Week 7

This is CS50. Harvard University. Fall 2014.

Cheng Gong

Table of Contents

How the Internet Works

Servers and TCP/IP

- Today we transition from lower-level C programming to higher-level web programming.
- Let's watch a quick (inaccurate) clip[1] from the TV show Numb3rs[2] that shows us "how the Internet works."
- The last frame of that clip is a screen of code with a reference to `crayon`, suggesting that the code that the "hacker" is using is probably just some drawing program.
- Additionally, the top of the screen shows `http://275.3.6.28` in the address bar, which is still wrong since it's invalid (probably to keep viewers from visiting a real server), but that number is called an IP address.
- More generally, **IP** is **Internet Protocol**, which really just means that every computer (and phone and tablet) on the Internet follows a certain set of rules.
- An **IP address** is a unique address that identifies these devices on the Internet (well, these days we're actually running out of addresses, but more on that in a second).

[1] http://youtu.be/5ceaqtWhdnI
[2] https://en.wikipedia.org/wiki/Numbers_(TV_series)

    # An IP address is like a postal address. The Maxwell Dworkin building on campus has an address of 33 Oxford Street, Cambridge, Massachusetts, 02138, USA. This is a unique address in the world, and likewise computers have unique addresses.

- IP address take the form of `#.#.#.#`, where each number is in the range of 0-255.

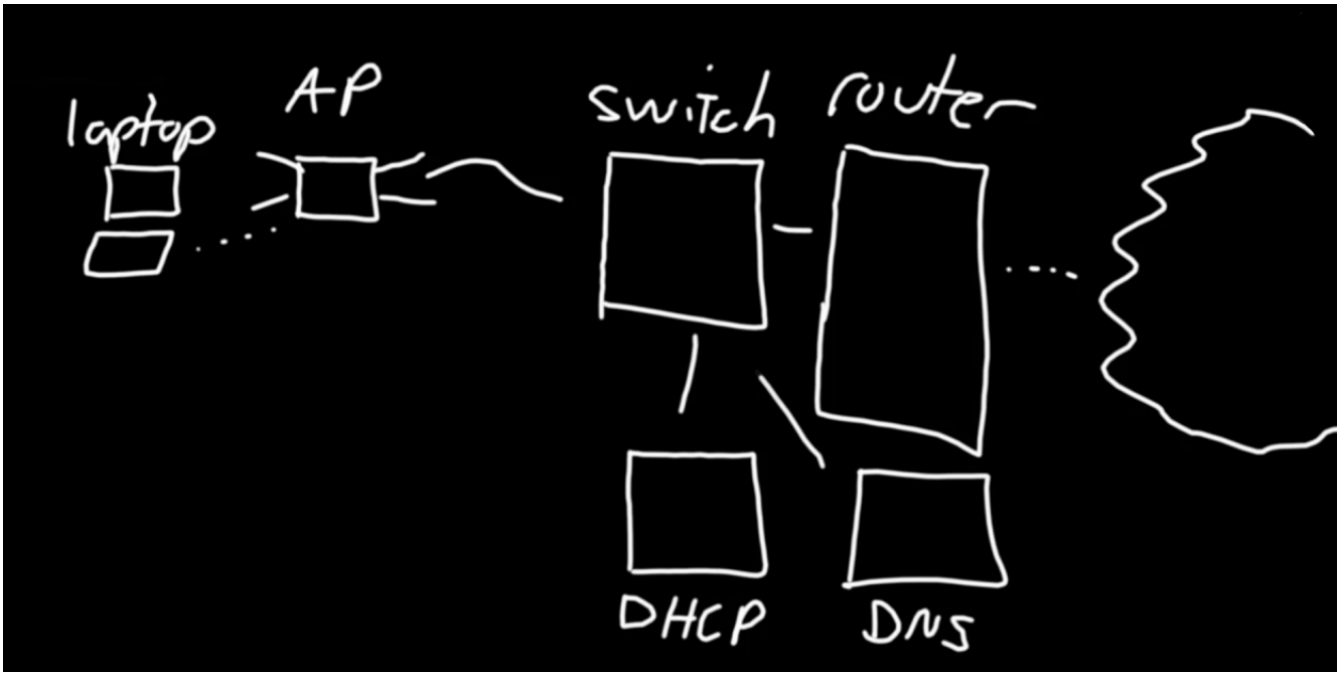
    # Each number, then, uses 8 bits, and in total the address is 32 bits, making for a total number of roughly 4 billion possible addresses.

- Though 4 billion is a high number, we have lots of servers and devices, which is starting to be problematic.

    # Specific ranges are also reserved for particular organizations or providers. For example, many of the computers at Harvard will be assigned an IP address that starts with 140.247.#.# or 128.103.#.#, with MIT down the street having its own range as well. Providers like Comcast also has a particular prefix that they own.

- Addresses that start with 10.#.#.#, 172.16.#.# - 172.31.#.#, or 192.168.#.# are **private IP addresses** that we have set aside to use within a particular network, but not on the Internet at large. (The CS50 Appliance likely has a private IP address starting with 192.168.#.# for your host operating system to communicate with.)

- Let's look at a simple picture:



    # Notice that the laptop connects wirelessly to an **AP**, **access point**, which typically comes with antennas. The access point allows devices to talk to the rest of the network wirelessly. At home, this might be called a home router, made by D-Link or Linksys or the like.

    # The access point is connected to a **switch**, which is connected to a **router**, and there is other equipment in between the router and the cloud on the right, representing the rest of the world.

        # A switch is just a simple device with many ports in it to connect a number of devices with cables.

    # We also have at least two other servers, one being a DHCP server and the other a DNS server.

    # At home, the switch and router and DHCP and DNS servers are replaced by a cable modem that connects to the equipment that the Internet provider, like Verizon or Comcast, is running somewhere else for all its customers.

- **DHCP** stands for **Dynamic Host Configuration Protocol**, which is how unique IP addresses are dynamically assigned. A DHCP server, then, just gives your computer an IP address.
- **DNS** is a bit more interesting, standing for **Domain Name System**. These servers translate the URLs of websites to IP addresses, and vice versa.

    # To draw an analogy, lots of companies buy 1-800 numbers with words at the end, like 1-800-COLLECT, so people don't need to remember those numbers but just one word.

    # **Hostnames**, or fully qualified domain names, allow us to do something similar, addressing servers by names rather than numbers.

- Let's open a Terminal window and do the following:

```
% nslookup facebook.com
Server:   140.247.233.163 ❶
Address:  140.247.233.163#53 ❷

Non-authoritative answer:
Name: facebook.com
Address: 173.252.120.6 ❸
```

The first part, lines 2-3, is the address of Harvard's DNS servers, and the last line, line 7, is its response to the question of what Facebook's IP address really is. And if we copied that number, and went to `http://173.252.120.6`, we'll indeed end up at facebook.com.

- If we did the same with google.com, we get this:

```
% nslookup google.com
Server:   140.247.233.163
Address:  140.247.233.163#53

Non-authoritative answer:
Name: google.com
Address: 74.125.226.68
Name: google.com
Address: 74.125.226.67
Name: google.com
Address: 74.125.226.78
Name: google.com
Address: 74.125.226.65
Name: google.com
Address: 74.125.226.71
Name: google.com
Address: 74.125.226.72
Name: google.com
Address: 74.125.226.70
Name: google.com
Address: 74.125.226.73
Name: google.com
Address: 74.125.226.69
Name: google.com
Address: 74.125.226.64
Name: google.com
Address: 74.125.226.66
```

Sometimes companies tell the world they have one IP address, which ends up being resolved, or mapped, to a whole bunch of servers after, or, like Google, they tell the world that there are a number of addresses, any of which you can contact.

- So that's what's been happening under the hood when you type in the name of a website: your operating system asks the DNS server what the address of this website is.

- The final device in the picture, a **router**, is in charge of "routing" stuff: sending packets, or envelopes of digital information, from sender to receiver.
- To demonstrate this, we have a picture of Rob that we want to send to Dan in the back of the lecture hall. In human terms, we'd say something like, "Can you pass this to Dan?" and pass it along until it finally reached him.
- Computers, rather than using "Dan," will use Dan's IP. We'll use an envelope and write "Dan's IP" on it in a "To:" field, since it doesn't matter what it is, and likewise we'll put "My IP" in the "From:" field.
- Routers on the Internet will see this envelope and know, by prior configuration, that if the IP address starts with a certain number it will go in a certain direction (this is a bit of a simplification but the gist is there).
- Routers also allow "guaranteed" delivery. **TCP**, **Transmission Control Protocol**, is another technology used on the Internet, often used together with IP (you may have seen TCP/IP).
  - # On the Internet, servers might lose, or drop, packets, and we want to avoid this.
- So let's take Rob's photo, cut it into quarters, and place each one into different envelopes, labeling them 1 of 4, 2 of 4, 3 of 4, and 4 of 4.
  - # But before we send that, let's remember that there are lots of services on the Internet beside regular web browsing. Email, chat, and file storage are all examples, and servers can do any of those things. So we need to somehow tell Dan about the type of message that we're sending him in the envelope, so he can open it with the right program.
- With TCP, we have a set of conventional numbers associated with certain services:
  - # ports
    21 FTP
    25 SMTP
    53 DNS
    80 HTTP
    443 HTTPS
  - # For example, **FTP**, file transfer protocol, was assigned a unique identifier of 21 some years ago.
  - # **SMTP**, for outbound email, is 25.

    # **DNS** uses 53 for its queries, or questions of what the address of a website might be.

    # And you may have seen that HTTP, web traffic, and HTTPS, secure web traffic, use 80 and 443.

        # The number for HTTPS can be greater than 255 because they have to do with TCP, not IP (which is 4 numbers, 0-255). A port number in TCP is a separate 16-bit integer value, so in theory can be really big, but in practice under a few thousand.

- So on the envelope we want to send Dan, we can write "Dan's IP:80", indicating that it is a webpage, in this case a webpage that contains Rob's picture.
- Now we can hand out each of the envelopes, even to separate routers, and in theory all four should make their way to the back of the lecture hall.
- Then Dan can reassemble the picture and realize that there should be 4 pieces.

    # But what if one router is broken or is powered off, and a packet doesn't make it to Dan? TCP tells computers to send a packet back - a message from Dan to David - telling him which packets in the original message were missing, since they were all numbered.

- There are other protocols and technologies that we rely on, but most likely TCP and IP are the ones used for the most popular of services.

Traceroute

- We can actually see the routers that our messages go through.
- If we wanted to see the servers between us and MIT, we can type `traceroute -q 1 www.mit-edu`, which runs the `traceroute` program in quiet mode, once, to MIT's website:

```
% traceroute -q 1 www.mit.edu
traceroute to www.mit.edu (23.10.80.128), 30 hops max, 40 byte packets ❶
 1  10.243.16.161 (10.243.16.161)  0.662 ms
 2  10.240.144.33 (10.240.144.33)  1.044 ms
 3  core-sc-1-gw-vl415.fas.harvard.edu (140.247.2.61)  1.302 ms ❷
 4  bdrgw1-te-4-2-core.net.harvard.edu (128.103.0.18)  1.321 ms ❸
 5  nox1sumgw1-vl-503-nox-harvard.net.harvard.edu (207.210.142.53)  2.001
 ms ❹
```

```
 6  192.5.89.21 (192.5.89.21)  1.714 ms
 7  et-5-0-0.120.ny0.tr-cps.internet2.edu (198.71.47.57)  6.144 ms ❺
 8  a23-10-80-128.deploy.static.akamaitechnologies.com (23.10.80.128)
7.349 ms ❻
```

Each of these rows is like a student in the audience between David and Dan, who passed the message along.

On line 2 we see the domain name typed in, and `23.10.80.128` is apparently the IP address of `www.mit.edu` that the computer figured out using DNS. We're also going to limit ourselves to `30 hops`, or by going through no more than 30 servers between us and MIT.

Then each row is a router, with the first two routers having no name, just some private (because they start with 10.#.#.#) IP address.

They only take a few milliseconds to reach, and when we get to the third router at **(3)**, we see that it has some cryptic name.Having been told how by Harvard's network administrators, we can break the name down as identifying this particular router as one in the `core` of the network, located in `sc`, the Science Center, and acts as a `gw`, gateway, which is a synonym for router.

That server is connected to another one at **(4)**, nicknamed `bdrgw1`, border gateway.

The next one at **(5)** is nicknamed "northern crossroads", which is just where lots of cables from lots of entities are connected.

The next one is unnamed and not very interesting, but router **(7)** has a part that says `ny` - implying New York, where this router is probably located (conventionally we name routers by the city or airport they're located near), and the fact that it took about 6 milliseconds (as opposed to one or two) makes sense.

Finally, the actual domain name for `www.mit.edu`, **(8)**, seems to indicate a server that's part of a company called Akamai that they've outsourced server hosting to.

- Let's try to contact our friend, Professor Nick Parlante, who runs `nifty.stanford.edu`:

```
% traceroute -q 1 nifty.stanford.edu
traceroute to nifty.stanford.edu (171.64.64.16), 30 hops max, 40 byte
 packets
```

```
 1  10.243.16.161 (10.243.16.161)  0.892 ms
 2  10.240.144.33 (10.240.144.33)  1.155 ms
 3  coregw1-vl-415-fas.net.harvard.edu (140.247.2.61)  1.383 ms
 4  bdrgw1-te-4-2-core.net.harvard.edu (128.103.0.18)  1.375 ms
 5  nox300gw1-vl-500-nox-harvard.nox.org (192.5.89.97)  1.880 ms
 6  192.5.88.22 (192.5.89.22)  1.875 ms
 7  nox1sumgw1-peer-nox-internet2-192-5-89-18.nox.org (192.5.89.18)
14.364 ms
 8  et-10-0-0.107.rtr.chic.net.internet2.edu (198.71.45.8)  23.959 ms ❶
 9  et-4-0-0.110.rtr.salt.net.internet2.edu (198.71.45.19)  55.141 ms ❷
10  et-5-0-0.112.rtr.losa.net.internet2.edu (198.71.45.22)  67.849 ms ❸
11  et-5-0-0.112.rtr.losa.net.internet2.edu (198.71.45.22)  67.826 ms
12  svl-hpr2--lax-hpr2-10g-2.cenic.net (137.164.25.50)  82.134 ms ❹
13  svl-hpr2--lax-hpr2-10g.cenic.net (137.164.25.38)  82.137 ms
14  csmx-west-rtr-vl9.SUNet (171.66.255.214)  82.605 ms
15  thneed.Stanford.EDU (171.64.64.16)  82.837 ms
```

So now we're going through a longer list of routers and cities, with router number **(8)** probably located in Chicago, **(9)** in Salt Lake City, **(10)** in Los Angeles, and **(12)** LAX. Finally, it goes from Southern California to Northern California to where Stanford is.

It would take about 82 milliseconds to send a message to California, but let's go further to `www.cnn.co.jp`, CNN in Japan:

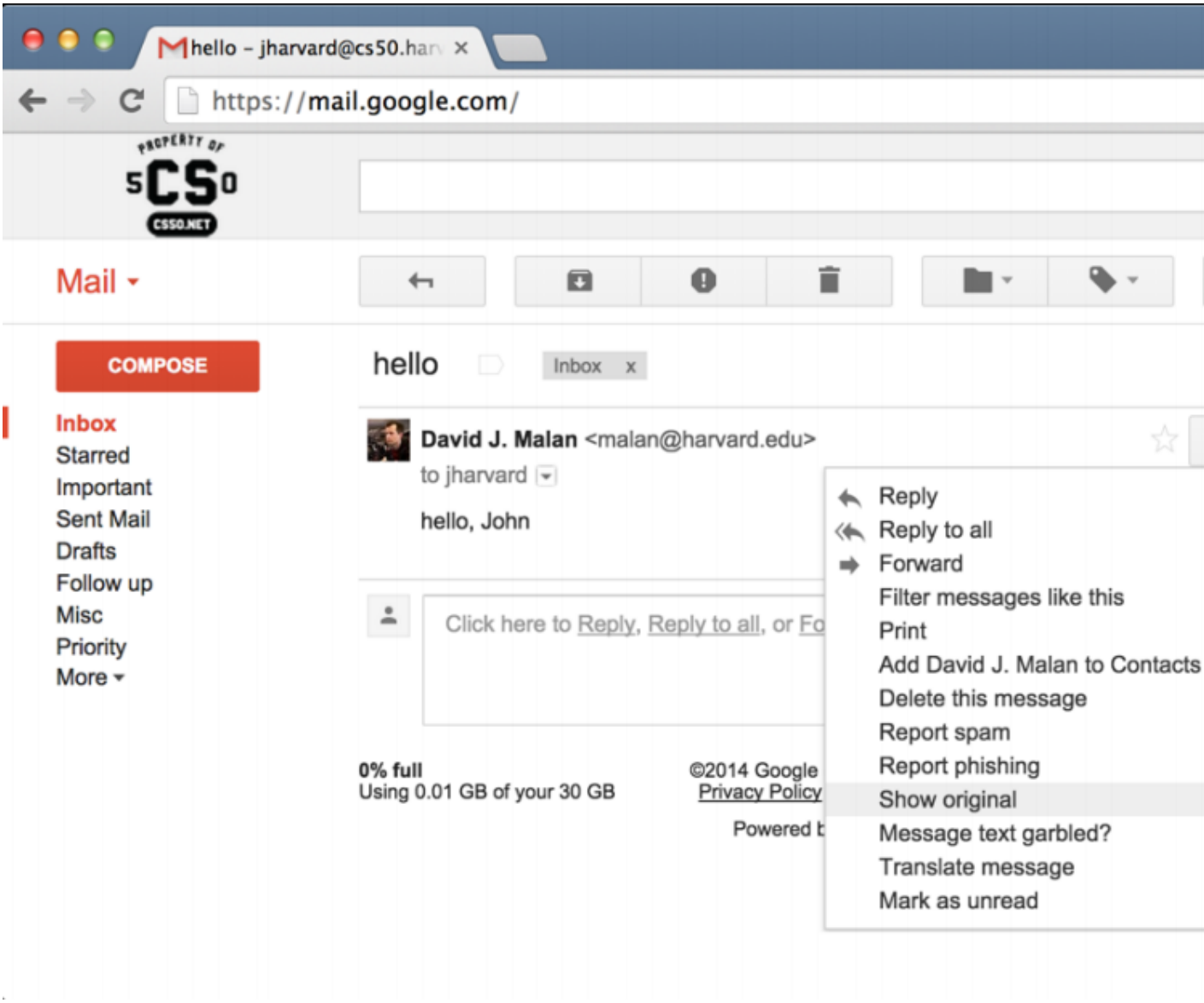
```
$ traceroute -q 1 www.cnn.co.jp
traceroute to www.cnn.co.jp (14.0.42.95), 30 hops max, 40 byte packets
 1  10.243.16.161 (10.243.16.161)  0.926 ms
 2  10.240.144.33 (10.240.144.33)  227.831 ms
 3  core-sc-1-gw-vl415.fas.harvard.edu (140.247.2.61)  1.460 ms
 4  bdrgw2-te-4-2-core.net.harvard.edu (128.103.0.2)  1.460 ms
 5  xe-11-2-0.bar2.Boston1.Level3.net (4.53.56.9)  0.879 ms
 6  * ❶
 7  124.215.192.77 (124.215.192.77)  79.800 ms ❷
 8  * ❸
 9  otejbb205.int-gw.kddi.ne.jp (203.181.100.137)  180.785 ms
10  sjkBBAC07.bb.kddi.ne.jp (106.162.175.154)  188.651 ms
11  obpBBAC03.bb.kddi.ne.jp (111.87.242.70)  192.322 ms
12  111.86.159.66 (111.86.159.66)  185.208 ms
13  14.0.40.86 (14.0.40.86)  187.124 ms
14  lajbb001.int-gw.kddi.ne.jp (59.128.2.209)  75.087 ms
15  otejbb206.int-gw.kddi.ne.jp (203.181.100.25)  185.154 ms
```

```
16  sjkBBAC07.bb.kddi.ne.jp (118.152.210.246)  172.626 ms
17  14.0.42.95 (14.0.42.95)  184.472 ms ❹
```

Looks like servers **(6)** and **(8)** aren't responding, if they're being private, but we can see that between routers **(7)** and **(17)** that there's a huge jump in time that it takes for the message to send, so between **7** and **9** there's probably a body of water, with transatlantic or transpacific cables connecting these servers.

But here, despite the number of hours it would take to fly to Japan, our message took under than 200 milliseconds to send.

- So you can play around, and some servers might give you a `*` as an answer for privacy's sake, but generally you can see the route your message takes.
- It turns out that Gmail has this little triangle at the top right of every email:



And you can click **Show original** which will show you lots of information like timestamps, IP addresses, and domain names. More importantly it will show you the headers that have been hidden in every email that you've ever sent or received, from which we can infer where and whom the email came from.

- We'll talk about how emails can be generated by programs, so websites can email their users, but also show how easy it is to forge emails unless you can verify headers (which is also not infallible these days).

- Let’s go up a layer, from IP (which addresses packets for us) and TCP (which ensures those packets are delivered), to another protocol (how computers talk to one another), HTTP.
- **HTTP** stands for **Hypertext Transfer Protocol**, or what web browsers use to speak to web servers.
- When you visit a website, your computer first translates its name, like "facebook.com", into an IP address, and then sends that server a message, saying something like "Give me today’s homepage" or "Give me the login page" or "Give me the default view".
- As another analogy, humans shake hands and introduce each other before having a conversation, and computers do something similar.

HTTP Requests

- Consider the following picture (it’s a bit dated as you can tell by the appearance of the computers):

  # The client is your machine that asks for information, and the server is the machine that responds with information.

- **GET** is a term for how computers get information. They make a request in the form of a textual message that literally says something like this:

```
GET / HTTP/1.1
```

```
Host: www.google.com
...
```

This simple message would be opened by the server on the other side, which then responds accordingly.

The `/` right after `GET` is just asking for the root directory, or the highest directory. To properly visit a website, we should really be typing `http://www.facebook.com/` with that final `/` meaning we want the root of the hard drive, or the default page.

The next part, `HTTP/1.1`, means that we're using version 1.1 of HTTP to talk to the server.

- So now we get something like this back as a response:

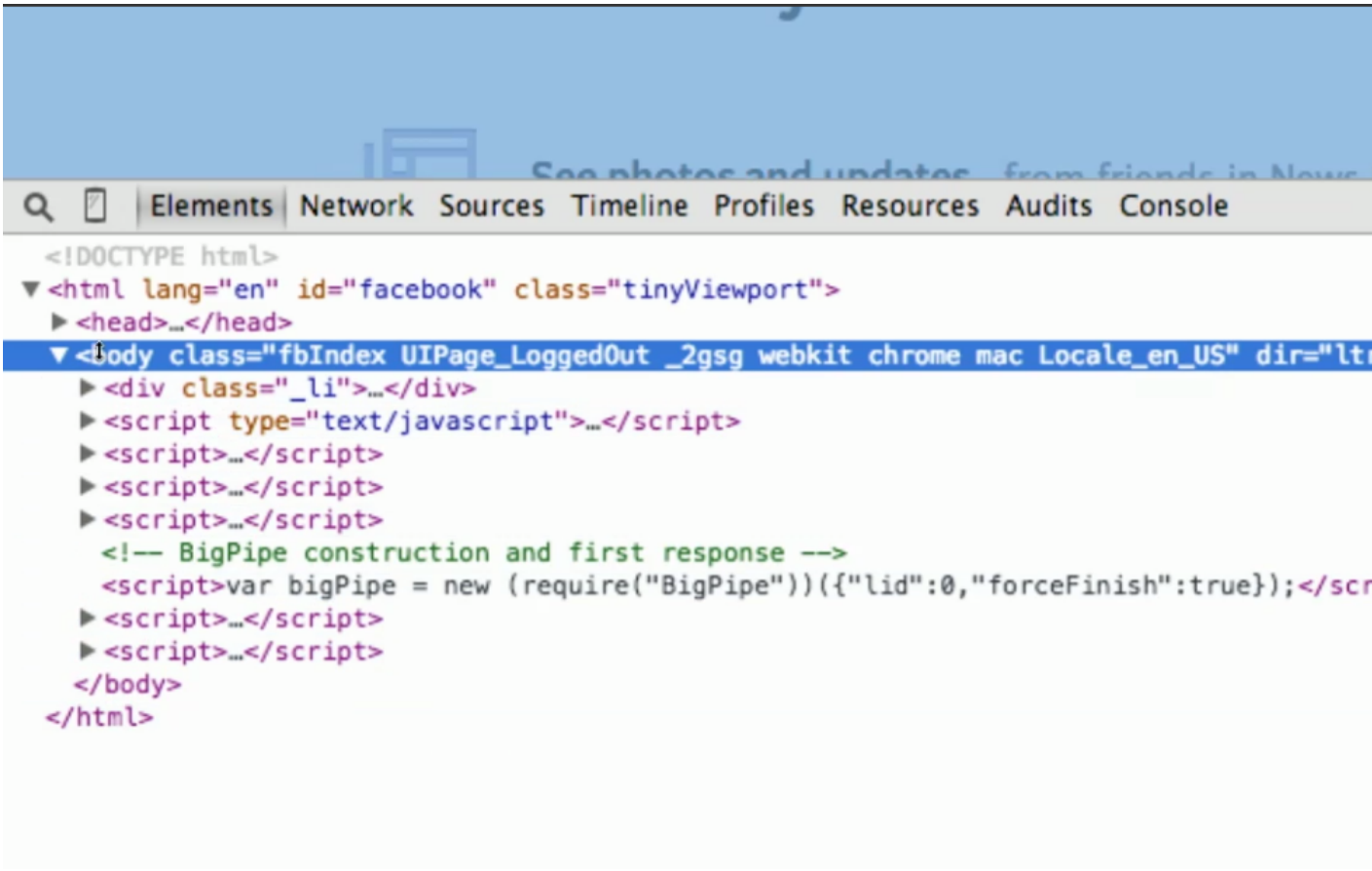
```
HTTP/1.1 200 OK
Content-Type: text/html
...
```

The first line is confirming that we're using version 1.1 of HTTP to communicate, and `200` is a status code that means `OK`: the server has the page we're looking for.

The second line is telling the web browser that you're getting back a webpage of the type `text/html` (as opposed to an image or video, for example).

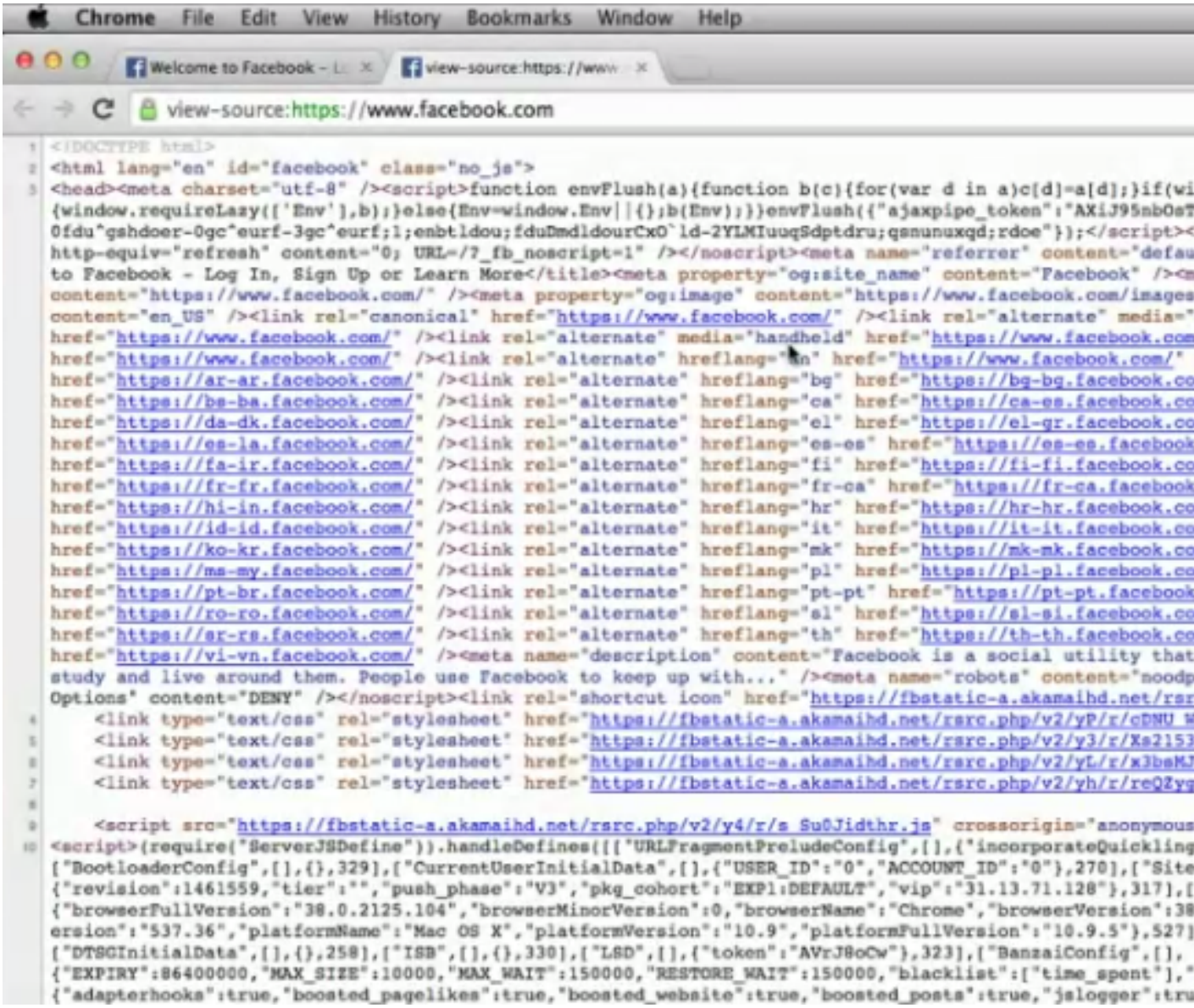
And then the `…` is the actual message that the server responds with.

- Let's open `www.facebook.com` with Chrome in the appliance, and under **View**, select **Developer** and then **Developer Tools**:



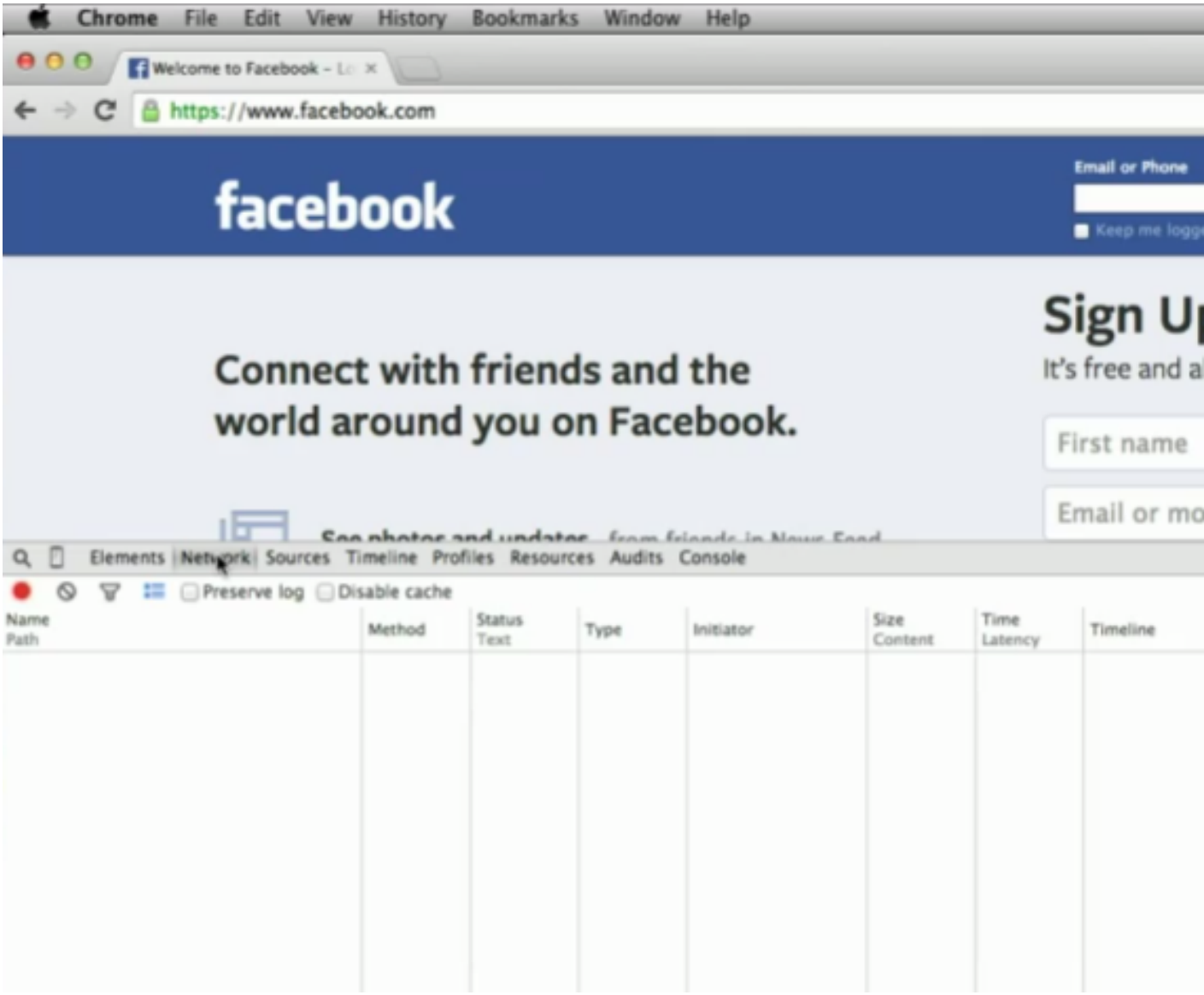
You'll see lots of cryptic text, and the response is in the language called **HTML**, **Hypertext Markup Language**. It's not a programming language because it doesn't have functions or loops, but a markup language in that it has tags and attributes that tell a browser what to display on the screen and how to display it, like centering or bolding text, or changing its color.

- In fact, we can go to **View**, **Developer**, and then **View Source** to see the full output of Facebook:



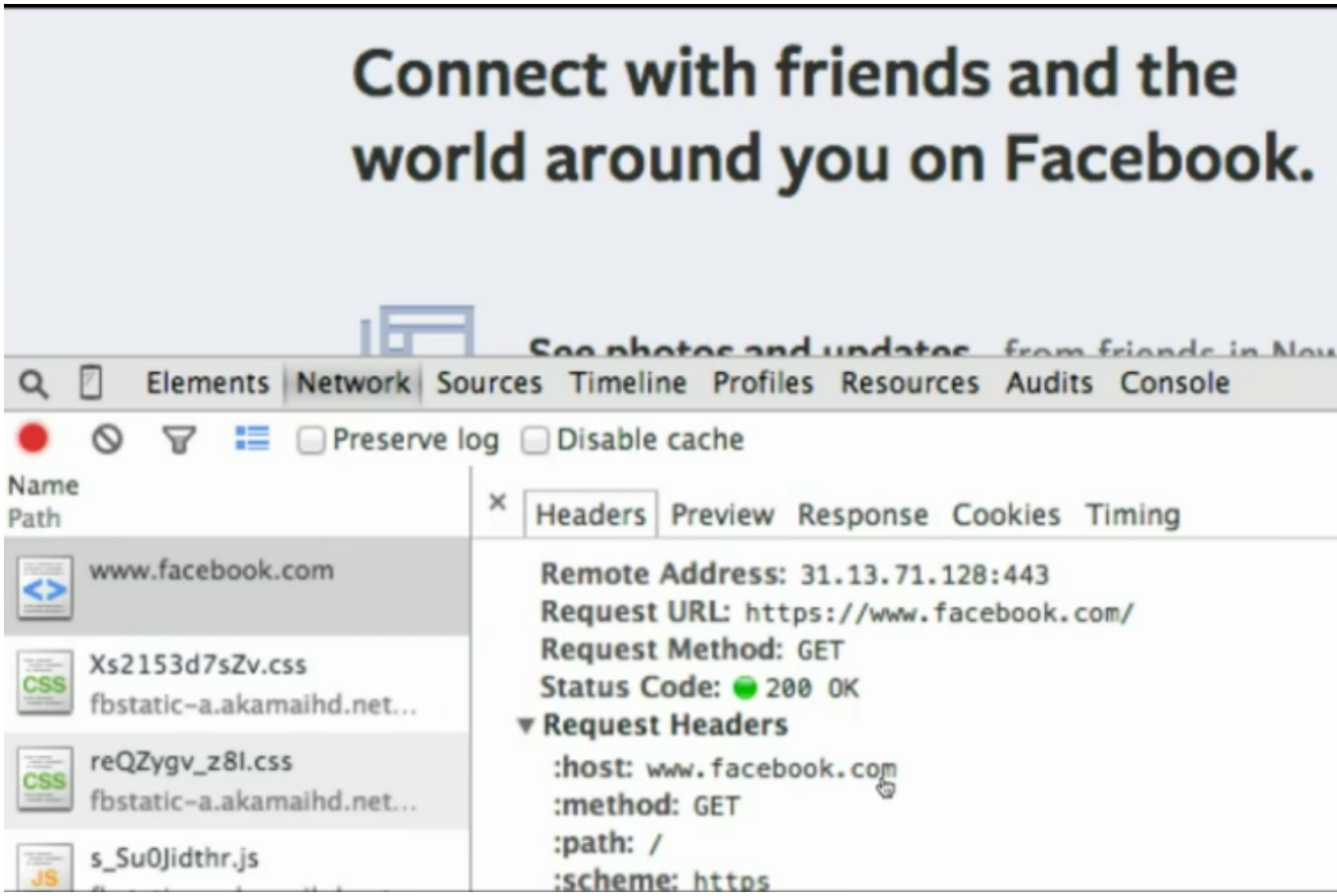
The output would get 0 out of 5 for style, but in this case since they're serving billions of pages, removing the unnecessary spaces and tabs saves time and bandwidth and ultimately money.

- In Developer Tools, Chrome is taking that minified HTML and formatting it so we can read it more easily, but it's the same code.
- Let's switch to the **Network** tab in Developer Tools:

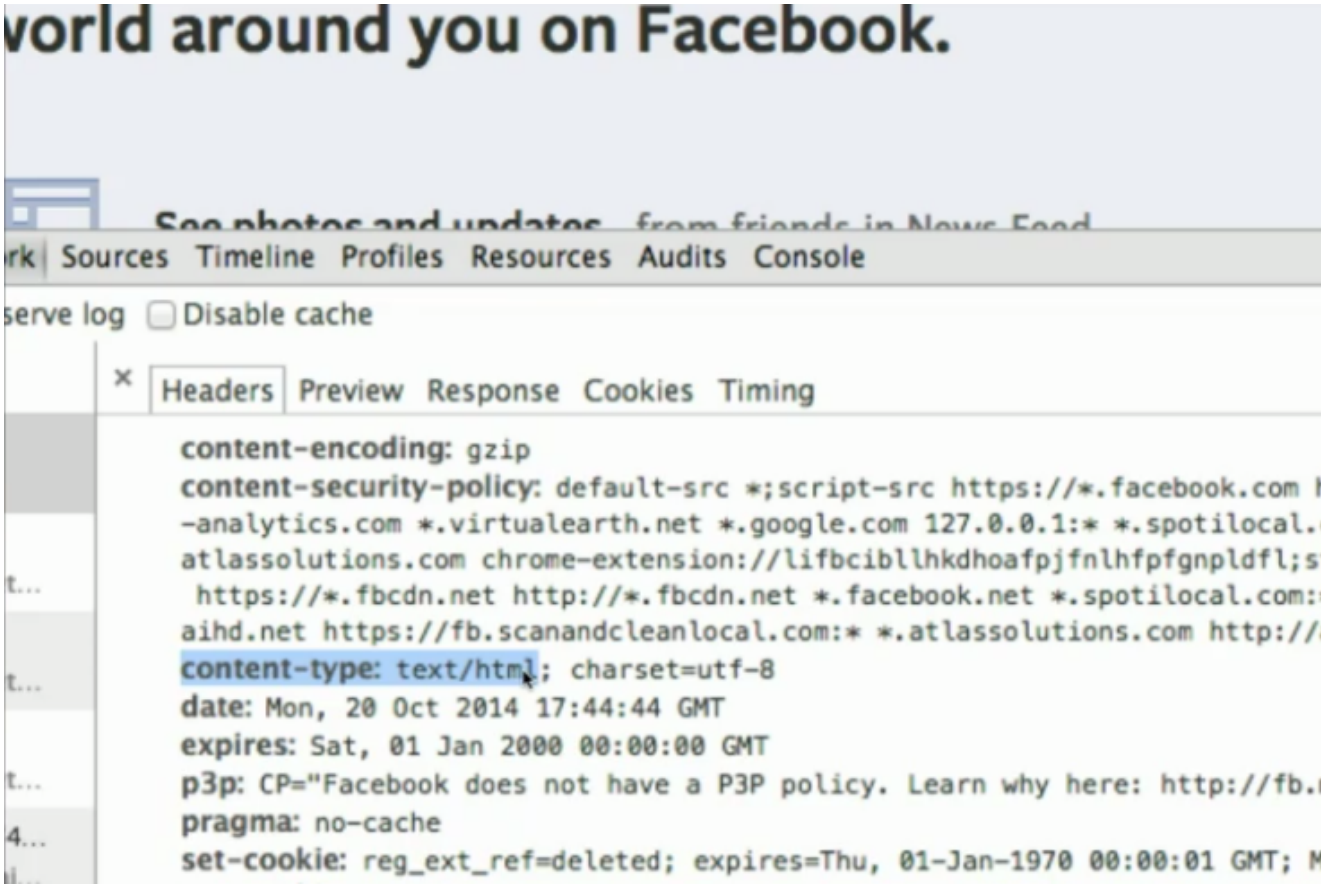


- This will show us all the requests between us and Facebook, so we can reload the page, and see a bunch of requests, but at the very top, this:

Notice that we are using a GET request that gave us a status code of 200, meaning we found the page. And if we click on that row, we can actually see the full request:



If we scroll down, we see that Facebook sends back a bunch of things, including `content-type: text/html` (The server is actually sending us `Content-Type: text/html`, but Chrome reformats it to all lowercase for aesthetic reasons.):



    - We'll take a look at cookies soon.

- In short, for years, every time you've visited a webpage, you've sent requests and receiving these responses, and on the outside of every message you've sent has been your IP address.

    # Someone somewhere will know who's registered to which IP address, and eventually the logs can be traced back to you, making for very little privacy.

- Here are some more possible status codes:

    # 200 OK
      301 Moved Permanently
      302 Found
      401 Unauthorized
      403 Forbidden
      404 Not Found
      500 Internal Server Error

You've probably seen at least the 404, which means the file doesn't exist.

HTML

- Let's also look at this bit of HTML:

```
<!DOCTYPE html>

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

It does nothing other than display `hello, world`, and we notice that the first line declares this piece of code as using HTML, followed by various tags beginning with `<` and ending with `>`.

- Let's go to the appliance, or anywhere you have a text editor, and save a file titled `hello.html` somewhere simple, like the Desktop.
- Then we can start with something like this:

```
<!DOCTYPE html>

<html>

</html>
```

Notice that we type out `</html>`, closing the tag, ahead of time, and remember to put everything else inside those tags.

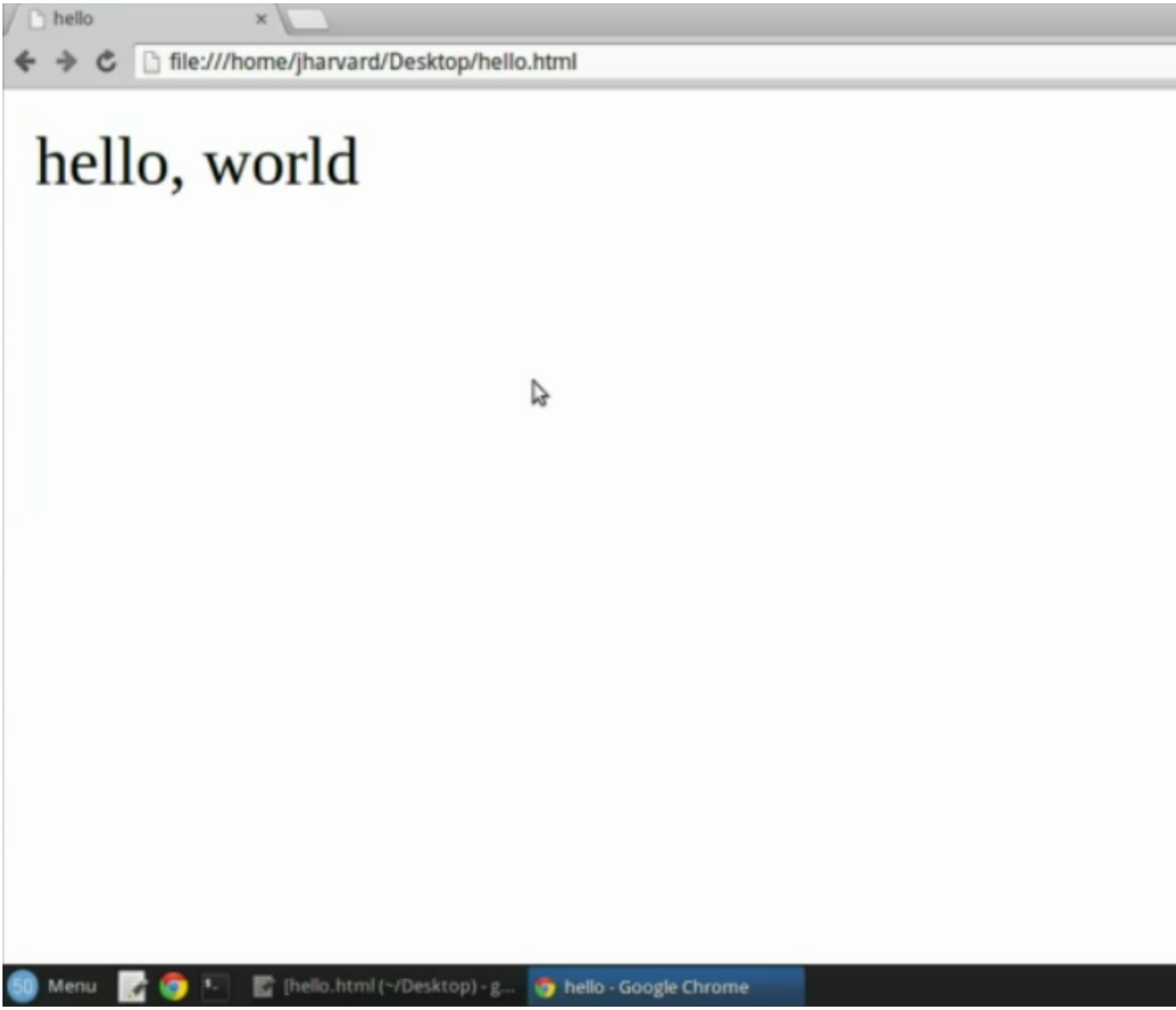
- Next we make a section called `<head>` that every HTML page has, and add a `<title>`:

```
<!DOCTYPE html>

<html>
  <head>
    <title>hello</title>
  </head>
</html>
```

- Every page also has a `<body>` section, which we can add here, like this:

```
<!DOCTYPE html>

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

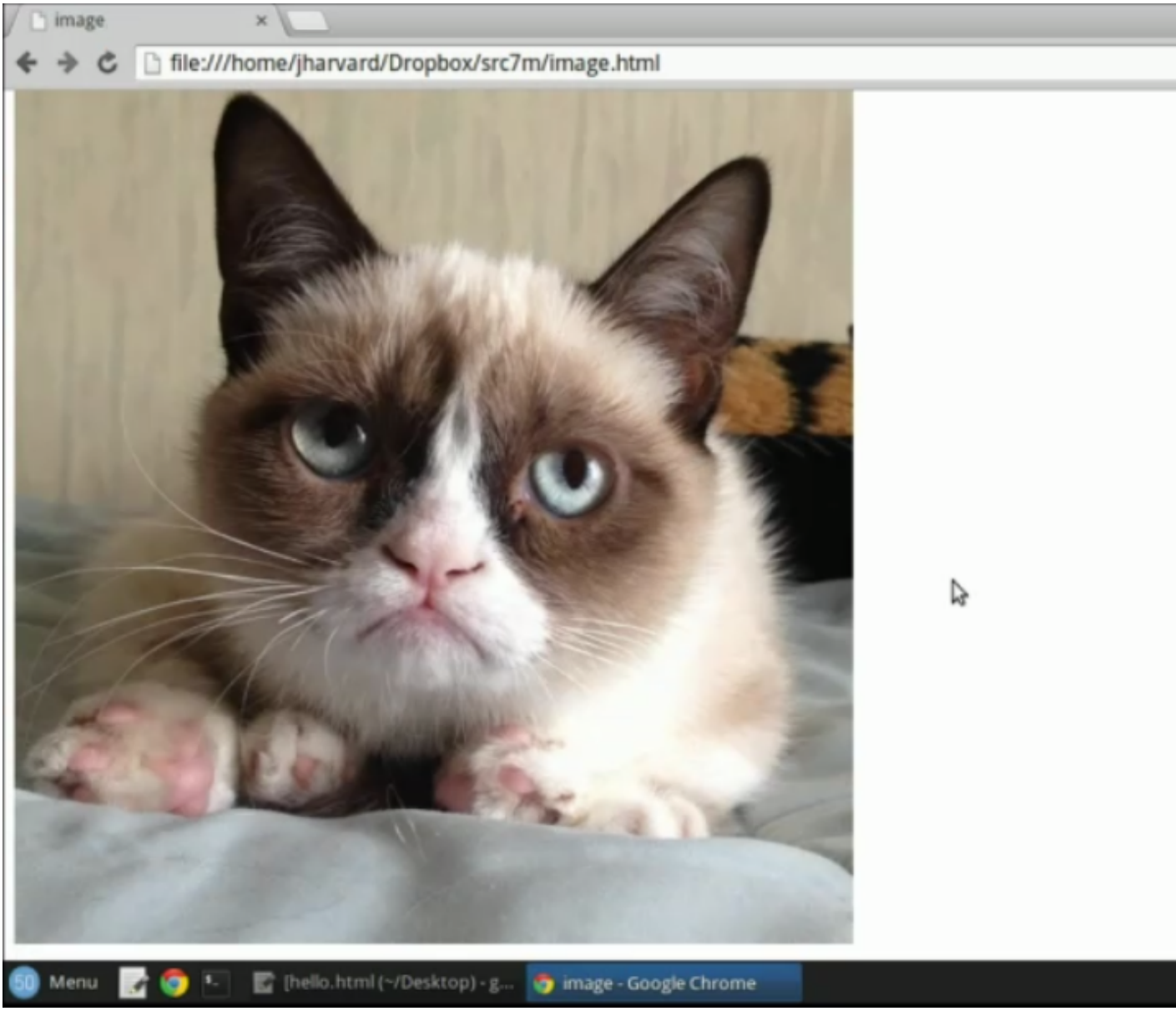
- We can save it, and even though the file will not be on a server but just our own computer, we can open a browser like Chrome and press `Ctrl-o`, and then open our webpage:




```
# Notice that the body is the large white space, and the title is at the top of the tab.
```

- We can go into the source code and open `image.html`[3]:

[3] http://cdn.cs50.net/2014/fall/lectures/7/m/src7m/image.html



- The source looks something like this:

```
<html>
    <head>
        <title>image</title>
    </head>
    <body>
        <!-- http://knowyourmeme.com/memes/grumpy-cat -->
        <img alt="Grumpy Cat" src="cat.jpg"/>
    </body>
</html>
```

    # We see on line 7 that we open a bracket with `<` and use the keyword `img` (read: image), followed by `alt`, alternative text for accessibility reasons, and then `src`, or the source file name, which is `cat.jpg`. And this tag is special in that it is closed not by `</img>` but just `/>` at its end.

- Now we're going to use webpages as containers for a graphical user interface like Breakout's, with languages like PHP and JavaScript and a database language like SQL to create dynamic user interfaces.
- Let's take a look at what happens under the hood of the Internet with Warriors of the Net[4], an animated short film that goes into detail about how information is packed into packets and sent across the Internet.

[4] http://youtu.be/PBWhzz_Gn10