**Laboratory 1 – HTTP**

**Student name:** Phan Phương Duy. **Student ID**: ITITIU16010

**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:

* Start up your web browser.
* 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).
* Wait a bit more than one minute (we’ll see why shortly), and then begin Wireshark packet capture.
* 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.
* Stop Wireshark packet capture.

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.

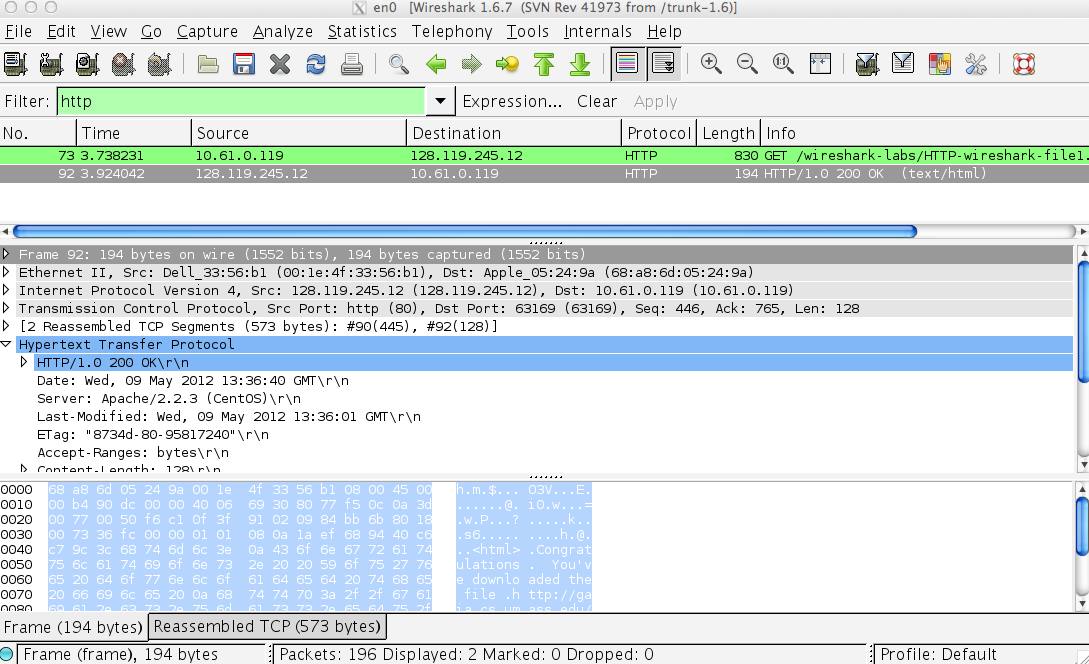


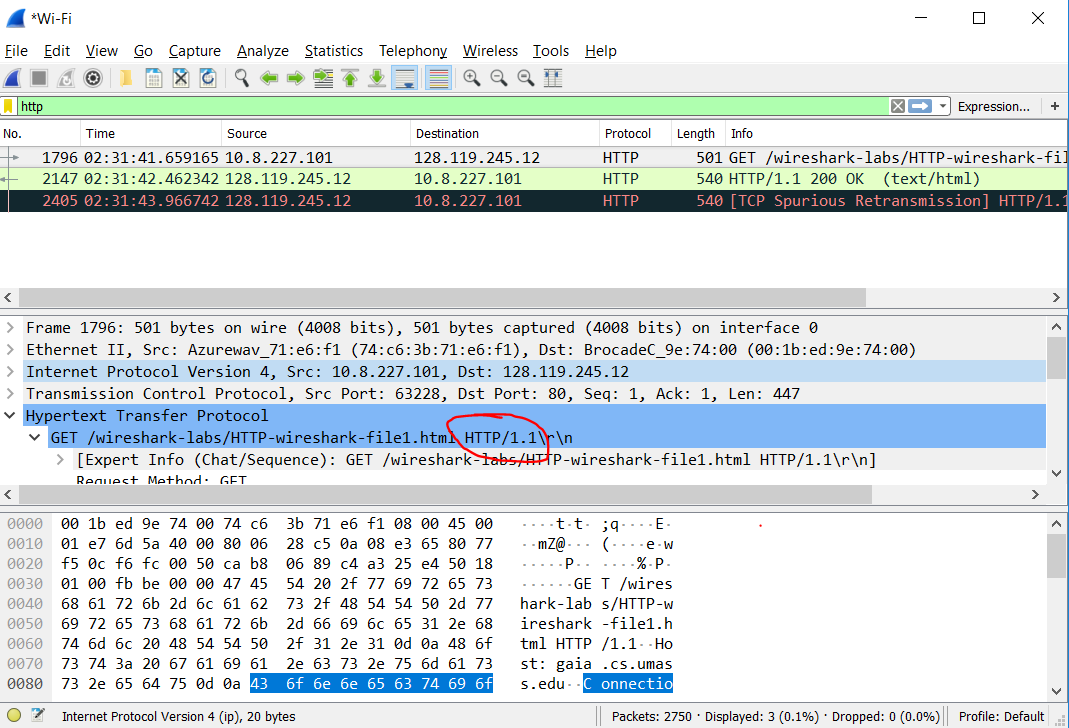
Figure 1: Wireshark Display after http://gaia.cs.umass.edu/wireshark-labs/ HTTP-wireshark-file1.html has been retrieved by your browser

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).

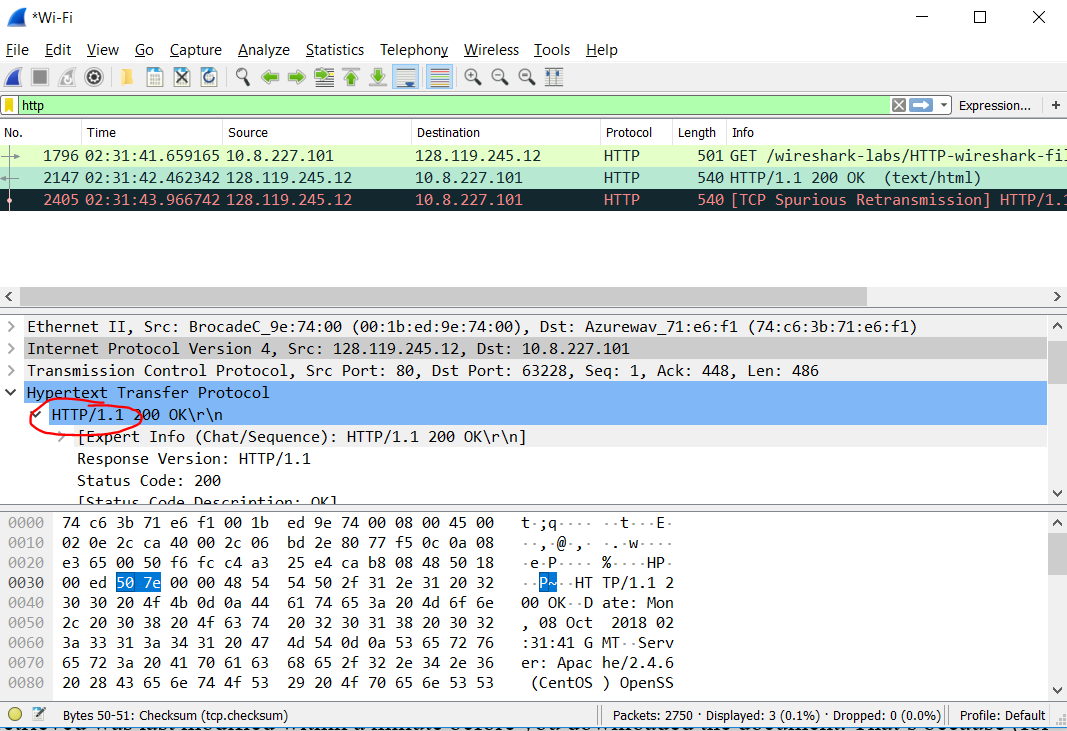
By looking at the information in the HTTP GET and response messages, answer the following questions. When answering the following questions, you should print out the GET and response messages (see the introductory Wireshark lab for an explanation of how to do this) and indicate where in the message you’ve 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).

**Q1. Is your browser running HTTP version 1.0 or 1.1? What version of HTTP is the server running?**

My browser is running HTTP 1.1

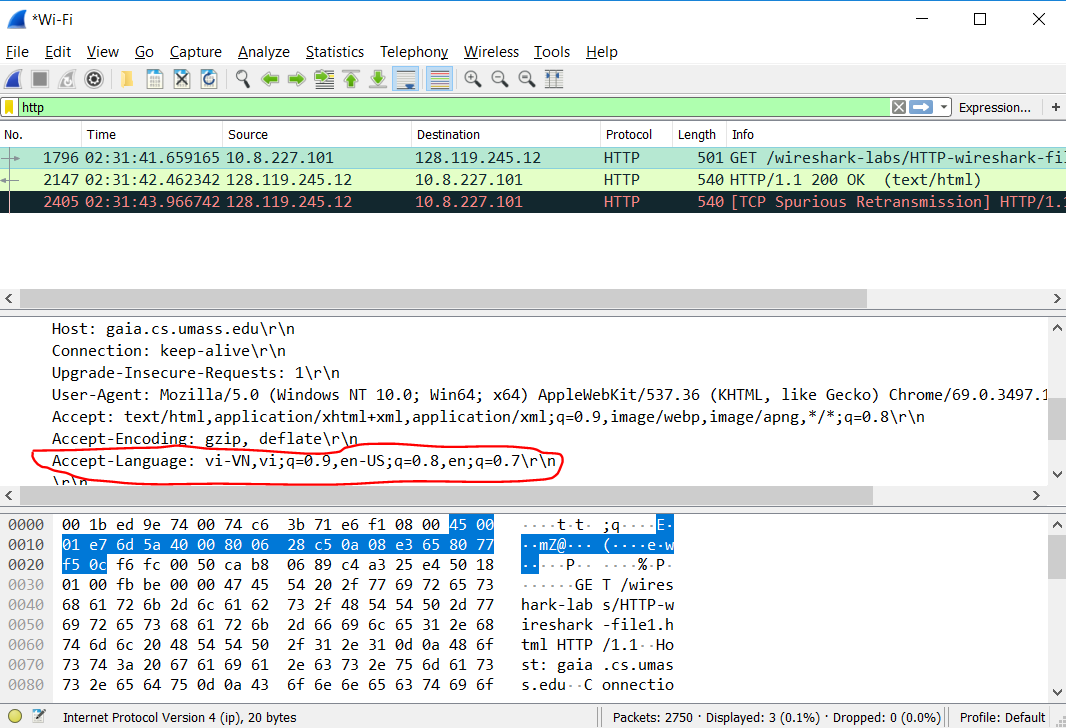


Server is running HTTP 1.1



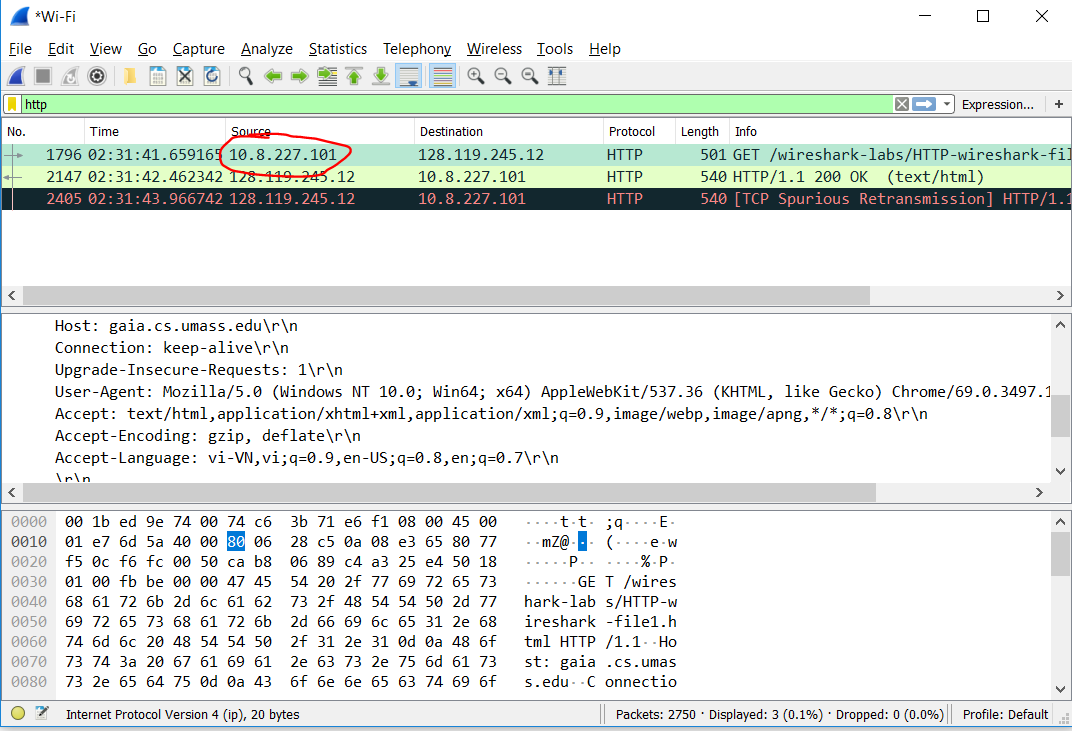
**Q2. What languages (if any) does your browser indicate that it can accept to the server?**

My browser indicates that it can accept Vietnamese (vi-VN) and English (en-US)

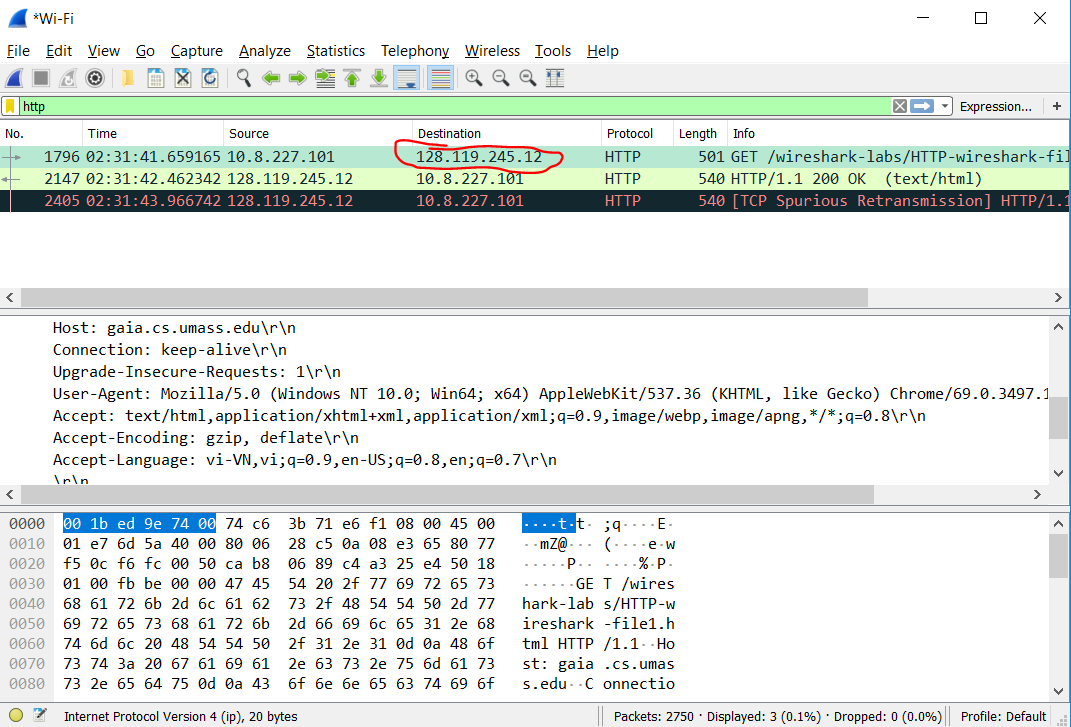


**Q3. What is the IP address of your computer? Of the gaia.cs.umass.edu server?**

IP address of my computer: 10.8.227.101

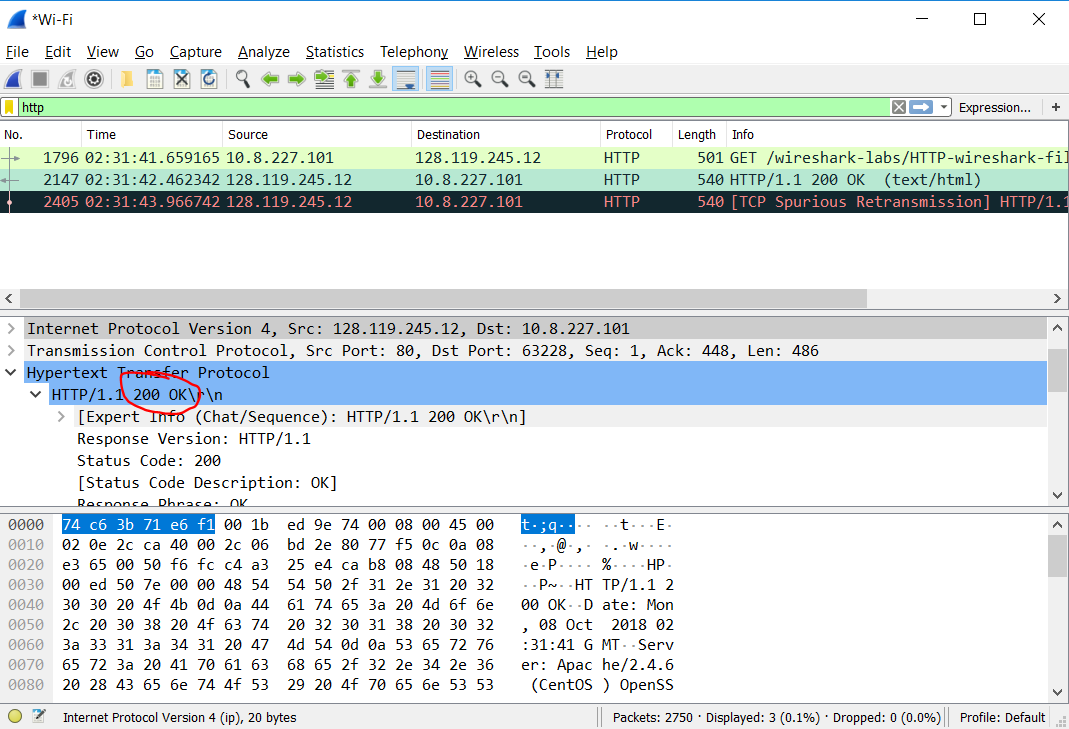


Ip address of gaia.cs.umass.edu server:



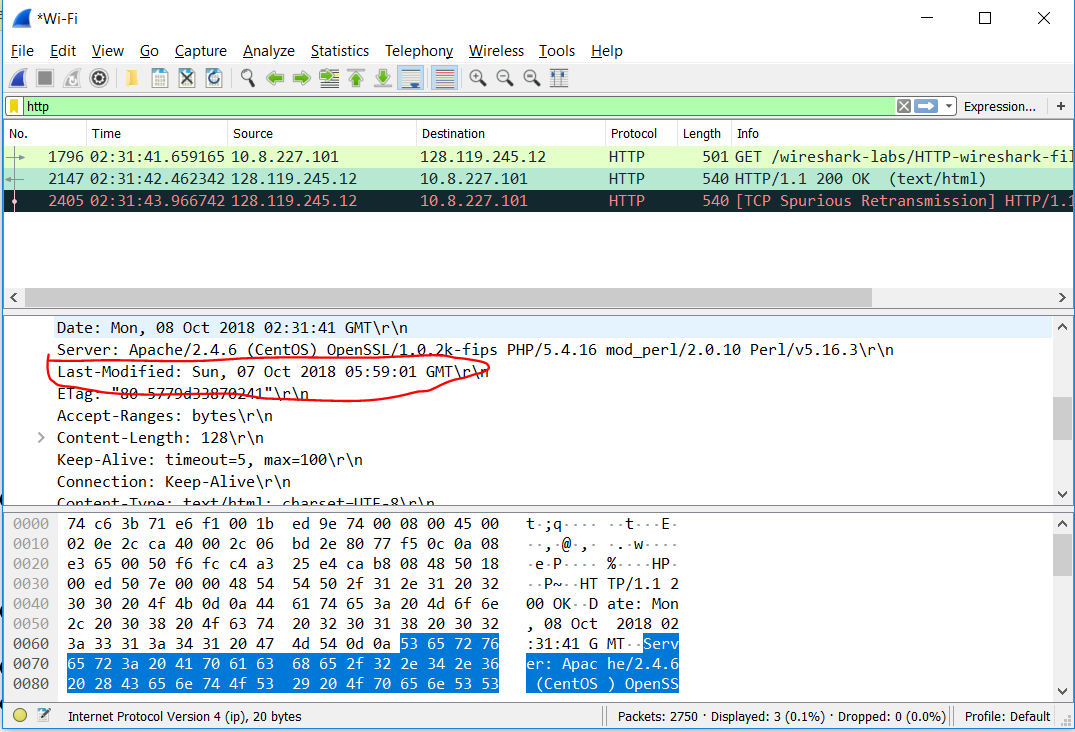
**Q4. What is the status code returned from the server to your browser?**

Status code: 200 OK



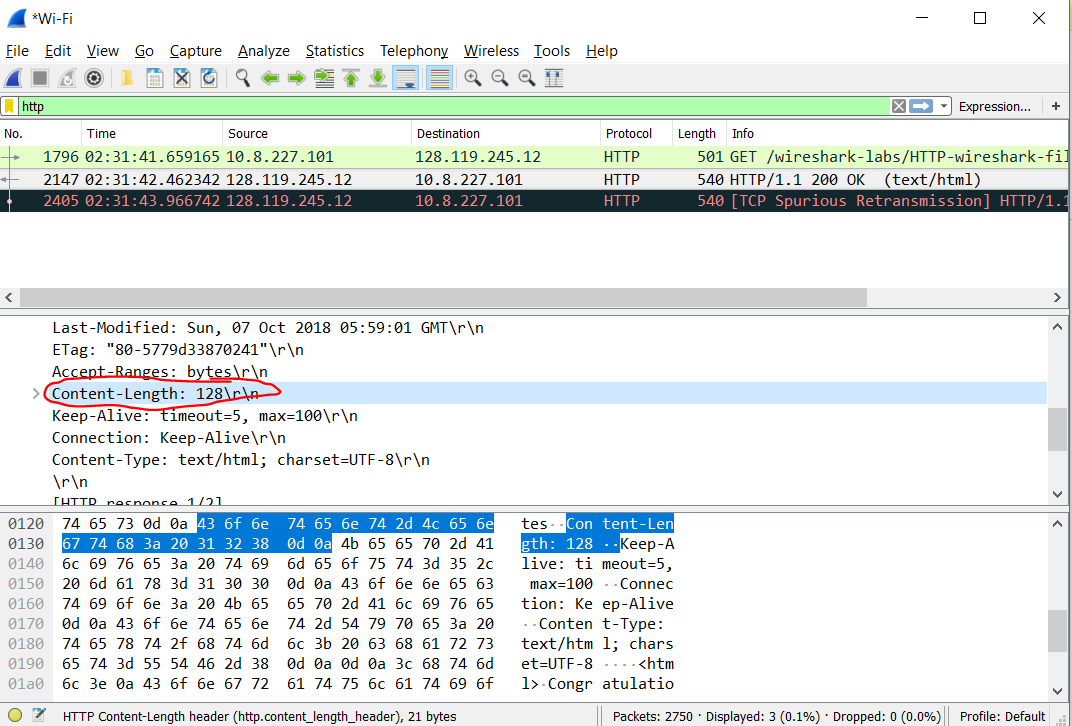
**Q5. When was the HTML file that you are retrieving last modified at the server?**

The HTML file is last modified at Sun, 07 Oct 2018 5:59:01 GMT



**Q6. How many bytes of content are being returned to your browser?**

128 bytes were returned to my browser



**Q7. 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 that are missing.

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.

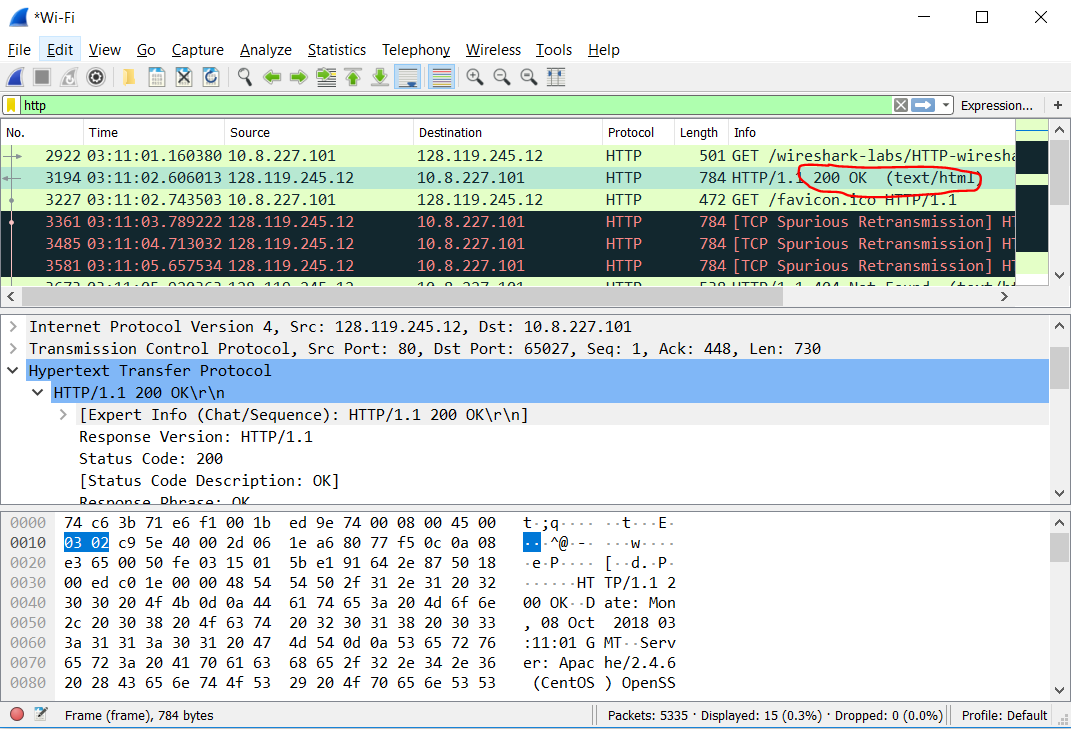
Answer the following questions:

**Q 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 do not see header “IF-MODIFIED-SINCE” in the HTTP GET

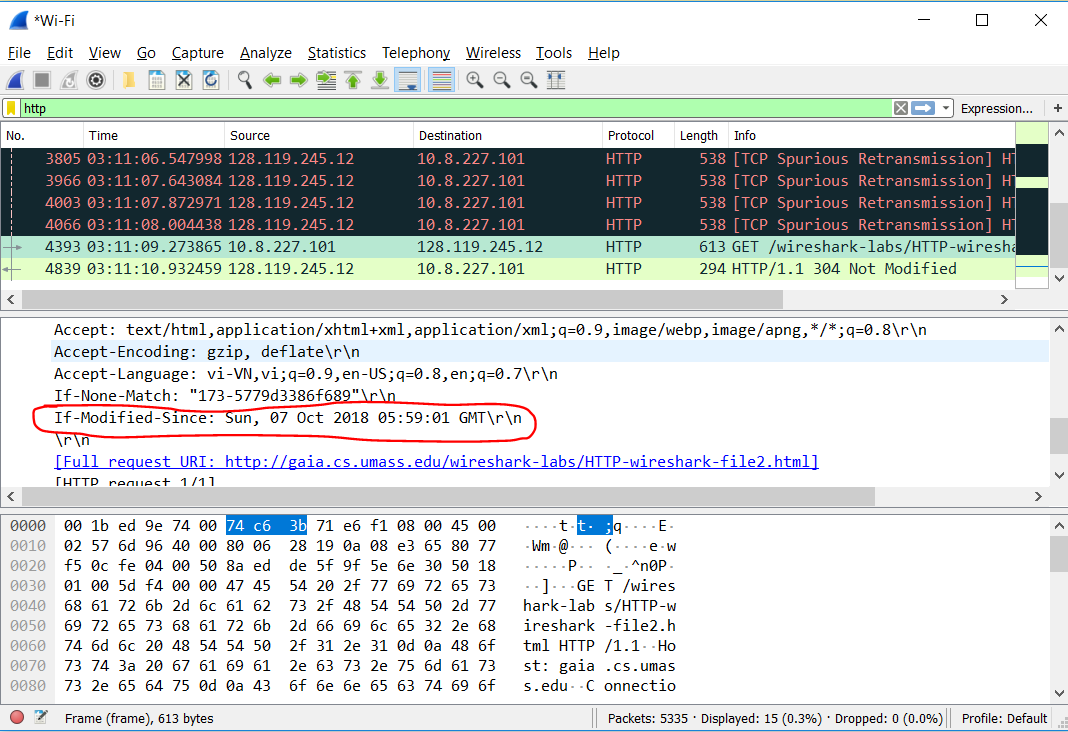
**Q 9. Inspect the contents of the server response. Did the server explicitly return the contents of the file? How can you tell?**

Yes, the server returned the contents of the file (text/html)



**Q 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?**

The header “IF-MODIFIED-SINCE” appeared in second HTTP GET request. The information followed was Sun, 07 Oct 2018 05:59:01 GMT.



**Q 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.**

HTTP status code and phrase: 304 Not Modified

The server did not return the contents of the file. Because the header “IF-MODIFIED-SINCE” of the second HTTP GET request was Sun, 07 Oct 2018 05:59:01 GMT and the header “LAST-MODIFIED” of the first HTTP GET response was Sun, 07 Oct 2018 05:59:01 GMT. It means that if the file cached was modified (updated) since Sun, 07 Oct 2018 05:59:01 GMT, then it returned to client directly (not making request to the origin server anymore).

**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:

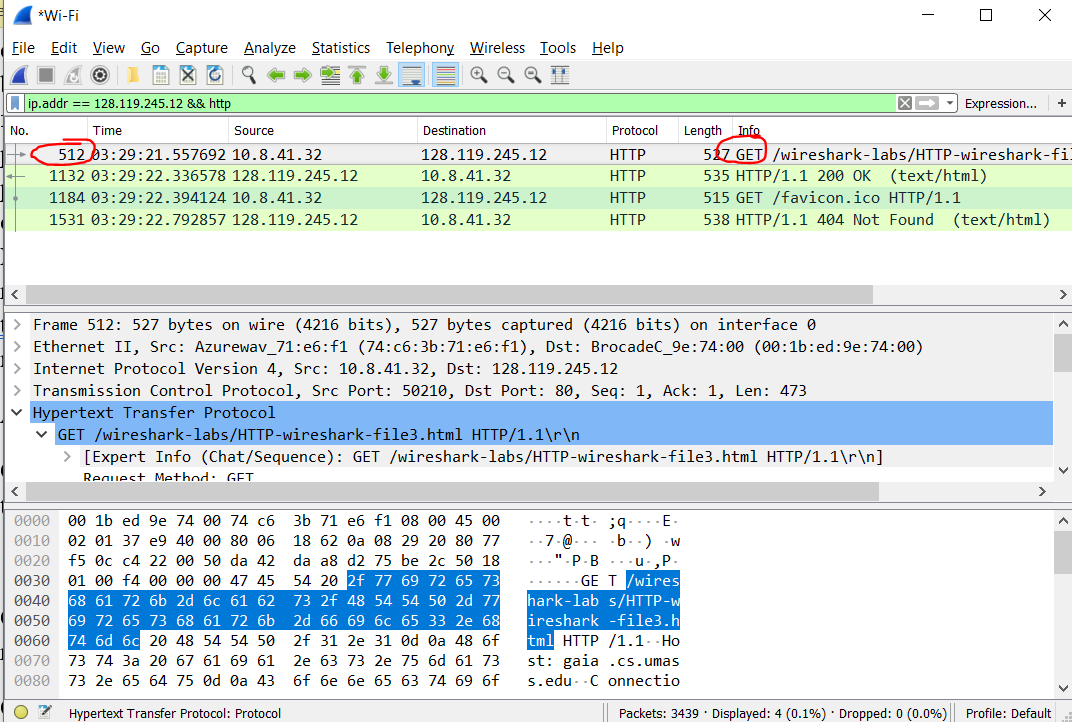
* Start up your web browser, and make sure your browser’s cache is cleared, as discussed above.
* Start up the Wireshark packet sniffer
* Enter the following URL into your browser <http://gaia.cs.umass.edu/wireshark-labs/HTTP-wireshark-file3.html>, your browser should display the rather lengthy US Bill of Rights.
* Stop Wireshark packet capture, and enter “http” in the display-filter-specification window, so that only captured HTTP messages will be displayed.

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:

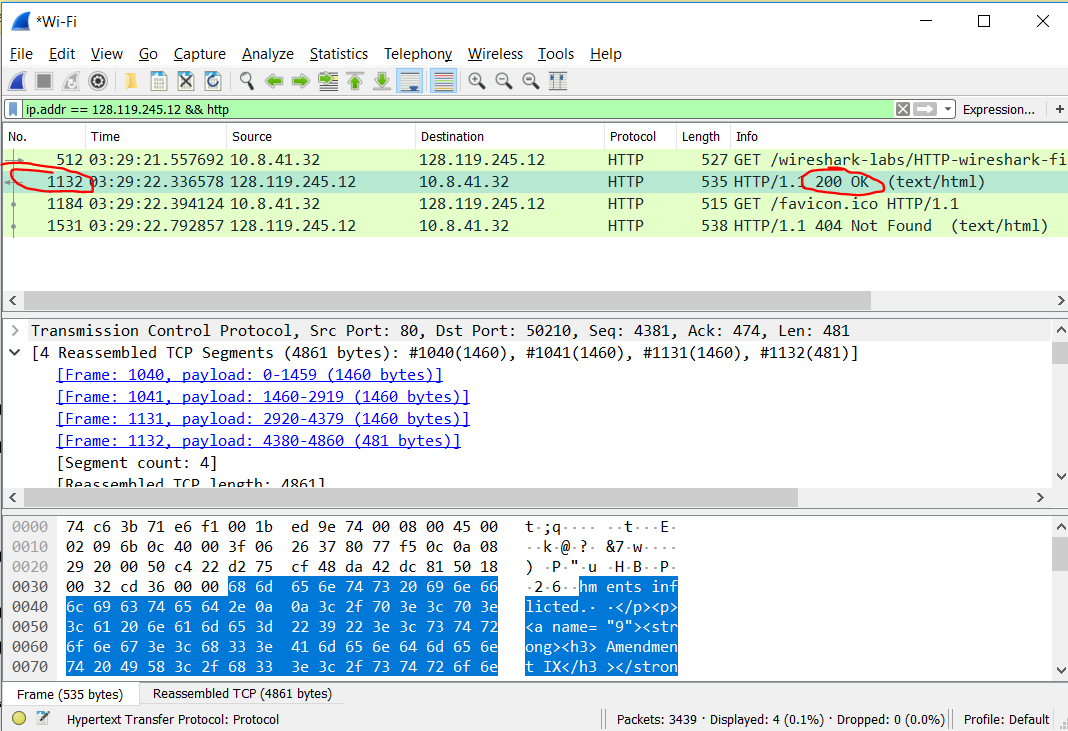
**Q 12. How many HTTP GET request messages did your browser send? Which packet number in the trace contains the GET message for the Bill or Rights?**

Only 1 HTTP GET request message that my browser sent (packet number: 512)



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

Packet number: 1132

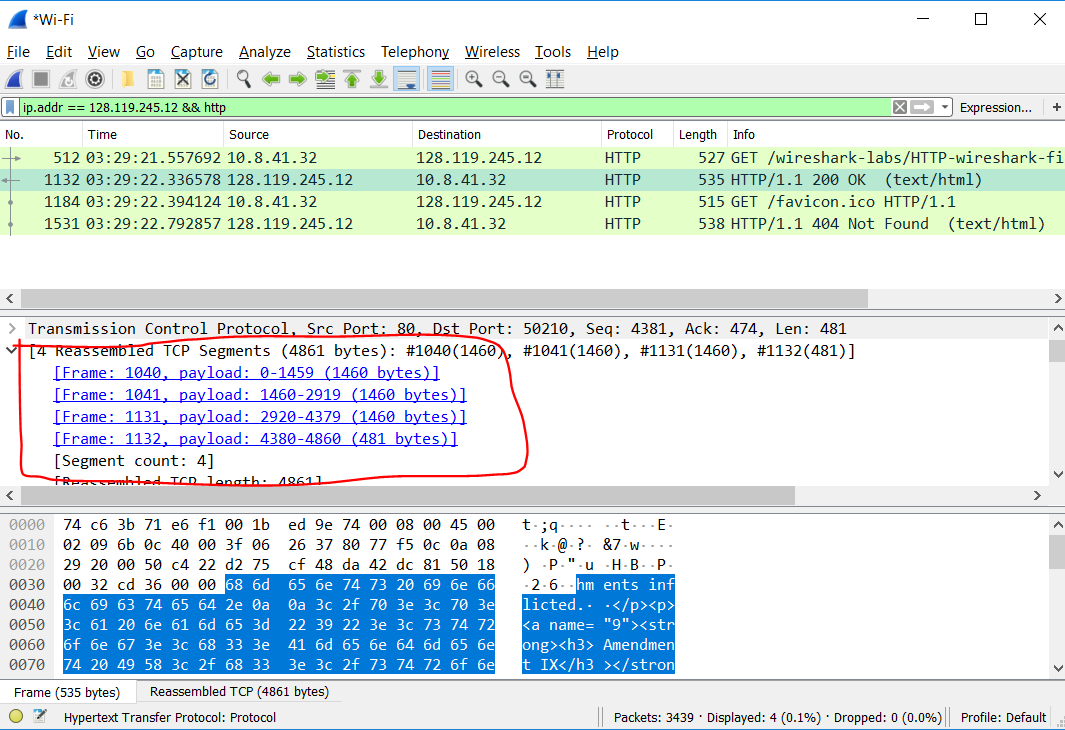


**Q 14. What is the status code and phrase in the response?**

Status code and phrase: 200 OK

**Q 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 that 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.

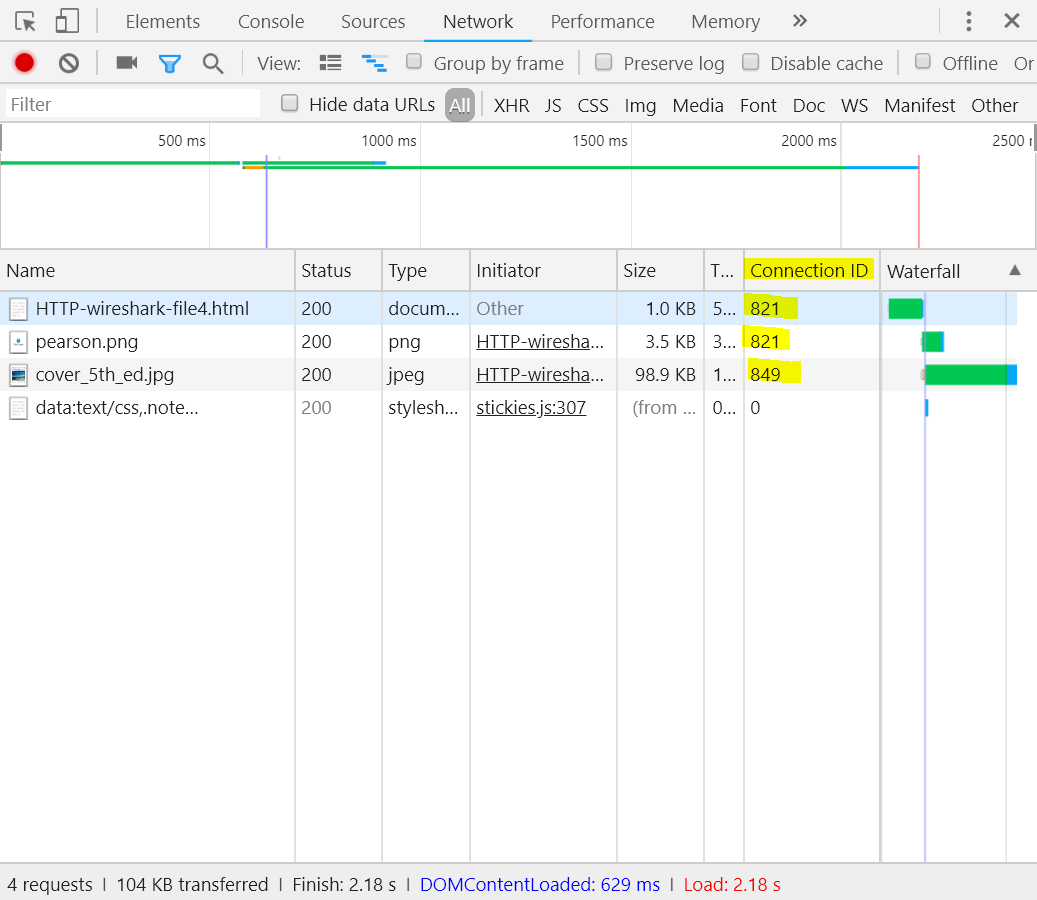
Answer the following questions:

**Q16. How many HTTP GET request messages did your browser send? To which Internet addresses were these GET requests sent?**

There were 3 HTTP GET requests that my browser sent. The destination IP address of 3 request was the same: 128.119.245.12 because all the objects and files were on the same host.

**Q17. 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 2 images in parallel.



In chrome developer tools, we can see that after the first request to get html file, two different parrallel connection whose id 821 and 849, respectively, were opened to get the next two images. The reason the first image (pearson.png) connection id was same as the html connection id (821) because the first request the browser set header “connection: keep-alive” that means the browser wanted to keep persistent connection. After getting the html file, the browser still used the same connection (id=821) to get first image and simultaneously opened new TCP connection (id=849) to get the second image.

**THE END.**