ICS 351: Today's plan

- IPv6
  - review and more details
- HTML
- HTTP
- web scripting languages
IPv6 uses 128-bit addresses instead of the 32-bit IPv4 addresses.

These are written as 8 groups of 4 hex digits separated by colons:

```
1234:5678:0000:0000:0000:0008:9ABC:DEF0
```

Leading zeros may be omitted:

```
1234:5678:0:0:0:8:9ABC:DEF0
```

A single sequence of all-zero groups can be omitted:

```
1234:5678::8:9ABC:DEF0
```

Networks are followed by a slash to indicate the number of bits in the network number:

```
1234:5678/32
```
Specific IPv6 addresses

- the loopback address is ::1
- an interface with a MAC address automatically has a non-routable IPv6 address: fe80::48 bits of MAC address+16 inserted bits
  - for example, with a hardware address of 00:01:03:a0:31:51, my non-routable IPv6 address will be fe80::201:3ff:fea0:3151 -- note the "u" bit is set to one to indicate universal scope
- globally routable unicast addresses have a network and subnetwork number in the most significant 64 bits
- ff00::/8 addresses are multicast addresses
the IPv6 header is twice as big as the (minimal) IPv4 header, but simpler (from RFC 2460):
IPv6 details

- instead of IP header options, there may be extension headers
- fragmentation is only done by the sender, and path MTU discovery is required
- upper layer is now required to checksum.
- IPv6 routing is essentially the same as IPv4 routing (perhaps minus netmasks)
- when sent over Ethernet, the Ethertype field is 0x86DD instead of 0x800. (RFC 2464)
- Neighbor Discovery Protocol (NDP, RFC 2461) replaces both ARP and DHCP, uses IPv6 packets
HTML

- HyperText Markup Language
- an in-line way of marking (hyper)text, similar in spirit to TeX/LaTeX, and inspiring the creation of XML
- part of the markings are about style and formatting: font, size, bold/italic, bullet lists, etc.
- some markings lead you to other pages or objects, e.g.
  <a href="http://www2.hawaii.edu/~esb/">home page</a>, or
  <img src="http://www2.hawaii.edu/~esb/pix/2009silts.jpg">
- objects are identified by URLs (all URLs are also URIs)
- each URL has a protocol (scheme name, e.g. http), a host identifier (DNS name or IP address), an optional port number (:80 if not specified), and the path given to the server
client is given a URL, splits it into domain name (port) and path
client resolves domain name to IP address
client opens a connection to the IP address (port 80, or the given port), server accepts connection (TCP 3-way handshake)
client sends HTTP request
server sends HTTP response
after parsing response and finding embedded images or other content, client sends new HTTP requests on same TCP connection
server replies to each request in sequence
client matches each response to its request, renders the page
after a time (typically 30s), the server closes the connection
HTTP request header

- all HTTP is rendered using ASCII. This makes it easy to read, a little harder to parse
- for example, an HTTP request might look like this:

  ```
  GET /~esb/ HTTP/1.1
  Host: www2.ics.hawaii.edu
  Accept: */*
  Connection: close
  ```
a corresponding HTTP reply might look like this:

HTTP/1.1 200 OK
Date: Thu, 19 Nov 2009 05:18:56 GMT
Server: Apache
Last-Modified: Wed, 02 Sep 2009 03:17:30 GMT
ETag: "19abf-2095-4728fb5090680"
Accept-Ranges: bytes
Content-Length: 8341
Connection: close
Content-Type: text/html

<html>
...

HTTP headers

- in each case, the first line describes the main request or result:
  - in the request, the method can be GET, HEAD, POST, or a few others,
  - the path is specified immediately after the request,
  - the protocol version follows the path
  - in the reply, the version comes first, followed by the result code, both as a number and as a string
- the remaining lines of the header give more details, sometimes essential details (e.g. the content type and content length)
- each header ends with an empty line
web scripting languages

- web content described by HTML was originally static, corresponding to files on the server
- since the server is a program, it can generate content that is generated dynamically, e.g. put the user's name (or bank balance) within the web page
- however, this requires the server administrator to modify the code of the server, which is error-prone
- so instead, the server program can execute a server-side script to generate new content to be served
- this script can be written in any language supported by the system on which the server is running
client-side scripts

- even with a server-side script, each change in the web page requires an HTTP request and reply, and requires that the page be rendered again
- and usually requires an explicit user action such as a mouse click
- to have more interactivity, many browsers have been designed to execute *client-side scripts* that can modify the displayed page and exchange data over the internet
- client-side scripts are usually in Java or Javascript
client-side scripts and security

- While client-side scripts do much to improve the appearance of pages, there can be concerns about security and reliability.
- Client-side scripts let servers execute code on a client – how does the client know what the code will do? Can the client trust the server?
- In an attempt to address these concerns, browsers limit what scripts are allowed to do.
- Not all browsers execute client-side scripts.
server-side scripts and security

- bugs in a server-side script can be exploited by attackers
- server-side scripts that do not thoroughly check their input are vulnerable, e.g. to SQL injection attacks
  
  http://xkcd.com/327/

- a server-side script lets the client execute code on the server
- the server controls what scripts are available, but not what the clients will do with the scripts