Networking Fundamentals:
IP, DNS, URL, MIME

Lecture 7
Internet Protocol (IP) Addresses

- A unique 32-bit number
  - Assigned to device connected to internet
  - An address for delivery of packets

- Written in “dotted-decimal” notation
  - Divided into 4 fields separated by “.”
  - Each field is 8 bits, ie 0-255 decimal
    - 10100100.01101011.01111011.00000110
    - 164.107.123.6

- Some are reserved: eg, 127.0.0.1
Representation and Abstraction

- Abstraction: 32-bit integer value
- Representations
  - Dotted decimal
  - Dotted hex
  - Dotted octal
  - Hexadecimal
  - Decimal
  - Binary
  - Etc...
Address Space

- Organizations are allocated blocks of contiguous address to use
- 32 bits means 4 billion addresses
  - Population of the earth: 7 billion
  - Not enough addresses to go around!
- The end is predictable
  - Techniques like NAT developed to help
- In fact, the end has come!
  - Feb 2011: Last block was allocated
IPv6

- 128 bits
  - \(~10^{40}\) addresses; we’re good for a while
  - A tiny fraction of IP traffic today

- Recommended format (canonical):
  - Divide into 8 fields separated by “:”
  - Each field 16 bits, ie 4 hex digits (0-FFFF)
  - Omit leading 0’s in a field
  - If there are consecutive fields with value 0, compress them as “::”
  - Compress at most 1 such set of 0’s
    - Otherwise rep could be ambiguous
  - Compress the longest sequence
Canonical Format: Uniqueness

2001:0db8:0000:0000:0000:ff00:0042:8329
2001:0db8:0000:0000:0000:ff00:0042:8329

2001:db8:0:0:0:ff00:42:8329
2001:db8:0:0:0:ff00:42:8329

2001:db8::ff00:42:8329
2001:db8::ff00:42:8329
Domain Names

- String corresponds to an IP address
  - `cse.ohio-state.edu` is easier than `164.107.123.6`
  - See host, whois
- Case insensitive: Lower-case standard
- A partial map (almost)
  - DNS maps lower-case strings → IP addresses
  - Multiple strings can map to same address!
  - Some strings map to multiple addresses (unusual)!
Domain Name Hierarchy

- Separated by .’s
  - Don’t confuse with dotted decimal!

- Right-to-left hierarchy
  - Top level domain is right-most field
    - edu, com, net, gov, countries (ca, it, ...)
  - Second-level domain to its left
  - Then third, fourth, etc, no limit
    www.sos.state.oh.us

- Hostname + Domain Name =
  Fully Qualified Domain Name (FQDN)
  stdlinux.cse.ohio-state.edu
Name Servers

- Act as a phonebook for lookup
- Given a FQDN, return IP address
- Hierarchical by domain, with local caching for recently retrieved items
Protocols

- Systematic ordering of messages
  - Phone rings
  - Callee answers by saying “Hello”
  - Caller answers by saying “Hello”

- Different protocols use different messages, different sequencing, etc
  - In Italy, callee answers by saying “Pronto”
Network Layering: Abstraction

- One protocol is built on top of another
  - Application level: FTP, HTTP, SSH, SMTP, TELNET
  - Transport: TCP, UDP
  - Internet: IP

- Each protocol assumes certain behavior from layer below
  - IP routes packets to destination (unreliable)
  - TCP creates a reliable, in-order channel
  - HTTP delivers web pages
Network Ports

- A single host has many ports
- Application-level protocols have default port
  - ftp 20
  - http 80
  - imap 143
  - ssh 22
  - smtp 25
  - telnet 23
- A running web server is waiting, listening for the phone (on port 80)
  - See telnet
URL

- Uniform Resource Locator
  scheme://FQDN:port/path?query#fragment

- Schemes include http, ftp, mailto, file...
  - Case insensitive, prefer lower case

- Port is optional (uses default)
  - 80 for http

- Variety of formats, depending on scheme
  ftp://doe@ftp.cse.ohio-state.edu
  mailto://brutus.1@osu.edu

- FQDN is case insensitive, prefer lower case
Concrete invariant (convention)
- No space, ;, :, & in representation
- To represent these characters, use %hh instead (hh is ASCII code in hex)
  - %20 for space
- Q: What about % in abstract value?
Document Root

- Web server configured to serve documents from a location in file system
  - “document root”: /class/3901
  - File: /class/3901/labs/lab2.html
  - URL: http://www.cse.osu.edu/labs/lab2.html

- Slashes in path should be for server’s OS (but forward slashes are common)

- Virtual servers: multiple doc roots
- Proxy servers: remote doc roots
MIME

- Multipurpose Internet Mail Extensions
  - Used to be for mail attachments

- How to interpret a file
  - File is a blob of bits (representation)
  - How to map this blob into (abstract) value? Colors, sounds, characters, etc?

- Syntax: type/subtype
  - plain/text, plain/html
  - image/gif, image/jpg
  - video/mpeg, video/quicktime
Determining MIME Type

- The sender (web server) determines MIME type of document being sent
  - Rules map extensions to MIME types
- If file arrives without MIME info, receiver has to guess
  - File extension may suggest the MIME type
- Some types handled by browser itself
- Others require plugin or application
- Experimental MIME subtypes: x-
  - application/x-gzip
Summary

- IP address are unique on network
  - IPv4 vs IPv6
- DNS maps strings to IP addresses
  - Domains nested hierarchically
- URLs identify resources on network
  - Scheme, host, path
- MIME type defines a file’s encoding
  - Correspondence
HTTP: Hypertext Transfer Protocol
HTTP

- Hypertext Transfer Protocol

- History
  - Developed at CERN, Tim Berners-Lee
  - Early 90’s
  - 1996: version 1.0
  - 1999: version 1.1 (current version!)

- Simple request/response
  - Client sends request to (web) server
  - (Web) server responds
  - “stateless” protocol
Request/Response Anatomy

- An HTTP request (or response) consists of
  1. Header: meta information
  2. Body (sometimes): the payload

- The header consists of
  1. Method (requests) / Status (response)
  2. Header fields (separated by newlines)
  3. Blank line
Protocol: Request, Response

- **Request**
  - Method
  - Header field 1
  - Header field 2
  - Body

- **Response**
  - Status
  - Header field 1
  - Header field 2
  - Header field 3
  - Body
Request Header: First Line

- Syntax of first line:
  - verb path version
  - Verb: GET, HEAD, POST, PUT, DELETE,…
  - Path: part of URL (path-to-document)
  - Version: HTTP/1.1

- Example:
  - URL
  - Request begins
    - GET /news/ HTTP/1.1
Request Header: Header Fields

- Each field on its own line, syntax: `name: value`
- Examples
  Host: cse.ohio-state.edu
  Accept: text/*
  Accept: image/gif
  If-Modified-Since: Sat, 12 May 2014 19:43:31 GMT
  Content-Length: 349
  User-Agent:

- Followed by blank line
"On the Internet, nobody knows you're a dog."
"Nobody knows you're a dog"

GET / HTTP/1.1
Host: news.osu.edu
User-Agent: Mozilla/5.0 (X11; Ubuntu;...etc

Request

$ telnet

$ curl
Header Fields cont’d

- **Host**
  - Only required field
  - Q: Why is host needed?

- **Accept**
  - Browser preference for MIME type(s) to receive

- **If-Modified-Since**
  - Send payload only if changed since date
  - Date must be GMT

- **Content-Length**
  - Required if request has a body
  - # number of bytes in body
Demo

- Send this HTTP request to host web.cse.ohio-state.edu (on port 80):
  
  ```
  GET /~paolo/ HTTP/1.1
  Host: web.cse.ohio-state.edu
  <blank line>
  ```

- At console
  
  ```
  $ telnet web.cse.ohio-state.edu 80
  Then type what to send (see above)
  ```
HTTP Response Anatomy

- Recall, four parts
  1. Status (one line)
  2. Header fields (separated by newlines)
  3. Blank line
  4. Body: the payload

- Parts 1-3 are the “header”

- Part 1, status line syntax:
  \textit{http-version status-code text}

  - Examples
    - HTTP/1.1 200 OK
    - HTTP/1.1 301 Moved Permanently
    - HTTP/1.1 404 Not Found
# Taxonomy of Status Codes

<table>
<thead>
<tr>
<th>Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1xx</td>
<td>Informational</td>
</tr>
<tr>
<td>2xx</td>
<td>Success</td>
</tr>
<tr>
<td>3xx</td>
<td>Redirection</td>
</tr>
<tr>
<td>4xx</td>
<td>Client Error</td>
</tr>
<tr>
<td>5xx</td>
<td>Server Error</td>
</tr>
</tbody>
</table>
Some Common Status Codes

- **200 OK**
  - All is good!
  - Response body is the requested document

- **301 Moved Permanently**
  - Requested resource is found somewhere else (please go there in the future)

- **304 Not Modified**
  - Document hasn’t changed since date/time in If-Modified-Since field of request
  - No response body

- **404 Not Found**
  - Server could not satisfy the request
  - It is the client’s fault (design-by-contract?)

- **500 Internal Server Error**
  - Server could not satisfy the request
  - It is the server’s fault (design-by-contract?)
Response Header: Header Fields

- Each field on its own line, syntax: 
  `name: value`

- Examples
  `Date: Mon, 22 Sep 2014 14:51:38 GMT`
  `Server: Apache/2.2.3 (Red Hat)`
  `Content-Type: text/html; charset=iso-8859-1`
  `Content-Length: 333`

- Followed by blank line
Demo: Using Terminal

- Use telnet to retrieve
  http://web.cse.ohio-state.edu/~paolo
  - Fails (see status code)
  http://web.cse.ohio-state.edu/~paolo/
  - Body is incomplete (no images)
  - Body is chunked

- Use curl to retrieve
  - Handles headers, redirection, chunking,...
  $ curl -L http://web.cse.ohio-state...
Demo: Using Firefox

- Developer > Network
- One GET results in many requests
  - http://www.cse.osu.edu/~paolo
- For each request, see:
  - Request headers
  - Response status code
  - Response headers
  - Response (and preview)
Demo: Using Ruby

- Mechanize: A Ruby gem for HTTP
  ```ruby
  require "mechanize"
  ```

- Create an agent to send requests
  ```ruby
  agent = Mechanize.new do |a|
    a.user_agent_alias = "Mac Safari"
  end
  ```

- Use agent to issue a request
  ```ruby
  page = agent.get "http://www.osu.edu"
  ```

- Follow links, submit forms, etc
  ```ruby
  page.link_with(text: "Carmen").click
  ```
Request Methods

- **GET, HEAD**
  - Request: should be *safe* (no side effects)
- **PUT**
  - Update (or create): should be *idempotent*
- **DELETE**
  - Delete: should be *idempotent*
- **POST**
  - Create (or update): changes server state
  - Beware re-sending!
- HTTP does not enforce these semantics
Passing arguments (GET)

- Arguments are key-value pairs
  Mascot: Brutus Buckeye
  Dept: CS&E

- Can be encoded as part of URL
  scheme://FQDN:port/path?query#fragment

- application/x-www-form-urlencoded
  - Each key-value pair separated by & (or ;)
  - Each key separated from value by =
  - Replace spaces with + (arcane!)
  - Then normal URL encoding
    Mascot=Brutus+Buckeye&Dept=CS%26E
Examples

- Wikipedia search
  http://en.wikipedia.org/w/index.php?
  search=ada+lovelace

- OSU job postings
  http://news.osu.edu/articles-by-category.html?
  tag=Press+release&
  key=article_category

- Random numbers
  http://random.org/passwords
  ▪ Use FF Developer to submit, edit request
Passing Arguments (POST)

- Encoded as part of the body

- Advantages:
  - Arbitrary length (URLs are limited)
  - Arguments not saved in browser history
  - Slightly more secure (not really)

- Content-Type header field specifies
  - application/x-www-form-urlencoded
    - Same encoding scheme as seen with GET
  - multipart/form-data
    - Better for binary data (else 1 byte → 3 bytes)
  - More options too:
    - application/xml, application/json, ...
Summary

- HTTP: request/response
- Anatomy of request
  - Methods: GET, PUT, DELETE, POST
  - Headers
  - Body: arguments of POST
- Anatomy of response
  - Status Codes: 200, 301, 404, etc
  - Headers
  - Body: payload
- Tools
  - Curl, FF Developer, Mechanize