Lab Exercise – HTTP

**Objective**

HTTP (HyperText Transfer Protocol) is the main protocol underlying the Web. HTTP is covered in §7.2.4 of your text. Review that section before doing this lab.

**Step 3: Inspect the Trace**

*To focus on HTTP traffic, enter and apply a filter expression of* “http”. This filter will show HTTP requests and responses, but not the individual packets that are involved. Recall that an HTTP response carrying content will normally be spread across multiple packets. When the last packet in the response arrives, Wireshark assembles the complete response and tags the packet with protocol HTTP. The earlier packets are simply TCP segments carrying data; the last packet tagged HTTP includes a list of all the earlier packets used to make the response. A similar process occurs for the request, but in this case it is common for a request to fit in a single packet. With the filter expression of “http” we will hide the intermediate TCP packets and see only the HTTP requests and responses. With this filter, your Wireshark display should be similar to the figure showing our example.

*Select the first GET in the trace, and expand its HTTP block*. This will let us inspect the details of an HTTP request. Observe that the HTTP header follows the TCP and IP headers, as HTTP is an application protocol that is transported using TCP/IP. To view it, select the packet, find the HTTP block in the middle panel, and expand it (by using the “+” expander or icon). This block is expanded in our figure.

*Explore the headers that are sent along with the request.* First, you will see the GET method at the start of the request, including details such as the path. Then you will see a series of headers in the form of tagged parameters. There may be many headers, and the choice of headers and their values vary from browser to browser. See if you have any of these common headers:

• Host. A mandatory header, it identifies the name (and port) of the server.

• User-Agent. The kind of browser and its capabilities.

• Accept, Accept-Encoding, Accept-Charset, Accept-Language. Descriptions of the formats that will be accepted in the response, e.g., text/html, including its encoding, e.g., gzip, and language.

• Cookie. The name and value of cookies the browser holds for the website.

• Cache-Control. Information about how the response can be cached.

The request information is sent in a simple text and line-based format. If you look in the bottom panel you can read much of the request directly from the packet itself!

*Select the response that corresponds to the first GET in the trace, and expand its HTTP block*. The Info for this packet will indicate “200 OK” in the case of a normal, successful transfer. You will see that the response is similar to the request, with a series of headers that follow the “200 OK” status code. However, different headers will be used, and the headers will be followed by the requested content. See if you have any of these common headers:

• Server. The kind of server and its capabilities.

• Date, Last-Modified. The time of the response and the time the content last changed.

• Cache-Control, Expires, Etag. Information about how the response can be cached.

You are likely to see a variety of other headers too, depending on your browser, server, and choice of content that you requested.

*Answer the following questions:*

*1. What is the format of a header line? Give a simple description that fits the headers you see.*

The format of all the header lines are really simple. They simple have what the correspond to following the information they carry. Ex. “Host : [www.mit.edu\r](http://www.mit.edu\r)\n”

*2. What headers are used to indicate the kind and length of content that is returned in a response?*

Length = Content-Length: 14846\r\n

Kind = Content-Type: image/x-icon\r\n

**Step 5: Complex Pages**

Now let’s examine the third fetch at the end of the trace. This fetch was for a more complex web page that will likely have embedded resources. So the browser will download the initial HTML plus all of the embedded resources needed to render the page, plus other resources that are requested during the ex-ecution of page scripts. As we’ll see, a single page can involve many GETs!

*To summarize the GETs for the third page, bring up a HTTP Load Distribution panel.* You will find this panel under “Statistics” and “HTTP”. You can filter for the packets that are part of the third fetch by re-moving the packets from the earlier part of the trace by either time or number. For example, use “frame.number>27” or “frame.time\_relative>24” for our trace.

Looking at this panel will tell you how many requests were made to which servers. Chances are that your fetch will request content from other servers you might not have suspected to build the page. The-se other servers may include third parties such as content distribution networks, ad networks, and ana-lytics networks. Our panel is shown below – the page fetch involved 95 requests to 4 different servers!



Figure 4: HTTP Load Distribution panel

*For a different kind of summary of the GETs, bring up a HTTP Packet Counter panel.* You will also find this panel under “Statistics” and “HTTP”, and you should filter for the packets that are part of the third fetch as before. This panel will tell you the kinds of request and responses. Our panel is shown in the figure below. You can see that it consists entirely of GET requests that are matched by 200 OK responses. However, there are a variety of other response codes that you might observe in your trace, such as when the resource is already cached.



Figure 5: HTTP Packet Counter panel

You might be curious to know what content is being downloaded by all these requests. As well as seeing the URLs in the Info column, you can get a summary of the URLs in a HTTP Request panel under “Statis-tics” and “HTTP”. Each of the individual requests and responses has the same form we saw in an earlier step. Collectively, they are performed in the process of fetching a complete page with a given URL.

For a more detailed look at the overall page load process, use a site such as Google’s PageSpeed or webpagetest.org. These sites will test a URL of your choice and generate a report of the page load activity, telling what requests were fetched at what times and giving tips for decreasing the overall page load time. We have shown the beginning of the “waterfall” diagram for the page load corresponding to our trace in the figure below. After the initial HTML resource is fetched there are many subsequent quick fetches for embedded resources such as JavaScript scripts, CSS stylesheets, images, and more.

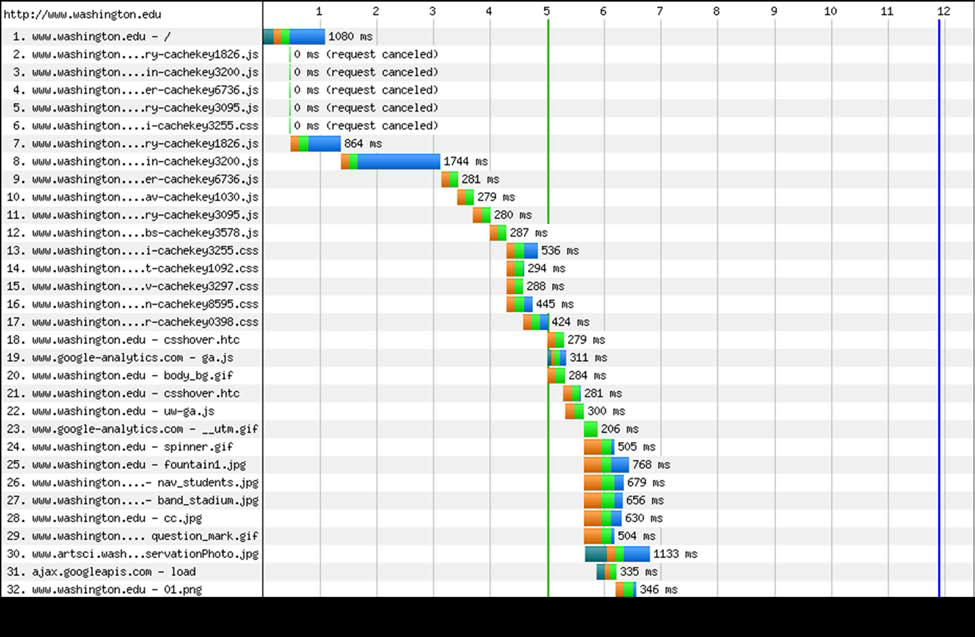


Figure 6: Start of waterfall graph for www.washington.edu (from pageloadtest.org)