Web

Remember that the internet consists of the architecture that information travels over and the protocols to use it. The web is a set of documents that is sent over the internet under some specific protocols at the application layer.

HTML

Web pages are written in a *markup language* called HTML (HyperText Markup Language).

We have created formatted documents in Word, which is a WYSIWYG[1] editor; in a WYSIWYG editor, you see the formatting of the document as you create it, and it is unusual for a human ever to look at the actual codes in the document that record this formatting; also, any images, sounds, or other elements to be included in the document become part of it, and are generally all stored in one file.

When formatting a document using a markup language, the creator inserts codes in the content to indicate how the document should be formatted, and later another program *renders* the document to show this formatting[2].   HTML documents are rendered by browsers. This can mean that a web page renders differently in different browsers, depending on how each one interprets the codes; this is intentional: you could set up a

[1] What You See Is What You Get

[2] It is possible to use a WYSIWYG editor that then saves using a markup language by automatically generating the correct codes. For example, Word can save to HTML, and can open and render an HTML document.

browser to render web pages into speech instead of visibly on-screen, and read differently formatted text at different volumes or in different voices, for visually impaired users.

Formatting in HTML is indicated by codes called *tags* which are surrounded by angle brackets[3]. Most tags come in pairs. A tag's formatting applies to the text after it, applying to everything until a matching end-tag (starting with a /).

```
<b> bold </b>

<i> italic </i>

<center> centered </center>

<img
src="http://www.harford.edu/resources/images/logo.png
" />

<a href="http://www.harford.edu">Link to HCC main
page</a>
```

The above code is rendered below. The <b> tag indicates bold, <i> italic, and <center> centering. The <img> tag is used to indicate where an image in a separate file should be included when rendering the document – the image is not part of the html document itself. The <a> tag indicates that text should be formatted as a link to another web document.

[3] a.k.a less-than and greater-than signs.

bold *italic*

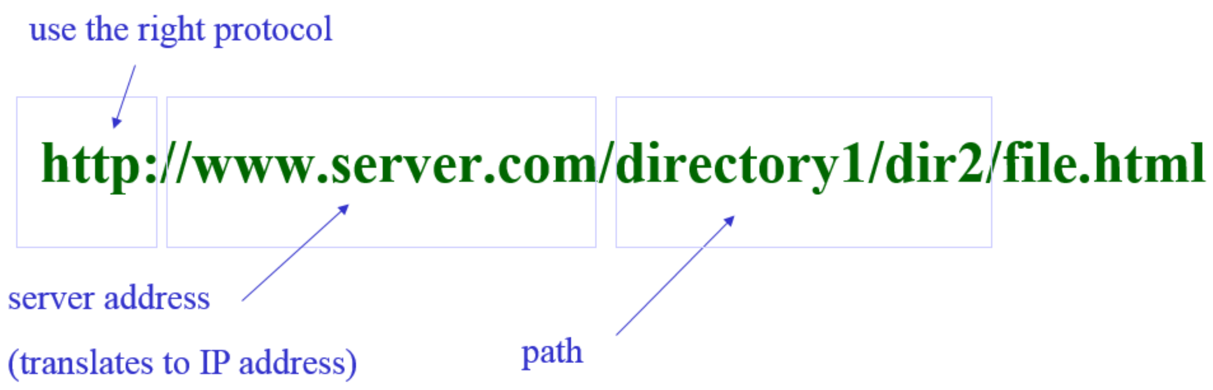
centered

Link to HCC main page

Links from one document to another are what makes web documents special. These allow a document in one file to point a reader at another document in another file, possibly on an entirely different computer.

URL

For links in web documents to work, and to indicate other elements such as images and sounds and videos stored in separate files, we need a way to uniquely identify any other file on the internet. To do this, we use URLs (Uniform Resource Locators). These look like


use the right protocol
http://www.server.com/directory1/dir2/file.html
server address
(translates to IP address)
path

The first part indicates that at the Application Layer, we are using the protocol for web pages[4], HTTP (HyperText Transfer Protocol). The next part indicates which computer on the internet the file we are linking to is stored on. This is the domain name, a unique identifier for the web server (we could instead use its IP address). After the name of the server comes the path to the file – if we start in the root directory of the server's hard drive, what path through directories we should take to find the file.

If you leave out the protocol, web browsers will generally assume you mean HTTP. If you leave out the file name, but include the path, most servers are set up to respond with some default file from that directory. If you leave out the file name and path altogether, most servers are set up to respond with a default file, often known as the home page.

DNS

Every computer on the internet must have an IP address. A URL would work if we put the IP address of the server in, instead of the domain name. In fact, domain names exist only for human convenience. It is much easier to remember "www.harford.edu" than 204.126.204.30. To actually send a request for a web page to a server, we need the IP address, so we need a way to look up the IP address based on the domain name. This is called the Domain Name System (DNS).

There are so many domain names in the world that it would be unreasonable to store them all on your computer. Instead, the DNS is a system of servers whose job is to translate domain

[4] For most things on the web, this is obvious, although we might, for instance, use a more secure version called HTTPS instead. URLs are also used for other applications that have other protocols.

names to IP addresses. Each DNS server has a table with some of the domain names and their corresponding IP addresses. Your ISP probably provides a DNS server, and there are many others all over the internet.

When you enter a new domain name for the first time on your computer, your browser sends a request for a DNS lookup to a DNS server. This might be simply the DNS server provided by your ISP, or your device might specify in its internet settings to use a specific DNS server.

The most popular addresses, such as Google and Yahoo, are probably in every table. If the address you want is at the first DNS server you ask, it can find it in the table and send it right back to your browser. Otherwise, your request is sent on to another DNS server. Domain names are organized into large categories based on the last part of the name, and some DNS servers specialize in .com, .gov, or .edu addresses. Eventually, the address you requested is found in the table of a DNS server, and that IP address is sent back to you.

Some DNS servers also specialize in *not* providing the IP addresses of certain domains. For example a DNS server might maintain a list of known ad servers and return an error or dummy IP address when those are requested. By setting your domain to use that DNS server you would avoid seeing many ads because when your browser began the process to request an ad it would never be told what IP number to request it from.

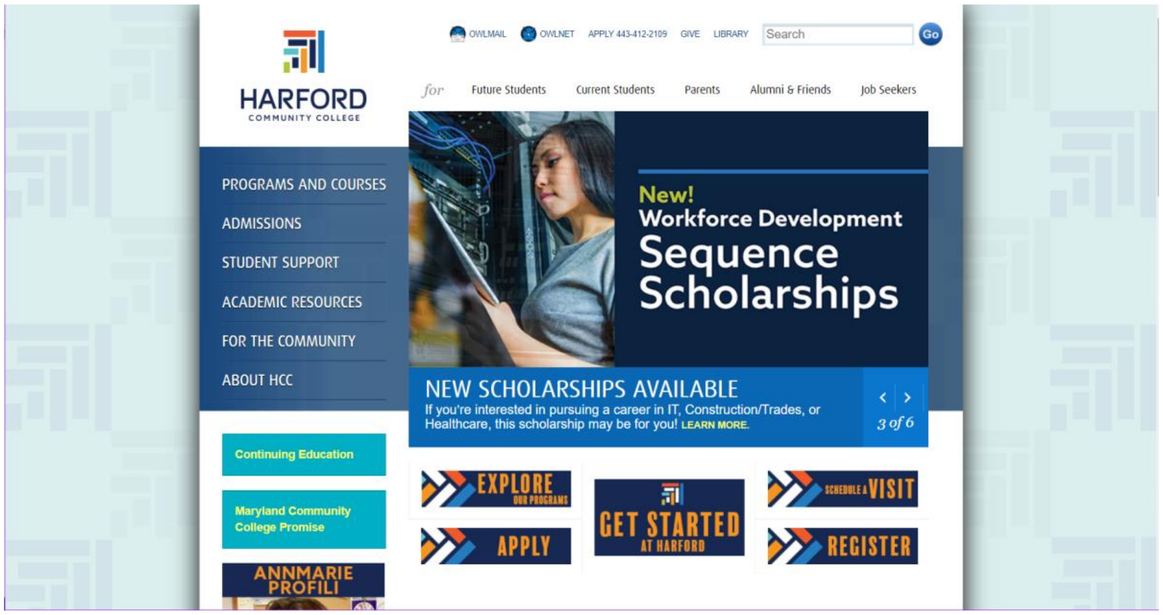
Note that all these steps – you enter the URL with domain name, your browser sends the DNS lookup request, the request passes through (potentially) several DNS servers, the IP address is sent back to your browser – has to happen before you can even send a request for a web page to a web server.

Once you have done this once, your computer can store the web server's IP address and won't have to do another DNS lookup for it next time.

Web Pages

Part of the HTTP sets out how we get a web page once we have the IP address of a server: the browser sends the server a GET command, which needs to include the URL for the page we want.

On receiving a GET command, the server chooses what file to send back based on this URL. It will usually be the file found at that URL, but what if there was a mistake, or that page is not currently available? Servers are set up to send different files back in response to different such circumstances; for instance a 404 page is sent back to indicate that the path and file name in the URL requested doesn't match any file on the server's hard drive.



If you're interested in pursuing a career in IT, Construction/Trades, or Healthcare, this scholarship may be for you!

A GET command only includes one URL, and the response is just one file. So if you send a request for http://www.harford.edu, you only get the HTML document for the Harford main page. Note that this web page has a large number of images on it. Remember that each image lives in its own file and is not part of the HTML document.

So, when the browser receives the HTML document, it begins to render it, showing the bold parts as bold and the italic parts as italic and so on. When it reaches an <img> tag, this specifies the URL of an image file, and the browser must send a new GET command to retrieve that file from the server. The image might, depending on the URL, actually be on a different server from the server that stored the original web page, in which case the computer might have to do another DNS lookup. Each time it finds another <img> tag, or any other such element, such as a video, sound, game, etc, the browser must do another GET.

Some more modern sites are more complex. For instance, some sites use infinite scrolling, so that instead of loading the entire page at once, you load a screenful at a time, triggering new GET commands for the new content each time you scroll.

Many sites have tools for interaction, such as text editors for posting or image editors that run within the site. Some of these make clever use of the more complex parts of HTML. Others send complex commands through GET commands back to a program on the server that sends back changes to the web page code in response. Others make use of extensions, programs that interact with your browser, running on your computer rather than on the server, to control what the browser shows you.

Cookies

Another part of the HTTP is that the protocol is stateless; this means that the server is not allowed to remember the client computers it has communicated with. Suppose you log in to OwlNet. Along with the URL in the GET command, your browser sends your username and password. Assuming you don't make a typo, the server then sends you back the page for your OwlNet.

But what happens when you click on one of the other tabs in OwlNet? Your browser sends a GET command for this page, which you need to be logged in to view. As far as the server is concerned, it has never heard from your IP address before, so why should it believe that this computer is logged in?

Since the fact that your computer is logged in as you cannot be stored on the server, it is stored instead on your computer. When you log in, the server sends back not only the first page from OwlNet, but a small file with any information it needs to remind itself that your IP address is currently allowed to view your OwlNet. This file is called a *cookie*.

Each time your browser sends a GET to the web server[5], it includes the cookie this server gave it last time. The server can then use this cookie (which it has, essentially, sent to itself through your computer) to verify that it has already logged you in.

Cookies almost always include time-stamps which the server checks to make sure they are current. If you don't interact with the sever for long enough, it may decide that when you do, the cookie has expired and instead of the page you wanted, it sends back the login page, so that you can login again and then receive a fresh cookie to that effect.

[5] Other events, like small programs that run within in a web page (for instance JavaScript) can also trigger cookies to be sent back and forth.

Every time your browser sends a GET to a web server, it checks if it has a cookie stored from that server, and if so sends this as part of the GET.

Note that it doesn’t just send every cookie to every server, so one server doesn’t get another server’s information (at least not through cookies). Also note that the cookie is a data file, not a program, so the cookie itself can’t do anything on your computer.

Cookies are used any time the server wants to associate information with your computer.

If you have an account on the website, then the cookie may just be used to tell the server that a certain IP address is you, and the rest of the information can be stored in your account instead of in the cookie. If you don’t have an account, lots of information can also just be stored in the cookie. Some shopping sites even use cookies to store whole temporary shopping carts – keeping track of what you clicked to buy -- so that you can shop even without an account.

What kinds of information can a cookie store about you? Anything that you (that is your computer) sent to the server. At its most general, this is really just a list of every GET command you sent to the server, each with a time stamp. But most of your interactions with a website – anything you click on and in some cases what your mouse moves over – will send a GET to the server.

GET can include extra information such as text you typed in, or the setting you chose on a slider, but just based on what you click a server’s cookie can store your choices and opinions any time you click light vs dark mode, click a heart or a thumbs-down, or just which pictures or posts you expand from thumbnails. Some sites use time-stamps to know what time of day you typically show interest in a certain topic, or to determine your level of interest in something based on how long the time between GET commands is (e.g. if the GET commands slow down as you are scrolling through a long list of media, they know what you spent more time looking at the most recently loaded media).

3rd Party Cookies

Because the images on a web page can be on a different server than the page itself, you can have cookies on your computer from servers whose actual pages you've never visited.

On many web pages you will see logos for social media sites such as twitter or facebook and these images live on the companies' own servers. Suppose there is a facebook[6] logo on HCC's home page. Then as part of loading the HCC web page, your browser sends a GET command to facebook for this image. Since you sent it a GET, the facebook server can send back a cookie to your computer along with the image file. This is a 3rd party cookie, a cookie that does not belong to the page you are visiting but to some 3rd party. Note that you *did not* have to click on the facebook logo for this to happen. If you can see the logo, the GET has already been sent and the cookie is already on your computer.

Suppose that later on, using the same computer, you visit a site on horses that also has a facebook logo. Then your computer will send another GET command to facebook, but this time it has a facebook cookie to send with it. Now facebook knows that someone on this computer was interested in HCC and in horses. It updates the cookie and sends it back to your computer. If later in the day you look up hotsauce, and visit a page with another

[6] I'm using facebook for this example since facebook pioneered the use of most kinds of personal data gathering, but most sites do the same thing.

facebook logo, that’s another element in the list of things facebook has listed (in the cookie) that the user of this computer is interested in.

If you happen to also be logged into facebook on the same computer, facebook can associate the contents of that cookie with your account, and you may find that facebook starts showing you ads for colleges, horse races, or hotsauce.

Consider the ad images you see[7] on most commercial web pages. These ads are stored on servers belonging to advertising companies. Suppose you visit two different pages with recipes for crabcakes, each of which has an ad that lives on the same ad server. Then later you visit a totally different page with another ad from the same server. Your computer sends the ad server a GET for the new ad image, with the cookie that it has been updating each time you visited the previous pages. The ad server, based on this cookie, can choose to send you an image that is an ad for Old Bay, because they have reason to believe you may be interested. It doesn’t matter that you never visited a web page on the ad server, they can still track what pages you have visited using cookies, through images that do live on their server.

Now that more people know about what cookies can be used for, they have started to come under legal restrictions – in some countries every site has to explicitly warn visitors that cookies are being used. So advertisers work on new technologies to extract information without technically breaking laws.

[7] In fact, many companies use “invisible pixels” – tiny one-pixel images that blend into the background but still trigger a cookie to be sent/updated, so you may have a cookie updated without even seeing an ad. This is just one of many tricks that a site could agree to do in order to make sure that your computer talks to the servers of advertisers paying to get cookies onto more computers.

Browser fingerprinting, for example, sends a request to the browser for detailed information that a web page might need to determine how to display graphical information (screen size, fonts available, plug-ins/extensions installed) and tries to use these to uniquely identify a device. This approach may also be used by banks to try to identify fraud: if multiple "different customers" are all logging in from an identical fingerprint, the bank may suspect it is actually one person using several stolen log-ins.

Visiting a Web Page

Let's put all of this together.

When you type a URL into your browser or click on a link or bookmark, your browser starts the process of getting the web page for you.

If you have never viewed a page on that server before, your computer doesn't know what IP address to send packets to yet, so it must do a DNS lookup. It sends a DNS lookup request to the nearest DNS server, and eventually gets back the IP address

Once you have the IP address (whether from a DNS lookup or stored from earlier) your browser can send the GET command. If it has communicated with this server before, it will include any cookies that the server sent it last time, along with the URL of the page.

The server receives the GET command, and chooses which file to send back, based on the URL and any cookies. It may also create new cookies or update existing cookies, and send them back along with the file.

Your browser begins to render the HTML of the web page it receives. For any images, sounds, etc., on that page, it must send another GET command, which may mean more DNS lookups or more cookies.

Finally the browser finishes rendering the page.

Remember that all this is happening at the Application Layer. Anytime something is sent, whether it is a DNS lookup request, a GET command, or a web page, that message is still broken into packets, moved across the internet by routers, reassembled at the other end, etc.