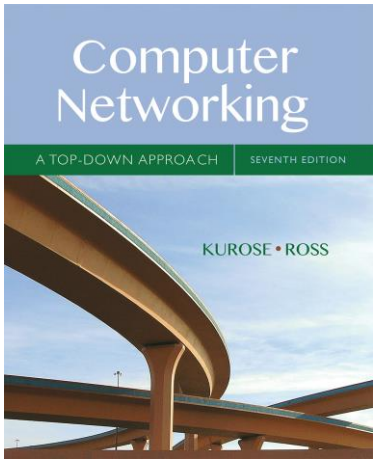
Name: _____Timur Guner_____

Wireshark Lab: HTTP v7.0

Supplement to *Computer Networking: A Top-Down Approach, 7th ed.*, J.F. Kurose and K.W. Ross



"Tell me and I forget. Show me and I remember. Involve me and I understand." Chinese proverb

---

Having gotten our feet wet with the Wireshark packet sniffer in the introductory lab, we're now ready to use Wireshark to investigate protocols in operation. In this lab, we'll explore several aspects of the HTTP protocol: the basic GET/response interaction, HTTP message formats, retrieving large HTML files, retrieving HTML files with embedded objects, and HTTP authentication and security. Before beginning these labs, you might want to review Section 2.2 of the text.[1]

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.

---

[1] References to figures and sections are for the 7th edition of our text, *Computer Networks, A Top-down Approach, 7th ed., J.F. Kurose and K.W. Ross, Addison-Wesley/Pearson, 2016.*

Your Wireshark window should look similar to the window shown in Figure 1. If you are unable to run Wireshark on a live network connection, you can download a packet trace that was created when the steps above were followed.[2]

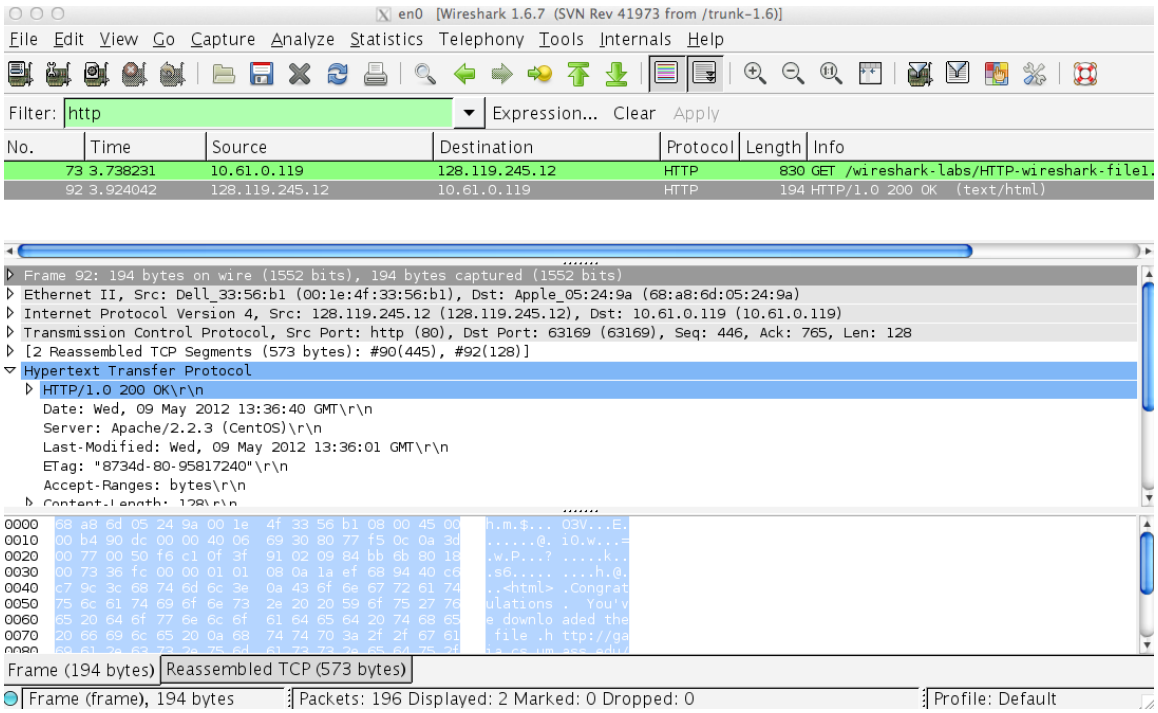


Figure 1: Wireshark Display after http://gaia.cs.umass.edu/wireshark-labs/ HTTP-wireshark-file1.html has been retrieved by your browser

The example in Figure 1 shows in the packet-listing window that two HTTP messages were captured: the GET message (from your browser to the gaia.cs.umass.edu web server) and the response message from the server to your browser. The packet-contents window shows details of the selected message (in this case the HTTP OK message, which is highlighted in the packet-listing window). Recall that since the HTTP message was carried inside a TCP segment, which was carried inside an IP datagram, which was carried within an Ethernet frame, Wireshark displays the Frame, Ethernet, IP, and TCP packet information as well. We want to minimize the amount of non-HTTP data displayed (we're interested in HTTP here, and will be investigating these other protocols is later labs), so make sure the boxes at the far left of the Frame, Ethernet, IP and TCP information have a plus sign or a right-pointing triangle (which means there is hidden, undisplayed information), and the HTTP line has a minus sign or a down-pointing triangle (which means that all information about the HTTP message is displayed).

---

[2] Download the zip file http://gaia.cs.umass.edu/wireshark-labs/wireshark-traces.zip and extract the file http-ethereal-trace-1. The traces in this zip file were collected by Wireshark running on one of the author's computers, while performing the steps indicated in the Wireshark lab. Once you have downloaded the trace, you can load it into Wireshark and view the trace using the *File* pull down menu, choosing *Open*, and then selecting the http-ethereal-trace-1 trace file. The resulting display should look similar to Figure 1. (The Wireshark user interface displays just a bit differently on different operating systems, and in different versions of Wireshark).

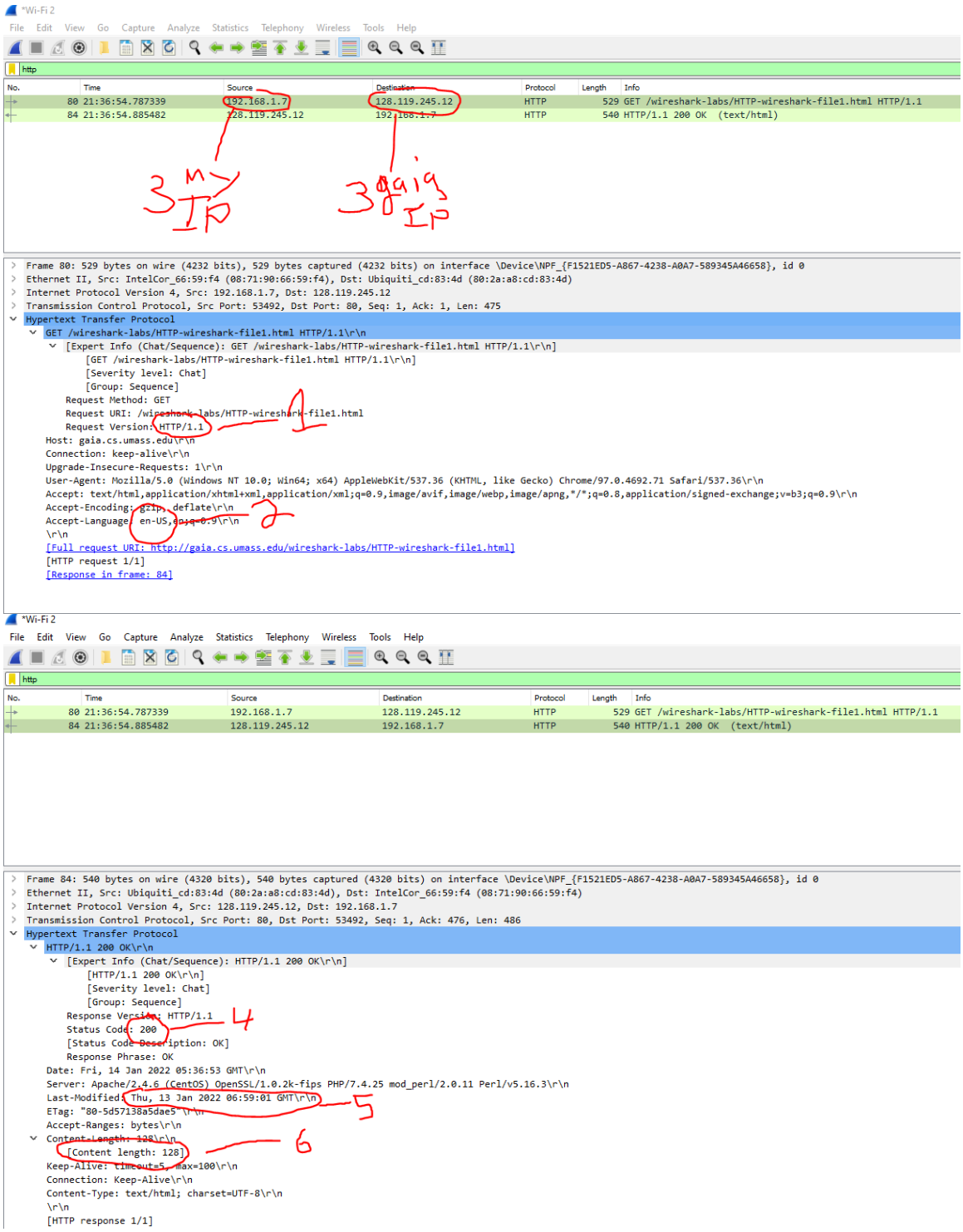
(Note: You should ignore any HTTP GET and response for favicon.ico. If you see a reference to this file, it is your browser automatically asking the server if it (the server) has a small icon file that should be displayed next to the displayed URL in your browser. We'll ignore references to this pesky file in this lab.)

By looking at the information in the HTTP GET and response messages, answer the following questions. When answering the following questions, you should screenshot 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?
    a. **HTML 1.1**
2. What languages (if any) does your browser indicate that it can accept to the server?
    a. **English**
3. What is the IP address of your computer? Of the gaia.cs.umass.edu server?
    a. **My IP 192.168.1.7**
    b. **Gaia IP 128.119.245.12**
4. What is the status code returned from the server to your browser?
    a. **200**
5. When was the HTML file that you are retrieving last modified at the server?
    a. **Jan 13th, 2020 06:59:01 GMT**
6. How many bytes of content are being returned to your browser?
    a. **128**
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.
    a. **I do not**

In your answer to question 5 above, you might have been surprised to find that the document you just retrieved was last modified the morning of the day that 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 day. Thus, if you wait a day between accesses, the file will appear to have been recently modified, and hence your browser will download a "new" copy of the document.

(Note: If you open the developer tools window on Chrome browser, then press and hold the reload button, you are presented with several options. Both "Hard Reload" and "Empty Cache and Hard Reload" will get all of the data, regardless if it is cached.)



2. The HTTP CONDITIONAL GET/response interaction

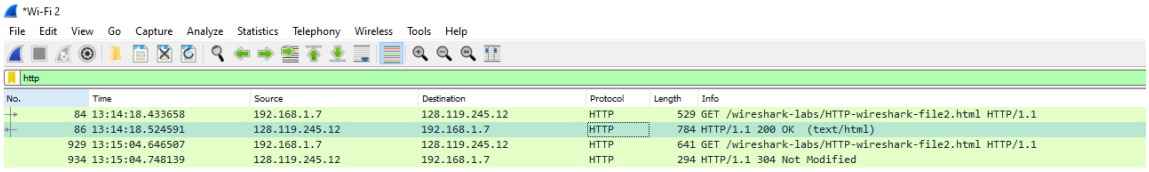
Recall from Section 2.2.5 of the text, that most web browsers perform object caching and thus perform a conditional GET when retrieving an HTTP object. Before performing the steps below, make sure your browser's cache is empty. (To do this under Firefox, select

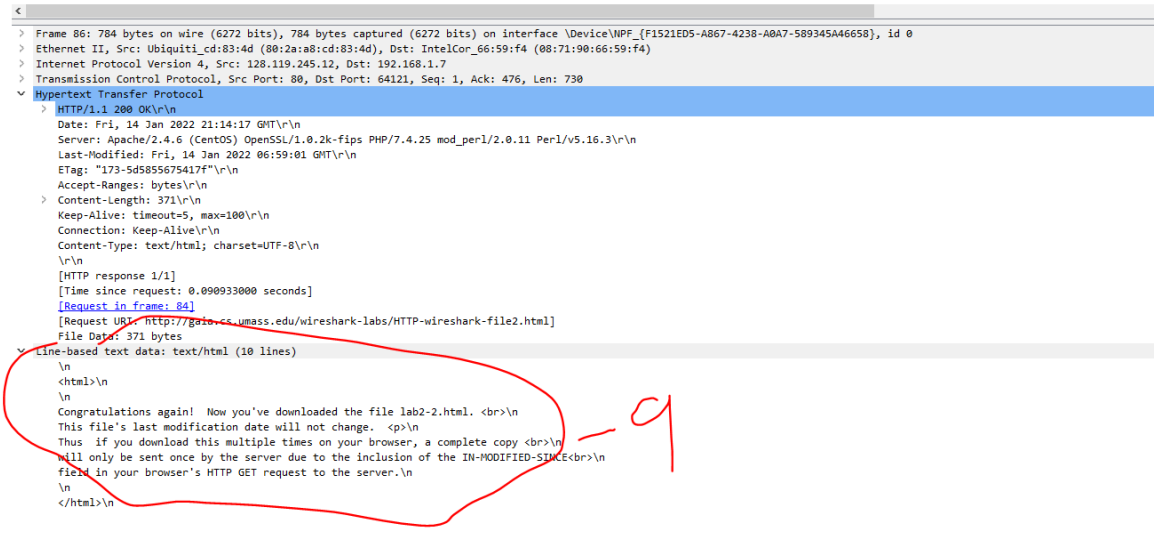
Tools->Clear Recent History and check the Cache box, or for Internet Explorer, select *Tools->Internet Options->Delete File;* these actions will remove cached files from your browser's cache.) Now do the following:

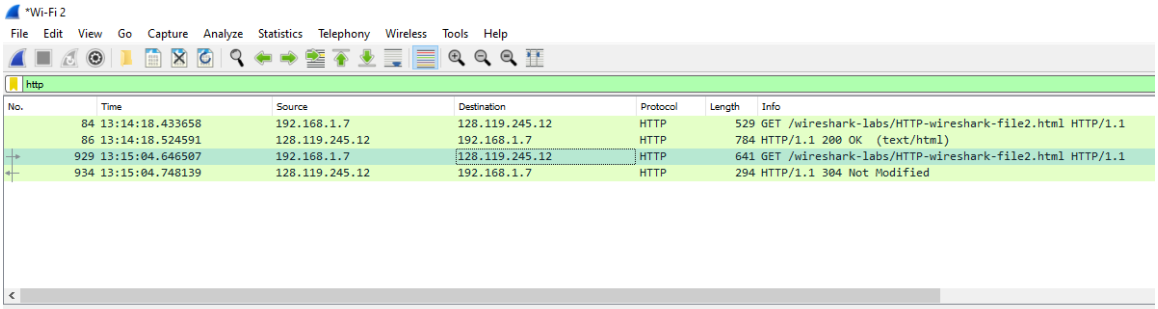
- Start up your web browser, and make sure your browser's cache is cleared, as discussed above.
- Start up the Wireshark packet sniffer
- Enter the following URL into your browser
  http://gaia.cs.umass.edu/wireshark-labs/HTTP-wireshark-file2.html
  Your browser should display a very simple five-line 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.)

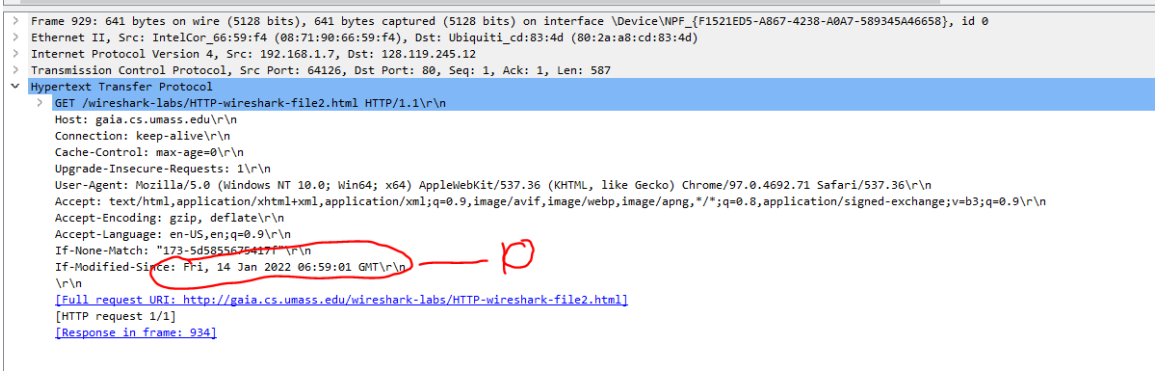
Answer the following questions:

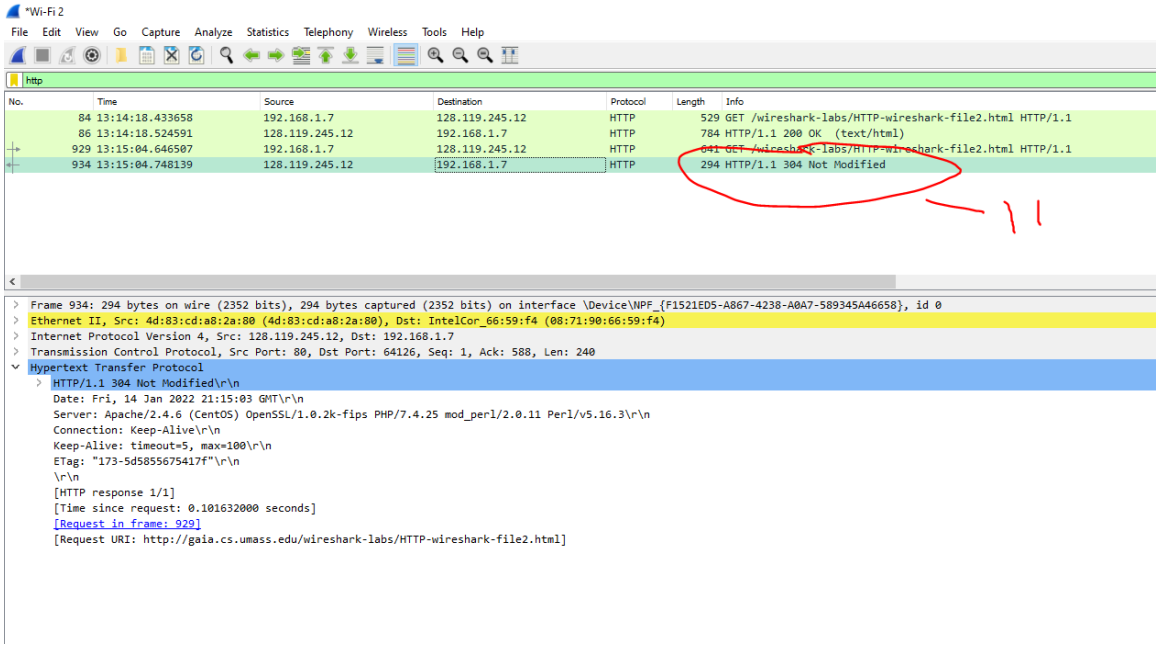
8. Inspect the contents of the first HTTP GET request from your browser to the server. Do you see an "IF-MODIFIED-SINCE" line in the HTTP GET?
   a. **NO**
9. Inspect the contents of the server response. Did the server explicitly return the contents of the file? How can you tell?
   a. **Yes because it is shown is 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?
    a. **Fri, 14 Jan 2022 06:50:01 GMT the last modification time of the file which is the also the date I last accessed the page**
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.
    a. **It was a 304 message which means the file was just retrieved from the cache not the server**


*Wi-Fi 2
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
http
No. Time Source Destination Protocol Length Info
84 13:14:18.433658 192.168.1.7 128.119.245.12 HTTP 529 GET /wireshark-labs/HTTP-wireshark-file2.html HTTP/1.1
86 13:14:18.524591 128.119.245.12 192.168.1.7 HTTP 784 HTTP/1.1 200 OK (text/html)
929 13:15:04.646507 192.168.1.7 128.119.245.12 HTTP 641 GET /wireshark-labs/HTTP-wireshark-file2.html HTTP/1.1
934 13:15:04.748139 128.119.245.12 192.168.1.7 HTTP 294 HTTP/1.1 304 Not Modified


Frame 86: 784 bytes on wire (6272 bits), 784 bytes captured (6272 bits) on interface \Device\NPF_{F1521ED5-A867-4238-A0A7-589345A46658}, id 0
Ethernet II, Src: Ubiquiti_cd:83:4d (80:2a:a8:cd:83:4d), Dst: IntelCor_66:59:f4 (08:71:90:66:59:f4)
Internet Protocol Version 4, Src: 128.119.245.12, Dst: 192.168.1.7
Transmission Control Protocol, Src Port: 80, Dst Port: 64121, Seq: 1, Ack: 476, Len: 730
Hypertext Transfer Protocol
HTTP/1.1 200 OK\r\n
Date: Fri, 14 Jan 2022 21:14:17 GMT\r\n
Server: Apache/2.4.6 (CentOS) OpenSSL/1.0.2k-fips PHP/7.4.25 mod_perl/2.0.11 Perl/v5.16.3\r\n
Last-Modified: Fri, 14 Jan 2022 06:59:01 GMT\r\n
ETag: "173-5d5855675417f"\r\n
Accept-Ranges: bytes\r\n
Content-Length: 371\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
[HTTP response 1/1]
[Time since request: 0.090933000 seconds]
[Request in frame: 84]
[Request URI: http://gaia.cs.umass.edu/wireshark-labs/HTTP-wireshark-file2.html]
File Data: 371 bytes
Line-based text data: text/html (10 lines)
\n
<html>\n
\n
Congratulations again! Now you've downloaded the file lab2-2.html.
\n
This file's last modification date will not change. <p>\n
Thus if you download this multiple times on your browser, a complete copy
\n
will only be sent once by the server due to the inclusion of the IN-MODIFIED-SINCE
\n
field in your browser's HTTP GET request to the server.\n
\n
</html>\n
9


*Wi-Fi 2
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
http
No. Time Source Destination Protocol Length Info
84 13:14:18.433658 192.168.1.7 128.119.245.12 HTTP 529 GET /wireshark-labs/HTTP-wireshark-file2.html HTTP/1.1
86 13:14:18.524591 128.119.245.12 192.168.1.7 HTTP 784 HTTP/1.1 200 OK (text/html)
929 13:15:04.646507 192.168.1.7 128.119.245.12 HTTP 641 GET /wireshark-labs/HTTP-wireshark-file2.html HTTP/1.1
934 13:15:04.748139 128.119.245.12 192.168.1.7 HTTP 294 HTTP/1.1 304 Not Modified


Frame 929: 641 bytes on wire (5128 bits), 641 bytes captured (5128 bits) on interface \Device\NPF_{F1521ED5-A867-4238-A0A7-589345A46658}, id 0
Ethernet II, Src: IntelCor_66:59:f4 (08:71:90:66:59:f4), Dst: Ubiquiti_cd:83:4d (80:2a:a8:cd:83:4d)
Internet Protocol Version 4, Src: 192.168.1.7, Dst: 128.119.245.12
Transmission Control Protocol, Src Port: 64126, Dst Port: 80, Seq: 1, Ack: 1, Len: 587
Hypertext Transfer Protocol
GET /wireshark-labs/HTTP-wireshark-file2.html HTTP/1.1\r\n
Host: gaia.cs.umass.edu\r\n
Connection: keep-alive\r\n
Cache-Control: max-age=0\r\n
Upgrade-Insecure-Requests: 1\r\n
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/97.0.4692.71 Safari/537.36\r\n
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=0.9\r\n
Accept-Encoding: gzip, deflate\r\n
Accept-Language: en-US,en;q=0.9\r\n
If-None-Match: "173-5d5855675417f"\r\n
If-Modified-Since: Fri, 14 Jan 2022 06:59:01 GMT\r\n
\r\n
[Full request URI: http://gaia.cs.umass.edu/wireshark-labs/HTTP-wireshark-file2.html]
[HTTP request 1/1]
[Response in frame: 934]
10



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:

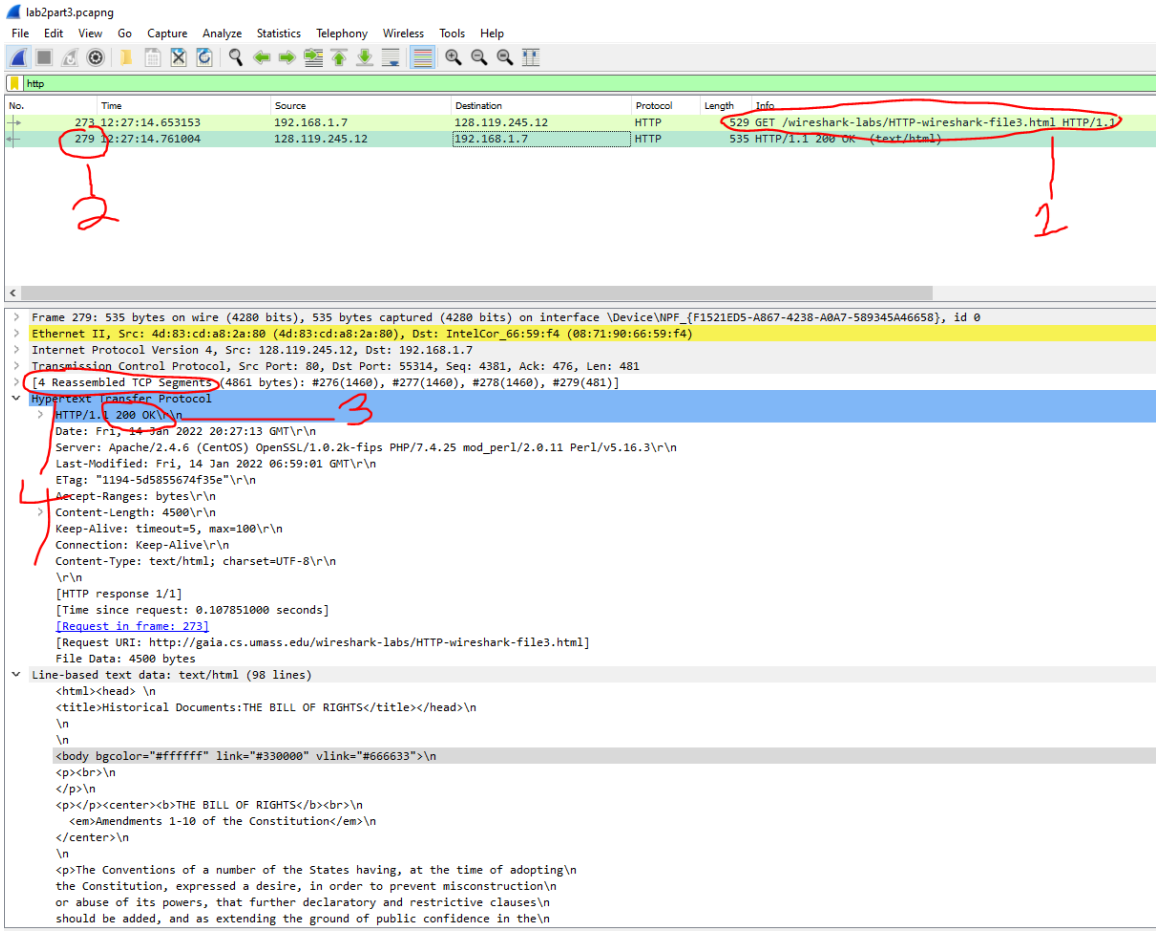
- Start up your web browser, and make sure your browser's cache is cleared, as discussed above.
- Start up the Wireshark packet sniffer
- Enter the following URL into your browser
  http://gaia.cs.umass.edu/wireshark-labs/HTTP-wireshark-file3.html
  Your browser should display the rather lengthy US Bill 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?
    a. **Only one message was sent at packet number 273**
13. Which packet number in the trace contains the status code and phrase associated with the response to the HTTP GET request?
    a. **Packet 279**
14. What is the status code and phrase in the response?
    a. **200 OK**
15. How many data-containing TCP segments were needed to carry the single HTTP response and the text of the Bill of Rights?
    a. **4**



4. HTML Documents with Embedded Objects

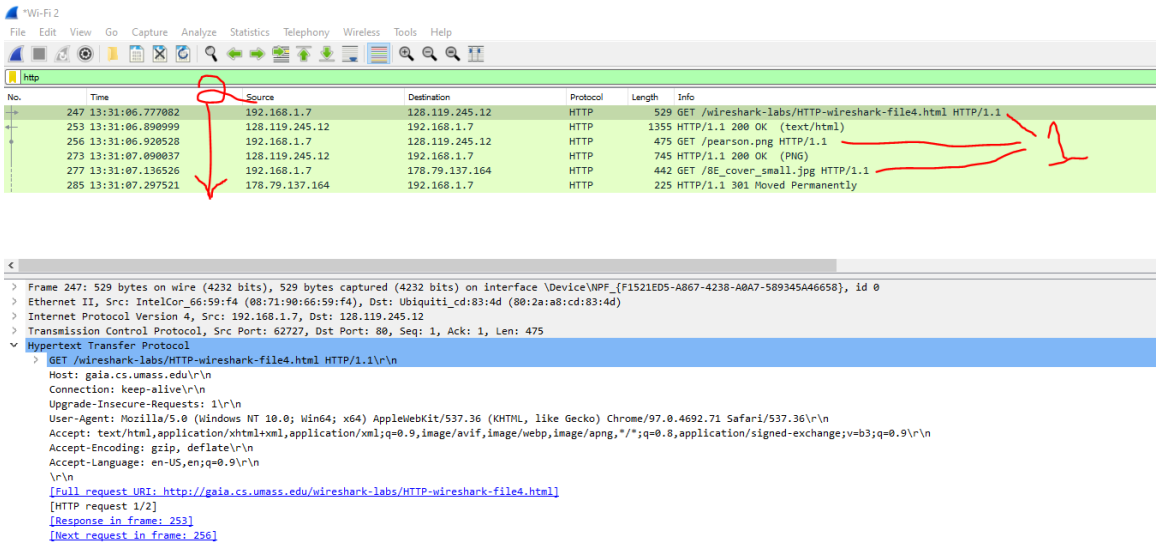
Now that we've seen how Wireshark displays the captured packet traffic for large HTML files, we can look at what happens when your browser downloads a file with embedded objects, i.e., a file that includes other objects (in the example below, image files) that are stored on another server(s).

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 gaia.cs.umass.edu 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. (These are two different web servers inside cs.umass.edu).
- 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?
    a. **There were 3 GET messages sent. The addresses were 128.119.245.12, 128.119.145.12, 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.
    a. **They were done serially because according to the time logs, a get request and response is finished before the next get request is sent. Everything is in chronological order.**



5 HTTP Authentication

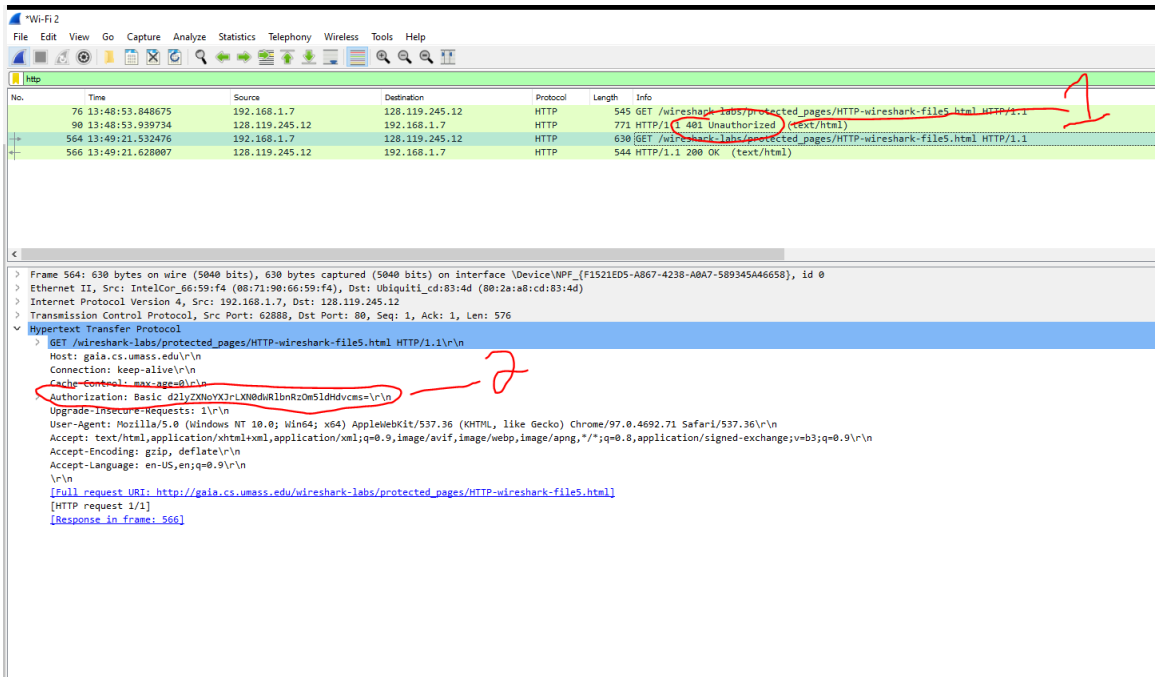
Finally, let's try visiting a web site that is password-protected and examine the sequence of HTTP message exchanged for such a site. The URL http://gaia.cs.umass.edu/wireshark-labs/protected_pages/HTTP-wireshark-file5.html is password protected. The username is "1" (without the quotes), and the password 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?
    a. **401 Unauthorized User**
19. When your browser's sends the HTTP GET message for the second time, what new field is included in the HTTP GET message?
    a. **The Authorization field**



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!

Changelog:

1. 03 July, 2020: Altered last-modified description in response to question 5 to describe it as changing once each day instead of each minute.