CS 3800 Computer Networks

Assignment # 2

Points: 15

- **Submission**
  - **Zip the documents for your solutions. Name the file Lastname_firstname_HW2.zip**
  - **Submit your zip on Blackboard under Assignments->Assignment2.**
- **Submission deadline**
  - **Deadline: 10/01 11:59 PM**
- **Late Submission Policy**
  - **You can do a late submission until 10/04 11:59PM with a 5% penalty on the total points for this assignment.**
  - **After that no submission will be accepted**
- **Early Submission**
  - **You can also get 5% extra point (on your score on the assignment) for early submission if you submit by 09/30 11:59PM.**
- **Getting help**
  - **From instructor during office hours or by email.**
- **Academic Dishonesty Policy**
  - **Submission will be checked for plagiarism. This is individual work.**

1. Wireshark Exercises (from the Instructor's resources of the textbook) [5pts]

Provide your solutions for Question 1 as a MS Word document.

The HTTP CONDITIONAL GET/response interaction

Recall from Section 2.2.5 of the textbook, 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:

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?
2. Inspect the contents of the server response. Did the server explicitly return the contents of the file? How can you tell?
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 information follows the "IF-MODIFIED-SINCE:" header?
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.

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:

5. 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?
6. Which packet number in the trace contains the status code and phrase associated with the response to the HTTP GET request?
7. What is the status code and phrase in the response?
8. How many data-containing TCP segments were needed to carry the single HTTP response and the text of the Bill of Rights?

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. The 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 caite.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.)

Answer the following questions:

9. How many HTTP GET request messages did your browser send? To which Internet addresses were these GET requests sent?
10. Can you tell whether your browser downloaded the two images serially, or whether they were downloaded from the two web sites in parallel? Explain.

2. Date and Time HTTP server [10pts]

You will implement a simple HTTP based server in JAVA using sockets that can accept requests from a web browser and provide the date and time in the time zones: GMT, EST and PST.

You will implement and submit the following classes

Server.java: The server will listen for HTTP connections from web browsers on the localhost on port 5000. The server will accept GET requests for current date and time for a set of time zones. The server will automatically retrieve the current date and time from the time server time.nist.gov on port 13 (as per the Daytime Protocol; see RFC 867 https://tools.ietf.org/html/rfc867) using your client implementation in class TimeClient.java (see below). The output from the time server will be send as an HTML page in response to the web browser. The server will respond to the following URLs:

1. *http://127.0.0.1:5000/time*
   The server will respond with an HTML page containing the dates/times in the time zones GMT, EST and PST.
2. *http://127.0.0.1:5000/time?zone=x* where x can take one of these values: all, est, pst
   The server will respond with an HTML page containing only the times as per the values of x. For x=all, the server will provide GMT, EST and PST dates and times.

For all other urls, the server will respond “Invalid Request”.

TimeClient.java: You will implement a client using JAVA sockets to connect to the time serer at time.nist.gov on port 13. This server provides the date and time in the GMT timezone. The Server implementation will use the TimeClient to query the timeserver for the current date and time. The server will convert the time to appropriate time zones, as needed, and display it as an html page.

Note the following:

1. Use the code stubs for Server.java and TimeClient.java provided as attachments. You will only use the JAVA packages imported in the stub codes, in your implementation. You can add additional variables, methods and user defined classes as needed.
2. The server should be able to respond to multiple client requests, however not simultaneously. In other words, the server should not exit after a request is completed. A good idea during implementation would be for your server to print out the GET requests to figure out how to parse for relevant fields.
3. Test your code using your web browsers for the various URLs. Some sample responses from the server are provided in Figs 1,2, 3 & 4.

Submission: Provide all the .java and .class files. Provide screen shots of your browser with responses as a MS Word file.

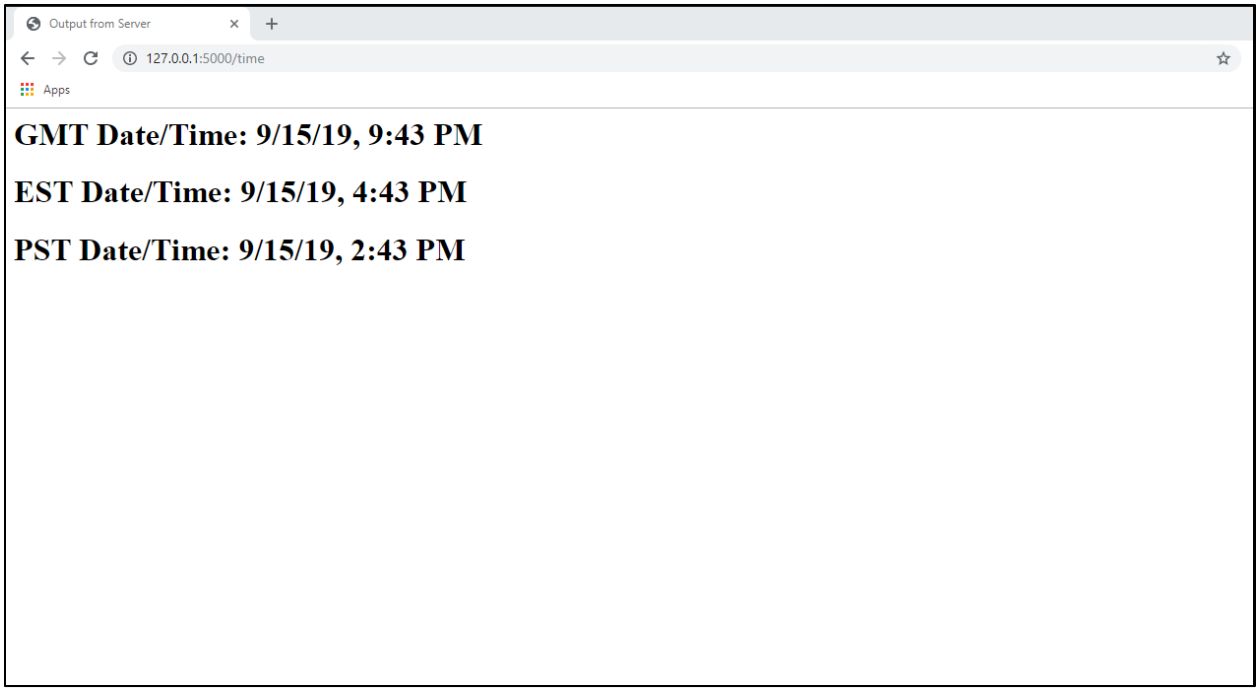

Figure 1 Response to URL http://127.0.0.1:5000/time

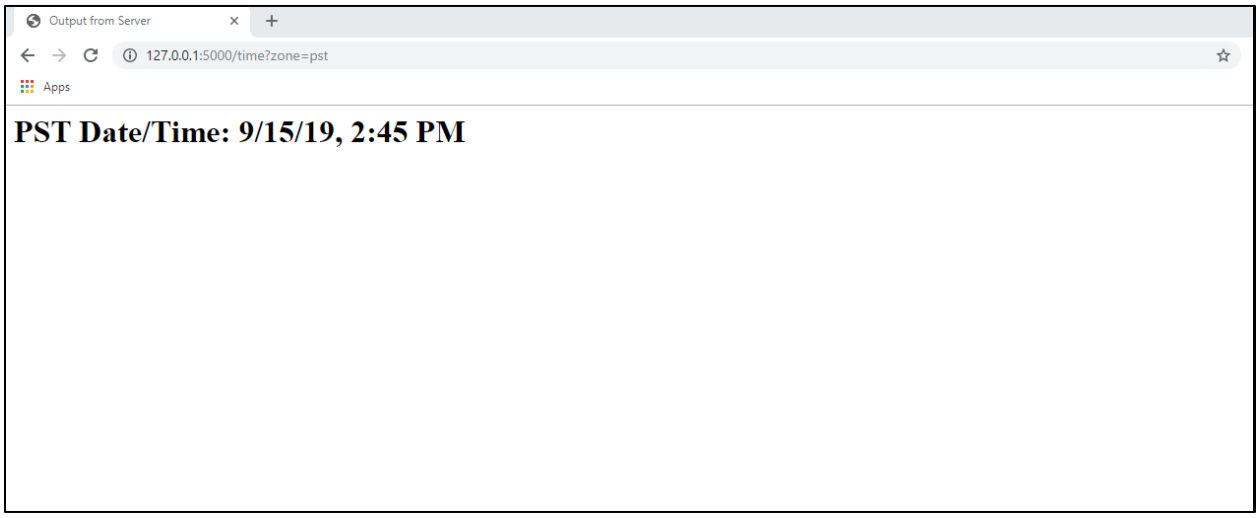

Figure 2 Response to URL http://127.0.0.1:5000/time?zone=pst

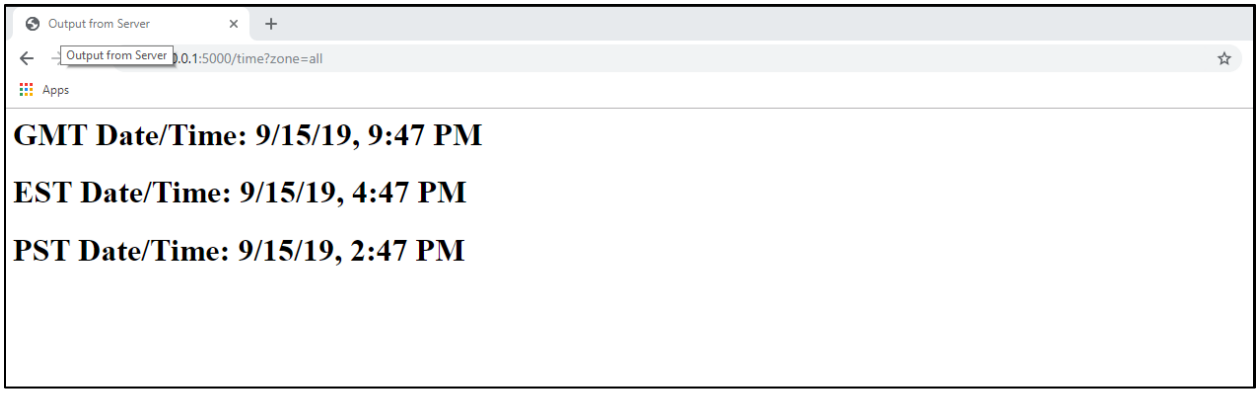

Figure 3 Response to URL http://127.0.0.1:5000/time?zone=all

Figure 4 Response to invalid URL http://127.0.0.1:5000/time?zone