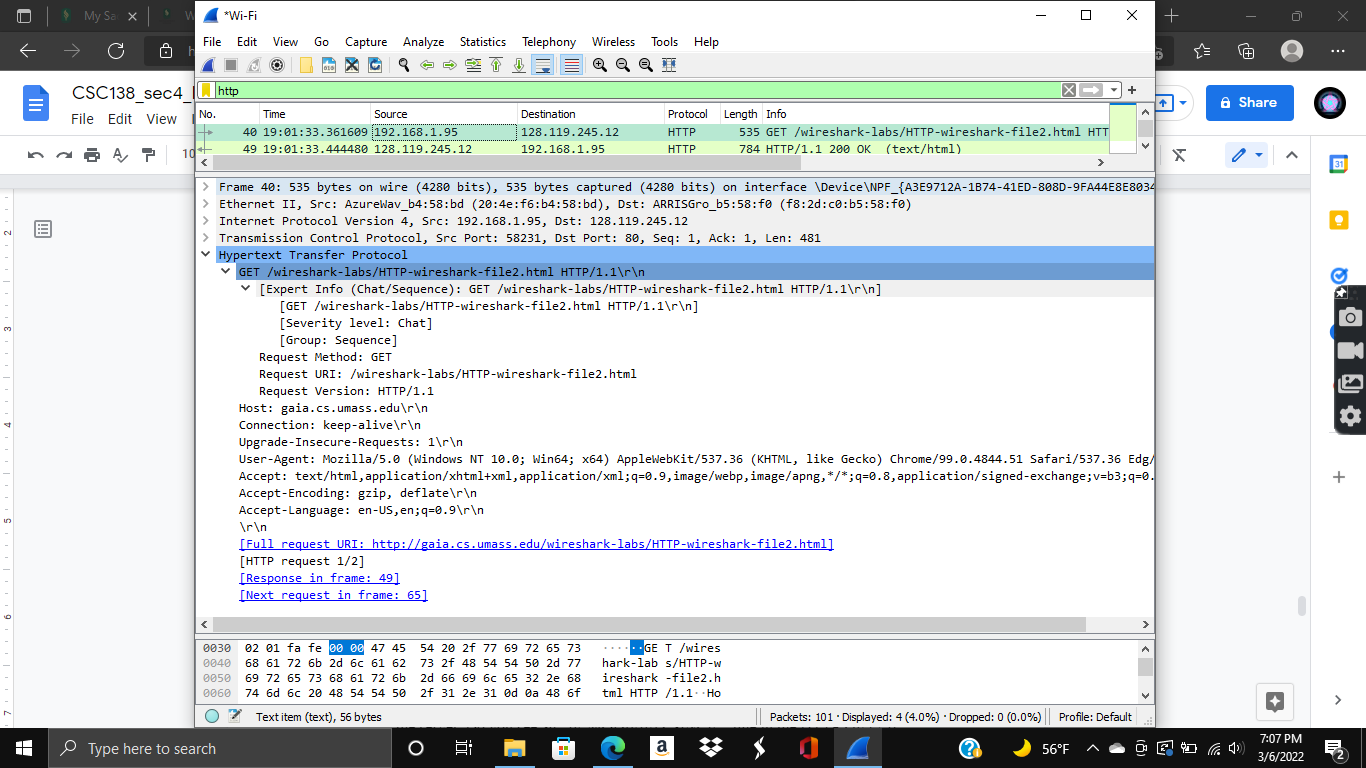
• (*Note:* If you are unable to run Wireshark on a live network connection, you can use the http-ethereal-trace-2 packet trace to answer the questions below; see footnote 1. This trace file was gathered while performing the steps above on one of the author’s computers.)

Answer the following questions:

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?

No, I have not seen an “IF-MODIFIED-SINCE” line. This line would only show if the site had been visited before.

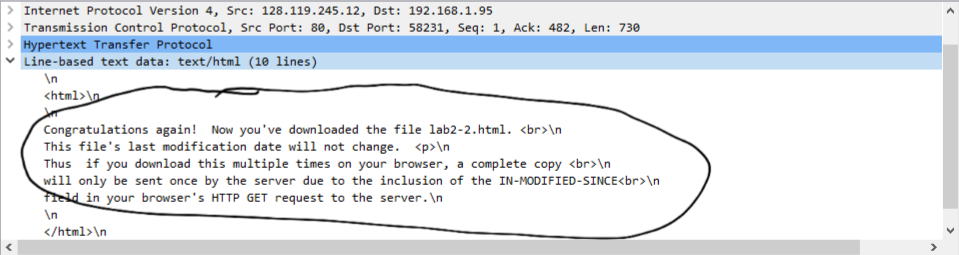
\*It’s the same case as with question #7.

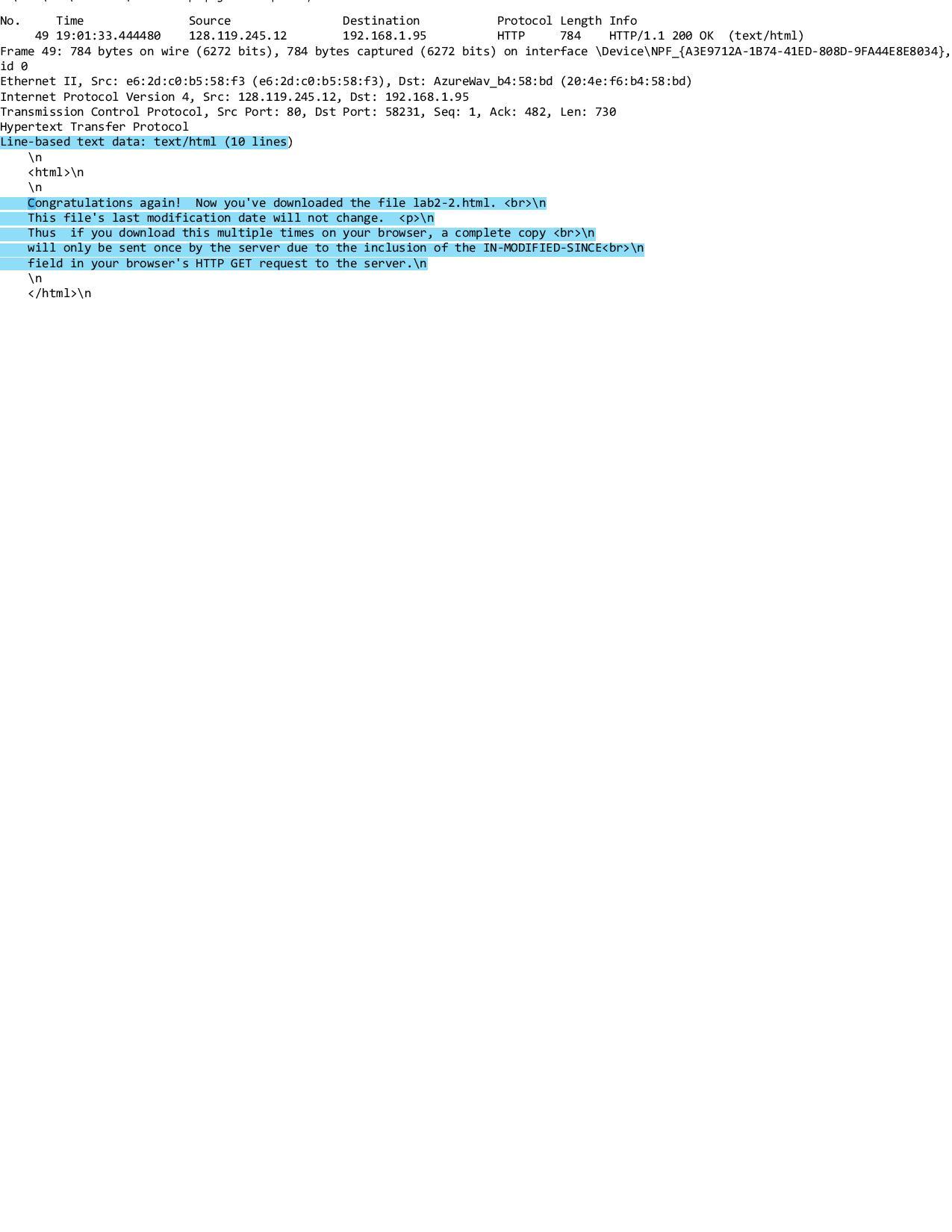




9. Inspect the contents of the server response. Did the server explicitly return the contents of the file? How can you tell?

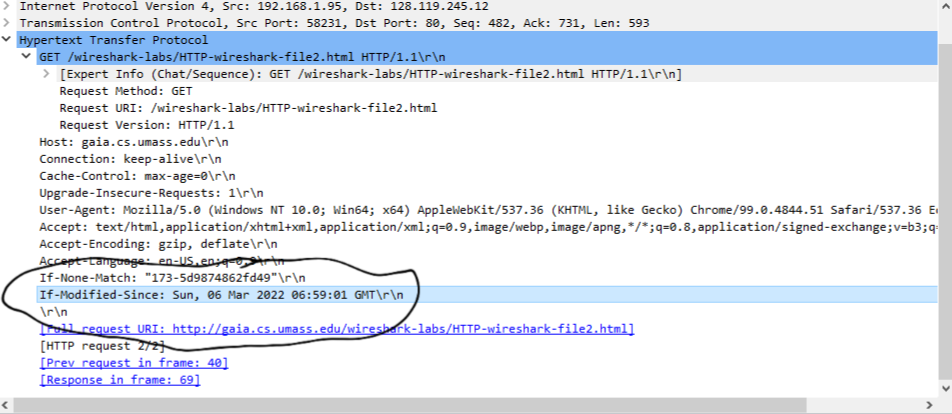
Yes, the server did in fact explicitly return the contents of the file. One can tell because of the line-based text data.

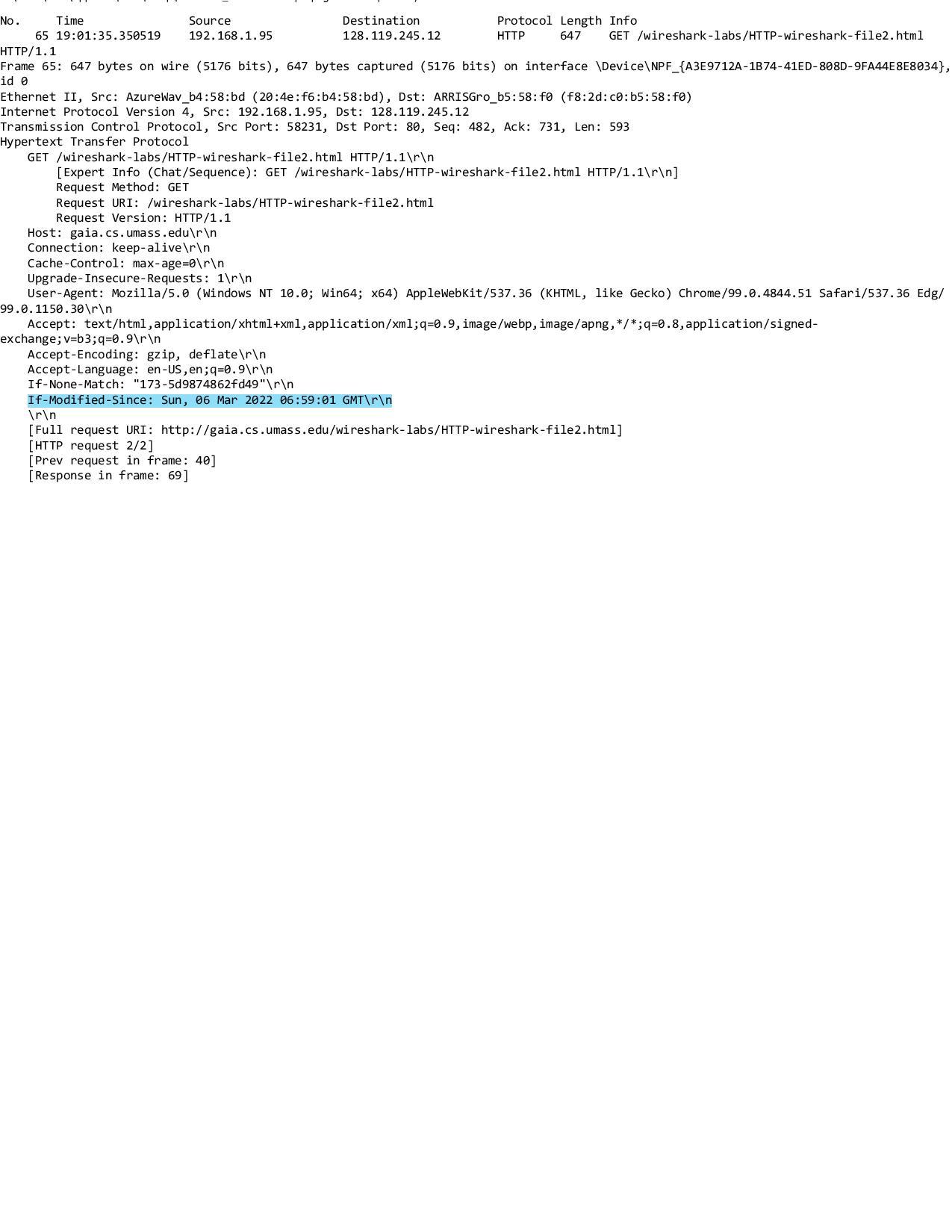




10. Now inspect the contents of the second HTTP GET request from your browser to the server. Do you see an “IF-MODIFIED-SINCE:” line in the HTTP GET? If so, what information follows the “IF-MODIFIED-SINCE:” header?

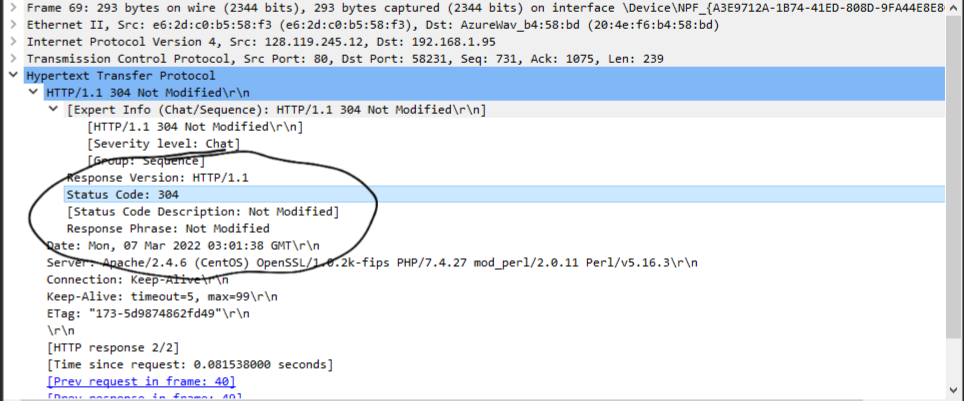
Yes. The information that that follows the “If-Modified-Since” line appears to be a timestamp, so we get: **If-Modified-Since: Sun, 06 Mar 2022 06:59:01 GMT\r\n**

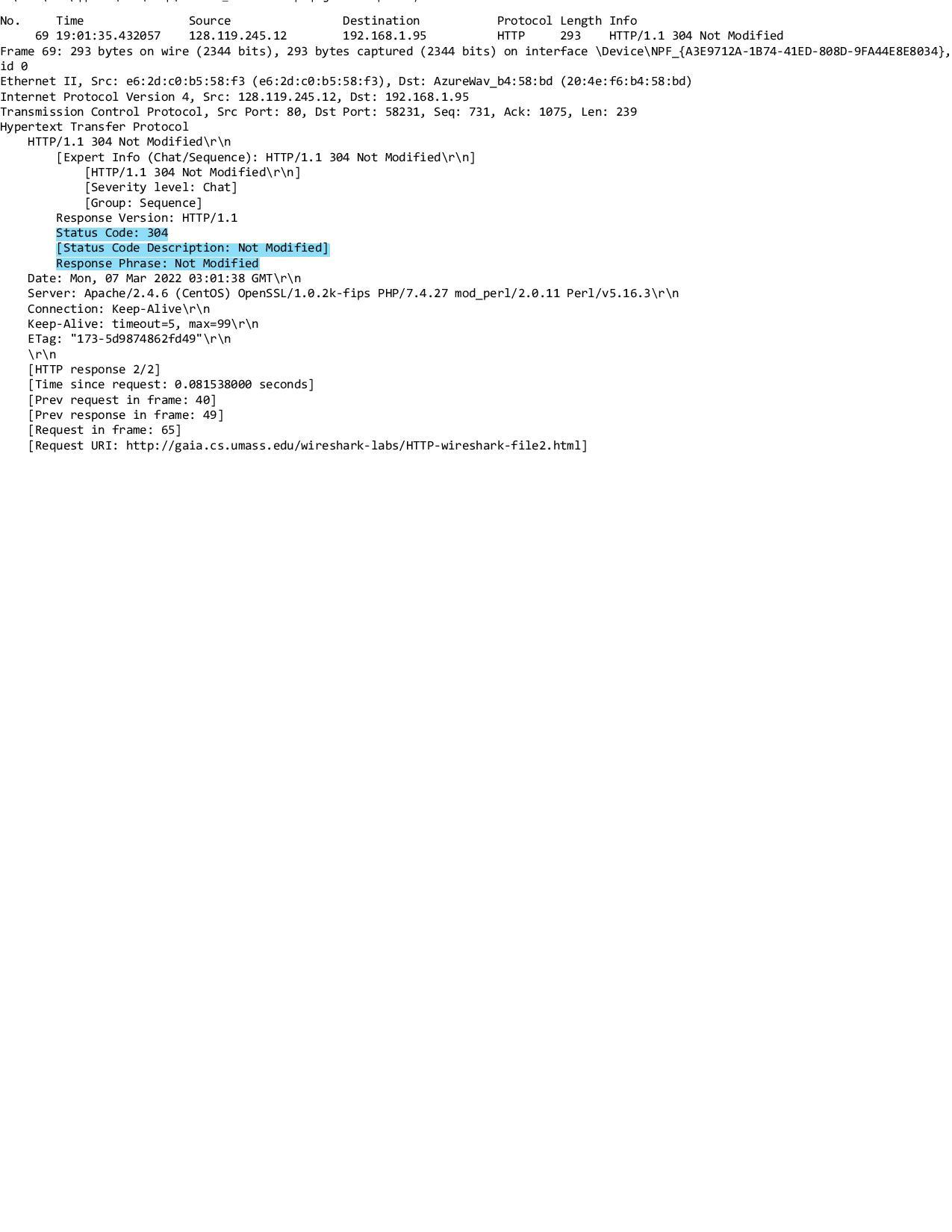




11. What is the HTTP status code and phrase returned from the server in response to this second HTTP GET? Did the server explicitly return the contents of the file? Explain.

The HTTP status code returned from the server is 304. The phrase returned is “Not Modified”. The server did not explicitly return the content because the browser already had it from its cache.





3. Retrieving Long Documents

In our examples thus far, the documents retrieved have been simple and short HTML files. Let’s next see what happens when we download a long HTML file. Do the following:

• Start up your web browser, and make sure your browser’s cache is cleared, as discussed above.

• Start up the Wireshark packet sniffer

• Enter the following URL into your browser

http://gaia.cs.umass.edu/wireshark-labs/HTTP-wireshark-file3.html Your browser should display the rather lengthy US Bill of Rights.

• Stop Wireshark packet capture, and enter “http” in the display-filter-specification window, so that only captured HTTP messages will be displayed.

• (*Note:* If you are unable to run Wireshark on a live network connection, you can use the http-ethereal-trace-3 packet trace to answer the questions below; see footnote 1. This trace file was gathered while performing the steps above on one of the author’s computers.)

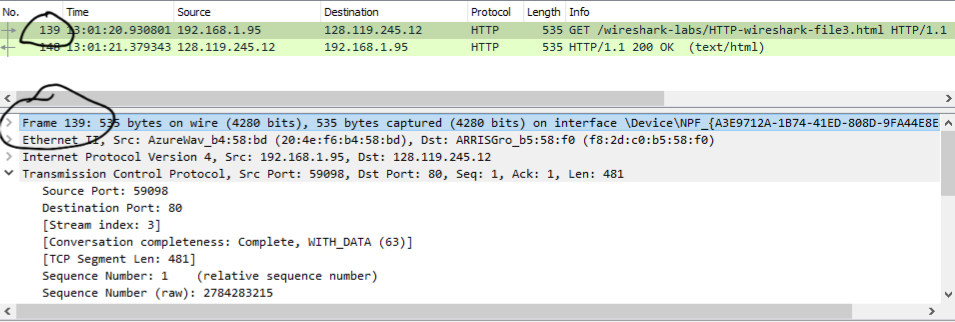
In the packet-listing window, you should see your HTTP GET message, followed by a multiple-packet TCP response to your HTTP GET request. This multiple-packet response deserves a bit of explanation. Recall from Section 2.2 (see Figure 2.9 in the text) that the HTTP response message consists of a status line, followed by header lines, followed by a blank line, followed by the entity body. In the case of our HTTP GET, the

entity body in the response is the *entire* requested HTML file. In our case here, the HTML file is rather long, and at 4500 bytes is too large to fit in one TCP packet. The single HTTP response message is thus broken into several pieces by TCP, with each piece being contained within a separate TCP segment (see Figure 1.24 in the text). In recent versions of Wireshark, Wireshark indicates each TCP segment as a separate packet, and the fact that the single HTTP response was fragmented across multiple TCP packets is indicated by the “TCP segment of a reassembled PDU” in the Info column of the Wireshark display. Earlier versions of Wireshark used the “Continuation” phrase to indicated that the entire content of an HTTP message was broken across multiple TCP segments.. We stress here that there is no “Continuation” message in HTTP!

Answer the following questions:

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

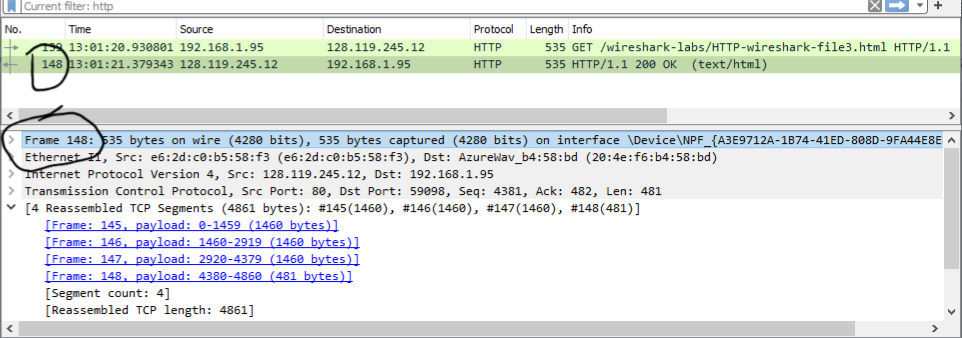
My browser sent only 1 GET request. The packet that contains the GET message is packet 139.

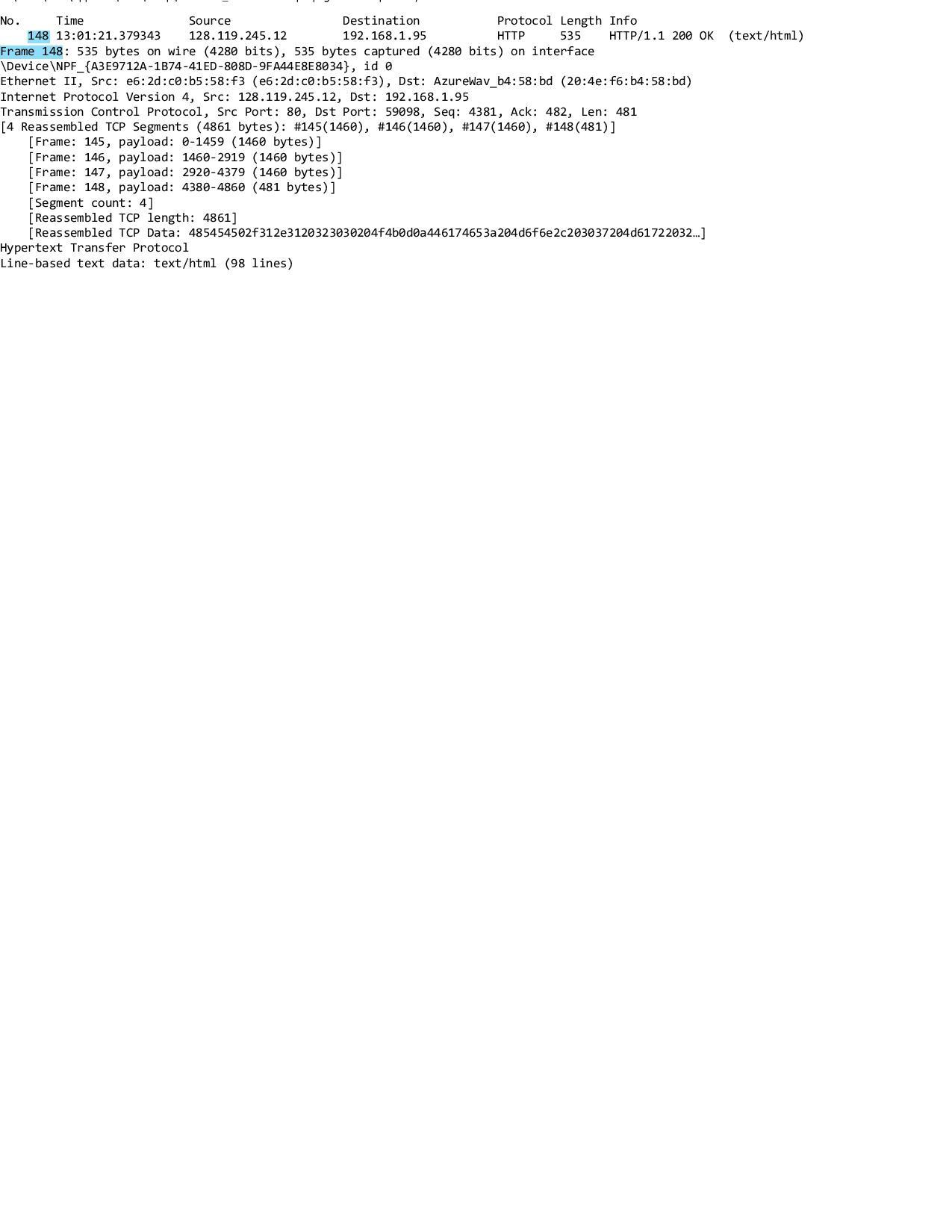




13. Which packet number in the trace contains the status code and phrase associated with the response to the HTTP GET request?

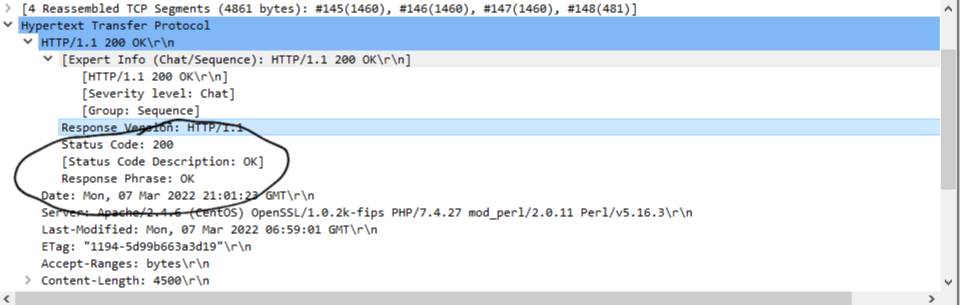
Packet 148 contains the status code and phrase associated with the response to the HTTP GET request.

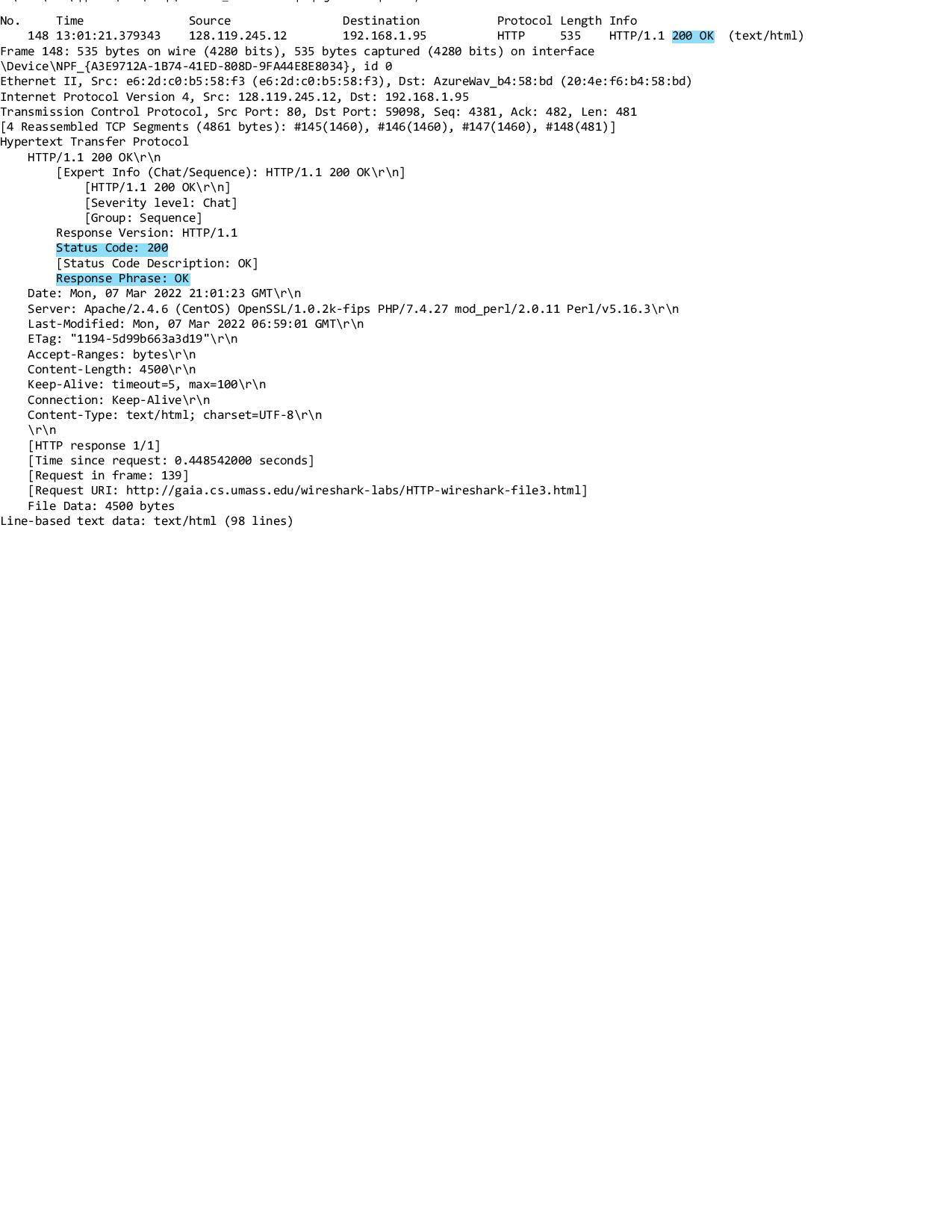




14. What is the status code and phrase in the response?

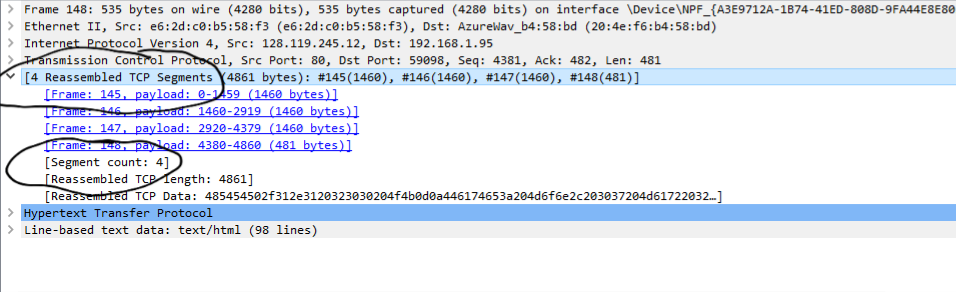
The status code was 200, and the response phrase was OK.





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

4 TCP segments were needed to carry the single HTTP response and the text of the Bill of Rights.





4. HTML Documents with Embedded Objects

Now that we’ve seen how Wireshark displays the captured packet traffic for large HTML files, we can look at what happens when your browser downloads a file with embedded objects, i.e., a file that includes other objects (in the example below, image files) that are stored on another server(s).

Do the following:

• Start up your web browser, and make sure your browser’s cache is cleared, as discussed above.

• Start up the Wireshark packet sniffer

• Enter the following URL into your browser

http://gaia.cs.umass.edu/wireshark-labs/HTTP-wireshark-file4.html

Your browser should display a short HTML file with two images. These two images are referenced in the base HTML file. That is, the images themselves are not contained in the HTML; instead the URLs for the images are contained in the downloaded HTML file. As discussed in the textbook, your browser will have to retrieve these logos from the indicated web sites. Our publisher’s logo is retrieved from the www.aw-bc.com web site. The image of the cover for our 5th edition (one of our favorite covers) is stored at the manic.cs.umass.edu server.

• Stop Wireshark packet capture, and enter “http” in the display-filter-specification window, so that only captured HTTP messages will be displayed.

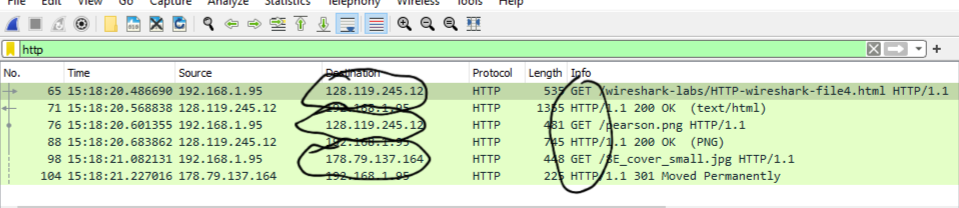
• (*Note:* If you are unable to run Wireshark on a live network connection, you can use the http-ethereal-trace-4 packet trace to answer the questions below; see

footnote 1. This trace file was gathered while performing the steps above on one of the author’s computers.)

Answer the following questions:

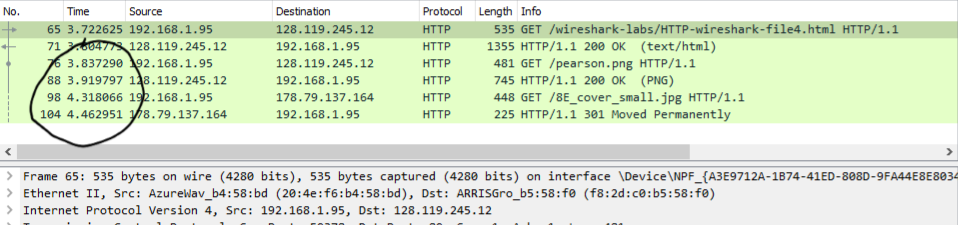
16. How many HTTP GET request messages did your browser send? To which Internet addresses were these GET requests sent?

My browser sent 3 GET request messages. The internet addresses that these GET requests were sent to were 128.119.245.12, 128.119.245.12, and 178.79.137.164.



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.

My browser downloaded the two images serially as there were two separate GETs and they were sent at different times. If you look closely, you can see that the GET request for the jpg image was sent after the response for the png image was sent. If the GET request for the jpg image was sent before the response for the png image, then we would say that it is parallel, but in this case, we say that the images were sent serially.



5 HTTP Authentication

Finally, let’s try visiting a web site that is password-protected and examine the sequence of HTTP message exchanged for such a site. The URL

http://gaia.cs.umass.edu/wireshark-labs/protected\_pages/HTTP-wireshark-file5.html is password protected. The username is “wireshark-students” (without the quotes), and the password is “network” (again, without the quotes). So let’s access this “secure” password-protected site. Do the following:

• Make sure your browser’s cache is cleared, as discussed above, and close down your browser. Then, start up your browser

• Start up the Wireshark packet sniffer

• Enter the following URL into your browser

http://gaia.cs.umass.edu/wireshark-labs/protected\_pages/HTTP-wireshark file5.html

Type the requested user name and password into the pop up box.

• Stop Wireshark packet capture, and enter “http” in the display-filter-specification window, so that only captured HTTP messages will be displayed later in the packet-listing window.

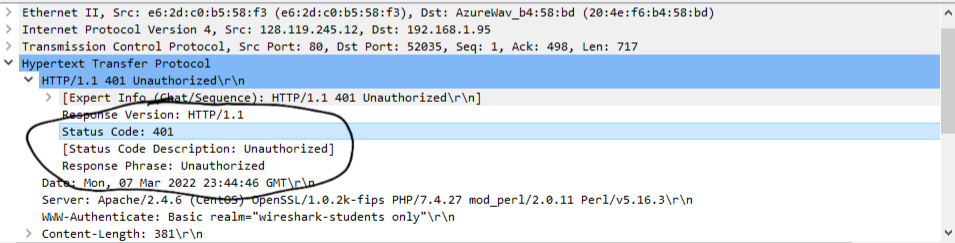
• (*Note:* If you are unable to run Wireshark on a live network connection, you can use the http-ethereal-trace-5 packet trace to answer the questions below; see footnote 2. This trace file was gathered while performing the steps above on one of the author’s computers.)

Now let’s examine the Wireshark output. You might want to first read up on HTTP authentication by reviewing the easy-to-read material on “HTTP Access Authentication Framework” at http://frontier.userland.com/stories/storyReader$2159

Answer the following questions:

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

In response to the initial HTTP GET message from my browser, the server’s response status code is 401. Its response phrase is “Unauthorized”.

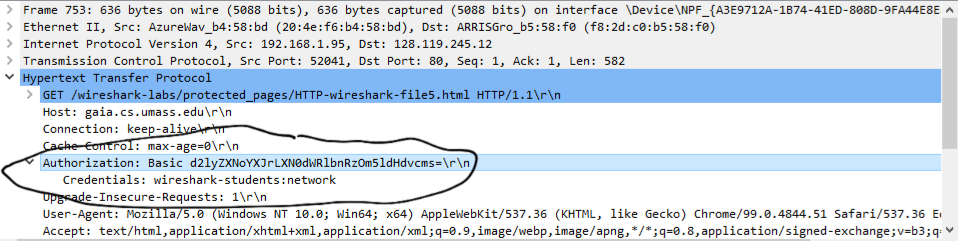




19. When your browser’s sends the HTTP GET message for the second time, what new field is included in the HTTP GET message?

The new field that is included in the HTTP GET message is Authorization. It looks like the following line listed below:

Authorization: Basic d2lyZXNoYXJrLXN0dWRlbnRzOm5ldHdvcms=\r\n





The username (wireshark-students) and password (network) that you entered are encoded in the string of characters (d2lyZXNoYXJrLXN0dWRlbnRzOm5ldHdvcms=) following the “Authorization: Basic” header in the client’s HTTP GET message. While it may appear that your username and password are encrypted, they are simply encoded in a

format known as Base64 format. The username and password are *not* encrypted! To see this, go to http://www.motobit.com/util/base64-decoder-encoder.asp and enter the base64-encoded string d2lyZXNoYXJrLXN0dWRlbnRz and decode. *Voila!* You have translated from Base64 encoding to ASCII encoding, and thus should see your username! To view the password, enter the remainder of the string Om5ldHdvcms= and press decode. Since anyone can download a tool like Wireshark and sniff packets (not just their own) passing by their network adaptor, and anyone can translate from Base64 to ASCII (you just did it!), it should be clear to you that simple passwords on WWW sites are not secure unless additional measures are taken.

Fear not! As we will see in Chapter 8, there are ways to make WWW access more secure. However, we’ll clearly need something that goes beyond the basic HTTP authentication framework!