

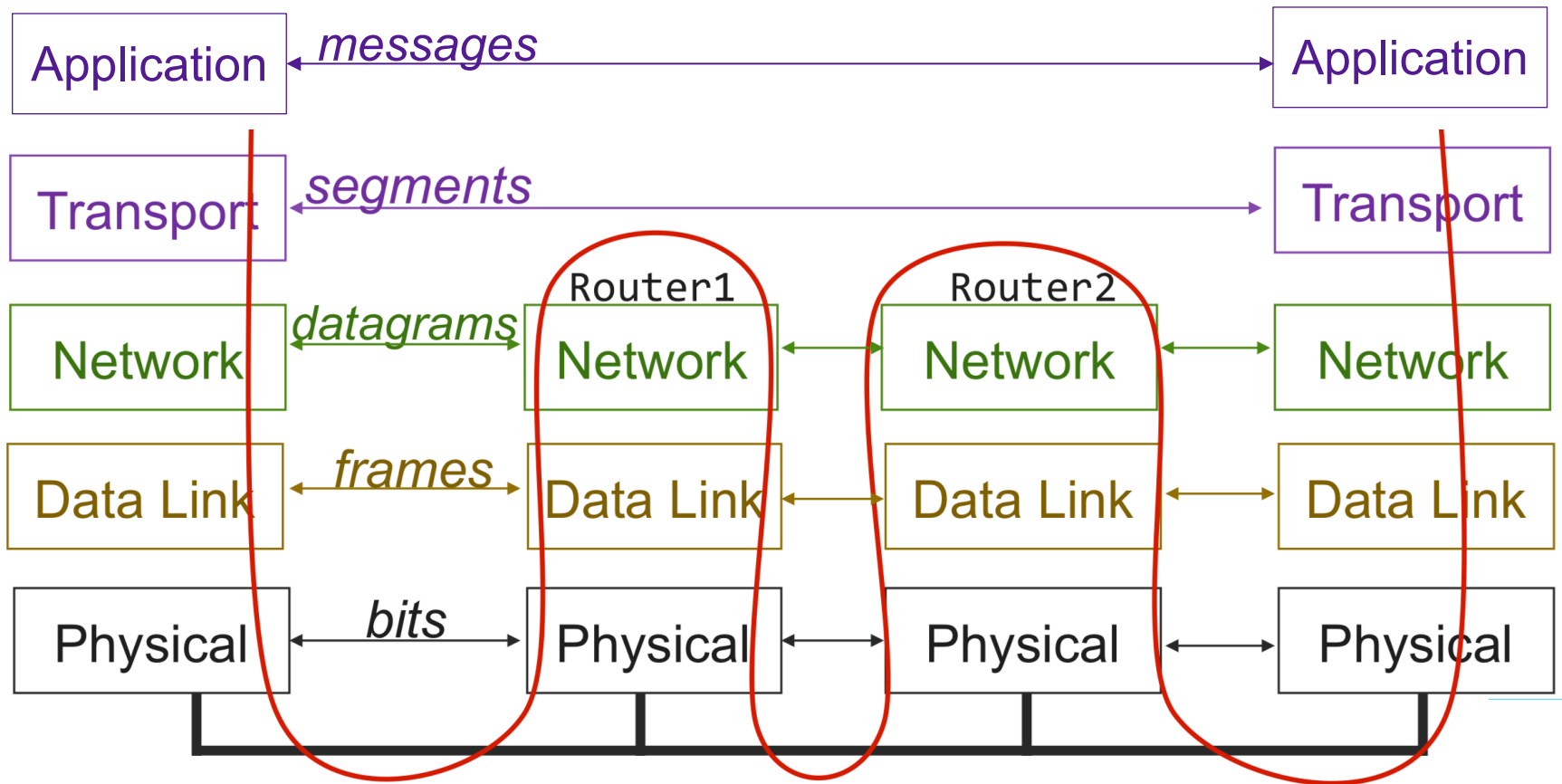
Lecture 25: Networking and the Internet

CS 105

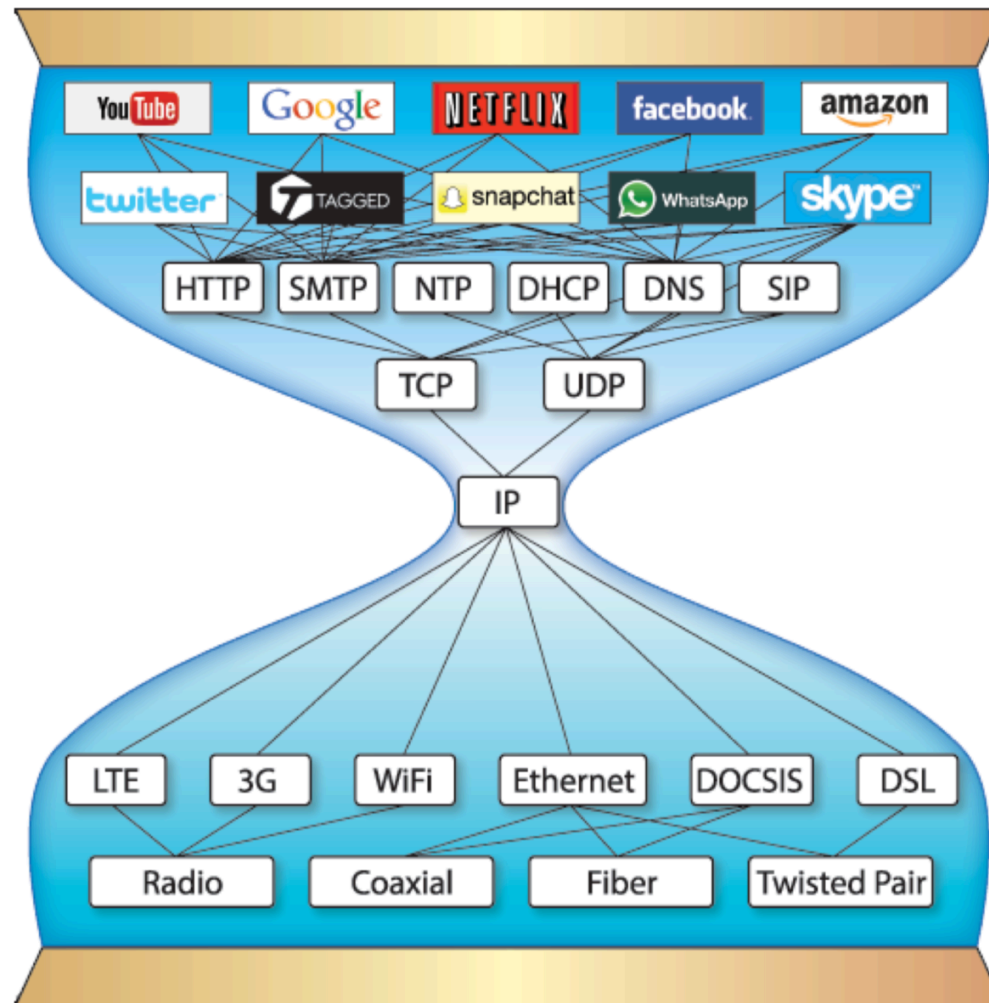
Fall 2020

What is the Internet?

The Big Picture



Continuing up the Network Stack...



URLs

- Unique name for a file: URL (Universal Resource Locator)
- Example URL: <http://www.cs.pomona.edu:80/classes/cs105/index.html>
- Clients use *prefix* (`http://www.cs.pomona.edu:80`) to infer:
 - Where the server is (`www.cs.pomona.edu`)
 - What port it is listening on (80)
 - What kind (protocol) of server to contact (HTTP)

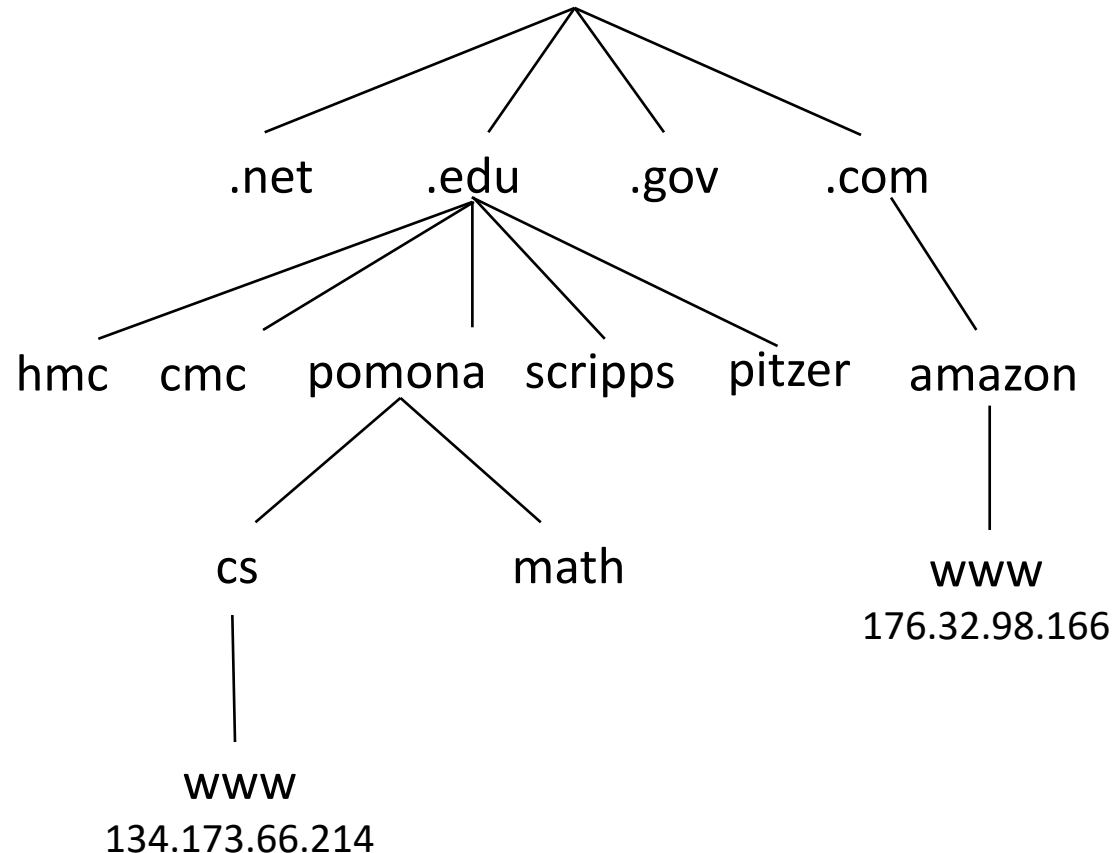
Domain Name System (DNS)

- Principals are identified by names
 - for web hosts, typically a domain name
 - e.g., www.cs.pomona.edu
- Internet hosts are identified by IP addresses
 - used by network layer to route packets between hosts
- The role of DNS is to translate between domain names and IP addresses



Domain Name System (DNS)

- Distributed, hierarchical database
- Application-level protocol: hosts and DNS servers communicate to resolve names
- Names are separated into components by dots
- lookup occurs top down

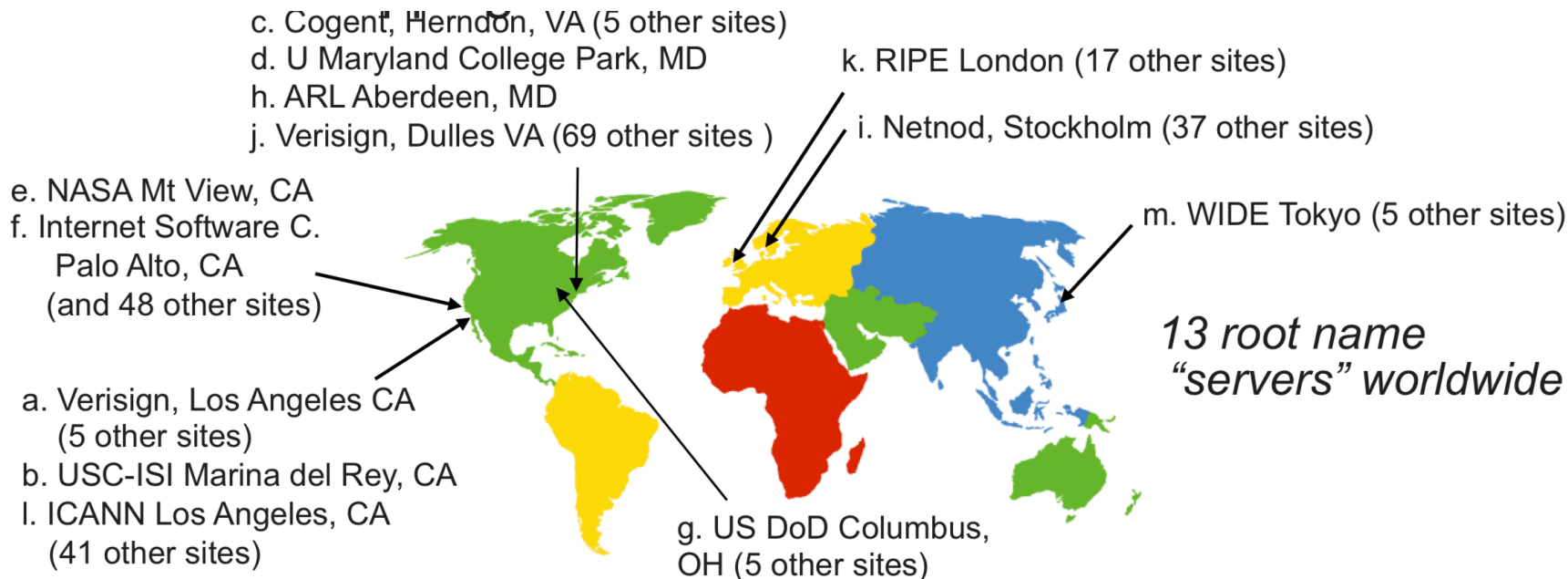


DNS Lookup

- the client asks its local nameserver
- the local nameserver asks one of the *root nameservers*

DNS Root Name Servers

- contacted by local name server that can't resolve name
- owned by Internet Corporation for Assigned Names & Numbers (ICANN)
- contacts authoritative name server if name mapping not known, gets mapping
- returns mapping to local name server



DNS Lookup

- the client asks its local nameserver
 - the local nameserver asks one of the *root nameservers*
 - the root nameserver replies with the address of the top level nameserver
 - the server then queries that nameserver
 - the top level nameserver replies with the address of the authoritative nameserver
 - the server then queries that nameserver
 - repeat until host is reached, cache result.
-
- Example: Client wants IP addr of `www.amazon.com`
 1. Queries root server to find `com` DNS server
 2. Queries `.com` DNS server to get `amazon.com` DNS server
 3. Queries `amazon.com` DNS server to get IP address for `www.amazon.com`

URLs

- Unique name for a file: URL (Universal Resource Locator)
- Example URL: <http://www.cs.pomona.edu:80/classes/cs105/index.html>
- Clients use *prefix* (`http://www.cs.pomona.edu:80`) to infer:
 - Where the server is (`www.cs.pomona.edu`)
 - What port it is listening on (80)
 - What kind (protocol) of server to contact (HTTP)

Well-known Ports and Service Names

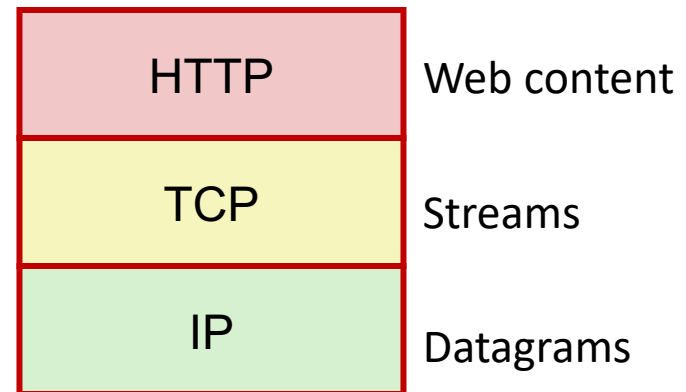
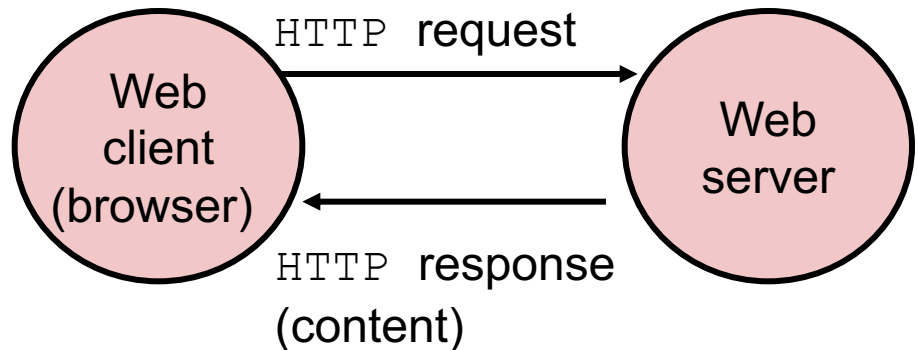
- Popular services have permanently assigned **well-known ports** and corresponding **well-known service names**:
 - echo server: 7/echo
 - ssh servers: 22/ssh
 - email server: 25/smtp
 - Web servers: 80/http or 443/https
- Mappings between well-known ports and service names is contained in the file `/etc/services` on each Linux machine.

URLs

- Unique name for a file: URL (Universal Resource Locator)
- Example URL: <http://www.cs.pomona.edu:80/classes/cs105/index.html>
- Clients use *prefix* (`http://www.cs.pomona.edu:80`) to infer:
 - Where the server is (`www.cs.pomona.edu`)
 - What port it is listening on (80)
 - What kind (protocol) of server to contact (HTTP)
- Servers use *suffix* (`/classes/cs105/index.html`) to:
 - Specify what content they want

HTTP

- Clients and servers communicate using the HyperText Transfer Protocol (HTTP)
 - Client and server establish TCP connection
 - Client requests content
 - Server responds with requested content
 - Client and server close connection (eventually)
- Current version is HTTP/2.0
 - RFC 7540, 2015
 - Includes protocol negotiation
 - HTTP/1.1 still in use (RFC 2616, 1999)
 - HTTP/3 proposed



HTTP Requests

- HTTP request is a **request line**, followed by zero or more **request headers**
- **Request line:** `<method> <uri> <version>`
 - `<method>` is one of GET, POST, OPTIONS, HEAD, PUT, DELETE, or TRACE
 - `<uri>` is typically URL for proxies, URL suffix for servers
 - A URL is a type of URI (Uniform Resource Identifier)
 - See <http://www.ietf.org/rfc/rfc2396.txt>
 - `<version>` is HTTP version of request (HTTP/1.0 or HTTP/1.1)
- **Request headers:** `<header name>: <header data>`
 - Provide additional information to the server

HTTP Responses

- HTTP response is a **response line** followed by zero or more **response headers**, possibly followed by **content**
 - a blank line (“\r\n”) separates headers from content.
- **Response line:** `<version> <status code> <status msg>`
 - `<version>` is HTTP version of the response
 - `<status code>` is numeric status
 - `<status msg>` is corresponding English text
 - 200 OK Request was handled without error
 - 301 Moved Provide alternate URL
 - 404 Not found Server couldn't find the file
- **Response headers:** `<header name>: <header data>`
 - Provide additional information about response
 - `Content-Type:` MIME type of content in response body
 - `Content-Length:` Length of content in response body

Web Content

- Web servers return content to clients
 - *content*: a sequence of bytes with an associated MIME (Multipurpose Internet Mail Extensions) type
- Example MIME types
 - `text/html` HTML document
 - `text/plain` Unformatted text
 - `image/gif` Binary image encoded in GIF format
 - `image/png` Binary image encoded in PNG format
 - `image/jpeg` Binary image encoded in JPEG format

You can find the complete list of MIME types at:

<http://www.iana.org/assignments/media-types/media-types.xhtml>

Static and Dynamic Content

- The content returned in HTTP responses can be either **static** or **dynamic**
 - *Static content*: content stored in files and retrieved in response to an HTTP request
 - Examples: HTML files, images, audio clips
 - Request identifies which content file
 - *Dynamic content*: content produced on-the-fly in response to an HTTP request
 - Example: content produced by a program executed by the server on behalf of the client
 - Request identifies file containing executable code
- Bottom line: *Web content is associated with a file that is managed by the server*

Tiny Web Server

- Tiny Web server
 - Tiny is a sequential Web server
 - Serves static and dynamic content to real browsers
 - text files, HTML files, GIF, PNG, and JPEG images
 - 239 lines of commented C code
 - Not as complete or robust as a real Web server
 - You can break it with poorly-formed HTTP requests (e.g., terminate lines with “\n” instead of “\r\n”)

Feedback

1. Rate how well you think this recorded lecture worked
 1. Better than an in-person class
 2. About as well as an in-person class
 3. Less well than an in-person class, but you still learned something
 4. Total waste of time, you didn't learn anything
2. How much time did you spend on this video lecture?
3. Do you have any comments or feedback?