67-328 Mobile to Cloud

Networking
Learning Goals

• Be comfortable with terminology used concerning the Internet
• Understand the role of protocols, and the layering of protocols, in the architecture of the Internet. And how this layering provides levels of abstraction below which a developer need not be (too) concerned.
• Understand the basic functionality of how packets of information travel between one system and another. This will inform design and configuration choices in building and maintaining systems.
Network Layered Architecture

• Important network “Layers”
  – Application (e.g. HTTP, IMAP)
    • E.g. A mail message from server to your laptop
  – Transport (TCP and UDP)
    • E.g. Reliable stream of data from one computer to another
  – Internet (Internet Protocol – IP)
    • 64K datagrams crossing multiple networks from one computer to another.
  – Data Link (e.g. Ethernet)
    • Frames of 1's and 0's passed on a single local network from one computer to another
  – Physical (e.g. fiber-optic cable, microwave, copper cable)
    • E.g. electrical signals on a wire, light pulses on a fiber, radio waves to a satellite.
Where are these networks?

- The cables are easy to see on campus, and in urban and suburban areas
- City to city terrestrial
  - Long distance cables often laid along railroad rights of way
  - http://www.youtube.com/watch?v=OtmKECNb78k
- Undersea:
  - http://www.youtube.com/watch?v=dW0Fp-bbKWI
Maps

• Undersea Cables:
  – http://www.submarinecablemap.com/
  – Where do they go?
  – ?Who owns them?

• Satellite Coverage:
  – o3b: http://www.o3bnetworks.com
    • and coverage map.
Microwave in Rural Alaska

Source: http://terra.gci.com/maps-locations/terra-vision-map

30 miles Nuiqsut to Oooguruk

Source: http://kundaliniandcelltowers.com/electrical-sensitivity.html
"Inter"-network addressing

• Assume you are an http request from your laptop to a server at 202.65.43.185

• How can you get from your browser to this server.
Route to 202.65.43.185

```
nut:- JoeMertz$ traceroute -I health.gov.ck
traceroute to health.gov.ck (202.65.43.185), 64 hops max, 72 byte packets
  1 pod-a-weh-v188.gw.cmu.net (128.2.80.3) 1.094 ms 0.652 ms 0.637 ms
  2 core255-vl919.gw.cmu.net (128.2.255.161) 0.934 ms 0.688 ms 0.677 ms
  3 pod-i-nh-vl987.gw.cmu.net (128.2.255.251) 0.929 ms 0.873 ms 0.823 ms
  4 ge-7-23.car1.pittsburgh3.level3.net (4.49.108.45) 1.174 ms 1.169 ms 1.114 ms
  5 ae-5-5.ebr1.washington1.level3.net (4.69.135.242) 17.289 ms 17.162 ms 17.947 ms
  6 ae-91-91.csw4.washington1.level3.net (4.69.134.142) 19.895 ms 16.998 ms 18.090 ms
  7 ae-4-99.edge2.washington4.level3.net (4.68.17.211) 7.646 ms 7.966 ms 7.682 ms
  8 if-8-0.icore1.aeq-ashburn.as6453.net (206.82.139.65) 12.407 ms 16.747 ms 18.806 ms
  9 if-0-0-0-459.core4.aeq-ashburn.as6453.net (216.6.42.21) 7.971 ms 8.485 ms 8.276 ms
 10 if-3-0-0-921.core1.cs8-chicago.as6453.net (66.110.27.53) 20.337 ms 22.613 ms 20.412 ms
 11 if-0-0-0-925.core1.s00-seattle.as6453.net (207.45.206.1) 91.743 ms 82.861 ms 81.951 ms
 12 if-5-0.core1.vcw-vancouver.as6453.net (207.45.196.54) 83.803 ms 83.718 ms 83.970 ms
 13 if-0-0.core1.lcn-lakecowichan.as6453.net (216.6.58.5) 86.043 ms 86.134 ms 86.489 ms
 14 if-5-0-0.bb2.lcn-lakecowichan.as6453.net (64.86.83.39) 89.394 ms 88.493 ms 87.112 ms
 15 ix-10-1-0.bb2.lcn-lakecowichan.as6453.net (64.86.84.2) 652.794 ms 605.698 ms 628.306 ms
 16 202.65.32.18 (202.65.32.18) 624.766 ms 642.546 ms 621.597 ms
 17 thealth.oyster.net.ck (202.65.43.185) 624.677 ms 740.438 ms 683.941 ms
```
Suppose H1 wants to send a message to H8.
H1 To H8

Protocol Layering

We will start at the lowest layer and move up
MAC address

• Each networking device has a MAC address
  – Media Access Control address
  – This has nothing to do with Apple, Inc.
• Fixed by manufacturer
  – (Can be spoofed, for good or by Eve/Mallory)
• My laptop has 3:
  – Ethernet: 60:fb:42:ff:fe:f8
  – Wifi: 60:fb:42:f8:b5:08
  – Bluetooth: 60:fb:42:72:08:4c
• My iPhone has 2:
  – Wifi
  – Bluetooth
Important distinction

• MAC addresses are used to pass messages around a single physical network segment
  – An ethernet segment is a network
  – A wifi access point forms a wireless network segment
  – A point-to-point microwave link is a network segment

• IP addresses are used to pass messages around between networks
  – I.e. between the wifi LAN in this room and google.com
  – IP == Internet Protocol
    • Internet meaning between networks
Where do IP addresses come from?

• So your MAC address is in hardware
• Where does your IP address come from?
  – Static – from a network administrator
  – Dynamic – Dynamic Host Configuration Protocol
To broadcast, send to MAC address: FF:FF:FF:FF:FF:FF

Source: http://docs.hp.com/en/B2355-90685/ch06s02.html
Network vs Internetwork

- An internetwork is an interconnected collection of networks.
- The Internet Protocol (IP) is the key tool used today to build scalable, heterogeneous internetworks.
Now let's move up a layer
To the Internet Protocol (IP) layer
IP packet layout

These must address every host on the planet!

(Some times we can cheat (see NAT))
32-bit IPv4 Addressing

IPv4 is the most common addressing today.
### Decimal representation of IP addresses

<table>
<thead>
<tr>
<th>Class</th>
<th>Range of addresses</th>
<th>Octet 1</th>
<th>Octet 2</th>
<th>Octet 3</th>
<th>Octet 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1.0.0.0 to 127.255.255.255</td>
<td>1 to 127</td>
<td>0 to 255</td>
<td>0 to 255</td>
<td>0 to 254</td>
</tr>
<tr>
<td>B</td>
<td>128.0.0.0 to 191.255.255.255</td>
<td>128 to 191</td>
<td>0 to 255</td>
<td>0 to 255</td>
<td>0 to 255</td>
</tr>
<tr>
<td>C</td>
<td>192.0.0.0 to 223.255.255.255</td>
<td>192 to 223</td>
<td>0 to 255</td>
<td>0 to 255</td>
<td>1 to 254</td>
</tr>
<tr>
<td>D (multicast)</td>
<td>224.0.0.0 to 239.255.255.255</td>
<td>224 to 239</td>
<td>0 to 255</td>
<td>0 to 255</td>
<td>1 to 254</td>
</tr>
<tr>
<td>E (reserved)</td>
<td>240.0.0.0 to 255.255.255.255</td>
<td>240 to 255</td>
<td>0 to 255</td>
<td>0 to 255</td>
<td>1 to 254</td>
</tr>
</tbody>
</table>

**E.g. IBM**

9.0.0.0-9.255.255.255

**E.g. CMU**

128.2.0.0-128.2.255.255

The Internet Assigned Numbers Authority allocates IP addresses
Every IP datagram contains the IP address of the destination host.

The “network part” of an IP address uniquely identifies a single physical network that is part of the larger Internet.

All hosts and routers that share the same network part of their address are connected to the same physical network and can thus communicate with each other by sending frames over the network, addressed with MAC addresses.

The network part of the IP address, therefore, helps a packet find the right physical network.

Every physical network that is part of the Internet has at least one router that, by definition, is also connected to at least one other physical network; this router can exchange packets with hosts or routers on either network.
Subnet Addressing

• You can take the bits that are not your network address, and use them to address your individual networks, and the hosts on each of those networks.

Source: http://en.wikipedia.org/wiki/Subnetwork

For a good supplemental reading on subnets, see the following through "Exercise 1"
Simple subnetting of 128.2.0.0 with subnet masks of 255.255.255.0
Similarly within CMU

- Network admins can use subnet masks to break 128.2.0.0 into multiple sub-networks (subnets).
  - A few top bits of the 3rd byte can be the subnet.
  - The remaining bits will be the host.
  - Note that this will not always fall only on “dot” boundaries.
Suppose H1 wants to send a message to H8.
Choose R1

H1 has the IP address of H8. Does H8 have the same network part address as my interface? No, so send to the router.

128.2.1.0

Router R1

128.2.2.0

Switch

H4

H5

128.2.3.0

Router R2

128.2.4.0

Router R3

H7

H8

H1

H2

H3

H6
The message is sent to R1.

R1 now has the packet with the IP address of H8. Does H8 have the same network part address as any of R1’s interfaces? No, so choose the best router R2.
R2 has the packet with the IP address of H8. Does H8 have the same network part address as any of my interfaces?
No, so choose the best router - R3.

The message is sent to R2.
R3 has the packet with the IP address of H8. Does H8 have the same network part address as any of R3’s interfaces? Yes, so send the packet to H8.
An application must know the _____ of H8 to send it a message.

H8 has a fixed _____ address, an assigned _____ address and its ______ address is the same as H7.

Each host on this network has the same IP ______ address and a different IP ______ address.

This interface has the same IP ______ address as H8.

These interfaces have the same ______ as H6.

These interfaces have the same IP network address because ______.___________
• How did R1 know R2 was the best path to H8?
• My diagrams only showed an obvious path, but the Internet looks like this:

Source: http://www.cheswick.com/ches/map/movie.mpeg
• Routers maintain routing tables, indicating the best paths to other networks.
# Initial Routing Tables for the Network

<table>
<thead>
<tr>
<th>Routings from A</th>
<th>Routings from B</th>
<th>Routings from C</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>To</strong></td>
<td><strong>Link</strong></td>
<td><strong>Cost</strong></td>
</tr>
<tr>
<td>A</td>
<td>local</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>C</td>
<td>-</td>
<td>$\infty$</td>
</tr>
<tr>
<td>D</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>E</td>
<td>-</td>
<td>$\infty$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Routings from D</th>
<th>Routings from E</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>To</strong></td>
<td><strong>Link</strong></td>
</tr>
<tr>
<td>A</td>
<td>3</td>
</tr>
<tr>
<td>B</td>
<td>-</td>
</tr>
<tr>
<td>C</td>
<td>-</td>
</tr>
<tr>
<td>D</td>
<td>local</td>
</tr>
<tr>
<td>E</td>
<td>6</td>
</tr>
</tbody>
</table>
RIP Routing Algorithm

**Fault:** If nothing is received on a link for a while, set cost to $\infty$ for each destination using that link and execute a send.

**Send:** Each $t$ seconds or when my local table changes, send my table on each non-faulty outgoing link.

**Receive:** Whenever a routing table is received on link $n$ from a neighboring router for each route (each row in the table) {
  if the route goes through me, ignore it, for I have better information than it does
  else if the route is to a destination not already in my table, add it to my table
  else compare the costs to the destination:
    if the received cost + 1 is less than my known route
      replace my route with the new route across link $n$, for it is a better route
      (To: destination, Link: $n$, Cost: received route cost + 1)
    if my known route to the destination begins over link $n$
      replace my cost to the destination with received cost + 1
      for the router over $n$ has better info (I go through it to the destination)
      (To: destination, Link: $n$, Cost: received route cost + 1)
**Update A With B**

<table>
<thead>
<tr>
<th>Routings from A</th>
<th>Routings from B</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>To</strong></td>
<td><strong>Link</strong></td>
</tr>
<tr>
<td>B</td>
<td>1</td>
</tr>
<tr>
<td>C</td>
<td>3</td>
</tr>
<tr>
<td>D</td>
<td>3</td>
</tr>
<tr>
<td>E</td>
<td>1</td>
</tr>
</tbody>
</table>

**Receive:** Whenever a routing table is received on link n from a neighboring router for each route (each row in the table) {
  if the route goes through me, ignore it, for I have better information than it does else if the route is to a destination not already in my table, add it to my table else compare the costs to the destination:
    if the received cost + 1 is less than my known route
      replace my route with the new route across link n, for it is a better route (To: destination, Link: n, Cost: received route cost + 1)
    if my known route to the destination begins over link n
      replace my cost to the destination with received cost + 1
      for the router over n has better info (I go through it to the destination) (To: destination, Link: n, Cost: received route cost + 1)
Receive: Whenever a routing table is received on link n from a neighboring router for each route (each row in the table) {
   if the route goes through me, ignore it, for I have better information than it does else if the route is to a destination not already in my table, add it to my table else compare the costs to the destination:
      if the received cost + 1 is less than my known route
         replace my route with the new route across link n, for it is a better route
         (To: destination, Link: n, Cost: received route cost + 1)
      if my known route to the destination begins over link n
         replace my cost to the destination with received cost + 1
         for the router over n has better info (I go through it to the destination)
         (To: destination, Link: n, Cost: received route cost + 1)
Types of Routing Algorithms

1. Vector-Distance Algorithms
   - RIP is the simplest example

2. Link-State Algorithms
   - Distribution and update of a DB at each node
   - Representing all or a substantial portion of an enterprises’ network
   - Each router has a map of the network, with which to calculate shortest path to each other node
   - Collection of shortest paths becomes the routing table
Network Abstractions

- IP – hosts communicate with routers and other hosts
- TCP & UDP – provides a socket and port abstraction across computers
- Application – application communicates with another application via a sockets connection on a port
Network Abstractions

- Similar with UDP
• But they don’t mix.
Transport Level Protocols

• Two common transport level protocols
  – UDP
  – TCP
• Both facilitate communication:
  – From a port at one IP address
  – To a port on another IP address
• The communication metaphor between the two ports is called a “socket”.
• E.g.
  – A UDP socket
  – from 128.2.25.11 port 5443
  – to 204.100.2.1 port 4568
  – Commonly noted 204.100.2.1:4568
• E.g.
  – A TCP socket
  – Between 128.2.25.22:4567
  – And 204.100.2.2:5432
Currently open sockets

netstat -f inet
Active Internet connections

<table>
<thead>
<tr>
<th>Proto</th>
<th>Recv-Q</th>
<th>Send-Q</th>
<th>Local Address</th>
<th>Foreign Address</th>
<th>(state)</th>
</tr>
</thead>
<tbody>
<tr>
<td>tcp4</td>
<td>0</td>
<td>0</td>
<td>192.168.1.106.61007</td>
<td>cyrus.andrew.cmu.imaps</td>
<td>ESTABLISHED</td>
</tr>
<tr>
<td>tcp4</td>
<td>0</td>
<td>0</td>
<td>192.168.1.106.60345</td>
<td>ool-4355d0ab.dyn.15641</td>
<td>ESTABLISHED</td>
</tr>
<tr>
<td>tcp4</td>
<td>0</td>
<td>0</td>
<td>192.168.1.106.60340</td>
<td>cyrus.andrew.cmu.imaps</td>
<td>ESTABLISHED</td>
</tr>
<tr>
<td>tcp4</td>
<td>0</td>
<td>0</td>
<td>192.168.1.106.60339</td>
<td>cyrus.andrew.cmu.imaps</td>
<td>ESTABLISHED</td>
</tr>
<tr>
<td>tcp4</td>
<td>0</td>
<td>0</td>
<td>192.168.1.106.60338</td>
<td>cyrus.andrew.cmu.imaps</td>
<td>ESTABLISHED</td>
</tr>
<tr>
<td>tcp4</td>
<td>0</td>
<td>0</td>
<td>192.168.1.106.60331</td>
<td>baymsg1010822.ga.msnp</td>
<td>ESTABLISHED</td>
</tr>
<tr>
<td>tcp4</td>
<td>0</td>
<td>0</td>
<td>192.168.1.106.60329</td>
<td>chatfarm-lm05b.e.aol</td>
<td>ESTABLISHED</td>
</tr>
<tr>
<td>tcp4</td>
<td>0</td>
<td>0</td>
<td>192.168.1.106.60328</td>
<td>oam-m11b.blue.ao.aol</td>
<td>ESTABLISHED</td>
</tr>
<tr>
<td>tcp4</td>
<td>0</td>
<td>0</td>
<td>192.168.1.106.60326</td>
<td>bos-d083b-rdr2.b.aol</td>
<td>ESTABLISHED</td>
</tr>
<tr>
<td>tcp4</td>
<td>0</td>
<td>0</td>
<td>192.168.1.106.60325</td>
<td>qw-in-f125.1e100.jabbe</td>
<td>ESTABLISHED</td>
</tr>
<tr>
<td>tcp4</td>
<td>0</td>
<td>0</td>
<td>192.168.1.106.60323</td>
<td>cs209p2.msg.ac4..mmcc</td>
<td>ESTABLISHED</td>
</tr>
</tbody>
</table>
## TCP & UDP Comparison

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>UDP</th>
<th>TCP</th>
</tr>
</thead>
<tbody>
<tr>
<td>General</td>
<td>Simple, high speed, low functionality wrapper that interfaces with IP</td>
<td>Full-featured protocol to reliably communicate data with another application across IP.</td>
</tr>
<tr>
<td>Connection</td>
<td>Connectionless, no setup</td>
<td>Connection-oriented; setup prior to transmission</td>
</tr>
<tr>
<td>Interface to Application</td>
<td>Discrete message based</td>
<td>Stream based</td>
</tr>
<tr>
<td>Reliability and ACK</td>
<td>No reliability guaranteed, best effort delivery without acknowledgement</td>
<td>Reliable, all messages acknowledged</td>
</tr>
<tr>
<td>Retransmissions</td>
<td>None. Application must detect if needed</td>
<td>All lost data retransmitted automatically</td>
</tr>
<tr>
<td>Flow control</td>
<td>None</td>
<td>Flow control on both ends (sliding window)</td>
</tr>
<tr>
<td>Overhead</td>
<td>Low</td>
<td>Low, but not as low as UDP</td>
</tr>
<tr>
<td>Speed</td>
<td>Very high</td>
<td>High, but not as high as UDP</td>
</tr>
<tr>
<td>Data quantity</td>
<td>Single datagram - up to 65K bytes</td>
<td>Small to very large (gigabytes)</td>
</tr>
<tr>
<td>Applicability</td>
<td>Speed matters more than completeness. Small discrete messages. <strong>Multicast or broadcast</strong></td>
<td>Data must be received reliably, in order.</td>
</tr>
</tbody>
</table>
Ports

- Port is a transport-layer software construct
- Port is a 16 bit integer
- 65,536 different Ports an application can communicate on
Official ports are assigned by IANA also
Ports

• 0 - 1023 Well Known Ports
  – Should not be used without IANA registration
  – Loosely, system-type-services ports

• 1024 - 49151 Registered Ports
  – Also should not be used without IANA registration
  – Loosely, application-level ports

• 49152 - 65535 Dynamic and/or Private Ports
  – Anything

• Readable list of port assignments:
Obvious simple example

• 80 - HTTP
• `http://cmu.edu` implies `http://cmu.edu:80`
• Can do `http://cmu.edu:12345`
  – But you probably won’t find http there.
Review – Most important overall view!

• Data Link Layer – e.g. Ethernet, PPP
  – Address: MAC
  – Communicates: frames (size depending on technology)
• Internet layer – Internet Protocol (IP)
  – Address: IP address
  – Communicates: 65K datagrams
• Transport layer – TCP or UDP
  – Address: port
  – Communicates:
    • TCP (streams, like a phone call)
    • UDP (<65K datagrams (they fit inside IP datagrams))
• Application layer - e.g. HTTP, SMTP, POP
  – Address: Depends on application
    • (e.g. @foo, joe@foo.com, foo.com/products/list)
  – Communicates: depends on application (tweet, mail, html page)
# Some Common Application Protocols

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Name</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP</td>
<td>Hypertext Transfer Protocol</td>
<td>Browser to server, web app to web service, etc.</td>
</tr>
<tr>
<td>HTTPS</td>
<td>HTTP Secure</td>
<td>Secure HTTP</td>
</tr>
<tr>
<td>DHCP</td>
<td>Dynamic Host Configuration Protocol</td>
<td>Server assigns IP address to a host</td>
</tr>
<tr>
<td>DNS</td>
<td>Domain Name System</td>
<td>Map domain name to IP address</td>
</tr>
<tr>
<td>RTP/RTCP</td>
<td>Real-time Transport Protocol / Real-time Transport Control Protocol</td>
<td>Delivers audio and video streams / statistics and control information about RTP stream</td>
</tr>
<tr>
<td>LDAP</td>
<td>Lightweight Directory Access Protocol</td>
<td>Querying and modifying directory services information</td>
</tr>
<tr>
<td>POP</td>
<td>Post Office Protocol</td>
<td>Retrieve email from a server</td>
</tr>
<tr>
<td>IMAP</td>
<td>Internet Message Access Protocol</td>
<td>Retrieve email from a server</td>
</tr>
<tr>
<td>SMTP</td>
<td>Simple Mail Transfer Protocol</td>
<td>Send mail from client to server, and send mail to/from server to server</td>
</tr>
<tr>
<td>SSH</td>
<td>Secure Shell</td>
<td>Terminal-type (shell) access to remote computers</td>
</tr>
</tbody>
</table>
HTTP Request

<table>
<thead>
<tr>
<th>General Format</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;method&gt; &lt;resource identifier&gt; &lt;HTTP Version&gt; &lt;crlf&gt;</code></td>
<td><code>GET /course/67-328/ HTTP/1.1</code></td>
</tr>
<tr>
<td><code>[&lt;Header&gt;: &lt;value&gt;] &lt;crlf&gt;</code></td>
<td><code>Host: www.andrew.cmu.edu</code></td>
</tr>
<tr>
<td><code>...</code></td>
<td><code>User-Agent: Joe typing</code></td>
</tr>
<tr>
<td><code>[&lt;Header&gt;: &lt;value&gt;] &lt;crlf&gt;</code></td>
<td><code>Accept: text/html</code></td>
</tr>
<tr>
<td><code>a blank line</code></td>
<td><code>This line intentionally left blank</code></td>
</tr>
<tr>
<td><code>[entity body]</code></td>
<td></td>
</tr>
</tbody>
</table>

- **Method**
  - GET, PUT, DELETE, HEAD, POST, etc.

- **Resource identifier** specifies the name of the target resource;
  - i.e. it's the URL stripped of the protocol and the server domain name.
  - When using the GET method,
    - this field will also contain a series of name=value pairs separated by ‘&’.
  - When using a POST method,
    - the entity body contains these pairs.

- **HTTP version** identifies the protocol used by the client.
## HTTP Response

<table>
<thead>
<tr>
<th>General Format</th>
<th>Example</th>
</tr>
</thead>
</table>
| `<HTTP Version> <Status> <crlf>`  
  
  `[
  `<Header>`: `<value>`
  ] <crlf>`  
  
  `[a blank line]`
  
  `[response body]` | HTTP/1.1 200 OK  
  Date: Mon, 24 Jan 2011 15:43:08 GMT  
  Server: Apache/1.3.39 (Unix) mod_throttle/3.1.2 ...  
  Set-Cookie: webstats-cmu=cmu128.2.87.50.8400; ...  
  Last-Modified: Sun, 23 Jan 2011 21:46:30 GMT  
  Accept-Ranges: bytes  
  Content-Length: 9014  
  Content-Type: text/html  
  This line intentionally left blank  
  `<HTML>`  
  `<HEAD>`  
  `<META http-equiv="Content-Type" content="text/html; charset=UTF-8">`  
  ... |

- **HTTP version** identifies the protocol used by the client.
- **Status** indicates the result of the request
Some Common HTTP Status Codes

• 301 Moved Permanently
• 400 Bad Request
• 401 Unauthorized
• 404 Not found
• 500 Internal Server Error
  – You get this when your server program throws an uncaught exception.
  – You are likely to see this frequently this semester!

• For more information, see the standard:
  – http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html
HTTP Request with a Query String

GET /alflickrbet?letter=e HTTP/1.1
Host: newalflickrbet1.appspot.com
User-Agent: Joe typing
Accept: text/html
HTTP Response Example

HTTP/1.1 200 OK
Content-Type: text/html; charset=utf-8
Vary: Accept-Encoding
Date: Mon, 23 Jan 2012 02:15:23 GMT
Server: Google Frontend
Cache-Control: private
Transfer-Encoding: chunked

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
  <title>alFLICKRbet</title>
</head>
<body>
  <h1>e is for elephant</h1>
  <br/>
  (Source: cached in datastore )
  <a href=http://www.flickr.com/67955941@N00/3885234912 >
  <img src=http://farm3.static.flickr.com/2488/3885234912_5278906961_z.jpg>
  </a><form action="alfllickrbet">
  <br/>
  <label for="letter">Type the letter you would like to play with next:</label>
  <input type="text" name="letter" value="" />
  <input type="submit" value="Submit" />
  </form>
</body>
</html>

• If response length is not known, chunks of data are sent
  • The size of the chunk is given, followed by that # of bytes
  • 0x2d6 = 726 (I edited out some whitespace for the slide)
  • There could be any number of chunks
  • End of chunks is indicated by length 0
  • Why is the length not known in this case?
Show example

- Show in Chrome
- Show via telnet
- Show in curl
  - MacOS: included
  - Windows: http://curl.haxx.se/download.html#Win32
- REST Console for Chrome
  - https://chrome.google.com/webstore/detail/rest-console/cokgbflfommojglbmbpenpphppikmonn
The genesis of the WWW

• Led by Sir Tim Burners Lee

• Defined 3 standards:
  1. Uniform Resource Identifiers (URI)
     • UR Locators (URL) – e.g. http://cmu.edu/dietrich
        – scheme://domain:port/path?query_string#fragment_id
     • UR Names (URN) e.g. urn:isbn:0132143011
        – urn:namespace_identifier:namespace_specific_string
  2. Hypertext Transfer Protocol (HTTP)
  3. Hypertext Markup Language (HTML)

• Then implemented to these standards:
  – A server program
  – A browser program
Who maintains the standards?

- The Internet Engineering Task Force (IETF) of the Internet Society, in cooperation with the World Wide Web Consortium (W3C)
- Published at http://www.w3.org/Protocols/

Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing

Abstract

The Hypertext Transfer Protocol (HTTP) is a stateless application-level protocol for distributed, collaborative, hypertext information systems. This document provides an overview of HTTP architecture and its associated terminology, defines the "http" and "https" Uniform Resource Identifier (URI) schemes, defines the HTTP/1.1 message syntax and parsing requirements, and describes related security concerns for implementations.
HTTP Methods

• Most frequently used:
  – GET – retrieve information
  – HEAD – query if there is information to retrieve
  – POST – append or modify information
  – PUT – add information
  – DELETE – delete information

• When choosing which HTTP methods to implement, consider:
  – Safety
  – Idempotence

• And when/how to use each
HTTP Methods in HTML 4 & 5

- HTTP defines at least 8 methods
- HTML 4 & 5 require only 2: GET and POST
- Therefore, simple HTML form submission only uses GET or POST
- The full set of HTTP methods are available (e.g. by JavaScript), however.
A safe request is one that does not change the state of the resource on the server
  – Except for trivial changes such as logging, caching, or incrementing a visit counter.

E.g. GET should be safe
  – GET should only retrieve information from a source
  – GET should not update that resource

GET *could* be used to update a resource, but as the HTTP protocol is defined, it *should not* be used in that way.
Head

• Another **safe** HTTP Method
• Requests headers only, but not content
• “If I were to GET this
  – Does it exist?
  – What would be its size?”
• Why might you want to use Head?
Idempotence

• Multiple identical requests are the same as a single request
  – Analog in math: taking the absolute value of a number.

• Nice to know on a potentially unreliable Internet:
  – If the first request didn’t work, just try again.
  – If the Google search never returned, just reload
Idempotent Example

- http://www.andrew.cmu.edu/course/67-328/
- HTTP GET

Repeat operation will get not change result
  - E.g. Repeated GET of course syllabus will not change the syllabus.
  - The resource may change on its own, so repeating might not get the same response.
    - But it would not be the GET which is changing the resource.
  - E.g. cnn.com is frequently updated, but not as a result of the GET.
Idempotent Methods

• Safe methods are idempotent
  – Because you are not changing the resource
  – Therefore GET and HEAD are idempotent
• PUT and DELETE are also idempotent
  – PUT adds or wholly replaces a resource
    • Do this a second time identically, you still have the same results.
  – DELETE removes a resource
    • DELETE a resource for the second time does not change its state.
Failure Models

• If you adhere to implementing GET, PUT, and DELETE as idempotent, then your failure model becomes easy:
  – If request fails, try it again.
Non-idempotent example:

- Create a new assignment on Blackboard
- https://blackboard.andrew.cmu.edu/webapps/blackboard/assignments/instructor/proc_edit_assignment.jsp?course_id=_1088365_1&content_id=_349041_1
- Uses HTTP POST
- Repeating operation will add another assignment.
POST

• POST is neither safe nor idempotent
  – POST changes the resource
  – If you POST twice, then the results can be different after each change.

• POST should be reserved for modifying or adding to (not replacing), an existing resource.

• You have seen POST sometimes when you hit “Purchase” on a web store.
  – If you hit “Purchase” again, or reload, you may be charged twice.
What method should be used?

- To create a calendar event?
- To retrieve a database record?
- To delete a calendar event?
- Add links to participants on a calendar event?
<table>
<thead>
<tr>
<th>Method</th>
<th>Purpose?</th>
<th>Safe?</th>
<th>Idempotent?</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PUT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DELETE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HEAD</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>POST</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Method</td>
<td>Purpose?</td>
<td>Safe?</td>
<td>Idempotent?</td>
</tr>
<tr>
<td>--------</td>
<td>---------------------------</td>
<td>-------</td>
<td>-------------</td>
</tr>
<tr>
<td>GET</td>
<td>Retrieve a resource</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>PUT</td>
<td>Insert or replace a resource</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>DELETE</td>
<td>Remove a resource</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>HEAD</td>
<td>Get header information only of a resource</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>POST</td>
<td>Append to or modify a resource</td>
<td>N</td>
<td>N</td>
</tr>
</tbody>
</table>
Misuse

• Misuse is common
• E.g. Flickr uses GET to do everything, including removing a photo from your favorites list
  – http://api.flickr.com/services/rest/?method=flickr.favorites.remove&...
Desirable API URL characteristics

Design goal: You want to expose straightforward, reliable, direct access to as much of your resource (web site) as possible.

• **Addressability**
  – Interesting aspects of your service are immediately accessible by a unique URI.

• **Granularity**
  – How fine-grained are the interesting aspects of your service addressable?

• **Transparency**
  – The meaning and context of a path are easily apparent to those who did not design the site

• **Persistence**
  – Addresses of resources on your site do not change over time.

• **Versioning**
  – Include API version in the URL to facilitate introduction of updates

Source: *Restful Web Services* by Leonard Richardson and Sam Ruby (O’Reilly Media, 2007).