

# **Lecture**

## **The HTTP Protocol & Networking Overview**

# What Does the WWW Server Do?

- Enables browser requests
- Mainly provides
  - Support for retrieving hypertext documents
  - Manages access to the Web site
  - Provides several mechanisms for executing server-side scripts
    - Common Gateway Interface (CGI)
    - Application Programmers Interface (API)
  - produces log files and usage statistics

# How Does a Web Server Communicate?

- Web browsers and servers communicate using the **HyperText Transfer Protocol** (HTTP)
- HTTP is a **lightweight** protocol
  - different from the ftp protocol
    - ftp sessions are long lived and there are two connections, one for control, one for data
- Current HTTP protocol is version 1.1
- W3C updates to HTTP (last update: June 2014):
  - <http://www.w3.org/Protocols/>
- HTTP 2.0 under the IETF httpbis Working Group
  - <http://datatracker.ietf.org/wg/httpbis/charter/>
- HTTP/2 Home page:
  - <https://http2.github.io/>

# HTTP History

- The Hypertext Transfer Protocol (HTTP) is an application-level protocol for distributed, collaborative, hypermedia information systems.
- The first version of HTTP, HTTP/0.9, was a simple protocol for raw data transfer across the Internet.
- **HTTP/1.0**, is defined by RFC 1945, see
  - <http://www.w3.org/Protocols/rfc1945/rfc1945>
- HTTP/1.0 allows messages to be in the format of **MIME**-like messages, containing meta-information about the data transferred and modifiers on the request/response semantics.
- **HTTP/1.1**, is defined by RFCs 7230-7237 (supersedes RFC 2616) , see
  - <http://tools.ietf.org/html/>
- HTTP/1.1 extends the protocol to handle:
  - the effects of hierarchical **proxies**
  - **caching**
  - the need for **persistent connections**
  - **virtual hosts**

## HTTP History (cont'd)

- **HTTP/2** is being worked on by **IETF Working Group**:
  - <http://tools.ietf.org/wg/httpbis/>
- HTTP/2 started as a copy of **Google SPDY** ("SPeeDY").
- HTTP/2 designed to speed up websites far larger than 10 years ago, using hundreds of requests/connections.
- One major feature of HTTP/2 is **header compression**:
  - <https://httpwg.org/specs/rfc7541.html>
- Google has dropped SPDY from Chrome and adopted HTTP/2:
  - <http://techcrunch.com/2015/02/09/google-starts-fading-out-spdy-support-in-favor-of-http2-standard/>
- See also:
  - <https://en.wikipedia.org/wiki/HTTP/2>
- Seen RFC 7540 (HTTP/2) & 7541 (HPACK):
  - <https://httpwg.org/specs/rfc7540.html>
- Dozens of implementations already available, including **Apache (2.4+)**, **Apache-Tomcat (8.5+)**, **Nginx (1.9.5+)**, etc.:
  - <https://github.com/http2/http2-spec/wiki/Implementations>
- **HTTP/3** is already being worked on!

## MIME MEDIA TYPES

- HTTP tags all data that it sends with its **MIME type**
- HTTP sends the MIME type of the file using the line  
**Content-Type: mime type header**

- For example, here are 2 MIME type messages

Content-type: image/jpeg

Content-length: 1598

- Some important MIME types are

- text/plain, text/html
- image/gif, image/jpeg
- audio/basic, audio/wav, audio/x-pn-realaudio
- model/vrml
- video/mpeg, video/quicktime, video/vnd.rn-realmedia, video/x-ms-wmv
- application/\*, application-specific data that does not fall under any other MIME category, e.g., application/vnd.ms-powerpoint

# Multipurpose Internet Mail Extensions

- MIME is an Internet standard for **electronic mail**
  - Traditional e-mail was limited to ASCII text, limited line length, and limited size
- MIME has extended Internet e-mail to include
  - Unlimited text line and message length
  - Messages with multiple body parts or objects enclosed
  - Messages that point to files on another server and are automatically retrievable
  - International character sets in addition to US-ASCII
  - Formatted text including multiple font styles
  - Images
  - Video clips
  - Audio messages
  - Application-specific binary data
- It was formalized in RFC 2046

## Facts About MIME

- MIME converts data that uses all **eight bits** into **7-bit ASCII**, sends it, and reconverts it at the other end. See:

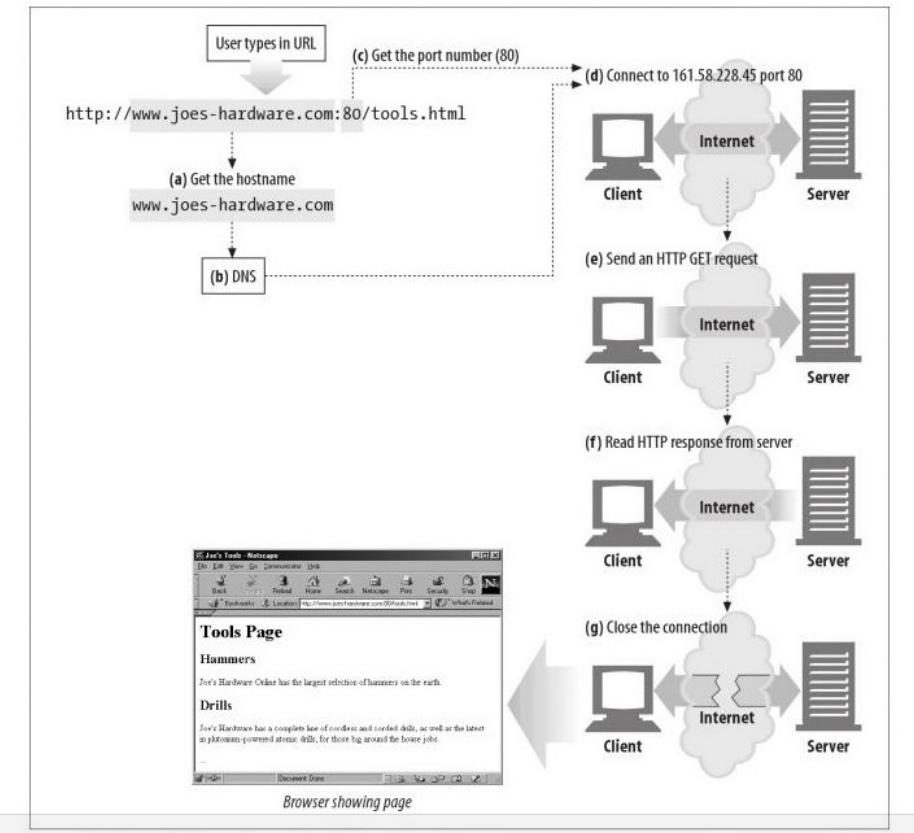
<https://tools.ietf.org/html/rfc1652>

- MIME headers at the front of the file define the type of data the message includes, e.g., here are a set of MIME types describing an attachment at an ftp site

```
Content-type: Message/External-Body  
name="classnotes.ps"  
site="ftp.usc.edu"  
access-type=anon-ftp  
directory="pub/cs665"  
mode="image"  
permission="read"  
expiration="Wed, 15 Mar 2009 07:00:00 -0400 (PST)"
```

# Description of a Browser Server Interaction

- (a) The browser extracts the server's hostname from the URL.
- (b) The browser converts the server's hostname into the server's IP address.
- (c) The browser extracts the port number (if any) from the URL.
- (d) The browser establishes a TCP connection with the web server.
- (e) The browser sends an HTTP request message to the server.
- (f) The server sends an HTTP response back to the browser.
- (g) The connection is closed, and the browser displays the document.



# An HTTP 1.0 “default” Scenario

- Communication takes place over a TCP/IP connection, generally on port 80

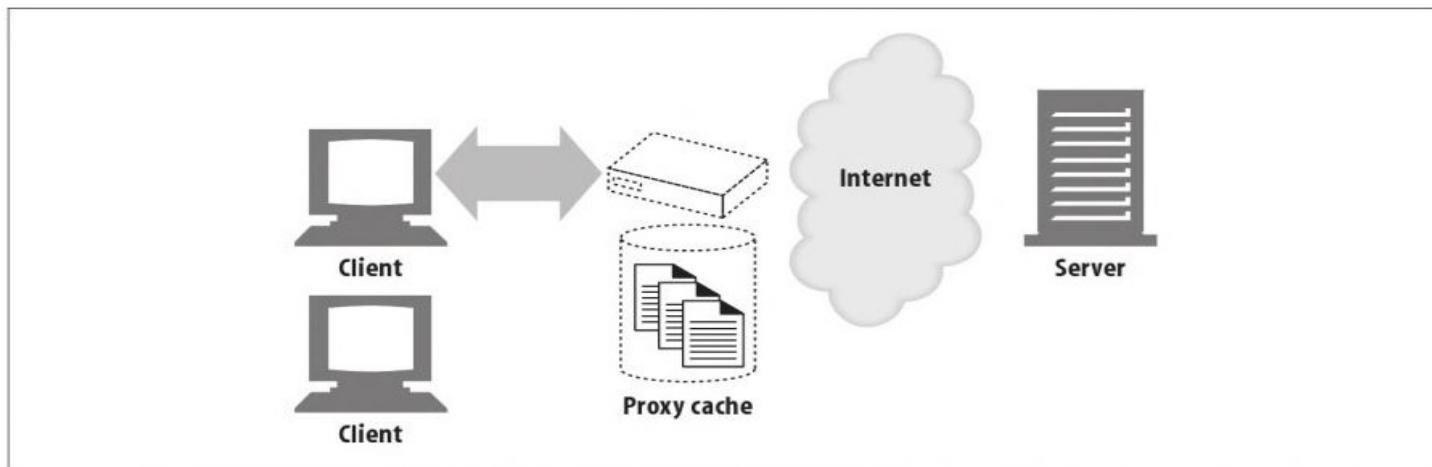
<b>Client action</b>	<b>Server response</b>
1. Client opens a connection	Server responds with an acknowledgment
2. Client sends HTTP request for HTML document	Server responds with the document and closes the connection
3. Client parses the HTML document and opens a new connection; it sends a request for an image	Server responds with the inlined image and closes the connection
4. Client opens a connection and sends another request for another image	Server sends the inlined image and closes the connection

## A More Complicated HTTP Scenario

- Actually, communication between a browser and a web server can be much more complicated; communication can go between one or more **intermediaries**.
- There are three common forms of intermediary: proxy, gateway, and tunnel.
  - A **proxy** is a forwarding agent, receiving requests for a URI in its absolute form, rewriting all or part of the message, and forwarding the reformatted request toward the server identified by the URI.
  - A **gateway** is a receiving agent, acting as a layer above some other server(s) and, if necessary, translating the requests to the underlying server's protocol.
  - A **tunnel** acts as a relay point between two connections without changing the messages; tunnels are used when the communication needs to pass through an intermediary (such as a firewall) even when the intermediary cannot understand the contents of the messages.

# Caching Proxies

A web cache or caching proxy is a special type of HTTP proxy server that keep copies of popular documents that pass through the proxy (**"forward" proxy**). The next client requesting the same document can be served from the cache's personal copy.

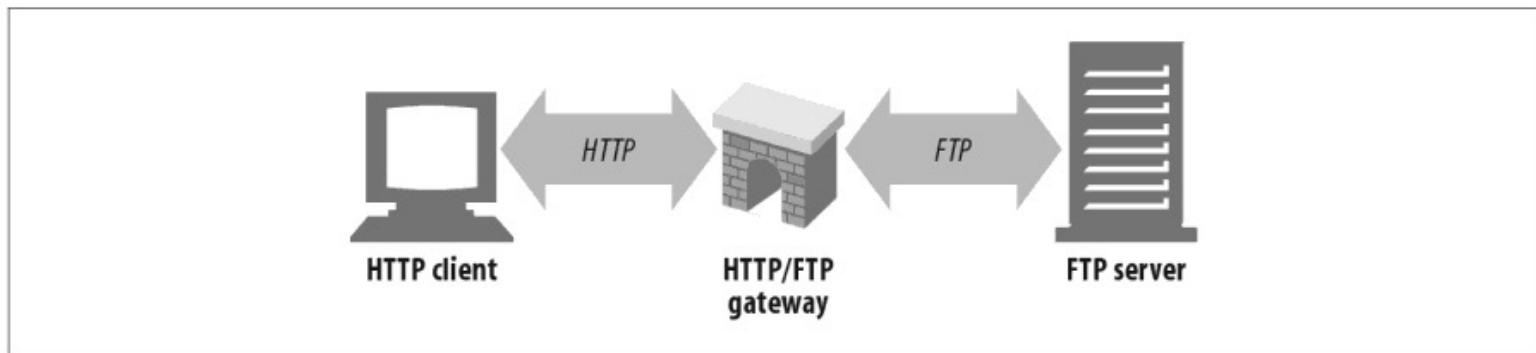


# Gateways

Gateways are special servers that act as intermediaries for other servers.

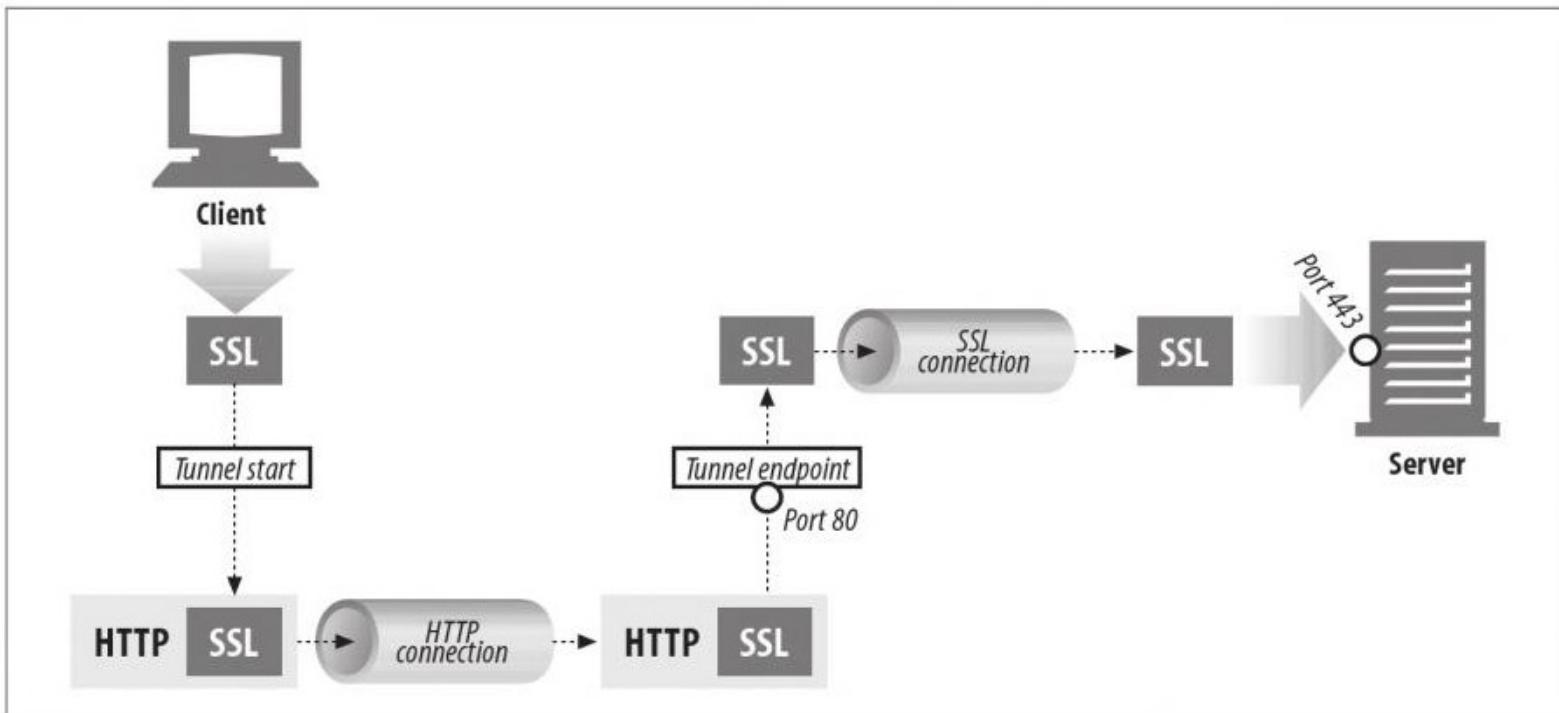
They are often used to **convert HTTP traffic to another protocol**. A gateway always receives requests as if it was the origin server for the resource. The client may not be aware it is communicating with a gateway.

For example, an HTTP/FTP gateway receives requests for FTP URIs via HTTP requests but fetches the documents using the FTP protocol. The resulting document is packed into an HTTP message and sent to the client.



# Tunnels

Tunnels are HTTP applications that, after setup, blindly relay raw data between two connections. HTTP tunnels are often used to transport non-HTTP data over one or more HTTP connections, without looking at the data. A **VPN** is an example of a tunnel.



# The Most General HTTP Scenario

Communication between browser and server should be regarded as a **request chain** goes left to right ---->

**UA** ---v--- **A** ---v--- **B** ---v--- **C** ---v--- **O**

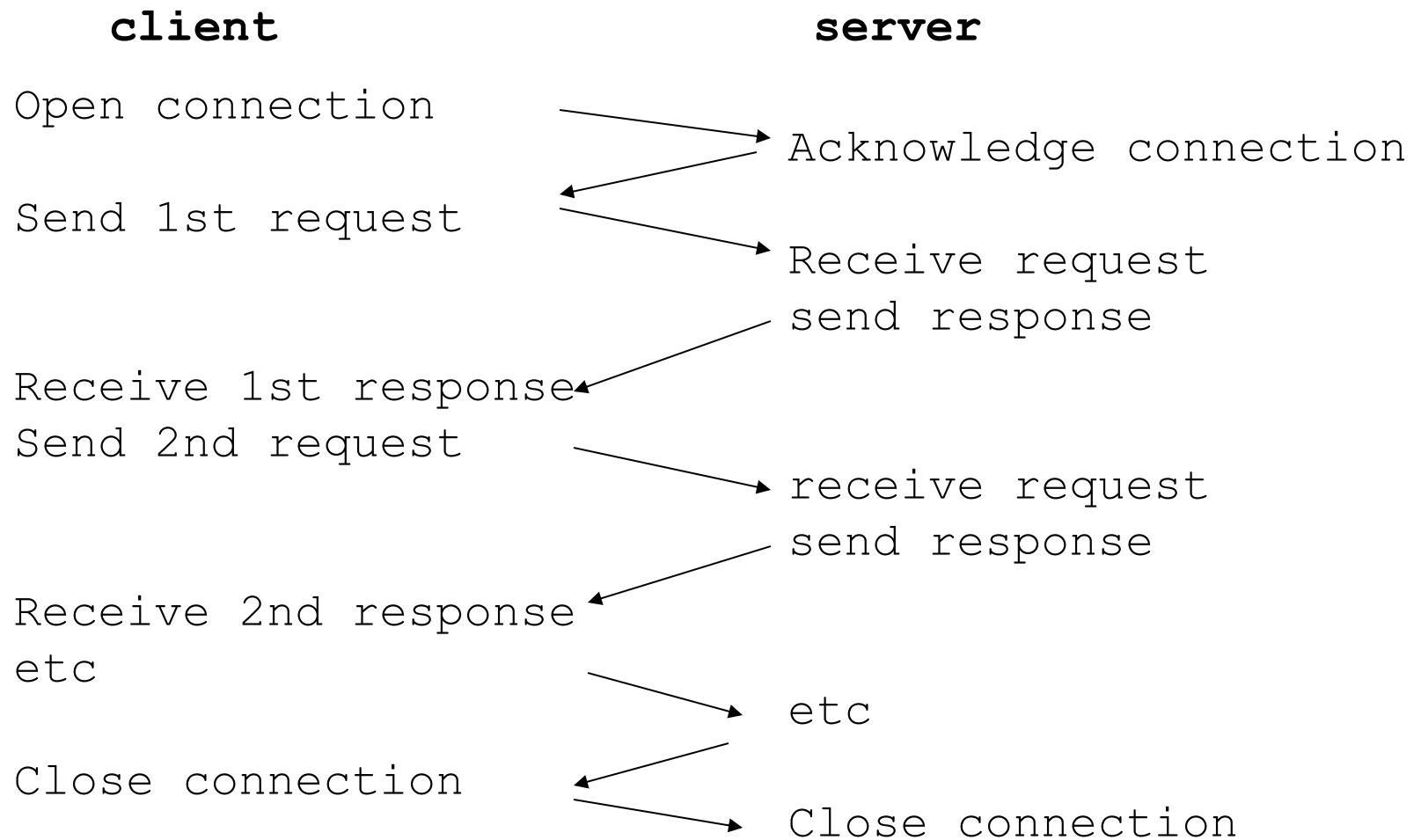
<----- and a response chain goes right to left

- **A**, **B**, and **C** are three intermediaries between the user agent and origin server. A request or response message that travels the whole chain will pass through four separate connections.
- **UA** stands for User Agent, typically a browser
- **O** stands for the origin server; the server that actually delivers the document

## Persistent Connections

- In the original HTTP protocol, each request was made over a new connection
  - so, an HTML page with  $n$  distinct graphic elements produced  $n+1$  requests
- TCP uses a **three-way handshake** when establishing a connection, so there is significant latency in establishing a connection
  - client sends SYN, server replies ACK/SYN, client responds with ACK
- HTTP 1.0 introduced a **keep-alive** feature
  - the connection between client and server is maintained for a period of time allowing for multiple requests and responses
  - a.k.a. **Persistent connection**

# HTTP/1.0 Keep Alive Connections



# HTTP/1.1 Keep Alive Extensions

- **Persistent connections** are now the default
- Request Header to set timeout (in sec.) and max. Number of requests, before closing:  
Keep-Alive: timeout=5, max=1000
- Client and server must explicitly say they do NOT want persistence using the header  
Connection: close
- HTTP permits multiple connections in parallel
  1. client requests a page and server responds
  2. client parses page and initiates 3 new connections, each requesting a different image
- Above scheme is NOT always faster, as multiple connections may compete for available bandwidth
- Generally, browsers severely limit multiple connections and servers do as well

## Example of a GET Request

- Clicking on a link in a web page or entering a URL in the address field of the browser causes the browser to issue a GET request, e.g.
- Suppose the user clicks on the link below:  
<A HREF="http://www.usc.edu/html/file.html">click here </A>
- The request from the client may contain the following lines

```
GET /html/file.html HTTP/1.1
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
User-Agent: Mozilla/5.0 (Windows NT 5.1; rv:15.0) Gecko/20100101
           Firefox/15.0.1
Referer: http://www.usc.edu/html/prevfile.html
If-Modified-Since: Wed, 11 Feb 2009 13:14:15 GMT
{there is a blank line here which terminates the input}
```

## **Response of the Server to GET**

- In response to the previous client request, the server responds with the following

### **HTTP/1.1 200 OK**

Date: Monday, 29-May-09 12:02:12 GMT

Server: Apache/2.0

MIME-version: 1.0

Content-Type: text/html

Last-modified: Sun, 28-May-09 15:36:13 GMT

Content-Length: 145

{a blank line goes here }

{the contents of file.html goes here }

# Client HTTP Requests

- The general form of an HTTP request has four fields:

HTTP\_method, identifier, HTTP\_version, Body

- **HTTP\_Method** says what is to be done to the object specified in the URL; some possibilities include GET, HEAD, and POST
- **identifier** is the URL of the resource or the body
- **HTTP\_version** is the current HTTP version, e.g. HTTP/1.1
- **Body** is optional text

# HTTP Request Methods

- Most common HTTP request methods are
  - **GET**, retrieve whatever information is identified by the request URL
  - **HEAD**, identical to GET, except the server does not return the body in the response
  - **POST**, instructs the server that the request includes a block of data in the message body, which is typically used as input to a server-side application
  - **PUT**, used to modify existing resources or create new ones, contained in the message body
  - **DELETE**, used to remove existing resources
  - **TRACE**, traces the requests in a chain of web proxy servers; used primarily for diagnostics
  - **OPTIONS**, allows requests for info about the server's capabilities

# HTTP Headers

- HTTP/1.1 divides headers into four categories:
  - **general**, present in requests or responses
  - **request**, present only in requests
  - **response**, present only in responses
  - **entity**, describe the content of a body
  - extension, new headers not already defined
- Each header consists of a name followed by a colon, followed by the value of the field, e.g.

Date: Tue, 3 Oct 2009 02:16:03 GMT

Content-length: 12345

Content-type: image/gif

Accept: image/gif, image/jpeg, text/html

## Examples of HTTP Headers - Request

- **Accept: text/html, image/\***  
indicates what media types are acceptable
- **Accept-Charset: iso-8859-5**  
indicates acceptable character sets. By default all are acceptable
- **Accept-Encoding: compress, gzip**  
indicates acceptable encodings
- **Accept-Language: en, fr=0.5**  
indicates language preferences, English preferred, but French also accepted
- **Authorization:**  
used to pass user's credentials to the server

## Examples of HTTP Headers - Request

- **From: name@site.com**  
requesting user's email address, rarely present
- **Host: www.usc.edu:8080**  
hostname and port of the requesting URL
- **Referer: http://www.usc.edu/index.html**  
the URL of the document that contains the reference to the requested URL
- **User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.8; rv:18.0) Gecko/20100101 Firefox/18.0**  
reports the client software name and version and possibly platform, see  
<http://www.javascriptkit.com/javatutors/navigator.shtml>

# Byte Range Headers

- Requests

- **If-Range: "entity-tag"**

used with byte range requests to guarantee that any new byte range responses are generated from the same source object. The entity-tag is quoted

- **Range: bytes=0-512, 2048-4096**

used to request a byte range

- Responses

- **Accept-ranges: bytes**

indicates the server can respond to range requests

- Entity

- **Content-Range: 0-399/2000**

response to byte range request giving the byte ranges actually returned, e.g. the first 400 bytes of a 2000 byte document

## Examples of HTTP Headers - Response

- **Age: 1246**  
age in seconds since response was generated
- **Location: http://www.myco.com/page.html**  
indicates that re-direction is desired
- **Public: GET, HEAD, POST, OPTIONS, PUT**  
methods supported by this web server
- **Server: Apache/1.3.1**  
identifies the server
- **WWW-AUTHENTICATE:**  
sent with 401 Unauthorized status code, it includes authorization parameters
- **Retry-after: 240**  
used with Service Unavailable status, indicates requested data will be available in 4 minutes

## Examples of HTTP Headers - Response

- A URL may point to a document with multiple representations: languages, formats (html, pdf), or html features based upon user-agent
- if a French version is requested and cached, then a new request may fail to retrieve the English version
- HTTP/1.1 introduces Vary: accept-language, user-agent the header specifies acceptable languages and browsers,
- **e.g., the request is**

GET http://www.myco.com/ HTTP/1.1

User-agent: Mozilla/4.5

Accept-language: en

- **the response is**

HTTP/1.1 200 OK

Vary: Accept-language

Content-type: text/html

Content-language: en

*The proxy must store the fact that this doc has variants and when requested, get the proper variant*

## **Examples of HTTP Headers - Response**

- Warning: 10 proxy-id “Revalidation failed” messages indicating status information of the resource; HTTP/1.1 defines the following status codes

### **Code Meaning**

10	Response is stale
11	Revalidation failed
12	Disconnected operation
13	Heuristic expiration
14	Transformation applied
99	Miscellaneous warning

# Entity Tags

- An **ETag** or **entity tag**.
  - one of several mechanisms that HTTP provides for web cache validation, and which allows a client to make conditional requests.
  - This allows caches to be more efficient, and saves bandwidth, as a web server does not need to send a full response if the content has not changed.
- An ETag is an opaque identifier assigned by a web server to a specific version of a resource found at a URL.
  - If the resource content at that URL ever changes, a new and different ETag is assigned.
  - ETags are similar to **fingerprints**, and they can be quickly compared to determine if two versions of a resource are the same or not.
- An ETag is a serial number or a checksum that uniquely identifies the file
  - caches use the **If-None-Match** condition header to get a new copy if the entity tag has changed
  - if the tags match, then a **304 Not Modified** is returned

## Examples of HTTP Headers - Entity

- **Allow:** `GET, HEAD, PUT`  
lists methods supported by the URL
- **Content-Base:** `http://www.usc.edu/somedir`  
all relative references are taken wrt the base
- **Content-Encoding:** `gzip`  
indicates the encoding of the entity body;  
content-type indicates the media after encoding
- **Content-Language:** `en`  
identifies the language of the entity
- **Content-Length:** `7890`  
specifies the length of the entity in bytes
- **Content-Location:** `http://www.usc.edu/myfile.htm`  
specifies the URL of the accessed resource

## Examples of HTTP Headers - Entity

- **Content-MD5: base-64 encoded MD5 signature**  
contains the MD5 signature of the body as created by the web server
- **Content-type:text/html**  
indicates the MIME type of the object
- **Etag: "7776cdb01f44354af8bfa40c56eebcb1378975"**  
specifies the entity tag for the object, which can be used for re-validation; tags are unique ids determined by the server; this line is normally sent as a response
- **Expires: Wed, 30 Dec 2002 03:43:21 GMT**  
specifies the expiration date/time of the object; a cached copy should not be used beyond; Expires 0/now is immediate
- **Last-Modified: Wed, 30 Dec 2002 01:20:34 GMT**  
specifies the creation or last modification time of the object on the web server

## **HTTP Status Codes - *Informational***

- After receiving and interpreting a request message, a server responds with an HTTP response message.
- Syntax of response is

Status-Line

\* ( general-header | response-header | entity-header ) CRLF [ message-body ]

where the Status line is composed of

Status-Line = HTTP-Version Status-Code Reason-Phrase CRLF

## **HTTP Status Codes - *Informational***

### **Code      meaning**

- |     |  |
|-----|--|
| 100 | Continue, the client may continue with its request; used for a PUT before a large document is sent |
| 101 | Switching Protocols, switching either the version or the actual protocol                           |

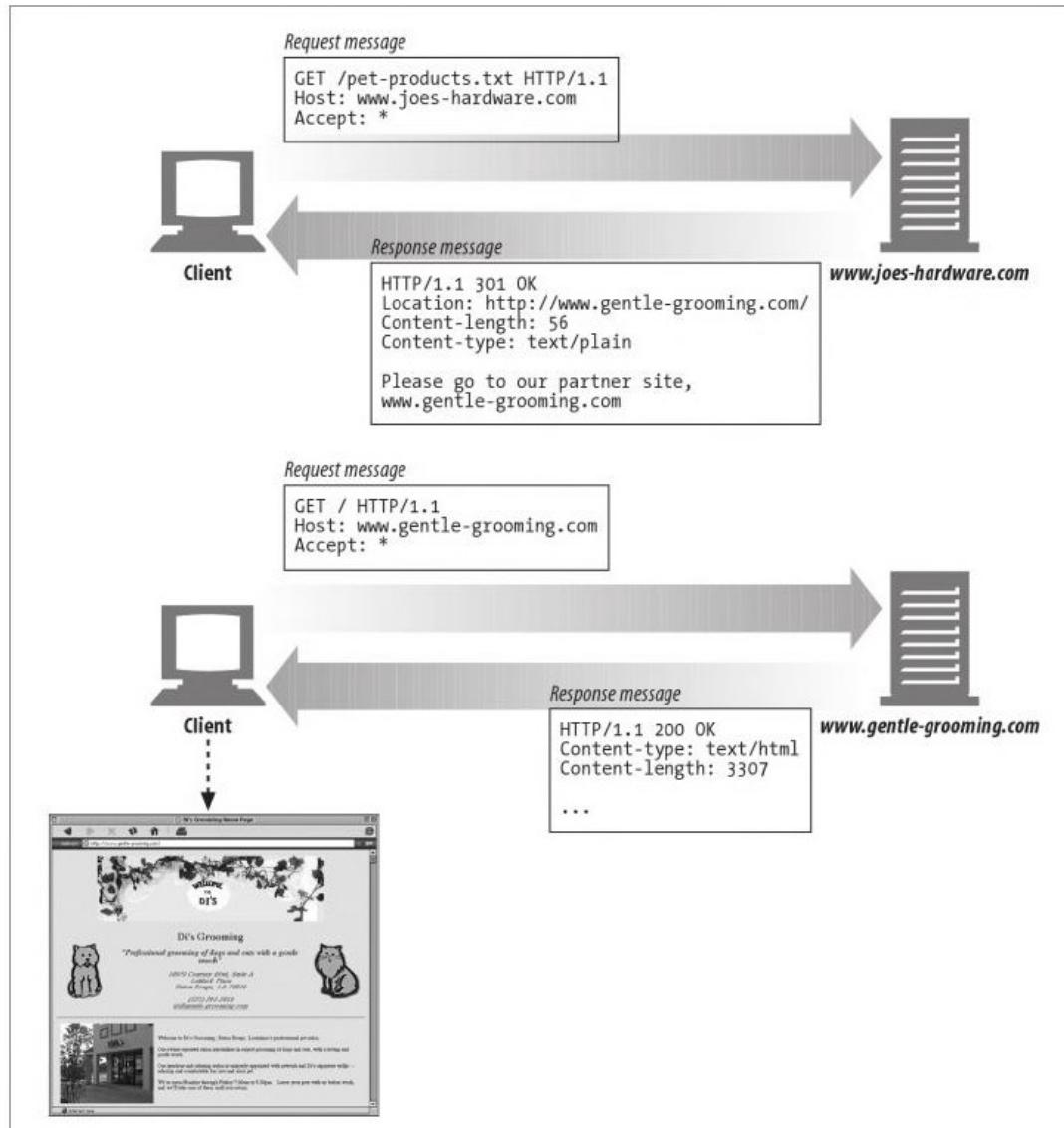
## HTTP Status Codes - *Successful*

<b>Code</b>	<b>meaning</b>
200	OK, request succeeded
201	Created, result is newly created
202	Accepted, the resource will be created later
203	Non-authoritative information, info returned is from a cached copy and may be wrong
204	No content, response is intentionally blank, so client should not change the page
205	Reset Content, notifies the client to reset the current document, e.g. clear a form field
206	Partial content, e.g. a byte range response

## HTTP Status Codes - *Redirection*

<b>Code</b>	<b>meaning</b>
300	Multiple choices, the document has multiple representations
301	Moved permanently, new location is specified in Location: header
302	Moved temporarily: similar to above
303	See Other: used to automatically redirect the client to a different URL
304	Not Modified: the client or proxy copy is still up-to-date
305	Use Proxy: make the request via the proxy
306	Proxy Redirection: a proposed extension to HTTP/1.1 still not specified
307	Temporary Redirect

# Redirection Example



## HTTP Status Codes - *Client Error*

<b>Code</b>	<b>meaning</b>
400	Bad request, server could not understand
401	unauthorized, authorization challenge
402	Payment Required, reserved for future use
403	forbidden, server refuses to fulfill request; e.g., check protections
404	Not found, document does not exist
405	Method not allowed, request method is not allowed for this URL
406	Not Acceptable, none of the available representations are acceptable to the client
407	Proxy Authentication Required, authentication is being challenged
408	Request Timeout, client did not send a request within a time given by server

## **HTTP Status Codes - *Client Error (cont'd)***

<b>Code</b>	<b>meaning</b>
409	conflict, the requested action cannot be performed
410	Gone, requested resource is no longer available
411	Length Required, Content-length header is missing
412	Precondition failed, a precondition for the request has failed, so request is canceled
413	request entity too large, entity is too large for server
414	Request URI too large, the request URL is too large
415	Unsupported media type, request entity is of an unsupported type

## HTTP Status Codes - *Server Error*

<b>Code</b>	<b>meaning</b>
500	Internal server error, generic error code for the server
501	Not implemented, request could not be serviced because server does not support it
502	Bad gateway, intermediate proxy server received a bad response
503	Service unavailable, due to high load or maintenance on the server
504	Gateway timeout, intermediate proxy server timed out waiting for response from another server
505	HTTP version not supported

# HTTP Authentication

- The web server can maintain secure directories and request authentication when someone tries to access them
- Procedure
  - web server receives a request without proper authorization
  - web server responds with 401 Authentication Required
  - client prompts for username and password and returns the information to the web server
- we will show how to cause an authentication request when we discuss the web server features



## **Example HTTP Basic Authentication**

- Client makes a request on a secure page**

GET /secure/mypage.html HTTP/1.0

- Server responds with**

HTTP/1.0 401 Unauthorized

Server: Apache 2.0

Date: Wed, 23 Dec 2002 15:42:00 GMT

WWW-Authenticate: Basic realm="notes server"

- Client prompts for username and password and uses base-64 printable encoding; it re-issues its request, e.g.**

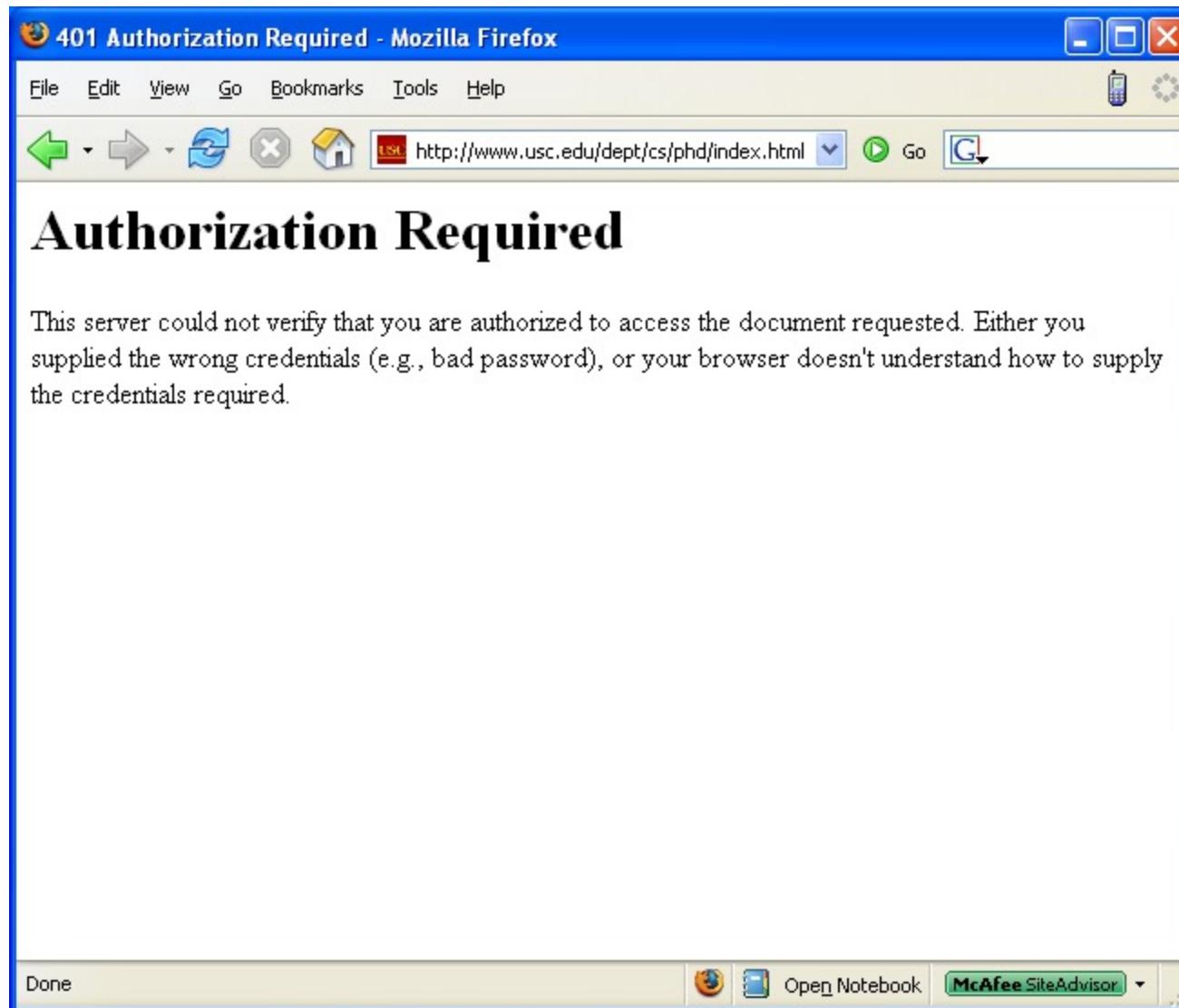
GET /secure/mypage.html HTTP/1.0

User-agent: Mozilla/4.5

Accept: text/html, image/gif, image/jpeg

Authorization: Basic MQDadmverWPUsvd=

# Authorization Denied



## META HTTP-EQUIV

- This is a mechanism for authors of HTML documents to set HTTP headers, in particular HTTP responses
- two common uses are
  - to set the **expiration time** of a document
  - to cause a **refresh** of a document
- E.g., here is an HTTP response header which causes the current page to be replaced by another page in 5 seconds

Refresh: 5; http://xyz.com/htdocs/mynewpage.html  
this is equivalent to the following HTML

```
<META HTTP-EQUIV="Refresh" CONTENT=5,  
http://csci571.com/index.html">  
</HEAD><BODY> This page will be replaced by the  
csci571 home page in 5 seconds.
```

## X-Frame-Options: SAMEORIGIN

Provides **Clickjacking protection**. Values: *deny* - no rendering within a frame, *sameorigin* - no rendering if origin mismatch, *allow-from: DOMAIN* - allow rendering if framed by frame loaded from *DOMAIN*

The X-Frame-Options HTTP response header can be used to indicate whether or not a browser should be allowed to render a page in a **<frame>** or **<iframe>**. Sites can use this to avoid clickjacking attacks, by **ensuring that their content is not embedded** into other sites; options include:

### DENY

The page cannot be displayed in a frame, regardless of the site attempting to do so.

### SAMEORIGIN

The page can only be displayed in a frame on the same origin as the page itself.

### ALLOW-FROM *uri*

The page can only be displayed in a frame on the specified origin.

### See:

<https://tools.ietf.org/html/rfc7034>

<https://developer.mozilla.org/en-US/docs/Web/HTTP/X-Frame-Options>

# HTTP Strict-Transport-Security (HSTS)

HSTS is a security feature that lets a web site tell browsers that it should only be **communicated with using HTTPS**, instead of using HTTP.

The HSTS feature lets a web site inform a browser that it should never load the site using HTTP, and browser should automatically convert all attempts to access the site using HTTP to HTTPS requests instead. Supported by all "modern" browsers. Ignored when site accessed using HTTP.

Enabling this feature for your site is as simple as **returning the Strict-Transport-Security HTTP header** when your site is accessed over HTTPS:

**Strict-Transport-Security: max-age=expireTime [; includeSubdomains]**

**expireTime**

The time, in seconds, that the browser should remember that this site is only to be accessed using HTTPS.

**includeSubdomains** (Optional)

If this optional parameter is specified, this rule applies to all of the site's subdomains as well.

See:

<https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Strict-Transport-Security>

[http://news.cnet.com/8301-1009\\_3-57524915-83/web-security-protocol-hsts-wins-proposed-standard-status/](http://news.cnet.com/8301-1009_3-57524915-83/web-security-protocol-hsts-wins-proposed-standard-status/)

# Cross-origin resource sharing (CORS)

Cross-Origin Resource Sharing (CORS) allows many resources (e.g., fonts, JavaScript, etc.) on a web page to be requested across domains.

With CORS, **AJAX calls can use XMLHttpRequest across domains**. Such “cross-domain” requests would otherwise be forbidden by web browsers.

The CORS standard adds new HTTP headers. To initiate a CORS request, a browser sends the request with an “Origin” HTTP header. Suppose a page from <http://www.social-network.com> attempts to access user data from [online-personal-calendar.com](http://online-personal-calendar.com). If the browser supports CORS, this header is sent:

**Origin: http://www.social-network.com**

If the server at [online-personal-calendar.com](http://online-personal-calendar.com) allows the request, it sends an Access-Control-Allow-Origin (ACAO) header in the response. The value of the header indicates what origin sites are allowed (**\* = all sites**). For example:

**Access-Control-Allow-Origin: http://www.social-network.com**

If the server does not allow the CORS request, the browser will deliver an error instead of the [online-personal-calendar.com](http://online-personal-calendar.com) response.

Firefox 3.5+, Safari 4+, Chrome3+, IE 10+, Opera 12+ support CORS. See:

[https://developer.mozilla.org/en-US/docs/Web/HTTP/Access\\_control\\_CORS](https://developer.mozilla.org/en-US/docs/Web/HTTP/Access_control_CORS)

[http://enable-cors.org/server\\_apache.html](http://enable-cors.org/server_apache.html)

# **Networking Overview**

- IP Addressing
- Routing
- DNS
- Ports
- NAT
- Subnets
- DHCP

# IP Addresses

- An IP address is a unique address that is required of all computers that communicate on a network
  - IPv4 was standardized in 1983
  - IPv6 was standardized as a draft in 1998 and became an official IETF standard in 2017
- IPv4 addresses (created in 1980) consist of 32 bits separated as 4 numbers of 8 bits each
  - `128.125.253.146`
- IPv6 (IETF Draft Standard in 1998) addresses consist of 128 bits separated as 8 sets of 4 hexadecimal values
  - `fe29:392A:3396:829E:5591:40d3:3495:109A`
  - NOTE that if all four hexadecimal values are 0, the term will be omitted
    - `fe29:392A:3396::5591:40d3` (4<sup>th</sup>, 7<sup>th</sup>, and 8<sup>th</sup> terms are 0000)

# **IPv4 - IPv6 Address Space**

- How many IPv4 addresses are there?
  - 32 bits =  $2^{32}$  addresses  
=  $2^2 * 2^{30}$  addresses  
= 4,294,967,296 addresses
- How many IPv6 addresses are there?
  - 128 bits =  $2^{128}$  addresses  
=  $2^8 * 2^{120}$  addresses  
=  $256 * 2^{120}$  addresses  
= 340,282,366,920,938,463,463,374,607,431,768,211,456 addresses

$$2^{10} = 1024 = \sim 10^3 = \text{thousand}$$

$$2^{20} = 1,048,576 = \sim 10^6 = \text{million}$$

$$2^{30} = \sim 10^9 = \text{billion}$$

$$2^{40} = \sim 10^{12} = \text{trillion}$$

$$2^{50} = \sim 10^{15} = \text{quadrillion}$$

$$2^{60} = \sim 10^{18} = \text{quintillion}$$

$$2^{70} = \sim 10^{21} = \text{sextillion}$$

$$2^{80} = \sim 10^{24} = \text{septillion}$$

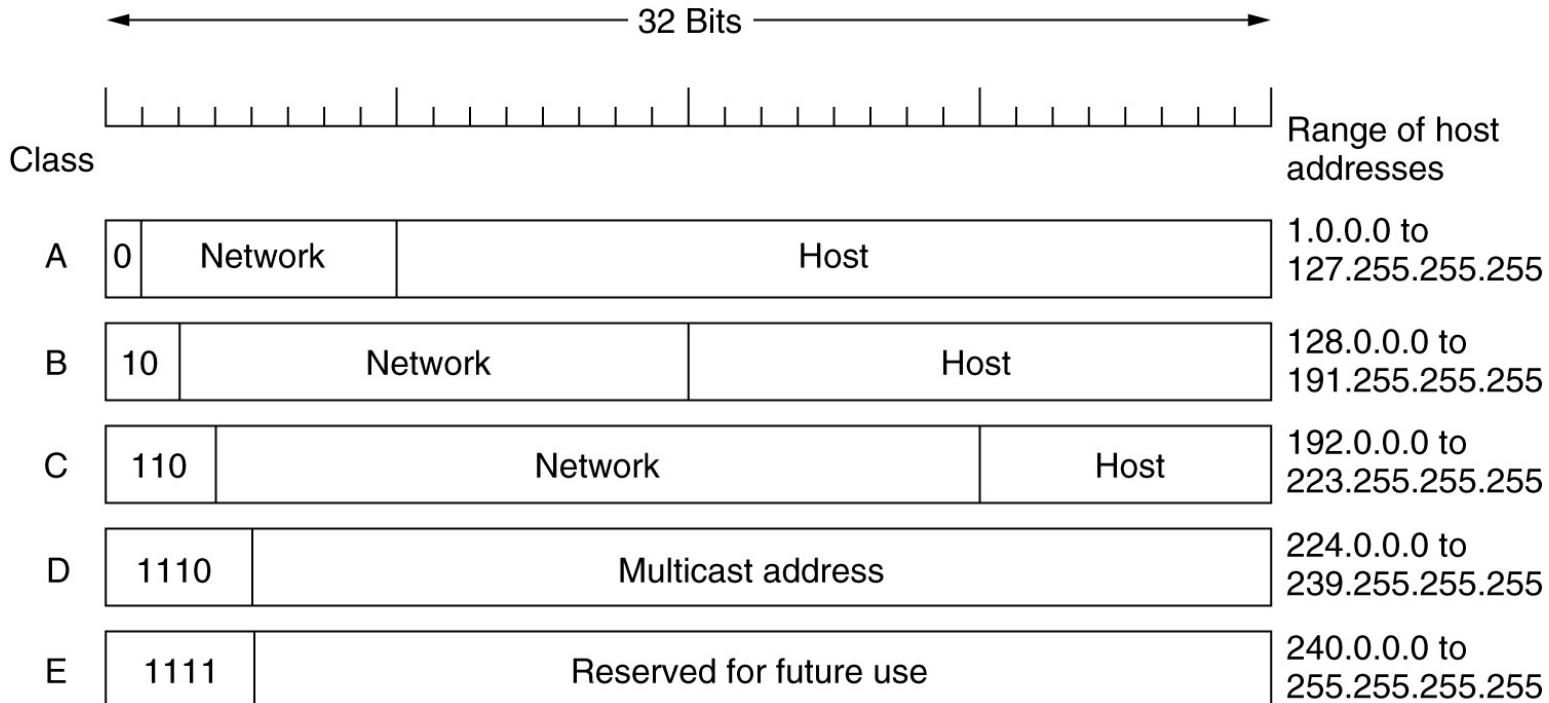
$$2^{90} = \sim 10^{27} = \text{octillion}$$

$$2^{100} = \sim 10^{30} = \text{nonillion}$$

$$2^{110} = \sim 10^{33} = \text{decillion}$$

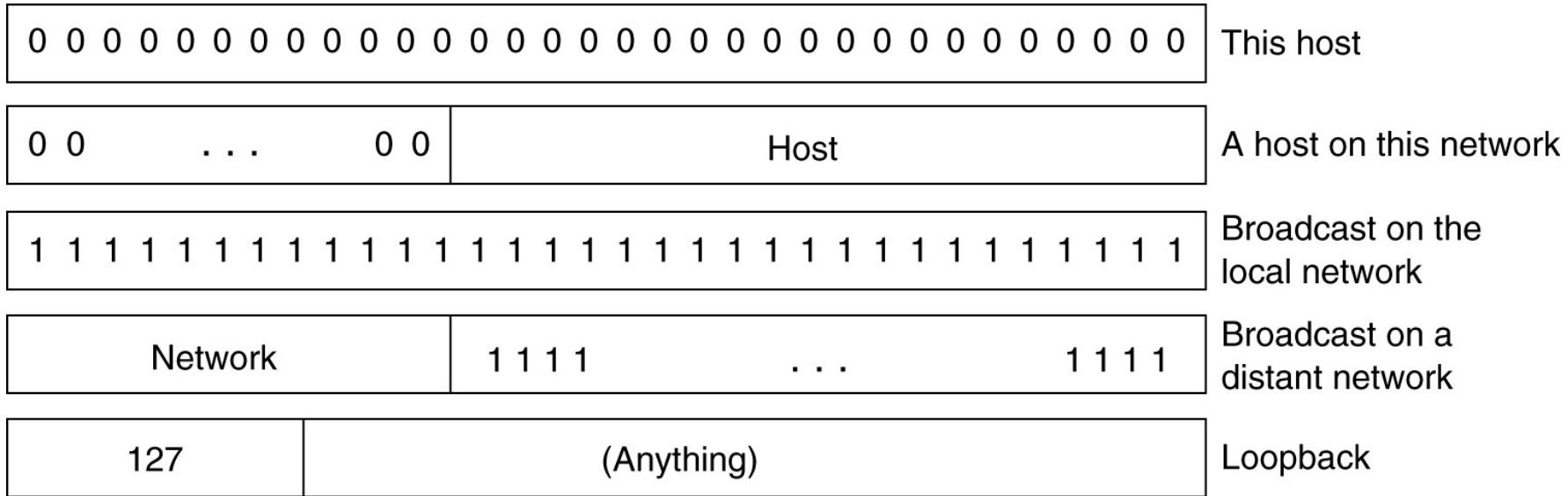
$$2^{120} = \sim 10^{36} = \text{undecillion}$$

# IPv4 Classes



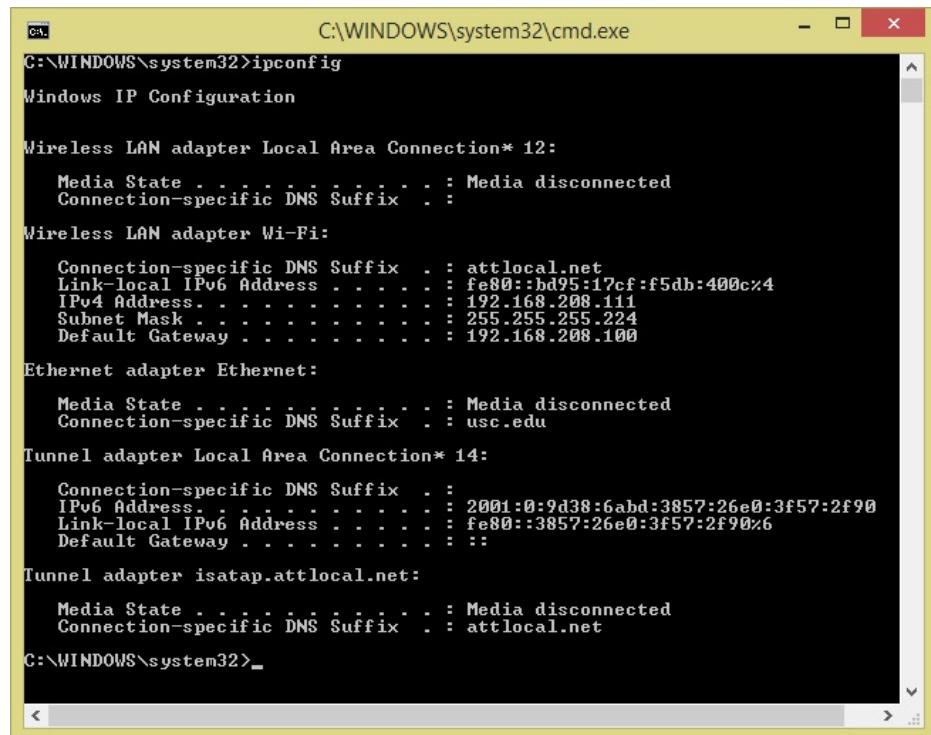
- Network numbers are managed by ICANN
- The Internet Assigned Numbers Authority (IANA) was founded by USC/ISI in 1976 by Jon Postel and transferred to ICANN in 1998
  - ICANN was founded primarily to take over control of IANA
- USC has Class B range

# IPv4 Special Addresses



# Finding Your IP Address

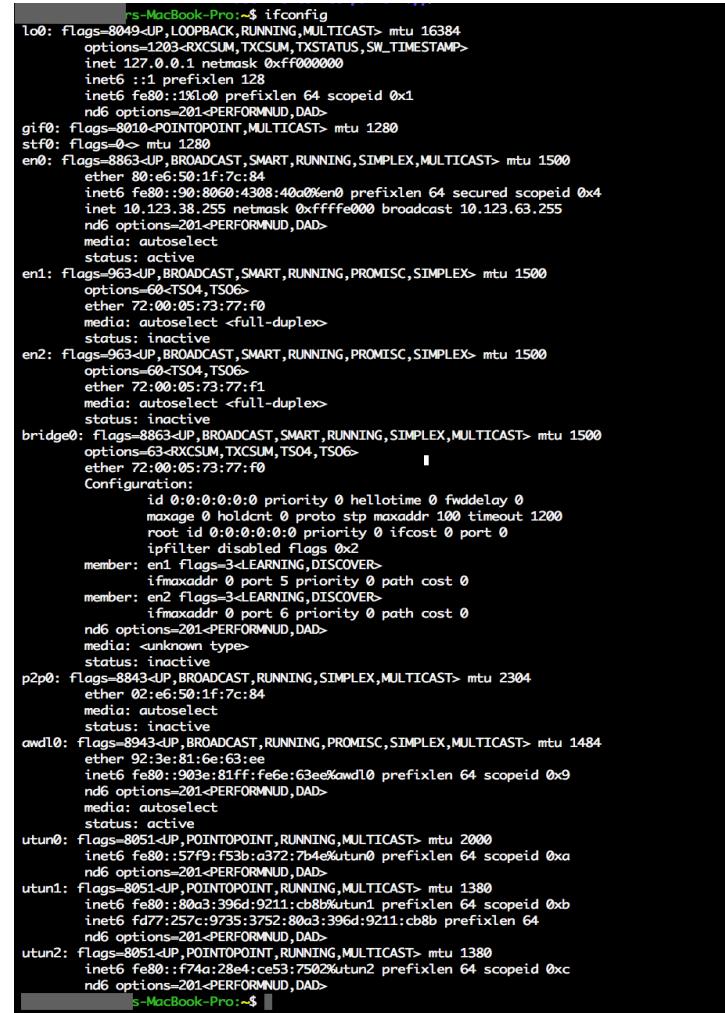
- Open a command line and type  
ipconfig in Windows  
ifconfig in macOS or Linux



A screenshot of a Windows Command Prompt window titled "C:\WINDOWS\system32\cmd.exe". The window displays the output of the "ipconfig" command. The output shows network configuration for several adapters:

- Wireless LAN adapter Local Area Connection\* 12:**
  - Media State : Media disconnected
  - Connection-specific DNS Suffix : attlocal.net
- Wireless LAN adapter Wi-Fi:**
  - Connection-specific DNS Suffix . : attlocal.net
  - Link-local IPv6 Address . . . . . : fe80::bd95:17cf:f5db:400c%4
  - IPv4 Address . . . . . : 192.168.208.111
  - Subnet Mask . . . . . : 255.255.255.224
  - Default Gateway . . . . . : 192.168.208.100
- Ethernet adapter Ethernet:**
  - Media State : Media disconnected
  - Connection-specific DNS Suffix : usc.edu
- Tunnel adapter Local Area Connection\* 14:**
  - Connection-specific DNS Suffix . . . . . : 2001:0:9d38:6abd:3857:26e0:3f57:2f90
  - Link-local IPv6 Address . . . . . : fe80::3857:26e0:3f57:2f90%6
  - Default Gateway . . . . . : ::
- Tunnel adapter isatap.attlocal.net:**
  - Media State : Media disconnected
  - Connection-specific DNS Suffix . . . : attlocal.net

C:\WINDOWS\system32>



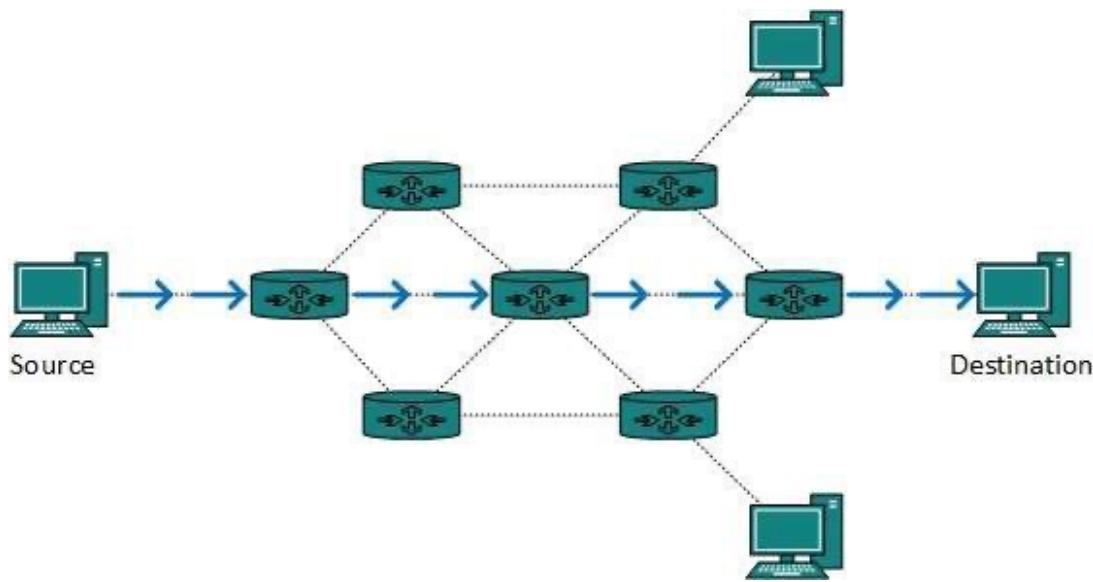
A screenshot of a macOS Terminal window titled "rs-MacBook-Pro:~\$". The window displays the output of the "ifconfig" command. The output shows network configuration for several interfaces:

- lo0:** flags=8049<UP,LOOPBACK,RUNNING,MULTICAST> mtu 16384 options=1203<RXCSUM,TXCSUM,TXSTATUS,SW\_TIMESTAMP>
  - inet 127.0.0.1 netmask 0xffffffff
  - inet6 ::1 prefixlen 128
  - inet6 fe80::1%lo0 prefixlen 64 scopeid 0x1
- inet6:** flags=0<NOARP> mtu 1280 options=201<PERFORMNUD,DAD>
- gif0:** flags=8010<POINTOPOINT,MULTICAST> mtu 1280
- stf0:** flags=0<NOARP> mtu 1280
- en0:** flags=8843<UP,BROADCAST,SMART,RUNNING,SIMPLEX,MULTICAST> mtu 1500
  - ether 00:0e:50:1f:7c:84
  - inet6 fe80::90:80e0%en0 prefixlen 64 secured scopeid 0x4
  - inet 10.123.38.255 netmask 0xffffffff broadcast 10.123.63.255
  - nd6 options=201<PERFORMNUD,DAD>
  - media: autoselect
  - status: active
- en1:** flags=963<UP,BROADCAST,SMART,RUNNING,PROMISC,SIMPLEX> mtu 1500
  - options=60<TS04,TS06>
  - ether 72:00:05:73:77:f0
  - media: autoselect <full-duplex>
  - status: inactive
- en2:** flags=963<UP,BROADCAST,SMART,RUNNING,PROMISC,SIMPLEX> mtu 1500
  - options=60<TS04,TS06>
  - ether 72:00:05:73:77:f1
  - media: autoselect <full-duplex>
  - status: inactive
- bridge0:** flags=8863<UP,BROADCAST,SMART,RUNNING,SIMPLEX,MULTICAST> mtu 1500
  - options=63<RXCSUM,TXCSUM,TS04,TS06>
  - ether 72:00:05:73:77:f0
  - Configuration:
    - id 0:0:0:0:0 priority 0 hellotime 0 fwddelay 0
    - maxage 0 holdcnt 0 proto stp maxaddr 100 timeout 1200
    - root id 0:0:0:0:0 priority 0 ifcost 0 port 0
    - ipfilter disabled Flags 0x2
  - member: en1 flags=3<LEARNING,DISCOVER>
  - ifmaxaddr 0 port 5 priority 0 path cost 0
  - member: en2 flags=3<LEARNING,DISCOVER>
  - ifmaxaddr 0 port 6 priority 0 path cost 0
  - nd6 options=201<PERFORMNUD,DAD>
  - media: <unknown type>
  - status: inactive
- p2p0:** flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> mtu 2304
  - ether 02:e6:50:1f:7c:84
  - media: autoselect
  - status: inactive
- awdl0:** flags=8943<UP,BROADCAST,RUNNING,PROMISC,SIMPLEX,MULTICAST> mtu 1484
  - ether 92:3e:81:6e:63:ee
  - inet6 fe80::903e:81ff%awdl0 prefixlen 64 scopeid 0x9
  - nd6 options=201<PERFORMNUD,DAD>
  - media: autoselect
  - status: active
- utun0:** flags=8051<UP,POINTOPOINT,RUNNING,MULTICAST> mtu 2000
  - inet6 fe80::57f9:f5b3:372:7b4%utun0 prefixlen 64 scopeid 0xa
  - nd6 options=201<PERFORMNUD,DAD>
- utun1:** flags=8051<UP,POINTOPOINT,RUNNING,MULTICAST> mtu 1380
  - inet6 fe80::80a3:396d:9211:cbb8%utun1 prefixlen 64 scopeid 0xb
  - inet6 fd77:257c:9735:3752:80a3:396d:9211:cb8b prefixlen 64
  - nd6 options=201<PERFORMNUD,DAD>
- utun2:** flags=8051<UP,POINTOPOINT,RUNNING,MULTICAST> mtu 1380
  - inet6 fe80::f74a:28e4:ce53:7502%utun2 prefixlen 64 scopeid 0xc
  - nd6 options=201<PERFORMNUD,DAD>

s-MacBook-Pro:~\$

# Routing

- Routing is the process of sending data from one computer to another
- Routers use two primary algorithms – **Distance Vector** and **Link State**
  - The specifics of those algorithms are outside the scope of this lecture

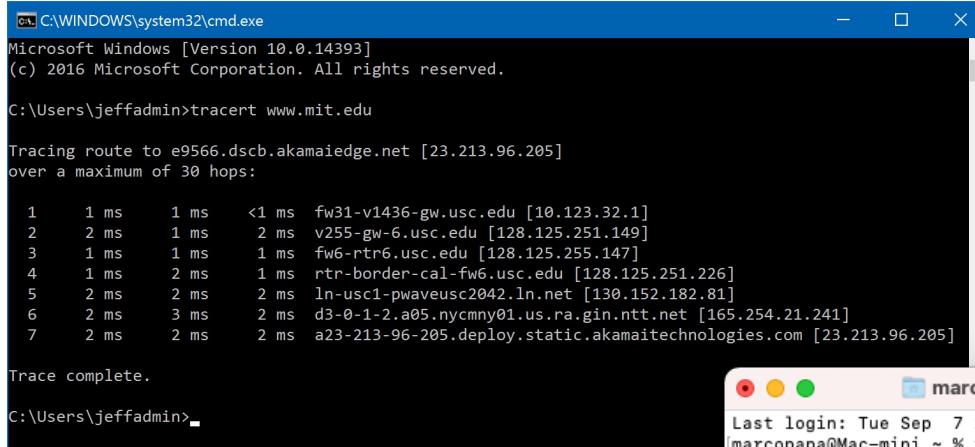


# Routing



# Finding a Route

- This is a very complicated process, but running **tracert** (Windows) or **traceroute** (macOS, Linux) from a command line or terminal will show you all the routers visited from your computer to a destination



```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. All rights reserved.

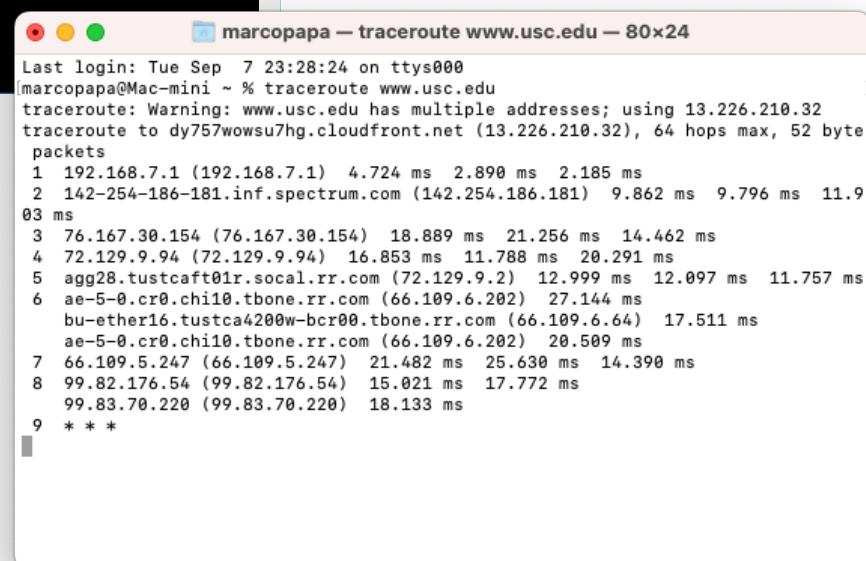
C:\Users\jeffadmin>tracert www.mit.edu

Tracing route to e9566.dscb.akamaiedge.net [23.213.96.205]
over a maximum of 30 hops:

 1  1 ms    1 ms    <1 ms  fw31-v1436-gw.usc.edu [10.123.32.1]
 2  2 ms    1 ms    2 ms  v255-gw-6.usc.edu [128.125.251.149]
 3  1 ms    1 ms    1 ms  fw6-rtr6.usc.edu [128.125.255.147]
 4  1 ms    2 ms    1 ms  rtr-border-cal-fw6.usc.edu [128.125.251.226]
 5  2 ms    2 ms    2 ms  ln-usc1-pwaveusc2042.ln.net [130.152.182.81]
 6  2 ms    3 ms    2 ms  d3-0-1-2.a05.nycmny01.us.ra.gin.ntt.net [165.254.21.241]
 7  2 ms    2 ms    2 ms  a23-213-96-205.deploy.static.akamaitechnologies.com [23.213.96.205]

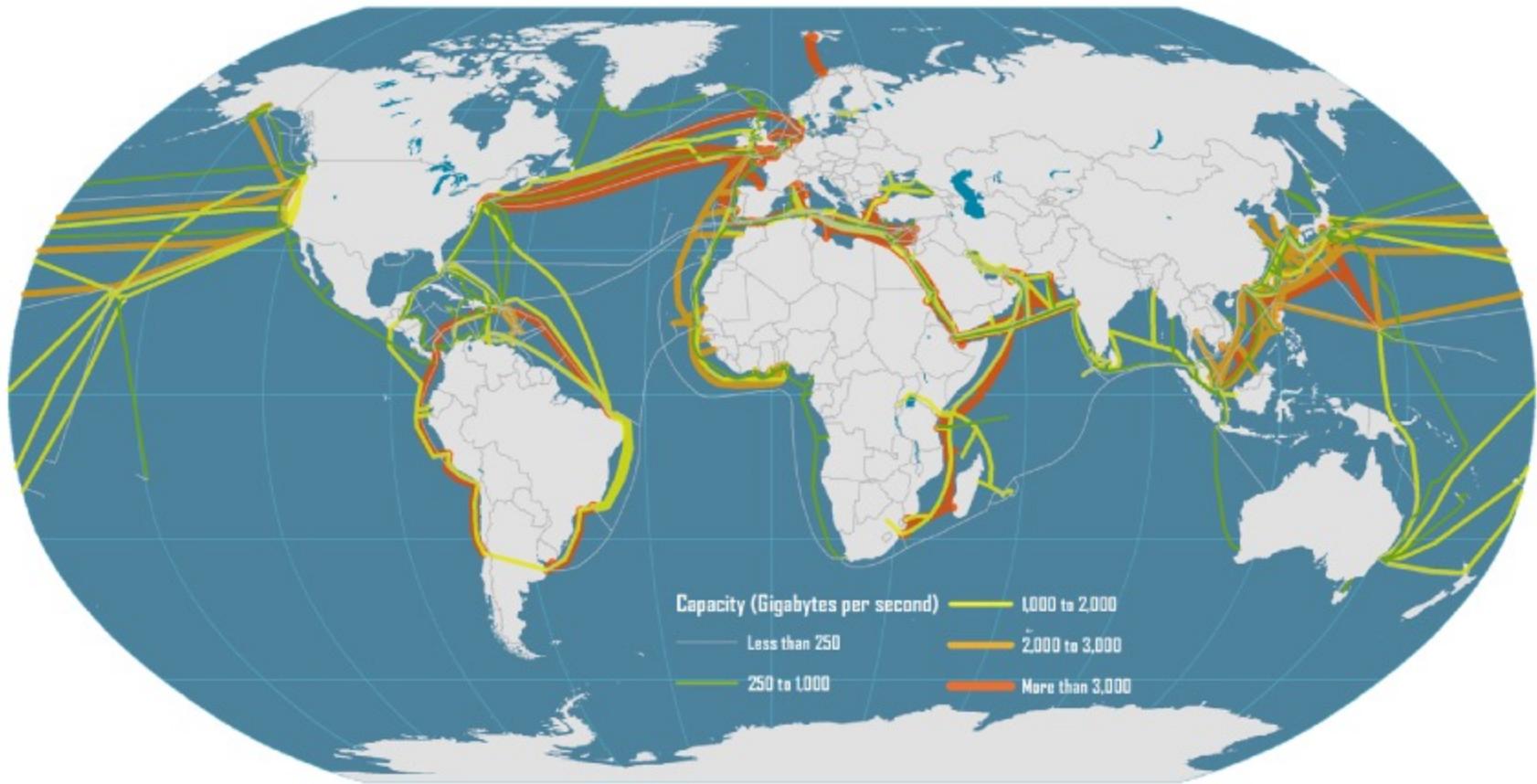
Trace complete.

C:\Users\jeffadmin>
```



```
marcopapa — traceroute www.usc.edu — 80x24
Last login: Tue Sep  7 23:28:24 on ttys000
[marcopapa@Mac-mini ~ % traceroute www.usc.edu
traceroute: Warning: www.usc.edu has multiple addresses; using 13.226.210.32
traceroute to dy757wou7hg.cloudfront.net (13.226.210.32), 64 hops max, 52 byte
packets
 1  192.168.7.1 (192.168.7.1)  4.724 ms  2.890 ms  2.185 ms
 2  142-254-186-181.inf.spectrum.com (142.254.186.181)  9.862 ms  9.796 ms  11.9
03 ms
 3  76.167.30.154 (76.167.30.154)  18.889 ms  21.256 ms  14.462 ms
 4  72.129.9.94 (72.129.9.94)  16.853 ms  11.788 ms  20.291 ms
 5  agg28.tustcaft01r.socal.rr.com (72.129.9.2)  12.999 ms  12.097 ms  11.757 ms
 6  ae-5-0.cr0.chi10.tbone.rr.com (66.109.6.202)  27.144 ms
     bu-ether16.tustca4200w-bcr00.tbone.rr.com (66.109.6.64)  17.511 ms
     ae-5-0.cr0.chi10.tbone.rr.com (66.109.6.202)  20.509 ms
 7  66.109.5.247 (66.109.5.247)  21.482 ms  25.630 ms  14.390 ms
 8  99.82.176.54 (99.82.176.54)  15.021 ms  17.772 ms
     99.83.70.220 (99.83.70.220)  18.133 ms
 9  * * *
```

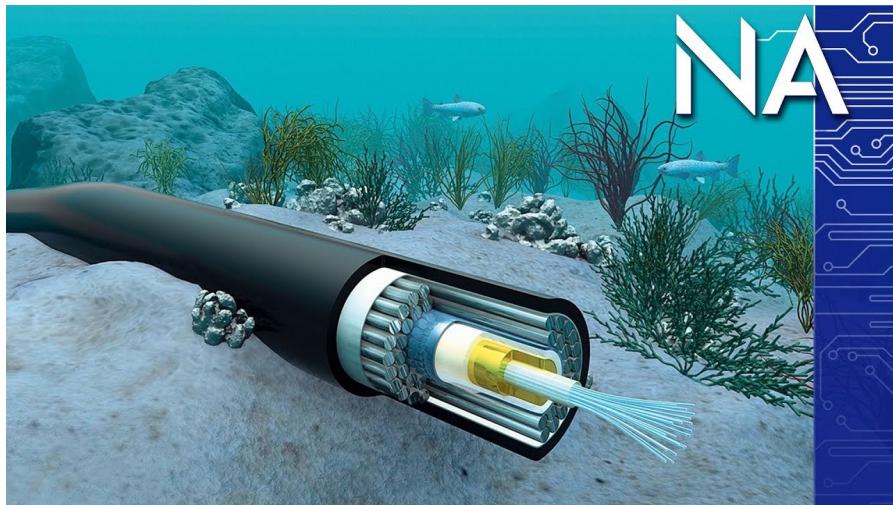
# Trans-Oceanic Pipelines



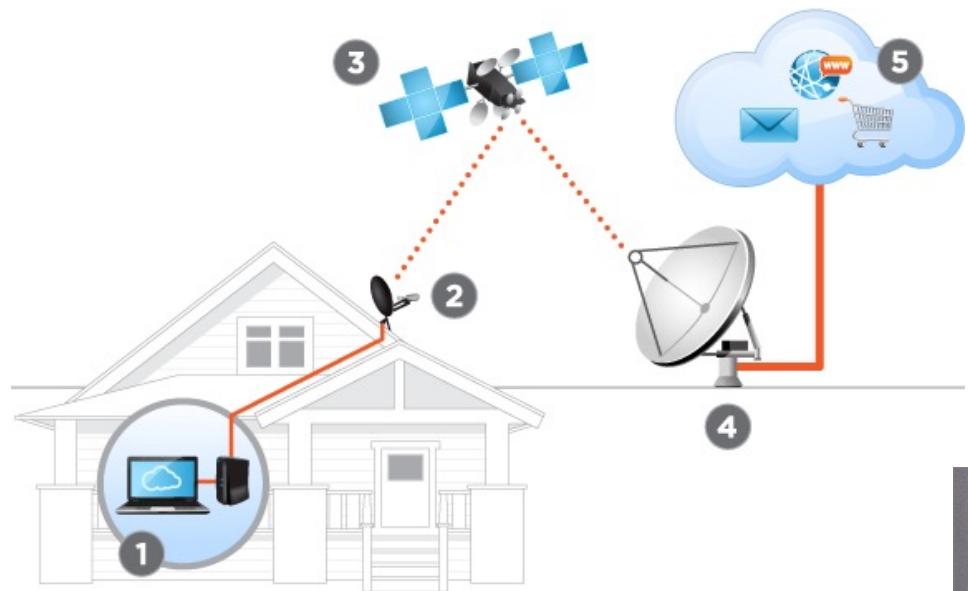
# Trans-Oceanic Pipelines



Microsoft and Facebook finished MAREA, a 4000-mile-long transatlantic communication cable, transmitting 160Tb/sec, connecting Virginia to Spain around 1700 feet below the surface, in February 2018



# Satellite Communication



# SpaceX Starlink

**STARLINK DEPLOYMENT STATUS**

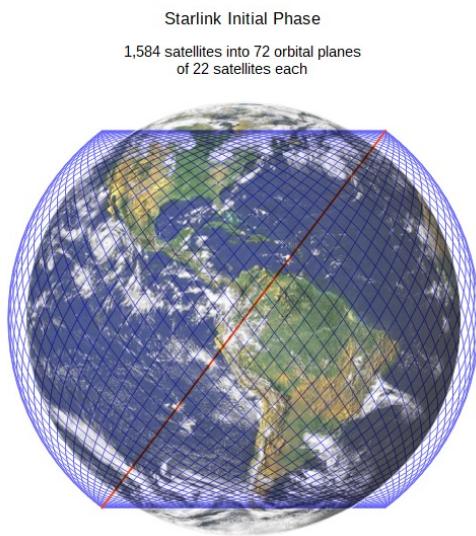
~1800 satellites launched

Starlink is rapidly rolling out internationally

- Currently serving ~140,000 users in 20+ countries (production slowed by ship shortage)
- Over 750,000 orders/deposits globally

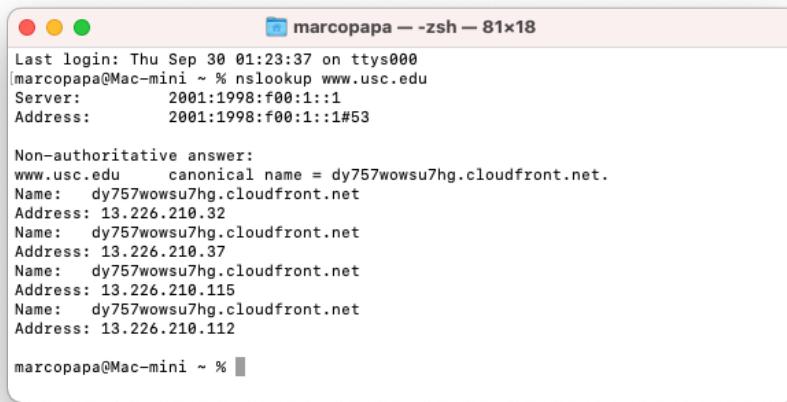


Satellite Internet Constellation  
Low Earth Orbit (LEO)  
1,892 satellites (as of 12/2/2021)  
145,000 internet users as of 1/2022 across 25 countries  
\$499 user terminal  
\$99 / month  
20ms latency  
50-150 Mbps  
<https://www.starlink.com/>



# Domain Name System (DNS)

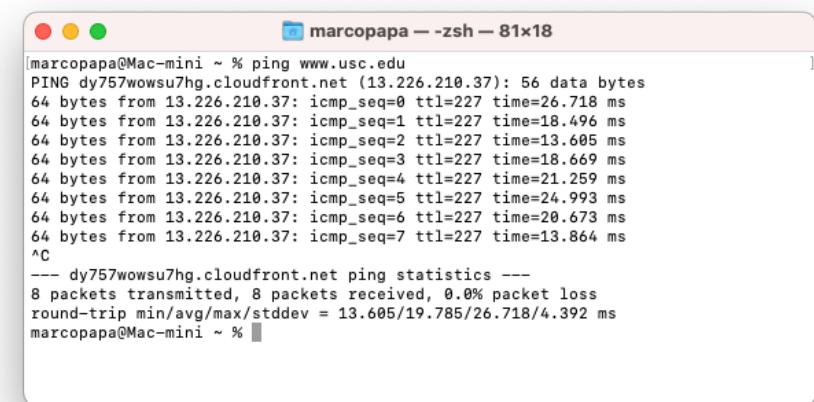
- Some servers can be identified by a hostname and domain name
  - An example would be `www.usc.edu`
  - `www` is the hostname (or an alias for a hostname)
  - `usc.edu` is the domain name
- To find the IP address of a hostname/domain name combination from a command line, run `ping` or `nslookup` from a command line or terminal



```
marcopapa@Mac-mini ~ % nslookup www.usc.edu
Server:  2001:1998:f00:1::1
Address: 2001:1998:f00:1::#53

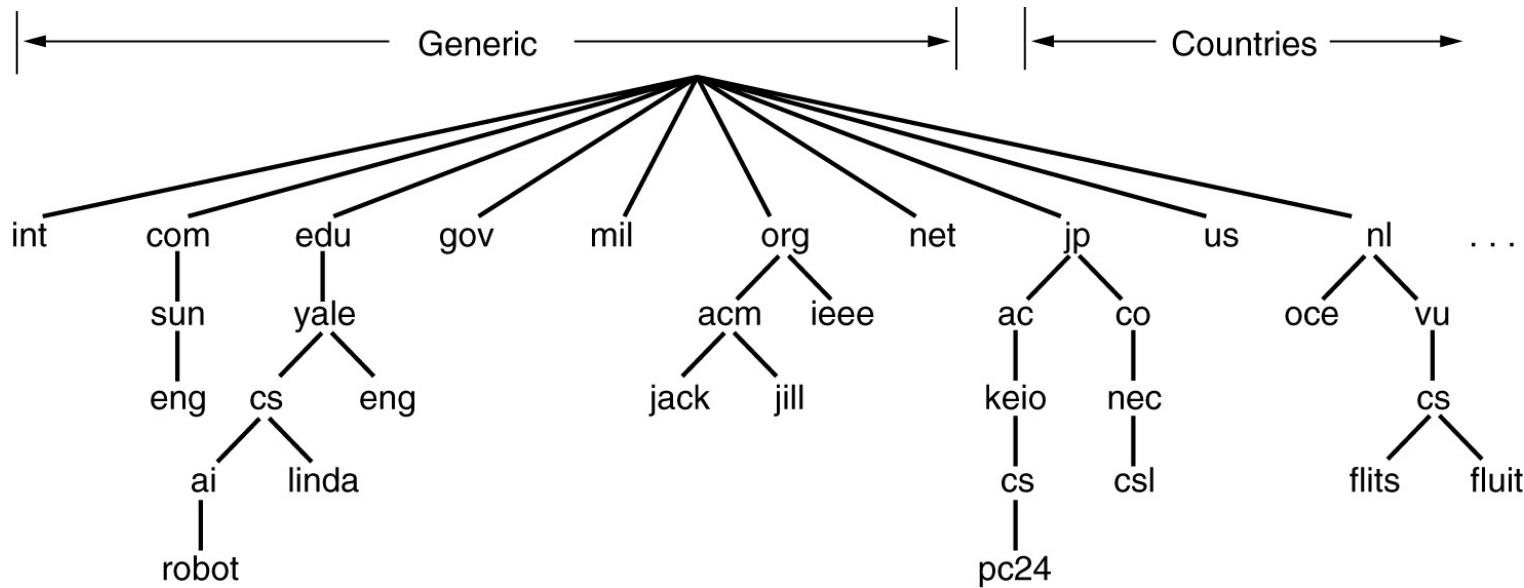
Non-authoritative answer:
www.usc.edu canonical name = dy757wowsu7hg.cloudfront.net.
Name:  dy757wowsu7hg.cloudfront.net
Address: 13.226.210.32
Name:  dy757wowsu7hg.cloudfront.net
Address: 13.226.210.37
Name:  dy757wowsu7hg.cloudfront.net
Address: 13.226.210.115
Name:  dy757wowsu7hg.cloudfront.net
Address: 13.226.210.112

marcopapa@Mac-mini ~ %
```



```
marcopapa@Mac-mini ~ % ping www.usc.edu
PING dy757wowsu7hg.cloudfront.net (13.226.210.37): 56 data bytes
64 bytes from 13.226.210.37: icmp_seq=0 ttl=227 time=26.718 ms
64 bytes from 13.226.210.37: icmp_seq=1 ttl=227 time=18.496 ms
64 bytes from 13.226.210.37: icmp_seq=2 ttl=227 time=13.605 ms
64 bytes from 13.226.210.37: icmp_seq=3 ttl=227 time=18.669 ms
64 bytes from 13.226.210.37: icmp_seq=4 ttl=227 time=21.259 ms
64 bytes from 13.226.210.37: icmp_seq=5 ttl=227 time=24.993 ms
64 bytes from 13.226.210.37: icmp_seq=6 ttl=227 time=20.673 ms
64 bytes from 13.226.210.37: icmp_seq=7 ttl=227 time=13.864 ms
^C
--- dy757wowsu7hg.cloudfront.net ping statistics ---
8 packets transmitted, 8 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 13.605/19.785/26.718/4.392 ms
marcopapa@Mac-mini ~ %
```

# DNS Namespace



- The hostname/domain name combination will be mapped to an IP address through DNS servers
  - DNS is a hierarchical domain-based naming scheme implemented through a distributed database system for implementing
  - **DNS was conceived at USC's ISI by Paul Mockapetris in 1983 and managed there until 1998**
  - DNS is now managed by ICANN

# Actual DNS Records

Screenshot of the GoDaddy DNS Manager - Zone File Editor interface for the domain SIGMACODING.COM.

The interface shows the following sections:

- Header:** Domain Details, DNS Manager - Zone File E..., GoDaddy.com, LLC (US), https://dns.godaddy.com/ZoneFile.aspx?zone=SIGMACODING.COM&zoneType=0&refer=dcc&prog\_id, Search, Welcome: jeffream, Log Out, My Account, GoDaddy.com.
- Breadcrumbs:** GoDaddy.com > DNSMANAGER > Zone File Editor > SIGMACODING.COM [change zone](#).
- Toolbar:** Home, Templates, Reports, Admin, Help.
- Dashboard:** Zone File Editor, Advanced Settings.
- Advertisement:** 25% off your order of \$75 or more\*, Save BIG on websites, domain names and more. [SHOP NOW!](#)
- Zone File Editor:** sigmacoding.com (Last saved 6/29/2014 5:55:36 PM MST). Shows 107 records remaining. [Upgrade now.](#)
- Record Types:**
  - A (Host):** Host Points to 104.49.170.234 TTL 1 Hour.
  - CNAME (Alias):** Host Points to autodiscover autodiscover.outlook.com TTL 1 Hour; Host Points to ftp @ TTL 1 Hour; Host Points to www @ TTL 1 Hour.
  - MX (Mail Exchanger):** Priority Host Points to 0 @ sigmacoding-com.mail.eo.outlook.com TTL 1 Hour.
- Filter:** Select: All, None, With Records. Options: A Record (Host) (checked), CNAME (Alias) (checked), MX (Mail Exchanger) (checked), TXT (Text) (checked), SRV (Service) (checked), AAAA (IPv6 Host) (checked), NS (Nameserver) (checked).
- Answer Center:** Why should I upgrade to Premium DNS?, Upgrading to Premium DNS, What are zone files and zone records?, Checking Your Domain Name's Zone File Records.
- Copyright © 1999-2015 GoDaddy.com, LLC. All rights reserved.

# Ports

- Since more than one networked program can run on a computer at the same time, we need a way to uniquely identify them
  - Ports allow us to do just that
- A client application will specify the port on the server with which to communicate, and that uniquely identifies the server application
- A port is in the range from 0 to 65535 (16 bits)
  - **0-1023** (inclusively) are **reserved** for well-known applications, so root or administrator access is required to run a program on a port in that range
  - **1024-49151** (inclusively) are registered ports and can be used by any application
  - **49152-65535** are dynamic or private ports and are typically used by the operating system when an application needs to pass an application off to a non-registered port

# Well Known Ports

Port	Application
20	FTP data
21	FTP control
22	SSH
23	Telnet
25	SMTP
53	DNS
80	HTTP
143	IMAP
443	HTTPS

- There are many others, but these are some of the more popular ones

# Public and Private IP addresses

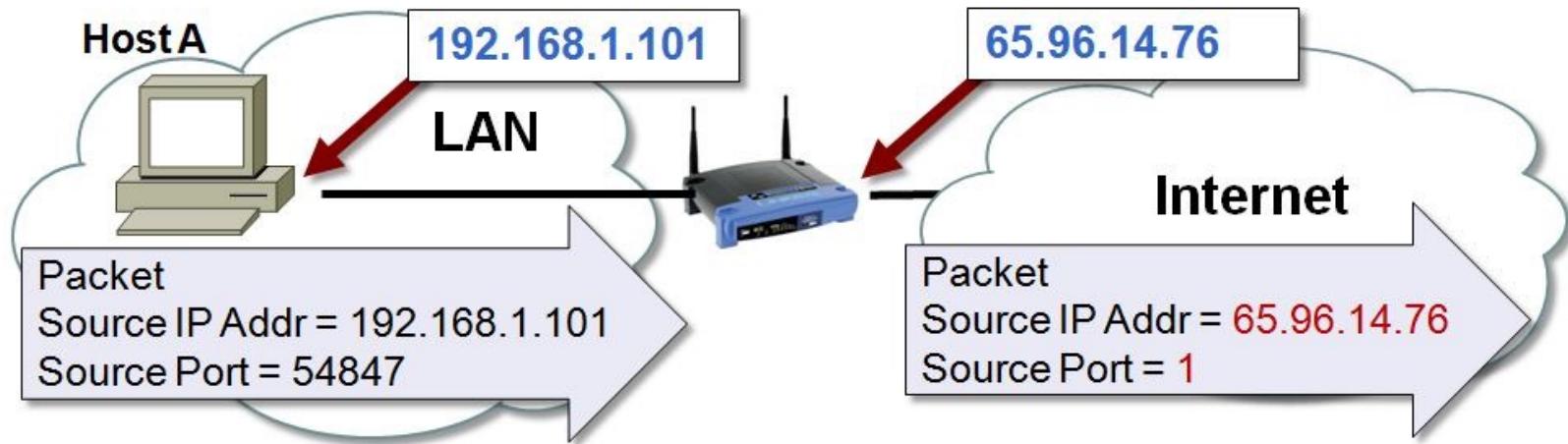
- **Public IP addresses** can be seen by any computer in the world and are required for communicating on the Internet
- Private IP addresses are typically secured behind a firewall
- Private IPv4 addresses are in the following ranges
  - **Class A** Private: **10.0.0.0 – 10.255.255.255**
    - $2^0=1$  network with  $2^{24}$  addresses
  - **Class B** Private: **172.16.0.0 – 172.31.255.255**
    - $2^4=16$  networks with  $2^{16}$  addresses
  - **Class C** Private: **192.168.0.0 – 192.168.255.255**
    - $2^8=256$  networks with  $2^8$  hosts on each network
- Private IP addresses must use NAT (see next slide) if accessing the Internet because **private IP addresses cannot communicate outside local networks**
- IPv6 doesn't have a need for private IP addresses (why?), but it still could use them if desired
- What does USC use for USC Guest Wireless and USC Secure Wireless?



# NAT

- **Network Address Translation** (NAT) allows a computer to have a private IP address (IP Masquerading)
  - Private IP addresses are not able to be accessed by hosts outside the local network
  - A NAT server (usually implemented in a router) substitutes its own public IP address in place of the computer's private IP address
  - The NAT server must maintain a NAT table that links the private IP address and TCP/UDP source port combination to the destination IP address
    - Since the NAT server's public IP address has been substituted for the computer's private IP address in the packet, the response from the destination computer will come back to the NAT server
    - The port will let the NAT server know to which computer to forward the response

# NAT Example



NAT Translation Table				
	Local IP Address	Source Port #	Internet IP Address	Source Port #
process X, Host A →	192.168.1.101	54,847	= 65.96.14.76	1
Host B →	192.168.1.103	24,123	= 65.96.14.76	2
process Y, Host A →	192.168.1.101	42,156	= 65.96.14.76	3
Host C →	192.168.1.102	33,543	= 65.96.14.76	4

## NAT Example (cont'd)

- Host A = MacBook, 192.168.1.101
- Process X = Chrome, port 54847
- Connecting to [www.usc.edu](http://www.usc.edu) home page: 13:35.90.69:80
- NAT maps 192.168.1.101:54847 to 65.96.14.76:1
- Browser requests page from www.usc.edu:
- **Source: 65.96.14.76:1, Destination: 13:35.90.69:80**
- USC web server responds with a page:
- **Source: 13:35.90.69:80, Destination: 65.96.14.76:1**
- Router uses NAT table to map 65.96.14.76:1 to 192.168.1.101:54847
- Chrome on the MacBook receives the page

# Subnets

- Since network addresses are so scarce in IPv4, we can take a few bits away from the host address to make a **subnet** within a network
- To implement subnetting, the router needs a subnet mask that indicates the split between the network/subnet combination and the host
- The **subnet mask** will consist of all 1's followed by all 0's
  - $255.255.255.0 = 11111111 \ 11111111 \ 11111111 \ 00000000$  **VALID**
  - $255.255.255.2 = 11111111 \ 11111111 \ 11111111 \ 00000010$  **NOT VALID**
- The subnet mask can also be written using slash notation
  - The number after the slash will represent the number of bits to be used in the subnet address
  - This would correspond to the number of 1's in the subnet mask
  - $255.255.255.0$  could be written as  $/24$ , a class C network

# **Subnets in IPv6**

- Subnetting exists in IPv6, but because there are so many bits in the IP address, we don't need to take bits from the host
  - Bits 1-48 - Network address
  - Bits 49-64 - Subnet address
  - Bits 65-128 - Host address
- Slash notation is used to identify the number of bits used in the network and subnet, but regardless, there are always  $2^{64}$  hosts on every subnet
  - A /64 means there is no subnetting
  - A /48 means there are  $2^{16}$  subnets available on that network
    - This is the maximum number of subnets available on an IPv6 network

# Subnet Example #1

- Assume a host has an IP address of 74.125.127.104 with a subnet mask of 255.255.255.192 (also written as 74.125.127.104/26)
  - What is the network address? (i.e., what class IP address is this?)
  - What is the subnet mask in binary?
  - How many hosts can be in the subnet?
  - 74.125.127.104 = 01001010 01111101 01111111 01101000
  - 255.255.255.192 = 11111111 11111111 11111111 11000000
- Answers:
  - **Network Address** - this is a Class A address, so the first 8 bits are allocated for the network: **74.0.0.0**
  - **Subnet Mask** in binary will have the first 26 bits as 1's with the last 6 bits as 0's
  - **# Hosts** - the last 6 bits are reserved for hosts, giving  $2^6 - 2 = 62$  host addresses for this subnet
    - Remember that the address with all 0s and all 1s are reserved

## Subnet Example #2

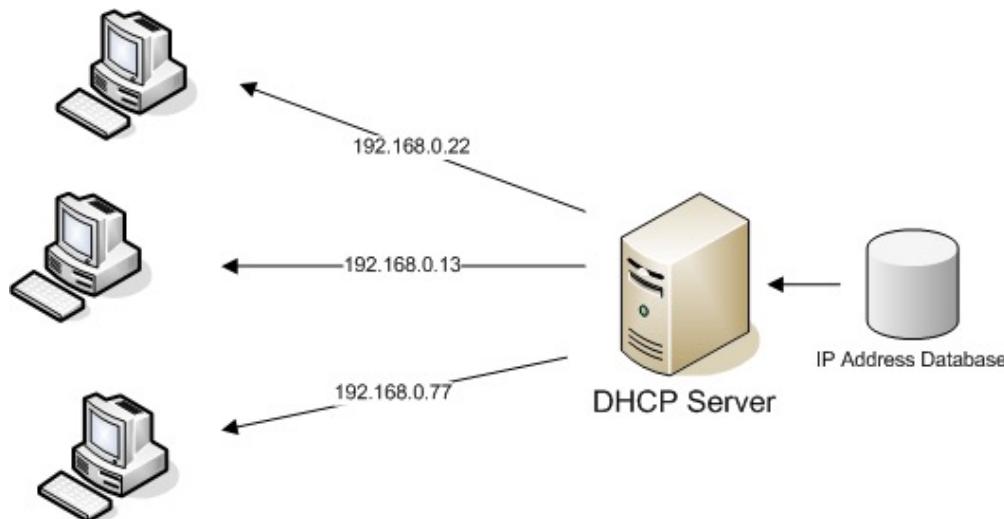
- To get the network/subnet address from an IP address and subnet mask, perform a **logical AND** operation between them

137.229.154.221	=	1000 1001 1110 0101 1001 1010 1101 1101
255.255.224.0	=	1111 1111 1111 1111 1110 0000 0000 0000
137.229.128.0	=	1000 1001 1110 0101 1000 0000 0000 0000

- The **network/subnet address** is 137.229.128.0
- This **subnet mask** provides **19** bits for the network address and 13 bits for the host address
- Another way we could have written the IP address with the subnet mask is 137.229.154.221/**19**
- Note that the subnet mask can never contain fewer 1 bits than the number of bits in the network address
  - Why does that make sense?

# DHCP

- **Dynamic Host Control Protocol** (DHCP) is used for a computer or router to automatically assign IP addresses and other network configuration (such as the gateway and subnet mask) to computers on the network
  - These addresses can be private or public IP addresses
  - Most routers assign private IP addresses, such as 192.168.1.101
  - Routers often have DHCP servers built into them

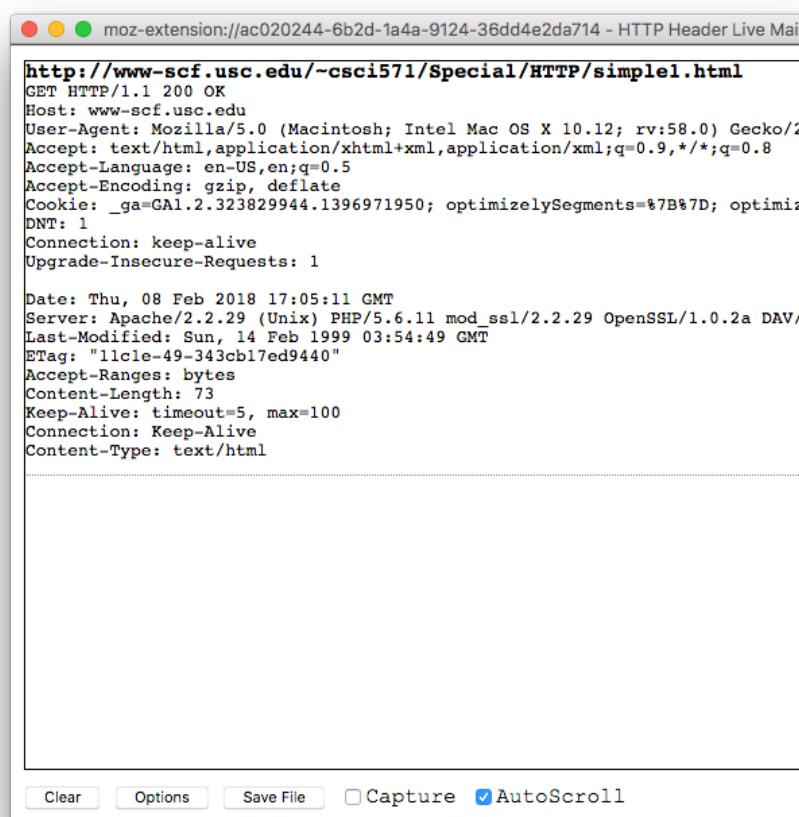


## **Exercises to Examine the HTTP Protocol**

1. HTTP Header Live
2. Wireshark
3. Postman

# Firefox “HTTP Header Live” add-on

- Download at
  - <https://addons.mozilla.org/en-US/firefox/addon/http-header-live/>
  - Compatible with Firefox Quantum 57+



The screenshot shows the 'HTTP Header Live Main' window. At the top, it displays the URL <http://www-scf.usc.edu/~csci571/Special/HTTP/simple1.html>. Below the URL, the page content is shown as a blank white area. Above the content, the full HTTP request and response headers are listed. The request headers include:

```
GET HTTP/1.1 200 OK
Host: www-scf.usc.edu
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.12; rv:58.0) Gecko/2
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
Cookie: _ga=GAI.2.323829944.1396971950; optimizelySegments=%7B%7D; optimiz
DNT: 1
Connection: keep-alive
Upgrade-Insecure-Requests: 1
```

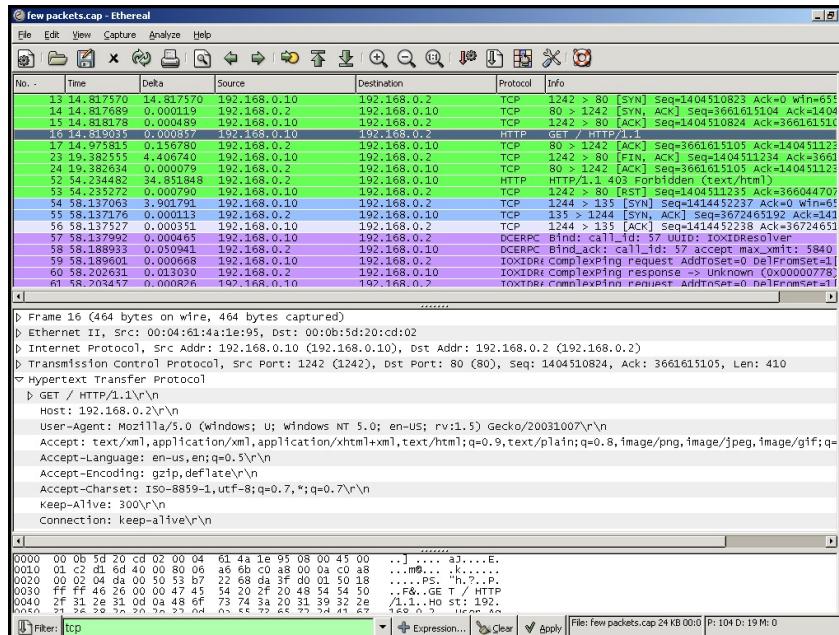
The response headers include:

```
Date: Thu, 08 Feb 2018 17:05:11 GMT
Server: Apache/2.2.29 (Unix) PHP/5.6.11 mod_ssl/2.2.29 OpenSSL/1.0.2a DAV/
Last-Modified: Sun, 14 Feb 1999 03:54:49 GMT
ETag: "11c1e-49-343cb17ed9440"
Accept-Ranges: bytes
Content-Length: 73
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html
```

At the bottom of the window, there are several buttons: 'Clear', 'Options', 'Save File', 'Capture' (unchecked), and 'AutoScroll' (checked).

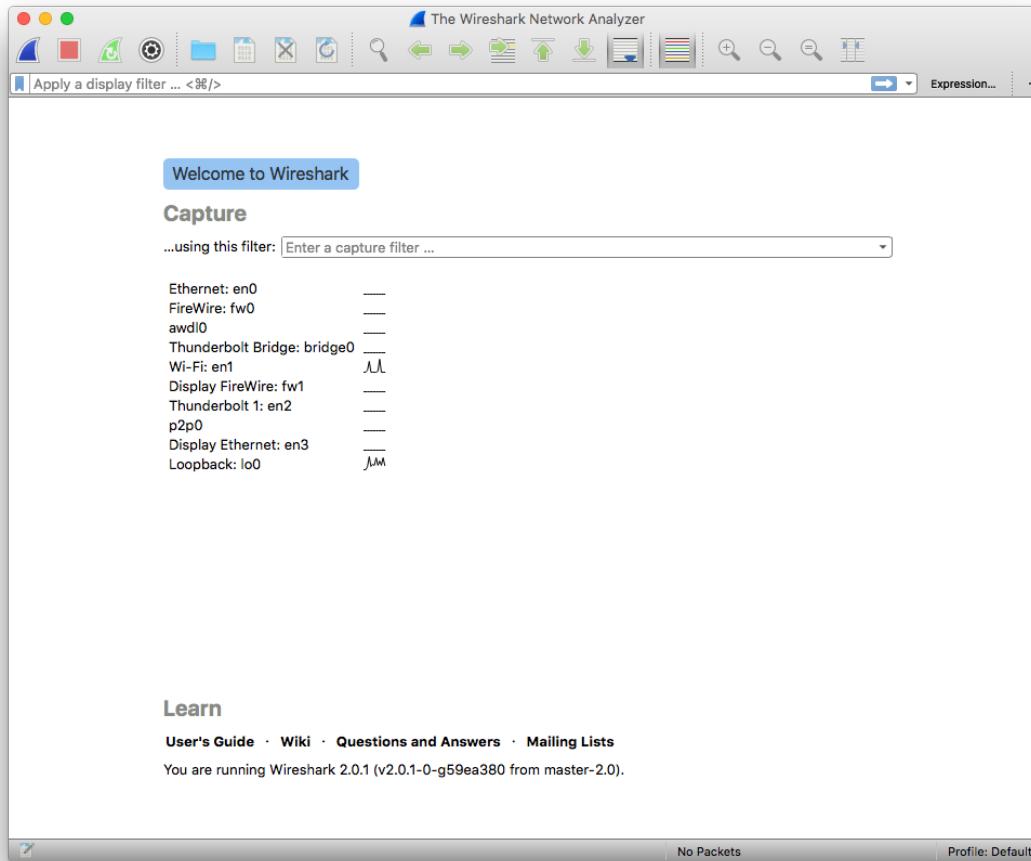
# Wireshark Protocol Analyzer

- Install a protocol analyzer (a.k.a. a “sniffer”), such as **WireShark** (formerly Ethereal), available at
  - <http://www.wireshark.org/>
- Captured network data can be browsed via a GUI
- Supports 1300+ protocols
- Multiplatform: Windows, macOS, Linux, and many more



# **www.wireshark.org**

- Download the latest “stable” release of Wireshark (currently 3.4.8)



Initial Screen (new interface)

## **Filter HTTP requests**

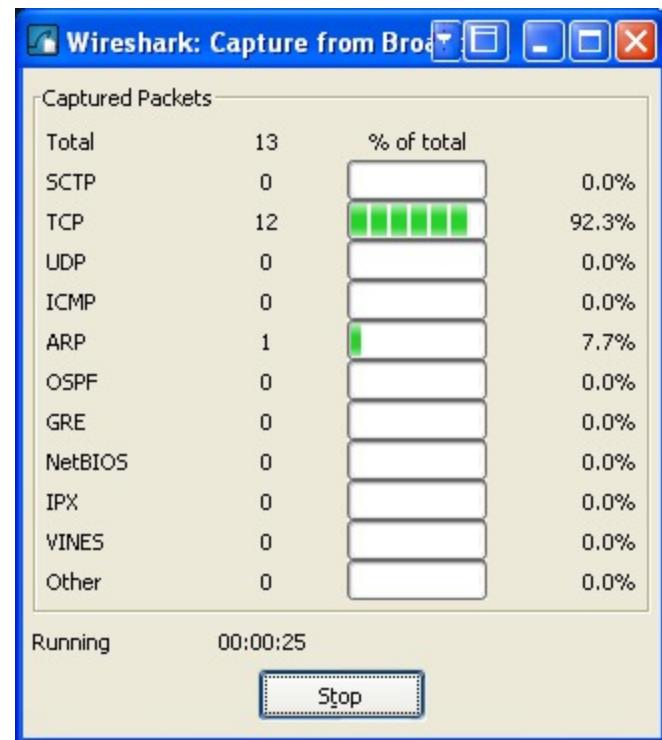
- In your browser open up the page  
<http://csci571.com/Special/HTTP/proxy.html>
- In Wireshark
  - Click on Expression on the right-hand side of the third row
  - Select HTTP and click on OK
  - Click on Apply on the third row at the right-hand end
- In Wireshark
  - Click on Capture | Interfaces and then click the Start button that is on the line related to your wired/wireless connection to the Internet
  - A window appears as shown on the next slide

## Start up the web Traffic

- Return to the browser and paste the first test URL into the address field

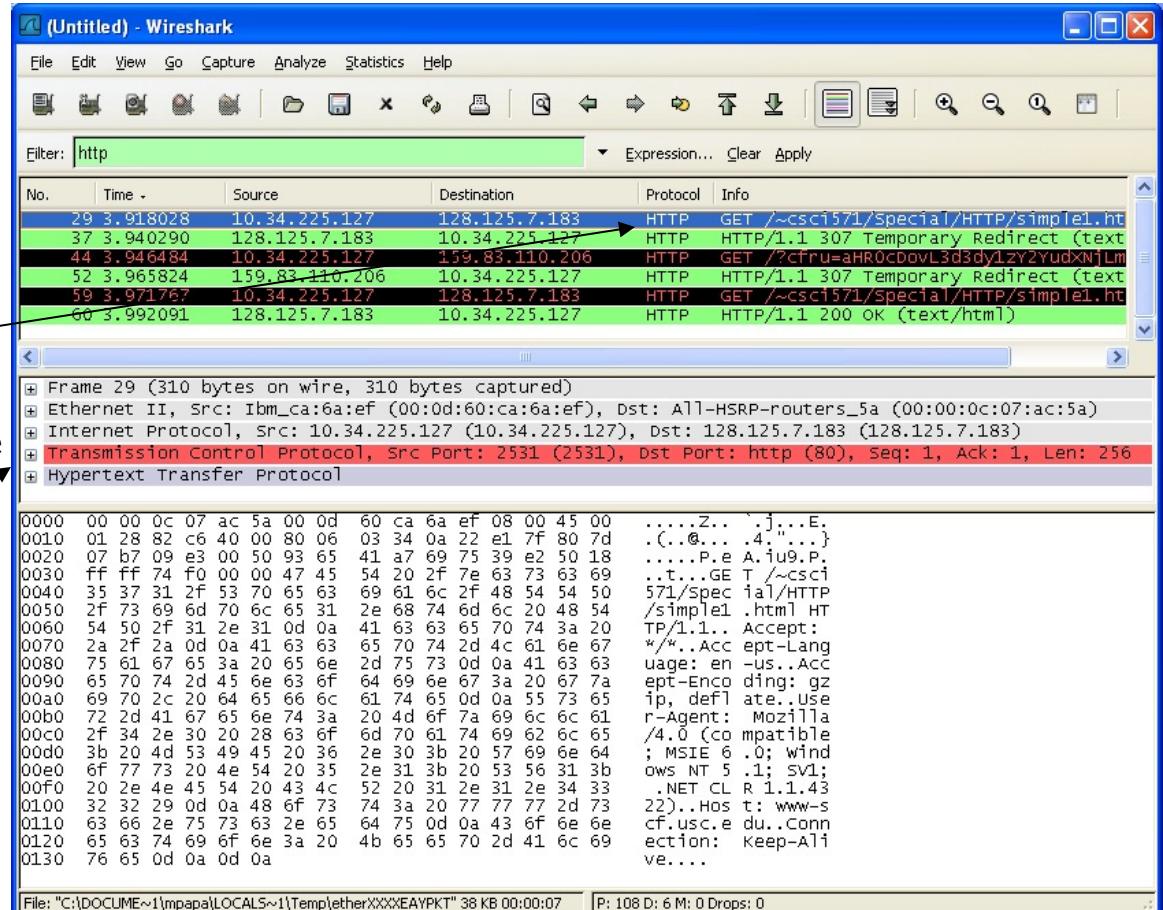
<http://csci571.com/Special/HTTP/simple1.html>

- Once the requested page appears in the browser return to the window that is shown on the right and click on the Stop button
- A new Wireshark screen appears as shown on the next slide



# WireShark Output

- Scroll the green window until a line with HTTP GET appears;
- Click on the line;
- Then, in the middle window expand the line labeled Hypertext Transfer Protocol



# Expansion of HTTP request

The screenshot shows the Wireshark interface with a captured session titled '(Untitled) - Wireshark'. A filter bar at the top is set to 'http'. The main pane displays three network frames. Frame 224 is selected and expanded, showing its detailed structure:

- Frame 224 (636 bytes on wire, 636 bytes captured)
- Ethernet II, Src: Intel\_7a:c9:d8 (00:11:11:7a:c9:d8), Dst: LinksysG\_e7:0d:e9 (00:06:25:e7:0d:e9)
- Internet Protocol, Src: 192.168.1.100 (192.168.1.100), Dst: 128.125.5.229 (128.125.5.229)
- Transmission Control Protocol, Src Port: 1782 (1782), Dst Port: http (80), Seq: 1, Ack: 1, Len: 582
- Hypertext Transfer Protocol
  - GET /~csci571/Special/HTTP/simple1.html HTTP/1.1\r\n
  - Host: www-scf.usc.edu\r\n
  - User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.8.1) Gecko/20061010 Firefox/2.0\r\n
  - Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,\*/\*;q=0.5,text/vnd.wap.wml;q=0.6\r\n
  - Accept-Language: en-us,en;q=0.5\r\n
  - Accept-Encoding: gzip,deflate\r\n
  - Accept-Charset: ISO-8859-1,utf-8;q=0.7,\*;q=0.7\r\n
  - Cookie: \_\_utma=185748653.462061095.1156035922.1165639737.1165785580.10; MintUnique=1; \_\_utmz=185748653.1156035922.1.1.utmcn=(direct/)\r\n

The bottom pane shows the raw hex and ASCII data of the selected frame.

Notice the:

- GET request
- Host
- User-Agent
- Accepts
- Cookie

# Expansion of HTTP response

- Scroll down again in the upper window and click on the HTTP/1.1 200 OK line, as shown to the right
- Notice the additional lines that are sent including
  - Cache-Control
  - Content-Type
  - Server
  - Etc.



The screenshot shows the Wireshark interface with the following details:

- Filter Bar:** http
- Selected Frame:** Frame 266 (284 bytes on wire, 284 bytes captured)
- Protocol Tree:** The tree view shows the structure of the HTTP response, starting with the Hypertext Transfer Protocol, then HTTP/1.1 200 OK, followed by Cache-Control, Content-Type, Server, Transfer-Encoding, Content-Encoding, and Date.
- Text View:** The expanded text view shows the raw HTTP response message:

```
HTTP/1.1 200 OK\r\nCache-Control: private\r\nContent-Type: text/html\r\nServer: GWS/2.1\r\nTransfer-Encoding: chunked\r\nContent-Encoding: gzip\r\nDate: Sat, 16 Dec 2006 00:07:59 GMT\r\n\r\n
```
- Hex View:** The bottom pane shows the hex dump of the captured frame, with bytes 0100 through 0110 displayed.
- Status Bar:** Frame (284 bytes) | De-chunked entity body (40 bytes) | Uncompressed entity body (14 bytes) | P: 270 D: 120 M: 0 Drops: 0

## **Repeat the Exercise**

- Using the remaining links on  
<http://csci571.com/Special/HTTP/proxy.html>  
do the following:
  - In Wireshark, select the Capture Start menu
  - Click on the button “Continue without Saving”
  - Enter the new URL into the browser
  - Once the requested page appears click on Stop in the Wireshark Capture window
  - Examine the result in the main Wireshark window
  - Remember to note the HTTP commands, request, response and general commands

# Postman

- Collaboration platform for API development
- Supports all HTTP commands. Download from:  
<https://www.getpostman.com>
- Free to individuals and small teams
- Free license includes 1,000 API calls / month
- Desktop app available for MacOS, Windows and Linux

