

---

# Internet

This is CS50. Harvard University. Fall 2015.

Cheng Gong

## Table of Contents

Recap .....	1
Internet .....	2

## Recap

- Last time, we talked about encryption, which is just changing data so that it can only be read with a key, like a password that can be represented as some number of bits.
- [Diffie–Hellman key exchange](#)<sup>1</sup> is one of the simpler ways to achieve the sharing of a key, using certain mathematical properties of numbers.
- And for Dropbox to de-duplicate data, they must be encrypting everyone’s data with the same key.
- We asked a few questions about how many possibilities there might be for each of the following:
  - # 4-symbol passcode, numeric?
    - # Since we have 10 digits, 0-9, there are  $10^4$  combinations.
  - # 6-symbol passcode, alphanumeric?
    - # Now we have  $62^6$ , since we have 26+26+10 characters, for uppercase and lowercase characters and numbers, which is a much higher number.
  - # Encryption systems might use 256- or 1024-bit keys, meaning that there are  $2^{256}$  or  $2^{1024}$  possibilities, and the length of the keys have increased because we (or our adversary) have more and more computational power to try each possibility. But we can’t increase it too much, since there are performance costs to using longer keys, too.

---

<sup>1</sup> [https://en.wikipedia.org/wiki/Diffie%E2%80%93Hellman\\_key\\_exchange](https://en.wikipedia.org/wiki/Diffie%E2%80%93Hellman_key_exchange)

# Gur nafjre vf 42?

# To crack this question, we'd have tried to rotate the letters until we found a message that made sense, and by rotating it 13 times we get the message "The answer is 42?"

- Based on everyone's feedback, we'll be holding optional, supplementary seminars on the following topics:

# iOS App Development with Swift

# Programming with Python

# Web Development with HTML and CSS

# Android App Development

# Git and Version Control

# These tools help programmers maintain different versions of code and collaborate on them more easily.

# SQL 101

# SQL is a database query language for the back-end.

# JavaScript

# Introduction to APIs

# This might help you build more interesting applications using other services.

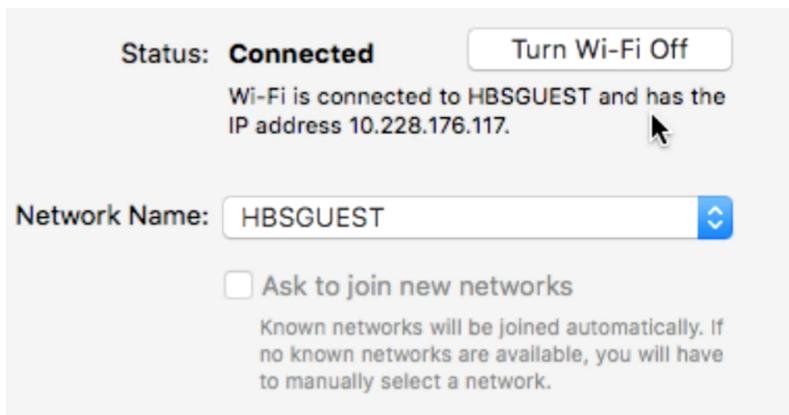
# Programming with Ruby

- We'll send out more information soon!

## Internet

- Today we'll talk about how the internet works.
- The internet is just many computers connected together.
- A LAN is a local area network, which is a group of computers close by connected by cables or wireless.
- A WAN is a wide area network, and there's no clear line but it's a bigger network than a LAN, perhaps all the computers on HBS' campus.

- If we open a browser and want to go to <http://www.stanford.edu>, for example, a lot happens.
- First, our device must support a protocol called DHCP, Dynamic Host Configuration Protocol, which allows it to get a unique address when it connects to a wired or wireless network. This address is like a postal address, and is called an IP address, which might look like `...`, where there are four numbers separated by periods. And if each number is in the range 0-255, then we might expect there to be a total of  $256^4$  unique addresses available. So there's about 4 billion addresses.
- If we open our network settings, we might see something that looks like this:



- And that value uniquely identifies our computer.
- There are servers somewhere that are DHCP servers, which means they give our device this address, and also DNS, Domain Name System, servers, which translate domain names (like web addresses) to IP addresses, so we can communicate with a server for <http://www.stanford.edu> without knowing its IP addresses directly.
- And the local DNS server doesn't know all the domain names in the world, so there's actually a hierarchy of servers that would be asked, in turn, if they do know. HBS has its own DNS servers, but so does the area's ISP, internet service provider, who might know more, and at the very top there are 13 root servers that know all the domain names in the world (or at the very least, what other servers might know). And to avoid making this entire request every time, HBS' DNS server would cache, or store, the answers for some period of time so we can get results faster, at the cost of potentially an outdated answer.
- So now our device has an IP address, and the IP address of <http://www.stanford.edu>. We can think of sending a request over the internet as sending

a message in an envelope, where the outside has the IP address of our destination written on it, and our own IP address, so it can get back to us. Then we need to put down what service we want, since there might be a website service, email service, or chat service, all using different protocols.

- Websites, for example, use HTTP, Hypertext Transport Protocol. And even though we don't type that out in front of addresses these days (our browser does it for us), it's sent along in the form of an additional number. (HTTP is equivalent to 80, HTTPS is 443, and there are standards for other services as well.) So we would write, if Stanford's IP address was `1.2.3.4`, `1.2.3.4:80` to get the web server. And our own address will also end with a `:` followed by some number, since we are likely to have many applications online at the same time, and the number will identify which application that the data we get back should go to.
- The message inside will look like this:

---

```
GET / HTTP/1.1
```

---

# There are different ways to interact with a server, and `GET` is just one such way.

# The `/` just means the default page of that website. We could write, for example, `/profile.php` to get a specific page.

# And finally, `HTTP/1.1` just specifies the version the protocol. A protocol is like meeting someone new in real life, where you extend a hand for a handshake and people know to respond with a handshake.

- But to get this envelope across the internet, we need the help of some devices in between.
- Routers send the envelope one step closer to its destination, perhaps based on the first or first few numbers of the address.
- And generally we should get there in no more than 30 hops. We can see this by opening a Terminal or command line window, and run some commands:

```
$ nslookup www.stanford.edu
Server:          140.247.233.195
Address:         140.247.233.195#53
```

```
Non-authoritative answer:
Name:   www.stanford.edu
Address: 52.35.204.173
Name:   www.stanford.edu
Address: 54.69.18.75
Name:   www.stanford.edu
Address: 52.26.65.143
```

# We can run `nslookup`, which stands for name server lookup, to see the IP addresses of a particular domain. `140.247.233.195` is the IP of the DNS server that provided us this answer, and below are multiple IP addresses for Stanford's, since they likely have multiple servers for their website. This helps them spread the load across them. In fact, if we run the command again, we see the order of the results change, so we visit a random server each time, rather than overloading one particular server. And we can see this works by putting one of these IP addresses into the address bar of our browser.

# As an additional aside, we're used to typing in `http://www.` before our addresses, but now we generally realize what a URL looks like, so we can just share or type in an address like `stanford.edu`, with our browser filling in the rest. `bit.ly` might not look like a standard URL, but its ending is a country code whereby each country has their own ending, and this particular website is actually a URL shortening service. And a country that happened to have `.tv` as its ending has been selling domains to video-related websites.

```
$ traceroute -q 1 www.stanford.edu
traceroute to www.stanford.edu (52.26.65.143), 30 hops max, 40 byte packets
 1 mr-sc-1-gw-vl-427-fas.net.harvard.edu (140.247.2.137) 1.380 ms
 2 core-sc-1-gw-vl429.fas.harvard.edu (140.247.2.237) 2.238 ms
 3 bdrwg2-te-2-3-core.net.harvard.edu (128.103.0.130) 2.498 ms
 4 aws-dx-sum1.net.harvard.edu (140.247.3.210) 18.043 ms
 5 *
 6 54.239.109.24 (54.239.109.24) 126.339 ms
 7 54.239.108.47 (54.239.108.47) 91.206 ms
 8 205.251.244.116 (205.251.244.116) 102.095 ms
 9 *
10 *
11 205.251.225.60 (205.251.225.60) 92.216 ms
12 205.251.232.151 (205.251.232.151) 92.710 ms
13 205.251.232.86 (205.251.232.86) 95.537 ms
14 205.251.232.169 (205.251.232.169) 96.706 ms
15 *
16 *
```

# Now that we have the IP address of our destination, we don't know how to send it there. So we run the command `traceroute`, and we see some output, where each row is a router that is forwarding our envelope along. The first few routers seem to all be on Harvard's network, and we see that some routers can respond with `*`, where they don't tell us their IP address. We also notice that the last number in each row, in milliseconds, seem to be increasing as we get further from campus. This is probably because there's a long physical cable between those servers.

```
$ traceroute -q 1 www.mit.edu
traceroute to www.mit.edu (23.78.251.122), 30 hops max, 40 byte packets
 1 mr-sc-1-gw-vl-427-fas.net.harvard.edu (140.247.2.137) 1.383 ms
 2 coregw1-vl-411-fas.net.harvard.edu (140.247.2.45) 1.732 ms
 3 bdrwg2-te-4-2-core.net.harvard.edu (128.103.0.2) 2.695 ms
 4 nox1sumgw1-vl-503-nox-harvard.nox.org (207.210.142.53) 3.117 ms
 5 et-5-0-0.120.rtr.eqny.net.internet2.edu (198.71.47.57) 7.358 ms
 6 *
 7 *
 8 ae5.cr2.lga5.us.zip.zayo.com (64.125.21.82) 8.996 ms
 9 ae3.er4.lga5.us.zip.zayo.com (64.125.31.246) 9.087 ms
10 a23-78-251-122.deploy.static.akamaitechnologies.com (23.78.251.122) 9.450 ms
```

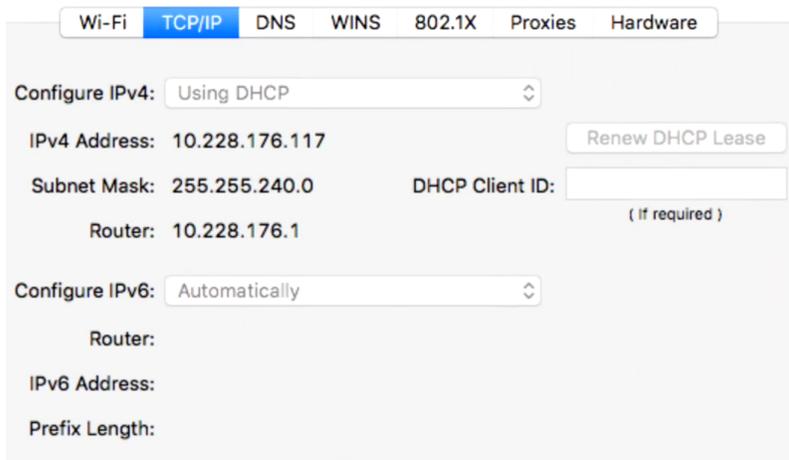
# We can try with a website closer, `www.mit.edu`, and we notice that the first hops are the same, but the end destination actually looks like a server owned by Akamai, a company that provides hosting services.

```
$ traceroute -q 1 www.cnn.co.jp
traceroute to www.cnn.co.jp (14.0.47.248), 30 hops max, 40 byte packets
 1 mr-sc-1-gw-vl-427-fas.net.harvard.edu (140.247.2.137) 1.617 ms
 2 coregw2-vl-3429-fas.net.harvard.edu (140.247.3.237) 1.557 ms
 3 bdrwg2-te-4-2-core.net.harvard.edu (128.103.0.2) 2.735 ms
 4 bst-edge-05.inet.qwest.net (63.145.1.133) 2.602 ms
 5 pax-brdr-01.inet.qwest.net (205.171.234.98) 67.879 ms
 6 124.215.192.77 (124.215.192.77) 74.135 ms
 7 111.87.3.117 (111.87.3.117) 74.087 ms
 8 otejbb205.int-gw.kddi.ne.jp (203.181.100.137) 172.479 ms
 9 sjkBBAC08.bb.kddi.ne.jp (27.93.254.86) 184.073 ms
10 obpBBAC03.bb.kddi.ne.jp (111.87.242.54) 191.380 ms
11 111.86.159.38 (111.86.159.38) 185.559 ms
12 14.0.40.190 (14.0.40.190) 182.789 ms
13 14.0.47.248 (14.0.47.248) 183.680 ms
```

# The first few hops are again the same, but by router `8` we see that there is a `.jp` ending in its name. And our data took much longer to get to router `8`, probably as a

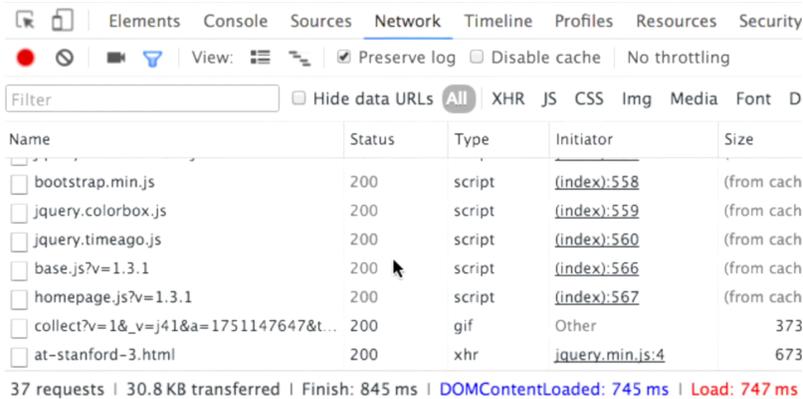
result of crossing the Pacific Ocean! And since these IP addresses are just routers who forward our envelope, we usually can't visit them directly in our web browser to find out more about them.

- So there's another protocol, TCP, that helps us ensure our messages are delivered. For example, if we have a picture of a cat that we want to send, we might split it into four pieces, place each one in an envelope, and send all 4 separately. TCP would be the equivalent of labeling each envelope 1/4, 2/4, etc, so our recipient will know if they got the entire picture.
- DHCP not only gives us an IP address, but also a DNS server and our default router. In Mac OS X, for example, we can find a window that tells us:



# And under the DNS tab, we can see the addresses of the DNS servers.

- With TCP, if an envelope disappears (perhaps a server along the way has been overwhelmed), our recipient can even ask us to re-transmit the lost portions.
- For some kinds of data, like live video, losing bits would be acceptable, so we wouldn't want to use TCP all the time.
- To have the internet work, we sort of build in layers. First, we need electricity, and then after that some physical cables. Next would be IP, then TCP, and on top of that, our application's data. But if we assume that everything below works, we can just worry about our application.
- Finally, in Chrome, we can right click a website, click **Inspect**, and then the **Network** tab and check **Preserve log**:



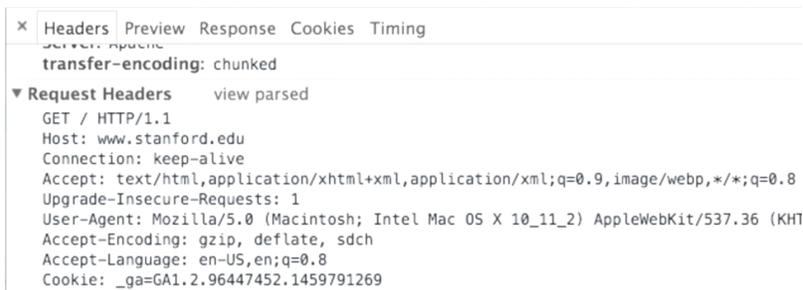
The screenshot shows the Network tab in Chrome DevTools. The 'All' filter is selected, and the table below lists 37 requests. The first six are JavaScript files, the next is a GIF, and the last is an XHR request for jQuery. Summary statistics at the bottom indicate 37 requests, 30.8 KB transferred, and a total load time of 747 ms.

Name	Status	Type	Initiator	Size
bootstrap.min.js	200	script	(index):558	(from cach
jquery.colorbox.js	200	script	(index):559	(from cach
jquery.timeago.js	200	script	(index):560	(from cach
base.js?v=1.3.1	200	script	(index):566	(from cach
homepage.js?v=1.3.1	200	script	(index):567	(from cach
collect?v=1&_v=j41&a=1751147647&t...	200	gif	Other	373
at-stanford-3.html	200	xhr	jquery.min.js:4	673

37 requests | 30.8 KB transferred | Finish: 845 ms | DOMContentLoaded: 745 ms | Load: 747 ms

# We see that just by visiting Stanford's home page, we're making 37 different requests. This is because the home page is made up of the page's text, but also images, sounds, and more. And each line is some asset that's included for this page.

- And if we look for our very first request to `www.stanford.edu` and click on it, we see:



The screenshot shows the Headers tab for the first request. The 'Request Headers' section is expanded, showing details such as the method (GET), host (www.stanford.edu), user agent (Mozilla/5.0), and various accept headers for HTML, XML, and images.

Header	Value
GET	HTTP/1.1
Host	www.stanford.edu
Connection	keep-alive
Accept	text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Upgrade-Insecure-Requests	1
User-Agent	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_2) AppleWebKit/537.36 (KHTML
Accept-Encoding	gzip, deflate, sdch
Accept-Language	en-US,en;q=0.8
Cookie	_ga=GA1.2.96447452.1459791269

# The first line looks familiar, and the rest includes more details about us and what we're looking for.

# Our IP address and computer information might even reveal us to advertisers.

- To actually purchase a domain name, we need to go to a registrar and pay them a small fee to use some domain name, or more precisely associate that domain name with some server of our choosing. And usually we rent out space on servers in the cloud from a hosting provider, who will actually run the DNS server and web server that holds our website.
- Now that we know how the internet is wired together, we'll look at how we might build a smaller network for our applications, next time!