Terminology

- A computer network is a set of interconnected computers
  - communication (send information to others, download from a company website, ...)
  - shared resources (backups, parallel computing, “the global grid”)
- A local area network (LAN) is a small group of systems, typically in one building
- A wide area network (WAN) can be very large, e.g. cover the whole US
- An internetwork (internet) is a collection of two or more networks with interfaces that transfer traffic between the subnets

History

- One of the first successful WANs was ARPANET
  - ARPA was a US DoD research organization
- Other large WANs of that period:
  - NSFNET
  - ESNET
  - Usenet

Protocols

- For a set of machines to exchange information they need to agree in advance on a communication protocol
- A protocol defines
  - the message format (size, encoding, structure, ...)
  - address information (sender and receiver)
  - a sequence of operations
    - “hello, are you ready to accept a message?”
    - “yes, go ahead”
    - “here is page 1 of 13”
    - “got it”
    - “here is page 2 of 13”
    - “...”
    - “sorry, I didn’t get that”
    - “here is page 5 again”
    - “...”
    - “that’s the last page of my message”
Protocol Stacks

- Communication typically involves a layer of cooperating protocols
  - operations at one layer are implemented by operations at a lower layer

Internet Protocols

- The Internet is based on a four-layer set of protocols commonly known as **TCP/IP**
  - TCP = transmission control protocol
  - IP = internet protocol

  The Internet (note the capital “I”) is loosely defined to be the collection of all systems that communicate via TCP/IP

- TCP and IP were first used in ARPANET, became a de facto standard for moving traffic to ESNET and other WANs

  For more info: “internet protocol suite” at Wikipedia

Internet Protocols

- The **application layer** defines interactions between user programs
  - HTTP (hypertext transfer protocol)
  - SMTP (simple mail transfer protocol)
  - FTP (file transfer protocol)
  - many others

Internet Protocols

- The **transport layer** protocols prepare messages for transmission
  - TCP
    - breaks messages into packets
    - attaches address information
    - adds checksums and other descriptions

  The TCP software on the receiver
  - checks for errors, and requests a retry if necessary
  - makes sure all packets are delivered (e.g. request a retry if a packet is lost)
  - reassembles the message from packets and passes it to the application

  Other transport protocols deal with streaming media, voice over IP (VOIP), ...
Internet Protocols

- Network layer protocols deal with routing between nodes in a network
  - breaks transport packets into smaller fixed size packets
  - nodes maintain tables to manage network traffic
  - may send several packets at once, over different routes

- Current standard: IPv4
  - 32-bit addresses
  - IPv6 will (some day) replace it

IP Addresses

- TCP/IP requires each system on the Internet to have a unique address
  - an IP address is a 32-bit number (IPv6 has 128-bit addresses)
  - usually written using “dotted decimal” notation
  - break the 32 bits into 4 8-bit pieces, write the equivalent decimal number for each segment
  - 8 bits => 0..255

Names

- To domain name system (DNS) assigns names to systems
  - top level domains have names like .edu, .gov, .com
  - an organization can register a new name through ICANN (e.g. faber.edu)
  - the organization assigns local names (cs.faber.edu, rotc.faber.edu, ...)
  - network administrators assign names within their local networks (www.cs.faber.edu)
Name Servers

- There are a lot of computers on the internet
  - not likely that each system knows the names of all the others
  - when a new system is installed, it registers its name with a **name server**
  - name servers share their information with other servers
  - a system will contact its local name server when it needs to look up a new name

Dynamic IP Addresses

- The dynamic host configuration protocol (DHCP) can assign IP addresses dynamically
  - some machines (e.g. teleost, fugu, wolferine) have permanent IP addresses
  - when *fintan* (my laptop) is in my office it gets a dynamic address from UO Secure
  - when *fintan* is at home, it gets a new IP address from a wireless router, which in turn gets its address from Comcast

Ports

- Operating systems use an abstraction known as a **port** to manage network communication
  - ports are implemented in software, inside the OS
  - when a message arrives at a host it includes a port number to identify the application layer protocol that will handle the message
  - the OS passes the message on to the application that is “listening” on the port

Clients and Servers

- Most internet communication is organized around **clients** and **servers**
  - a client initiates a conversation by sending a message to a server
  - the client knows the server's IP address
  - example: the web server at **www.cs.uoregon.edu** (128.223.4.25) is listening on port 80

- The server is a program that is always running
  - many servers are started by the system administrator when the OS starts (e.g. mail servers, web servers)
  - the standard application level protocols that are part of every TCP/IP implementation (smtp, ftp, etc) have predefined port numbers
  - users can start their own server software (e.g. database server) on other ports
  - may need to check first to see a port isn’t already being used
**Daemons**

- The OS/X Activity Monitor application shows O/S jobs as well as user jobs.
- In Unix an O/S job that listens for traffic on a port is a **daemon**
  - `httpd` is the web server (responds to incoming HTTP requests).
  - `ntpd` is the "network time protocol daemon".

* pronounced "demon"

---

**The World Wide Web**

- The term **hypertext** refers to a piece of text that includes links to other text.
- Ideas for cross-referencing text have been around for a long time.
- Adding links between text files on computers was an active research topic in the 1970s.
- Apple released an application called HyperCard in 1987.
  - Documents were known as "stacks".
  - A "card" was a single page in a stack.
- Sound, images or pieces of text could be links to other cards.
- Example: a stack on Beethoven's Ninth Symphony.

---

**URL**

- Hypertext links on the world wide web are **uniform resource locators** (URLs).
  - The first part identifies a **scheme** for locating the resource.
  - The remaining text depends on the scheme.
  - The scheme is not always a protocol (e.g., file:// means the resource is a local file).

Examples from Perkovic text:

```plaintext
http://www.w3.org/Consortium/mission.html
```

- Scheme
- Host
- Pathname

Other examples:

- `https://webmail.cdm.depaul.edu/`
- `ftp://ftp.server.net/`
- `mailto:lperkovic@cs.depaul.edu`
- `file:///Users/lperkovic/`

*Note: pathname format varies, depends on scheme.*
Web Servers

- HTTP interactions use a client-server model
  - a user runs a client program (a browser)
  - the client sends requests to a server
- The server is an application running on the host specified by the DNS portion of the URL
- By default the request is sent to port 80 on the server
- If a user knows a server is listening on another port the port number can be included with the URL
  - if I’m running my own special purpose server I can tell it to listen on port 8000
  - to get information from this server I would use a URL with the port number: 
    http://fugu.cs.uoregon.edu:8000/index.html

Web Browsers

- The first web browsers had a command line interface
  - text-only browsers are still in use (e.g. lynx, curl)
- A program named Mosaic (1993) was the first browser to use a graphical user interface
  - images linked with a document were displayed along with the document
  - was a free download from NCSA (Univ of Illinois)
- Mosaic sparked the growth of WWW and the Internet

<table>
<thead>
<tr>
<th>Date</th>
<th># Web</th>
<th># Hosts</th>
<th>% .com</th>
</tr>
</thead>
<tbody>
<tr>
<td>6/93</td>
<td>130</td>
<td>400,000</td>
<td>1.5</td>
</tr>
<tr>
<td>1/94</td>
<td>3,000</td>
<td>600,000</td>
<td>4.6</td>
</tr>
<tr>
<td>1/95</td>
<td>23,500</td>
<td>1,000,000</td>
<td>18.3</td>
</tr>
<tr>
<td>1/96</td>
<td>100,000</td>
<td>1,700,000</td>
<td>50</td>
</tr>
<tr>
<td>1/97</td>
<td>650,000</td>
<td></td>
<td>62.6</td>
</tr>
</tbody>
</table>

"host" means "system with an IP address and DNS name"

HTML

- The messages passed between HTTP clients and servers are text files
- How does a browser figure out how to display a page?
  - how does it know which parts are plain text and which are links?
  - how does it know which pieces of text are titles (to be displayed in large bold letters)?
  - where do the images come from?
  - how does the browser know to put some images on the upper left and embed others in the text?

A browser expects the documents sent by the server to be formatted in HTML

- HTML = hypertext markup language
- A "markup language" has special symbols to indicate formatting commands
  - example: I use a language named LaTeX when I write technical papers
  - the $ symbol marks the beginning and end of an equation
  - a backslash marks the beginning of a markup command

output produced by pdflatex
HTML

- The special characters that mark the beginning and end of markup commands in HTML are `<` and `>`
  - `<b>xxx</b>` means “display xxx in bold”
  - `<!--xxx-->` means “xxx is an anchor (hyperlink)”

Anchors

- Markup commands can have arguments
- Inside an anchor `href = "http://xxx"` means “when this text is clicked have the browser send a request for the URL `http://xxx`”

Python Networking Library

- Python has several modules for writing both clients and servers
- Our projects in the next week(s):
  - fetch, analyze HTML documents (“web crawler” described in Perkovic text)
  - use REST web service to fetch, analyze XML documents

HTML: “human web”

XML: “programmable web”

Example (from Perkovic)

- Use the `urllib` module to fetch a web page
  ```python
  >>> from urllib.request import *
  >>> response = urlopen('http://www.w3.org/Consortium/facts.html')
  >>> for h in response.getheaders() ...
  ... print(h)
  >>> doc = response.read()
  ```

See demos/HTTP/urllib.py