Computer Science 50
Fall 2011
Scribe Notes

Week 8 Monday: October 24, 2011
Andrew Sellergren

Contents

1 Announcements and Demos (0:00–11:00) 2

2 The Internet (11:00–41:00) 2
2.1 DNS and Routers 2
2.2 HTTP 5

3 HTML (41:00–72:00) 5

Computer Science 50
Fall 2011
Scribe Notes

Week 8 Monday: October 24, 2011
Andrew Sellergren

1 Announcements and Demos (0:00–11:00)

- Check out what is possible in web development these days using technologies like WebGL! If you right click on this page to View Source, you'll see the JavaScript which makes all the magic happen.
- If you're wondering how to design a website poorly, look no farther than David's UC President campaign website, of which we've provided a screenshot in this week's slides.
- To see what we're getting ourselves into with the internet, take a look at this trailer from Warriors of the Net.
- The Problem Set 5 Scavenger Hunt is afoot! Time to get together with your section and find the staff members whose photos you recovered. The deadline is Monday at noon.
- The Big Board is live! This completely optional contest pits you against your fellow classmates to implement the fastest spellchecker possible in Problem Set 6. To submit your code to the Big Board, simply run the command `check50`. Your code will be checked for memory leaks and then benchmarked as it spellchecks the standard set of texts.
- The Final Project specification has now been posted. Think of it as your chance to show off what you've learned! Historically, 80% of students have tackled web-based projects.
- To help you quickly pick up various technologies that might be of use to you as you implement your Final Project, we offer a series of seminars hosted by staff. All seminars will be taped and put online in case you are unable to attend in person.
- If you're short on ideas for your Final Project, check out projects.cs50.net, a repository of ideas from across campus.
- Interested in tech internships? Attend the Tech Interview Seminar on Friday, from 2 to 4 p.m. in Pierce 209 to help you prepare for the kinds of questions you will face during the application process. Word to the wise: don't try to show off when a recruiter asks you what your favorite programming language is. David chose assembly language and was then subjected to follow-up questions about this quite obscure programming language.

2 The Internet (11:00–41:00)

2.1 DNS and Routers

- What is the internet? According to Ted Stevens, it's a series of tubes. More properly, though, it is an infrastructure that connects millions of computers across the world.

- What is the World Wide Web? Although it is often conflated with the internet itself, the World Wide Web is actually the interconnected collection of web pages that is accessible via the internet.
- The relationship between your computer and the computer that hosts a web page is one of client and server. The server constantly listens for requests and when it receives one, it responds with the appropriate information. The protocol that describes how this exchange occurs is called HTTP (hypertext transfer protocol). Other protocols describe how to use the internet to send e-mail, make video calls, and more. Consider an analogy to the real world: when you meet someone for the first time, protocol dictates that you and he or she shake hands.
- In its simplest form, a request for a web page takes the following form:

  ```
  GET / HTTP/1.1
  ```

  The `/` means that the root, or homepage, of the site is being requested. `GET` is the request type and finally, `HTTP/1.0` describes the format of the request, the language that the client and server should speak to one another.
- You can think of web page requests as letters. The nature of the request is formatted as above and written on a piece of paper that is stuffed in an envelope. The envelope is then addressed to the web server's IP address. IP stands for internet protocol and IP addresses take the form w.x.y.z, where w, x, y, and z are numbers 0 through 255.[1] Given that they consist of 4 numbers, each of which requires 8 bits to store, IP addresses are only represented by 32 bits and thus can only take on one of approximately $2^{32} \approx 4$ billion values. That might sound like more than enough, but the truth is that we're running out, especially in light of the recent surge in smartphone usage. The newest version of IP, IPv6, will address this shortage by specifying IP addresses using 128 bits rather than 32. You can find your current IP address in the network settings of your computer. David's right now is 140.247.118.68. Most Harvard IPs start with 140.247.
- We'd be happy to address our web page request to Google's IP if only we knew it. Thankfully, DNS, or domain name system, allows us to look it up. DNS refers to a series of computers across the internet whose sole purpose is to translate domain names like google.com into IP addresses. The advantage of this, of course, is that your web browser doesn't need to know Google's IP address (which might change periodically) because it can always look it up.
- Now that we know how to address our envelope, how do we send it? We begin by passing it along to a *router*. Just as our computer knows its own

[1] At least IPv4 addresses. IPv6 addresses take a different form.

IP address, it also knows its router's IP address. In order to send our envelope to its final destination, we'll first address it to a router, which will then address it to another router, and so on. Each router along the way might not know about all of the other routers on the internet, but it maintains some mapping of IP addresses to directions so that it knows how to forward a request. Generally speaking, your envelope will pass through 10 to 30 intermediate routers before reaching its final destination.

- By the way, from here on out, we'll refer to our envelope by a more technical term: packet.
- We can actually watch a packet traveling from router to router using a command called `traceroute` from the Appliance. Note that in order for this command to work, David has to `ssh` into the Harvard Computer Society's servers in order to bypass Harvard's firewalls and their restrictions. Having done so, he runs this command:

  ```
  traceroute www.stanford.edu
  ```

  This command will spit out a list of the domain names of the routers which our data passes through on its way to Stanford. In total, it looks like our packet passes through 17 steps on its way to Palo Alto during a trip which takes less than 90 milliseconds. Alongside each of the displayed domain names are three independent measurements of the time it took our data to reach this router. The first few lines list routers that belong to Harvard with values on the order of 2-3 milliseconds next to them. Next, the domain names begin to contain the prefix `nox`, which stands for Northern Crossroads, a large datacenter in the Northeast. A few steps later we see the prefixes `kans`, `hous`, and `lax`, which we can guess stand for Kansas, Houston, and Los Angeles. You'll notice that the time measurements are somewhat proportional to the physical distances between these routers, but not always since congestion can enter into the equation. Note also that if we run this same command again, we might get different results: for good reason, the routes that packets take on the internet are dynamic.
- Similarly, we can run `traceroute` on `cnn.com` to find out its latency and the location of its nearest datacenter.[2] Note that if some of the rows show `* * *`, it's because the corresponding gateways didn't respond to us, possibly for privacy reasons. What's perhaps more interesting is running `traceroute cnn.co.jp`, which will connect us to the Japanese version of CNN. When we do so, we'll see that the request takes much longer, on the order of hundreds of milliseconds, presumably because it actually has to traverse the United State and then the Pacific Ocean!

[2] Alternatively, you can use `traceroute` to discover someone's IP address and connection speed!!!!1111 Okay, actually, please please know that this video is just plain wrong. I don't want to have to give points back to someone on a quiz.

2.2 HTTP

- As we begin our foray into web programming, we'll be relying on a multitude of development tools. One of these tools is Live HTTP Headers, a Firefox plugin that allows us to sniff the packets that our browser is sending to various web servers. When we open this plugin and navigate to `http://www.google.com/`, we see a series of plaintext messages that resemble the following:

```
GET / HTTP/1.1
Host: www.google.com.com
User-Agent: Mozilla/5.0 (X11; Linux i686; rv:7.0.1) Gecko/20100101 Firefox/7.0.1
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Connection: keep-alive
```

  We hinted at this earlier when we said that web requests typically begin with `GET`, but as we can see here, these requests are actually somewhat more complicated. Most of the so-called HTTP headers that comprise web requests are uninteresting, but we'll touch upon a few right now. The Host header specifies the full domain of the web page we're requesting. This is useful on shared servers which host more than one web site: without this reminder, the server wouldn't know which files to retrieve. The User-Agent header sends the version of your browser and operating system to the web server. The Cookie header is like a handstamp: it tells the web server that you've been there before and you'd like some of your preferences remembered.

- If we scroll down in our Live HTTP Headers window, we can see the HTTP response headers, excerpted below:

```
HTTP/1.1 200 OK
Date: Mon, 24 Oct 2011 17:44:22 GMT
Expires: -1
Cache-Control: private, max-age=0
Content-Type: text/html; charset=UTF-8
```

  The number 200 is an HTTP response code that indicates everything's okay. You might be more familiar with response code 404, which means that the page you're looking for wasn't found.

3 HTML (41:00–72:00)

- If you have some preconceived notion of web servers as gigantic computers in warehouses, you will be surprised to find out that even your own laptop

can act as a web server, albeit a slightly inconvenient one. Because your IP address is constantly changing and is not known to the outside world combined with certain security settings, your laptop isn't all that useful in serving up web pages to other users. However, it works just fine for serving them up to you as you develop them.

- To begin configuring our laptop to act as a web server, we'll `cd` to our home directory on the Appliance and create a new folder named `public_html`. Within this folder, we'll run `gedit index.html` and type the following:

```
<!DOCTYPE html>
```

This header declares a file written in the latest and greatest version of HTML, version 5.

- HTML consists of *tags* that are nested within each other to create a page's structure. The outermost tag in an HTML document is the `html` tag. Within this tag, the document is separated into two parts, the head and the body:

```
<!DOCTYPE html>

<html>
  <head>
  </head>
  <body>
  </body>
</html>
```

For now, we're doing our best to pretty print our HTML, although it actually doesn't matter to the interpreter: you can get rid of all the whitespace in an HTML document and it will display the same as if it were indented perfectly.

- To begin adding actual content to our page, we'll insert a `title` tag:

```
<!DOCTYPE html>

<html>
  <head>
    <title>hello, world</title>
  </head>
  <body>
    hello, world, again
  </body>
</html>
```

The contents of the `title` tag will be displayed in the browser bar and the contents of the `body` tag will be displayed in the actual browser window.

- Woohoo, our first web page! Just like our C source code, our HTML source code is nothing but a text file. Unlike our C source code, however, our HTML source code won't be compiled. Keep in mind that HTML is not a programming language, per se, because there is no notion of functions or statements.

- To access our first web page in our browser, we'll navigate to `http://localhost/~jharvard/` in our Appliance's web browser. Oops, we get a 403 Forbidden error. Believe it or not, this is a good thing. What it means is that the files you create on your computer are not, by default, readable by everyone.

- To examine the permissions that are set for our `public_html` directory, we execute the `ls -l` command from our home directory:

  ```
  drwx------ 2 jharvard students 4096 Oct 24 14:01 public_html
  ```

  The first `d` indicates that this is a directory. The next 9 placeholders dictate the permissions for the owner, the group, and everyone else. The owner of this directory is `jharvard` and the group of this directory is `students`. The three letters `rwx` indicate the the owner of this directory has read, write, and execute permissions. The `---` for both group and everyone else indicate that they have no permissions at all. In order for our web page to be visible on the internet, we'll need to give "everyone else" execute permissions for our `public_html` directory. To do this, we execute the following command:

  ```
  chmod a+x public_html
  ```

  Once we've done this, `ls -l` reveals the change we've made:

  ```
  drwx--x--x 2 jharvard students 4096 Oct 24 14:01 public_html
  ```

- When we try reloading our web page, we still get a 403 Forbidden error. At this point, we might suspect that the permissions of `index.html` are the culprit, but when we examine them, we see that it is world-readable. However, when we navigate to our home directory and execute `ls -alh`, the `-a` meaning "all," we see that the following at the top:

  ```
  drwx------ 28 jharvard students 4096 Oct 24 13:56 .
  ```

  The `.` indicates that this line refers to the present working directory, in this case, the home directory. As you can see, our home directory is not world-executable. To fix this, we run:

```
chmod a+x .
```

Finally, when we reload our web page, we see the text "hello, world, again."

- The `~jharvard` in our URL tells the web server to look in the `public_html` directory belonging to the user `jharvard`. If we want to access this same web page from outside the Appliance (but still on our laptop), we can navigate to `http://192.168.56.50/~jharvard/`, as will be documented in the next problem set. This IP address might be reminiscent of IP addresses you've seen associated with your home computer. That's because 192.168 is a convention for designating private IP addresses. Note that for the next problem set and your final project, you'll have the opportunity to host your website on the cloud so that it will be accessible as a subpage of `http://cloud.cs50.net`.
- Let's spice up our website a little bit with some bold text and color:

```
<!DOCTYPE html>

<html>
  <head>
    <title>hello, world</title>
  </head>
  <body bgcolor="pink">
    <b>hello</b>, world, again
  </body>
</html>
```

HTML tags can have parameters or *attributes* provided to them within their declaration. Here, we're specifying the `bgcolor` attribute with a value of `pink`. As the name implies, this attribute controls the background color of our body.

- As we add more content to our page, we may decide that we want to style some parts differently than others. We can do this by segmenting our content into tags, specifically `div` tags. If we place our "hello, world, again" message within a `div`, we can specify its style separately from other parts of the page

```
<!DOCTYPE html>

<html>
  <head>
    <title>hello, world</title>
  </head>
  <body bgcolor="pink">
    <div style="color: red">
```

```
      <b>hello</b>, world, again
    </div>
  </body>
</html>
```

Within the style attribute, we can specify a number of different rules of the form `rule: value`, separated by semicolons. Here, we've indicated that we want the text color of this `div` to be red.

- Next, let's add another line of text:

```
<!DOCTYPE html>

<html>
  <head>
    <title>hello, world</title>
  </head>
  <body bgcolor="pink">
    <div style="color: red">
      <b>hello</b>, world, again
      goodbye, world!
    </div>
  </body>
</html>
```

We might expect that "goodbye, world!" would appear on a separate line of our page, but recall that whitespace is ignored in HTML. In order to start a new line, we have to explicitly insert a line break:

```
<!DOCTYPE html>

<html>
  <head>
    <title>hello, world</title>
  </head>
  <body bgcolor="pink">
    <div style="color: red">
      <b>hello</b>, world, again
      <br>
      goodbye, world!
    </div>
  </body>
</html>
```

The line break tag is an empty tag, meaning that a single set of angle brackets both opens and closes it. That is, it's written as `<br>`, not `<br></br>`, although technically there's nothing wrong with the latter.

- It would be a shame if our website were a dead end, so let's add some links to it using the anchor tag:

```
<!DOCTYPE html>

<html>
  <head>
    <title>hello, world</title>
  </head>
  <body bgcolor="pink">
    <div style="color: red">
      <b>hello</b>, world, again
      <br>
      goodbye, world!
    </div>
    <div>
      Here are some of my favorite websites.
      <ul>
        <li><a href="https://cs50.net/">cs50.net</a></li>
        <li><a href="http://myspace.com/">facebook.com</a></li>
      </ul>
    </div>
  </body>
</html>
```

  The `ul` tag also enables us to create an unordered list dotted with bullet points. The `href` attribute of the `a` tag designates where you want your link to point. In this example, we've tricked the user into thinking she's going to Facebook when in fact the link points to MySpace. If she's savvy, though, she'll know that hovering over the link often gives a small tooltip that indicates where it points.

- Adding images to our website requires using the `img` tag. Incidentally, to upload images to the Appliance, you can follow the instructions here, which are quite simple. In order for our image to display properly, we must add it to our `public_html` directory and also set its permissions so that it is world-readable. As it turns out, we can even make our images into links:

```
<!DOCTYPE html>

<html>
  <head>
    <title>hello, world</title>
  </head>
  <body bgcolor="pink">
```

```
    <div style="color: red">
      <b>hello</b>, world, again
      <br>
      goodbye, world!
    </div>
    <div>
      Here are some of my favorite websites.
      <ul>
        <li><a href="https://cs50.net/">cs50.net</a></li>
        <li><a href="http://myspace.com/">facebook.com</a></li>
      </ul>
    </div>
    <div>
      <a href="http://www.facebook.com/rbowden91"><img src="eyes.png"></a>
    </div>
  </body>
</html>
```

- And on that note, we'll leave you to dream about Rob and the possibilities of HTML!