HTTP and TCP/IP

Intro

HTTP stands for HyperText Transfer Protocol and TCP/IP stands for Transmission Control Protocol/Internet Protocol. Discussing how TCP/IP works is beyond the scope of this class. For now it can be thought of as something like the telephone system. You dial a number to connect to a server, then you connect to a port which is like an extension. If a program is waiting for that port to 'ring' then it will accept the transmission and start a conversation with the client who called.

HTTP is more fundamental to this class. It is more like that language that is spoken on the 'call' between the client and the server. We previously talked about how requests are made to servers and how servers respond to the requests. HTTP is what defines how that process works.

Within HTTP there are lots of different parts of the specification but for basic web interactions we only care about a handful of topics.

HTTP Request and Response

There are two part to a usual browser/server web interaction. The *request* and the *response*. The request gets sent from the browser to the server. It is composed of two parts, the headers and the body. Often times the body is blank as there is no additional information that needs to get sent along. But if something like a form is submitted to the server by the client then that information is included in the request body sent by the client.

An Example Request

Request Headers

Here is a sample request that is submitting a POST request to a page hosted at OSU. This first block of code are the request headers. This is where most of the magic happens when a client is communicating with a server.

```
POST /about HTTP/1.1
Host: main.oregonstate.edu
Connection: keep-alive
Content-Length: 128
Cache-Control: max-age=0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Origin: http://main.oregonstate.edu
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/44.0.240
Content-Type: application/x-www-form-urlencoded
Referer: http://main.oregonstate.edu/about
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.8
Cookie: mp_fe42a3507c097e9a9d1e9f881d833cfb_mixpanel=%7B%22distinct_id%22%3A%20%2214af9e5614bf2-08c9f53
```

This is a very typical looking request header and we are going to ignore most of the fields in here for now. However, there are a few that we really want to pay attention to.

The first line

This specifies the request type (often a GET or a POST), the resource being requested (in this case the /about page) and the version of HTTP being used.

Host
This is who the client is sending the request to.

Accept
This is the type of file the client is OK getting back from the server. They are listed in order of preference. So here it is saying it wants a text file containing HTML. If it can't have that application specific xhtml and xml is OK and so on. Until it hits */* which means any file will work. There are a TON on content types that can be listed here.

Cookie
This is information, stored on the client that is sent to the server with every request. It can be used to keep track of which client is which when several clients are interacting with a server. It is important to know that browsers are sending this with pretty much every HTTP request.

Referer
This is the page the client came from. It can be occasionally useful for things like analytics but it can be fooled so it is not good to rely on it for important tasks. Referer is the actual spelling in this context.

Request Body

In this case I search for the term "foobar" so it can be seen in the data sent in the request body seen below. In addition there was quite a bit of other data sent along in the body. You will often be inspecting the request body to help debug issues when dealing with HTML forms or AJAX requests.

This is a standard way to format forms that browsers and many JavaScript libraries will take care of automatically. Key value pairs will be in the form of key=value and they will be separated by & symbols.

```
term=foobar&op=Search&group_path=&form_build_id=form-i4fdI34k9NbfdPoXctJDBbagH31-y2MkKhTl0ZS-u4M&form_i
```

It may be somewhat surprising that the body is so much smaller than the headers. We often think of headers as being a summary. And usually this would be correct. But often the request is very simple so there is no need for a substantive body, there are exceptions, like when uploading a file in which case the body is packed with information.

An Example Response

Here is the response that the server gives back to the above request.

```
HTTP/1.1 302 Found
Server: Apache/2.2.15 (CentOS)
X-Powered-By: PHP/5.4.45
X-Drupal-Cache: MISS
Cache-Control: no-cache, must-revalidate, post-check=0, pre-check=0
Content-Language: en
Location: http://main.oregonstate.edu/search/osu/foobar/0/3/
Content-Type: text/html; charset=utf-8
Content-Length: 18
Accept-Ranges: bytes
Date: Tue, 08 Sep 2015 23:58:05 GMT
X-Varnish: 305863683
Age: 0
Via: 1.1 varnish
Connection: close
X-Varnish-Cache: MISS
```

Lets take a look at the important pieces of this response.

The Fist Line
The first line of the response has probably the most important piece of information. The HTTP status code. In this case 302 (which Drupal misuses so signify that a redirect happened). You can read about the various codes in the Wikipedia entry (https://en.wikipedia.org/wiki/List_of_HTTP_status_codes) on HTTP status codes. Generally speaking values strictly less than 400 are OK and values of 400 or greater are a problem.
Location
This is where you actually end up after the server does its thing. In this case the address that shows up in the address bar is not the same one the link pointed to when we did the search. The server generated a page and sent us there after we ran the search. This is basically the sign post that lets the browser know what to display in the address bar.
Content-Type
This lets the browser know what kind of data was sent back. This is how the browser knows if it should try to parse HTML, just display plain text or render an image.
Content-Length
This is how much data was sent back in the response body. At times this can be tremendously helpful with debugging. Sometimes weird bugs will crop up and it is because hidden characters that the browser can't display get sent back. You can use this to compare how much data you expected to get sent back with how much was actually sent back.

Next lets look at the response body. As it turns out, the first thing we got back was the 302 response which was basically an instruction for the browser to go look elsewhere for the information it needed. So the response body was empty. But, we can look at the response body of the place we ended up at.

```
<!DOCTYPE HTML>
<!--[if lte IE 7]> <html lang="en" class="ie ie7 classic"> <![endif]-->
<!--[if IE 8]> <html lang="en" class="ie ie8 classic"> <![endif]-->
<!--[if gt IE 8]><!--><html class="classic" lang="en"><!--<![endif]-->

<head profile="http://www.w3.org/1999/xhtml/vocab">

...

</div> <!-- /page-wrapper -->
  </body>
</html>
```

In this case the response body was the HTML content of the page we asked for. It is heavily abbreviated above so that it could fit easily on this page. There is noting terribly unique about it. It is the same thing you would see if you opened the file in a text editor. The response body is usually just the content of a file. Nothing more, nothing less.

The Life-cycle of a Web Page

You have probably requests tens of thousands of web pages but may not have considered what is actually involved in getting that page to your computer and displaying it so lets break it down.

User Clicks a Link ›

This one is pretty straight forward. You click on a button. This fires off an even that the browser responds to.

Browser Sends a Request

Your browser gets together a fair amount of information and sends it off to the server in a *request*. This request includes a header that includes the request method, the identifier of the thing being requested, what format the client wants the data back in, information about the browser and page state and a bunch of other stuff. It can also include a body with yet more information like stuff to upload or more detailed information about the request.

Server Responds to the Request

After getting routed from server to server, eventually the request ends up at its destination. At this point the server looks for the requested resource, probably on its hard drives somewhere. It might return the thing directly or it might modify it then return it or it might build the entire page on-the-fly and return it based on the state of the server or the contents of the request.

Client Receives the Response

The client gets the response and parses it. It may render it directly or it might change how it renders it based on logic included in the file it got back. Additionally the file it got back might tell the browser it needs yet more files (for example, images to display in the page) and this will cause the browser to fire off more requests.

Client Renders the Page

Finally as the requests finish it will start rendering the page. Depending on the speed of the connection and size of the requests page it may render it instantly or you may see things render one at a time. In any case, once it all finishes the server can do nothing else to influence the page. However, the browser now can make changes, like causing a drop down menu to drop down when you hover over it or change the color of a link when you hover over it. At some point you, the user, will do something that will cause this whole process to kick off again and it will clear the current page and go through the process again to load a new page

Review

For now you should just be generally familiar with the idea of HTTP and understand at a high level what a Request, Response, Header and Body are. Throughout the course you will need to get a deeper and deeper understanding of the concepts. Often times you will need to provide the content of the request header or the response header in order to help debug issues so you need to be comfortable with the terms and the process to go about getting them.