To Ponder

An exercise in mental arithmetic:

\[ 2^{32} = ? \]

\[ 2^{128} = ? \]
Networking Fundamentals: IP, DNS, URL, MIME

Lecture 8
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
    - 164.107.123.6
- Some are reserved: eg, 127.0.0.1
Abstract Value vs Encoding

- Abstraction: 32-bit integer value
- Encodings
  - 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
  - \( \sim 10^{40} \) addresses; we’re good for a while
  - A growing fraction of IP traffic
    - GoogleIPv6 statistics

- Recommended format (canonical):
  - Divide into 8 fields separated by “:”
  - Each field is 4 hex digits (0-FFFF), ie 16 bits
  - Omit leading 0’s in a field
  - If there are consecutive fields with value 0, compress them as “::”
  - Compress at most one such set of 0’s
    - Otherwise encoding 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
Domain Names

- String corresponds to an IP address
  - `web.cse.ohio-state.edu` is easier than `164.107.123.6`

- Case insensitive: Lower-case standard

- A partial map (almost)
  - DNS maps lower-case strings $\rightarrow$ 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
ru.wikipedia.org.
Name Servers

- Act as a phonebook for lookup
- Client view:
  - Given a FQDN, return IP address
  - Partial map: FQDNs → IP addresses
  - See host, whois
- Implementation view:
  - Hierarchical by domain
  - Local caching for recently retrieved items
- Command line tools
  - $ host web.cse.ohio-state.edu
  - # web.cse.ohio-state.edu has address 164.107.129.176
  - $ whois campusparc.com
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: TPC, 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 “web server” is just a program, running, waiting, listening for a call (on port 80)
  - See telnet
URL

- Uniform Resource Locator
  scheme://FQDN:port/path?query#fragment
- Schemes include http, ftp, mailto, file...
  - Case insensitive, but prefer lower case
- Port is optional (each scheme has 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
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
Encoding (and Decoding)

- A single value can be viewed at two levels, eg:
  - HELLO
  - ..... .-. .-.
- Different uses: reading vs transmission
- Different alphabets (letters vs dot-dash) and/or requirements
  - Eg. Message has only upper case letters
- Encoding/decoding is the translation between these levels
  - c.f. encrypting/decrypting
- Abstract value vs concrete representation
  - Correspondence maps between the two
Example: URL Encoding

- Invariant on encoding (convention)
  - Small set of valid characters, no arbitrary ones (space, ~, newline...)

- Invariant on value (constraint)
  - Reserved metacharacters (;, :, &, #, @...)

- So some characters in abstract value are encoded as %hh (ASCII code in hex)
  - %20 for space, %7E for ~
  - %3B for ;, %40 for @

- Q: What about % in abstract value?
  - A: Encode it too! %25

- aka “percent encoding”
URL Encoding

Reserved characters after percent-encoding

| ! | # | $ | % | & | ' | ( | ) | * | + | , | / | : | ; | = | ? | @ | [ | ] |
| %21 | %22 | %23 | %24 | %25 | %26 | %27 | %28 | %29 | %2A | %2B | %2C | %2D | %2E | %2F | %3A | %3B | %3C | %3D | %3E | %3F | %40 | %5B | %5D |

Common characters after percent-encoding (ASCII or UTF-8 based)

| newline | space | " | % | - | . | < | > | \ | ^ | _ | ` | { | | } | ~ |
| %0A     | %20  | %22 | %25 | %2D | %2E | %3C | %3E | %5C | %5E | %60 | %7B | %7C | %7D | %7E |

<table>
<thead>
<tr>
<th>%0D</th>
<th>%0A</th>
</tr>
</thead>
</table>

Value

Mascot "address": brutus@osu.edu

Encoding

Mascot%20%22address%22%3A%20brutus%40osu.edu
MIME

- Multipurpose Internet Mail Extensions
  - Used to be for mail attachments
- Content Type: How to interpret a file
  - File is a blob of bits (encoding)
  - How should we decode this blob into an (abstract) value? Colors, sounds, characters?
  - Recall: correspondence relation
- Syntax: type/subtype
  - text/plain, text/html, text/css, text/javascript
  - image/gif, image/png, image/jpeg
  - video/mpeg, video/quicktime
- Transfer encoding: A layered encoding
  - quoted-printable, base64
Example: Multiple Parts

MIME-Version: 1.0
Content-Type: multipart/mixed; boundary=aFrontierString

This is a message with multiple parts in MIME format.
--aFrontierString
Content-Type: text/plain

This is the body of the message.
--aFrontierString
Content-Type: application/octet-stream
Content-Transfer-Encoding: base64

PGh0bWw+CiAgPGhlYWQ+CiAgPC9oZWFkPgogIDxib2R5PgogICAgPHAgPC90aGU+VGhpcyBpcyB0aGUg
Ym9keSBvZiB0aGUgbWVzc2FnZS48L3A+CiAgPC9ib2R5Pgo8L2h0bWw+Cg==
--aFrontierString--
Example: Content Type

MIME-Version: 1.0
Content-Type: multipart/mixed; boundary=aFrontierString

This is a message with multiple parts in MIME format.
--aFrontierString
Content-Type: text/plain

This is the body of the message.
--aFrontierString
Content-Type: application/octet-stream
Content-Transfer-Encoding: base64

PGh0bWw+CiAgPGhlYWQ+CiAgPC9oZWFkPgogIDxib2R5Pgo8L2h0bWw+Cg==
--aFrontierString--
Example: Transfer Encoding

MIME-Version: 1.0
Content-Type: multipart/mixed; boundary=aFrontierString

This is a message with multiple parts in MIME format.
--aFrontierString
Content-Type: text/plain

This is the body of the message.
--aFrontierString
Content-Type: application/octet-stream
Content-Transfer-Encoding: base64

PGh0bWw+CiAgPGhlYWQ+CiAgPC9oZWFkPgogIDxib2R5PgogICAgPHA+VGhpcyBpcyB0aGUgYm9keSBvZiB0aGUgbWVzc2FnZS48L3A+CiAgPC9ib2R5Pgo8L2h0bWw+Cg==
--aFrontierString--
Layered Encoding

source
(image)

content
(bits)

encoded
(safe for channel)
ASCII

Content-Type
image/jpeg

Content-Transfer-Encoding
???

/9j/4AAQSk...

ffd8ffe000104a464946...
Encoding (Binary) Data in ASCII

- Binary data: Any byte value is possible
  - 00 to FF (i.e. xxxx xxxx)

- ASCII data: bytes start with 0
  - 00 to 7F (i.e. 0xxx xxxx)

- Problem: a channel that needs ASCII
  - Encoding must use ASCII alphabet

- Hex: every 4 bits becomes 1 character
  1101 0110 1100 1111 0011 1001
  D 6 A F 2 5

- Problem?
Quoted-Printable Encoding

- Observation: bytes that happen to be ASCII already, don't need to be encoded
  - If most data is text, savings are significant
- For each byte:
  - If first bit is 0, do nothing
  - If first bit is 1, encode with 3 bytes: \( =XY \) where XY is the hex value being encoded
- Limit line length to 76 characters
- Finish lines with "="
- Q: What if data contains the byte "="?
J'interdis aux marchands de vanter trop leur marchandises. Car ils se font vite pédagogues et t'enseignent comme but ce qui n'est par essence qu'un moyen, et te trompent ainsi sur la route à suivre les voilà bientôt qui te dégradent, car si leur musique est vulgaire ils te fabriquent pour te la vendre une âme vulgaire.
Encoding Binary Data

- What if most data is *not* ASCII?
  - Raw (base 256): 8 bits are a digit/char
    
    1101 0110 1100 1111 0010 0101
    
    ? ? %
  
  - Hex (base 16): 4 bits → digit/char/byte
    
    1101 0110 1100 1111 0011 1001
    
    D 6 A F 2 5
  
  - Quoted-Printable: 8 bits → 3 bytes
    
    1101 0110 1100 1111 0011 1001
    
    =D 6 =A F %
  
- Can we do better?
What if most data is not ASCII?

- **Raw (base 256):** 8 bits are a digit/char
  
  \[
  \begin{align*}
  1101 & 0110 1100 1111 0010 0101 \\
  \hline
  ? & ? & %
  \end{align*}
  \]

- **Hex (base 16):** 4 bits → digit/char/byte
  
  \[
  \begin{align*}
  1101 & 0110 1100 1111 0011 1001 \\
  \hline
  D & 6 & A & F & 2 & 5
  \end{align*}
  \]

- **Base 64:** 6 bits → digit/character/byte
  
  \[
  \begin{align*}
  1101 & 0110 1100 1111 0011 1001 \\
  \hline
  1 & s & 8 & 5
  \end{align*}
  \]
# Base64 Alphabet

<table>
<thead>
<tr>
<th>Value</th>
<th>Char</th>
<th>Value</th>
<th>Char</th>
<th>Value</th>
<th>Char</th>
<th>Value</th>
<th>Char</th>
<th>Value</th>
<th>Char</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>A</td>
<td>16</td>
<td>Q</td>
<td>32</td>
<td>g</td>
<td>48</td>
<td>w</td>
<td>49</td>
<td>x</td>
</tr>
<tr>
<td>1</td>
<td>B</td>
<td>17</td>
<td>R</td>
<td>33</td>
<td>h</td>
<td>49</td>
<td>x</td>
<td>50</td>
<td>y</td>
</tr>
<tr>
<td>2</td>
<td>C</td>
<td>18</td>
<td>S</td>
<td>34</td>
<td>i</td>
<td>50</td>
<td>y</td>
<td>51</td>
<td>z</td>
</tr>
<tr>
<td>3</td>
<td>D</td>
<td>19</td>
<td>T</td>
<td>35</td>
<td>j</td>
<td>51</td>
<td>z</td>
<td>52</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>E</td>
<td>20</td>
<td>U</td>
<td>36</td>
<td>k</td>
<td>52</td>
<td>0</td>
<td>53</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>F</td>
<td>21</td>
<td>V</td>
<td>37</td>
<td>l</td>
<td>53</td>
<td>1</td>
<td>54</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>G</td>
<td>22</td>
<td>W</td>
<td>38</td>
<td>m</td>
<td>54</td>
<td>2</td>
<td>55</td>
<td>3</td>
</tr>
<tr>
<td>7</td>
<td>H</td>
<td>23</td>
<td>X</td>
<td>39</td>
<td>n</td>
<td>55</td>
<td>3</td>
<td>56</td>
<td>4</td>
</tr>
<tr>
<td>8</td>
<td>I</td>
<td>24</td>
<td>Y</td>
<td>40</td>
<td>o</td>
<td>56</td>
<td>4</td>
<td>57</td>
<td>5</td>
</tr>
<tr>
<td>9</td>
<td>J</td>
<td>25</td>
<td>Z</td>
<td>41</td>
<td>p</td>
<td>57</td>
<td>5</td>
<td>58</td>
<td>6</td>
</tr>
<tr>
<td>10</td>
<td>K</td>
<td>26</td>
<td>a</td>
<td>42</td>
<td>q</td>
<td>58</td>
<td>6</td>
<td>59</td>
<td>7</td>
</tr>
<tr>
<td>11</td>
<td>L</td>
<td>27</td>
<td>b</td>
<td>43</td>
<td>r</td>
<td>59</td>
<td>7</td>
<td>60</td>
<td>8</td>
</tr>
<tr>
<td>12</td>
<td>M</td>
<td>28</td>
<td>c</td>
<td>44</td>
<td>s</td>
<td>60</td>
<td>8</td>
<td>61</td>
<td>9</td>
</tr>
<tr>
<td>13</td>
<td>N</td>
<td>29</td>
<td>d</td>
<td>45</td>
<td>t</td>
<td>61</td>
<td>9</td>
<td>62</td>
<td>+</td>
</tr>
<tr>
<td>14</td>
<td>O</td>
<td>30</td>
<td>e</td>
<td>46</td>
<td>u</td>
<td>62</td>
<td>+</td>
<td>63</td>
<td>/</td>
</tr>
</tbody>
</table>

[en.wikipedia.org/wiki/Base64](en.wikipedia.org/wiki/Base64)
Layered Encoding: Base64

source (image)

content (bits)

encoded (alphabet)

transmission (bits)

---

Content-Type image/jpeg

Content-Transfer-Encoding base64

ASCII
## Base64 Encoding

<table>
<thead>
<tr>
<th>source ASCII (if &lt;128)</th>
<th>M</th>
<th>a</th>
<th>n</th>
</tr>
</thead>
<tbody>
<tr>
<td>source octets</td>
<td>77 (0x4d)</td>
<td>97 (0x61)</td>
<td>110 (0x6e)</td>
</tr>
<tr>
<td>Bit pattern</td>
<td>0 1 0 0 1 1 0 1</td>
<td>0 1 1 0 0 0 0 1</td>
<td>0 1 1 0 1 1 1 0</td>
</tr>
<tr>
<td>Index</td>
<td>19</td>
<td>22</td>
<td>5</td>
</tr>
<tr>
<td>Base64-encoded</td>
<td>T</td>
<td>W</td>
<td>F</td>
</tr>
<tr>
<td>encoded octets</td>
<td>84 (0x54)</td>
<td>87 (0x57)</td>
<td>70 (0x46)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Text content</th>
<th>M</th>
<th>a</th>
<th>n</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASCII</td>
<td>77 (0x4d)</td>
<td>0 (0x00)</td>
<td>0 (0x00)</td>
</tr>
<tr>
<td>Bit pattern</td>
<td>0 1 0 0 1 1 0 1</td>
<td>0 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>Index</td>
<td>19</td>
<td>16</td>
<td>0</td>
</tr>
<tr>
<td>Base64-encoded</td>
<td>T</td>
<td>Q</td>
<td>=</td>
</tr>
</tbody>
</table>

[en.wikipedia.org/wiki/Base64](en.wikipedia.org/wiki/Base64)
Determining MIME Content Type

- The sender (web server) determines MIME (content) type of document being sent
  - Rules map file extensions to MIME types
- If file arrives without MIME info, receiver has to guess (see `file` command)
  - File extension may help
  - Contents may help: magic number at start
    - JPG: ff d8...
    - PDF: 25 50 44 46... (ie %PDF)
    - PNG: 89 50 4e 47 0d 0a 1a 0a... (ie .PNG...)
- 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
  - Layered encodings are possible too
To Ponder

Before the Internet
http://xkcd.com/1348/
HTTP: Hypertext Transfer Protocol
HTTP

- Hypertext Transfer Protocol
- History
  - Early 90's: developed at CERN, Tim Berners-Lee
  - 1996: version 1.0
  - 1999: version 1.1 (ubiquitous today!)
  - May 2015: version 2
    - Performance improvements: binary, server push...
    - Backwards compatible
  - Coming soon: version 3 (IETF draft)
    - More performance improvements, same semantics

w3techs.com/technologies/overview/site_element

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

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

- The header consists of
  1. Method/Status (for request/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:
  
  \textit{verb} path version

  - Verb: \texttt{GET}, \texttt{HEAD}, \texttt{POST}, \texttt{PUT}, \texttt{DELETE},...
  
  - Path: part of URL (path and query)
    
    \texttt{scheme://FQDN:port/path?query#fragment}
  
  - Version: \texttt{HTTP/1.1}, \texttt{HTTP/2}

- Example:
  
  - For URL
    
    \texttt{https://news.osu.edu/campus/}
  
  - First line of request is
    
    \texttt{GET /campus/ HTTP/1.1}
Request Header: Header Fields

- Each field on its own line, syntax:
  
  \textit{name: value}

- Examples (only Host field is required)
  
  \textbf{Host}: cse.ohio-state.edu
  \textbf{Accept}: text/*
  \textbf{Accept}: image/gif
  \textbf{If-Modified-Since}: Sat, 12 May 2021 19:43:31 GMT
  \textbf{Content-Length}: 349
  \textbf{User-Agent}: Mozilla/5.0 (X11; Linux x86_64) Chrome/93.0.4577.82 Safari/537.0

- Blank line indicates end of headers
Header Fields cont’d

- Host
  - Only required field
  - Q: Why is host field even 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

- User-Agent
  - Identifies application making request
Steiner, The New Yorker (1993)

"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
"Nobody knows you're a dog"

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

$ curl -A "Mozilla/5.0" news.osu.edu
Demo: HTTP Request with telnet

- Example URL
  - web.cse.ohio-state.edu/~sivilotti.1/

- At console
  - $ telnet web.cse.ohio-state.edu 80
  - Opens connection to port 80, where a web server is listening

- Send the following HTTP request:
  - GET /~sivilotti.1/ HTTP/1.1
  - Host: web.cse.ohio-state.edu
  - <blank line>
HTTP Traffic Transparency

- Everything is visible to an eavesdropper
  - HTTP headers are plain text
  - HTTP payload may be binary

- To protect communication, use encryption
  - SSL, TLS: protocols to create secure channel
  - Initial handshake between client and server
  - Subsequent communication is encrypted

- HTTP over secure channel = HTTPS
  - Default port: 443
Demo: HTTPS with openssl

- Use openssl instead of telnet
  - Negotiates initial handshake with server
  - Handles encryption/decryption of traffic

- Example URL
  - https://www.osu.edu/

- At console
  
  $ openssl s_client -connect www.osu.edu:443
  - Note connection to port 443 (ie https)

- Syntax of subsequent request is the same

- Send the following HTTP request:
  
  GET / HTTP/1.1
  Host: www.osu.edu

  <blank line>
HTTP Response Anatomy

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

- Parts 1-3 collectively called “the header”

- Part 1 (status line) syntax:
  
  \texttt{http-version status-code text}

- Examples
  
  \texttt{HTTP/1.1 200 OK}
  \texttt{HTTP/1.1 301 Moved Permanently}
  \texttt{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, 16 Sep 2019 14:51:38 GMT
   Server: Apache/2.4.6 (Red Hat)
   Content-Type: text/html; charset=UTF-8
   Content-Length: 333

☐ Blank line indicates end of headers
Demo: Using Terminal

- Use telnet to retrieve
  http://web.cse.ohio-state.edu/~paolo/
  - Fails (see status code)
  http://web.cse.ohio-state.edu/~sivilotti.1/
  - Body is incomplete (no images)
  - Body is chunked

- Use cURL to retrieve
  - Handles https, headers, redirection, chunking,…
  $ curl -Li http://web.cse.ohio-state.edu/~paolo
Demo: Using Chrome

- Developer > Network
- One GET results in many requests
  http://web.cse.ohio-state.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
  s = page.form_with action: /search/
  ```
Request Methods

- GET, HEAD
  - Request: should be *safe* (no side effects)
  - Request has header only (no body)

- 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
HTTP is Stateless

- Every request looks the same
- But maintaining state between requests is really useful:
  - User logs in, then can GET account info
  - Shopping cart “remembers” contents
- One solution: Keep a shared secret
  - Server's response contains a unique session identifier (a long random value)
  - Subsequent requests from this client include this secret value
  - Server recognizes the secret value, request must have come from original client
HTTP Session
HTTP Session

Request

Store secret

38afes7a8
HTTP Session

Request

Response
Secret: 38afes7a8

Store secret

38afes7a8
HTTP Session

Request

Response

Secret: 38afes7a8

Request id: 38afes7a8

Response

Request id: 38afes7a8

Response

Store secret

Check id

38afes7a8

Check id
HTTP Cookies

- Popular mechanism for session management
- Set in *response* header field
  - `Set-Cookie: session=38afes7a8`
  - Any name/value is ok
  - Options: expiry, require https
- Client then includes cookie(s) in any subsequent request to that domain
- Sent in *request* header field:
  - `Cookie: session=38afes7a8`
- Cookies also used for
  - Tracking/analytics: What path did they take?
  - Personalization
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 news articles
  https://news.osu.edu/
  ?
  q=Rhodes+Scholarship&search.x=1

- Random numbers ([link](https://random.org/)
  passwords/?
  num=5&len=8&format=plain

  - Demo: use FF Dev to edit/resubmit request or Chrome Dev to copy as cURL
  - See guidelines and [API](#) for http clients
Passing Arguments: POST

- Encoded as part of the request body

- Advantages:
  - Arbitrary length (URLs are limited)
  - Arguments not saved in browser history
  - Result not cached by browser
  - Slightly more secure (not really)
    - Args not in location bar, so less likely to be accidentally shared

- Content-Type indicates encoding used
  - application/x-www-form-urlencoded
    - Same encoding as used in GET
  - multipart/form-data
    - Better for binary data (else 1 byte → 3 bytes)
  - More options too:
    - application/xml, application/json, ...
Passing Args: GET vs POST

- GET
  ```
  GET /passwords/?num=5&len=8&format=plain
  HTTP/1.1
  Host: www.random.org
  ```

- POST
  ```
  POST /passwords/ HTTP/1.1
  Host: www.random.org
  Content-Type: application/x-www-form-urlencoded
  Content-Length: 24
  num=5&len=8&format=plain
  ```
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