CS144 – Introduction to Computer Networking

Instructors: Philip Levis and David Mazières

CAs: Juan Batiz-Benet, Behram Mistree, Hariny Murli, Matt Sparks, and Tony Wu

Section Leader: Aki Kobashi

cs144-staff@scs.stanford.edu

http://cs144.scs.stanford.edu/

Networks class

- **Goal: Teach the concepts underlying networks**
  - How do networks work? What can one do with them?
  - Give you a basic understanding of the Internet
  - Give you experience using and writing protocols
  - Give you tools to understand new protocols & applications
- **Prerequisites:**
  - CS110 or equiv; class assumes you are comfortable with C and gdb, some socket programming helpful (e.g., CS110 web server)

Administrivia

- **All assignments are on the web page**
- **Text: Kurose & Ross, *Computer Networking: A Top-Down Approach, 4th or 5th edition***
  - Instructors working from 4th edition, either OK
  - Don't need lab manual or Ethereal (used book OK)
- **Syllabus on web page**
  - Gives which textbook chapters correspond to lectures (Lectures and book topics will mostly overlap)
  - Extra (not required) questions for further understanding
  - Papers sometimes, to make concepts more concrete (Read the papers before class for discussion)
  - Subject to change! (Reload before checking assignments)

Online Resources

- **We are using CourseWare as a more user-friendly web page**
  - It's a copy of the content on http://cs144.stanford.edu
- **We are trying a new web tool, Piazzza: please send all assignment questions there**
  - Link on http://cs144.stanford.edu
  - Piazzza allows you to answer questions and rate answers
- **Send all staff communication to `cs144-staff` list**
  - Goes to whole staff, so first available person can respond
  - CCing list ensures we give students consistent information
  - Also, some of us get lots of email…much easier for us to prioritize a specific mailing list

Grading

- **Exams: Midterm & Final**
- **Homework**
  - 5 lab assignments implemented in C
  - If you are not comfortable with C and gdb they will be painful
- **Grading**
  - Exam grade = $\max(\text{final}, (\text{final} + \text{midterm})/2)$
  - Final grade will be computed as:

$$\frac{max(2 \cdot exam + lab, exam + 2 \cdot lab)}{3}$$

- **Administrative handout has more details, please feel free to ask questions**

Labs

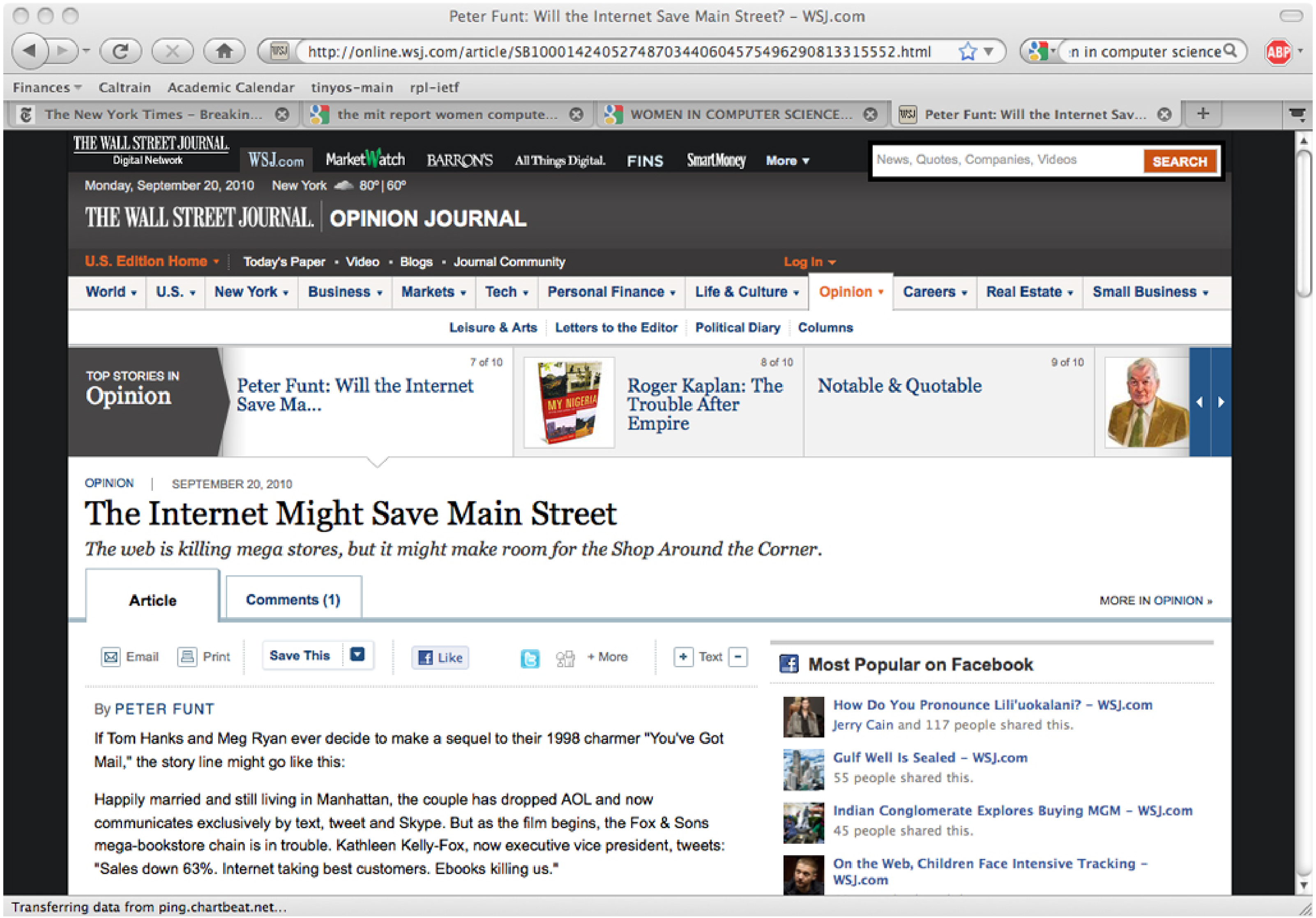
- **Labs are due by the beginning of class**
  - Lab 1: Stop & wait
  - Lab 2: Reliable transport
  - Lab 3: Static routing
  - Lab 4: Dynamic routing
  - Lab 5: NAT
- **All assignments due at start of Thursday lecture**
  - Late policy: can turn in late until 11:59PM that Saturday, grade capped at 90%

Section

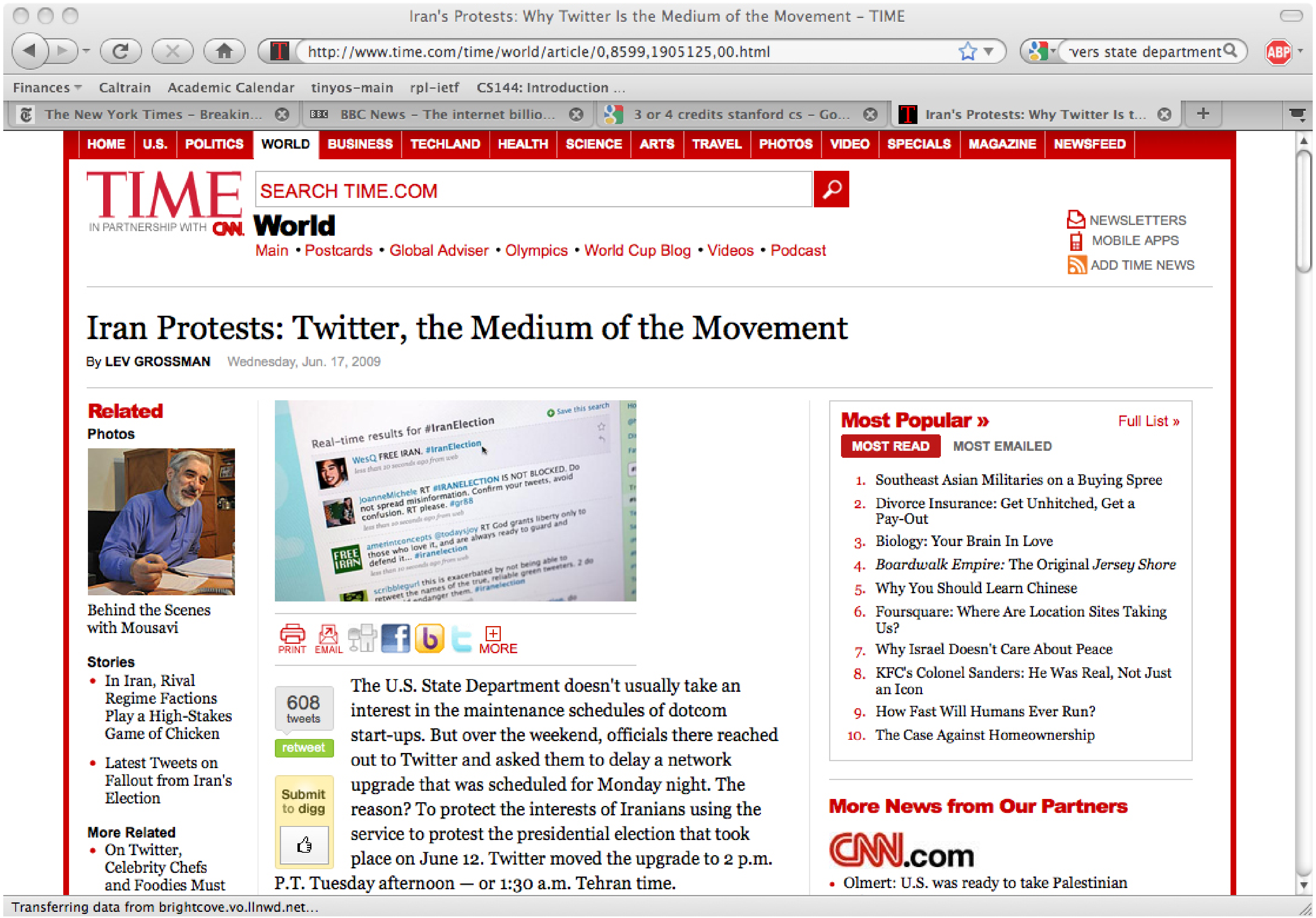
- Friday 11-11:50, Huang 108
- Led by TAs and section leaders
- Practical help with assignments, going over example exam problems
- This week: gdb tutorial

Why You Should Care About the Internet

Societal Change



Political Change



Iran's Protests: Why Twitter Is the Medium of the Movement – TIME

http://www.time.com/time/world/article/0,8599,1905125,00.html

Finances Caltrain Academic Calendar tinyos-main rpl-ietf CS144: Introduction ...

The New York Times – Breakin... BBC News – The internet billio... 3 or 4 credits stanford cs – Go... Iran's Protests: Why Twitter Is t...

HOME U.S. POLITICS WORLD BUSINESS TECHLAND HEALTH SCIENCE ARTS TRAVEL PHOTOS VIDEO SPECIALS MAGAZINE NEWSFEED

TIME IN PARTNERSHIP WITH CNN

SEARCH TIME.COM

World

Main • Postcards • Global Adviser • Olympics • World Cup Blog • Videos • Podcast

NEWSLETTERS
MOBILE APPS
ADD TIME NEWS

Iran Protests: Twitter, the Medium of the Movement

By LEV GROSSMAN Wednesday, Jun. 17, 2009

Related

Photos

Behind the Scenes with Mousavi

Stories

- In Iran, Rival Regime Factions Play a High-Stakes Game of Chicken
- Latest Tweets on Fallout from Iran's Election

More Related

- On Twitter, Celebrity Chefs and Foodies Must

Real-time results for #IranElection

PRINT EMAIL MORE

608 tweets retweet

Submit to digg

The U.S. State Department doesn't usually take an interest in the maintenance schedules of dotcom start-ups. But over the weekend, officials there reached out to Twitter and asked them to delay a network upgrade that was scheduled for Monday night. The reason? To protect the interests of Iranians using the service to protest the presidential election that took place on June 12. Twitter moved the upgrade to 2 p.m. P.T. Tuesday afternoon — or 1:30 a.m. Tehran time.

Most Popular » Full List »

MOST READ MOST EMAILED

1. Southeast Asian Militaries on a Buying Spree
2. Divorce Insurance: Get Unhitched, Get a Pay-Out
3. Biology: Your Brain In Love
4. *Boardwalk Empire:* The Original *Jersey Shore*
5. Why You Should Learn Chinese
6. Foursquare: Where Are Location Sites Taking Us?
7. Why Israel Doesn't Care About Peace
8. KFC's Colonel Sanders: He Was Real, Not Just an Icon
9. How Fast Will Humans Ever Run?
10. The Case Against Homeownership

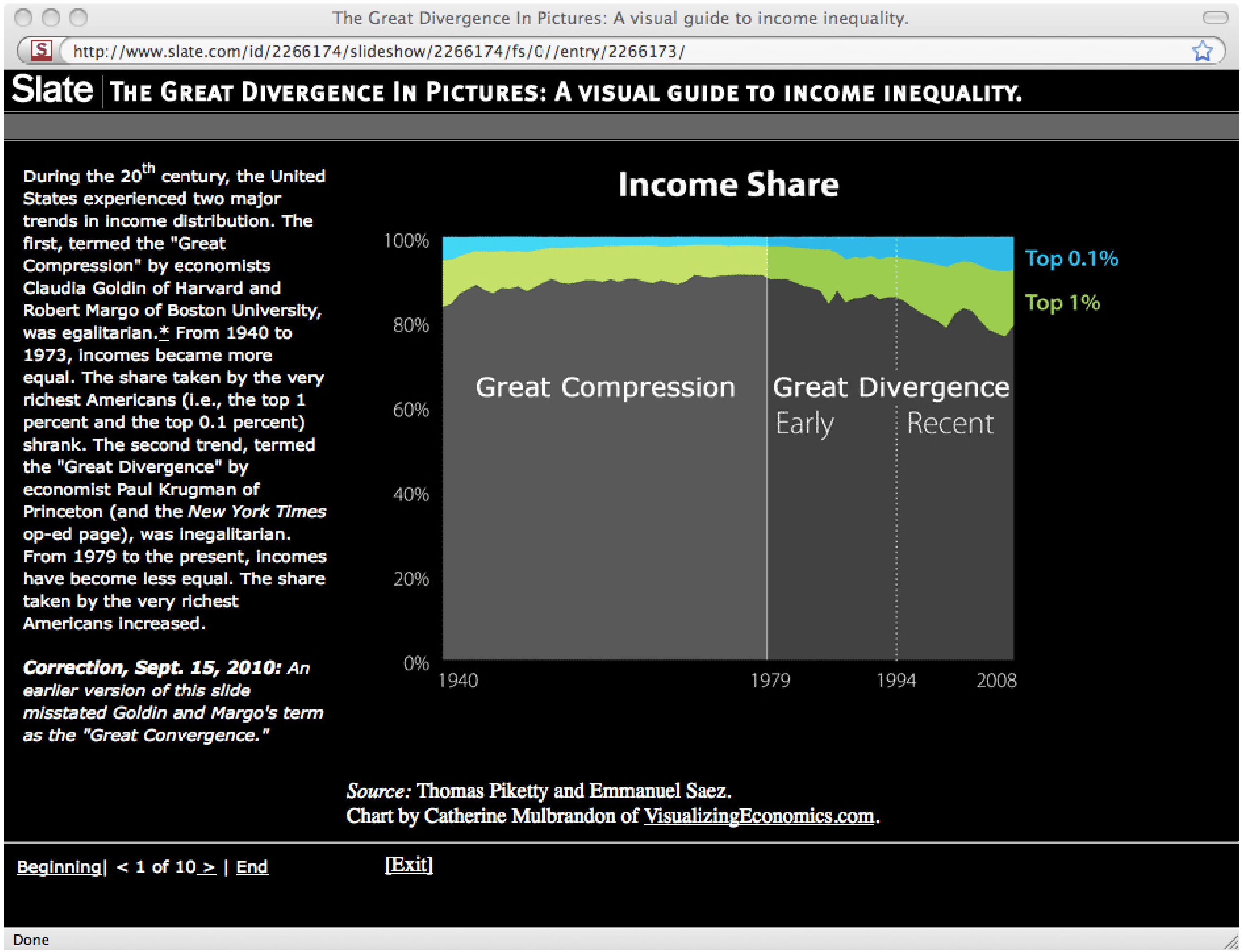
More News from Our Partners

CNN.com

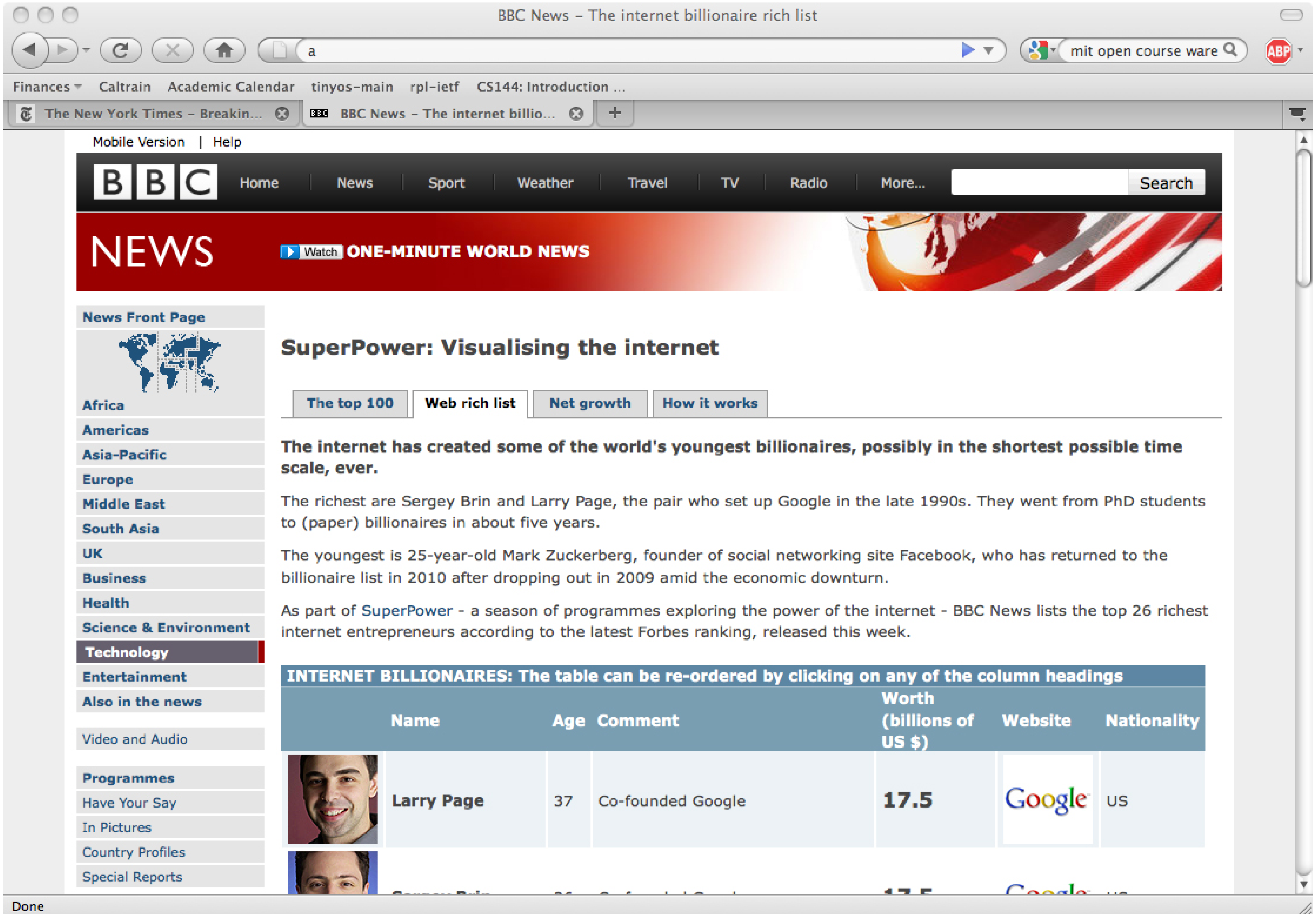
- Olmert: U.S. was ready to take Palestinian

Transferring data from brightcove.vo.llnwd.net...

Economic Change



Economic Change 2



BBC News – The internet billionaire rich list

Mobile Version | Help

BBC | Home | News | Sport | Weather | Travel | TV | Radio | More... | Search

NEWS

Watch ONE-MINUTE WORLD NEWS

News Front Page
Africa
Americas
Asia-Pacific
Europe
Middle East
South Asia
UK
Business
Health
Science & Environment
Technology
Entertainment
Also in the news

Video and Audio

Programmes
Have Your Say
In Pictures
Country Profiles
Special Reports

SuperPower: Visualising the internet

The top 100 | Web rich list | Net growth | How it works

The internet has created some of the world's youngest billionaires, possibly in the shortest possible time scale, ever.

The richest are Sergey Brin and Larry Page, the pair who set up Google in the late 1990s. They went from PhD students to (paper) billionaires in about five years.

The youngest is 25-year-old Mark Zuckerberg, founder of social networking site Facebook, who has returned to the billionaire list in 2010 after dropping out in 2009 amid the economic downturn.

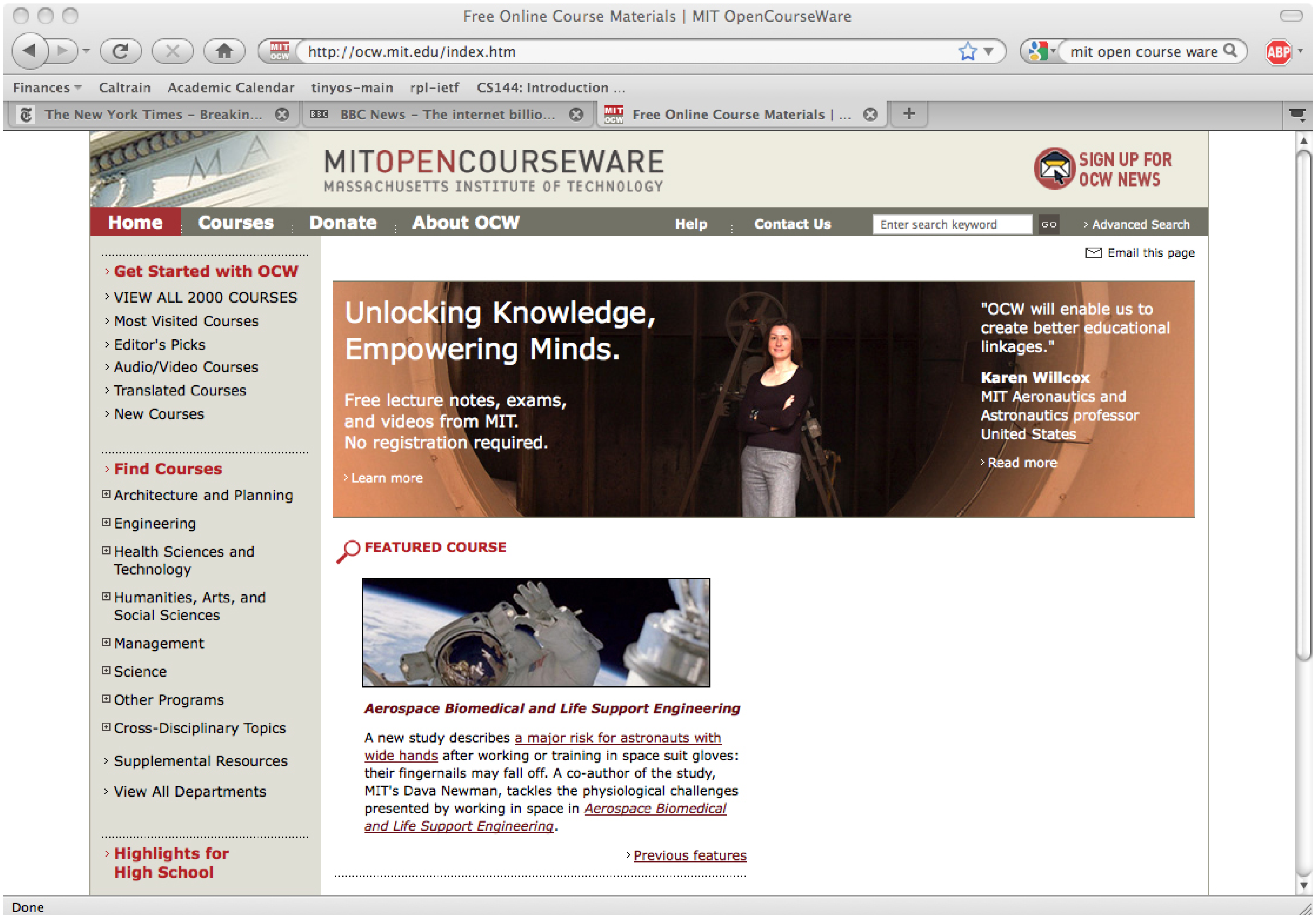
As part of SuperPower - a season of programmes exploring the power of the internet - BBC News lists the top 26 richest internet entrepreneurs according to the latest Forbes ranking, released this week.

INTERNET BILLIONAIRES: The table can be re-ordered by clicking on any of the column headings

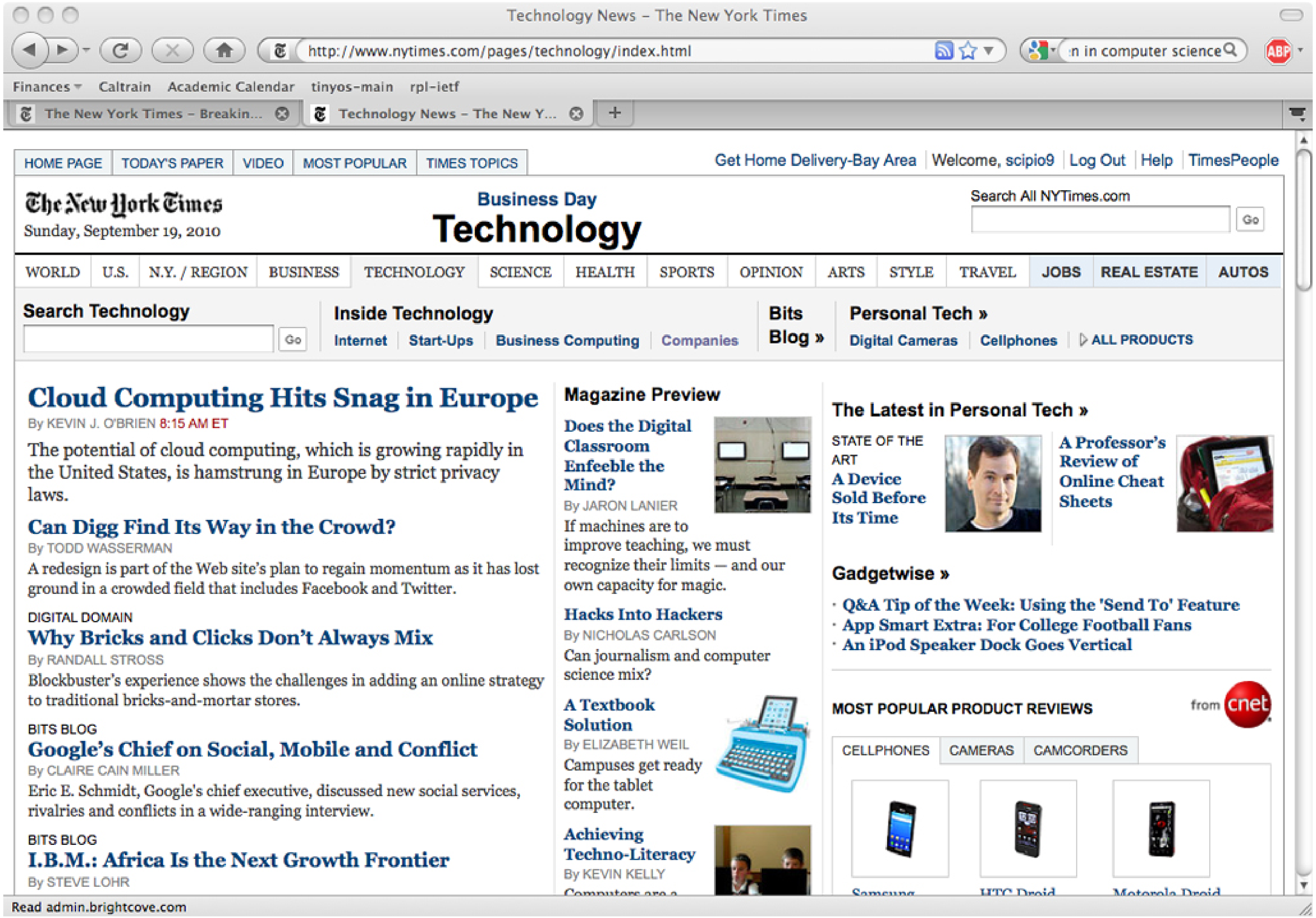
	Name	Age	Comment	Worth (billions of US $)	Website	Nationality
	Larry Page	37	Co-founded Google	17.5	Google	US

Done

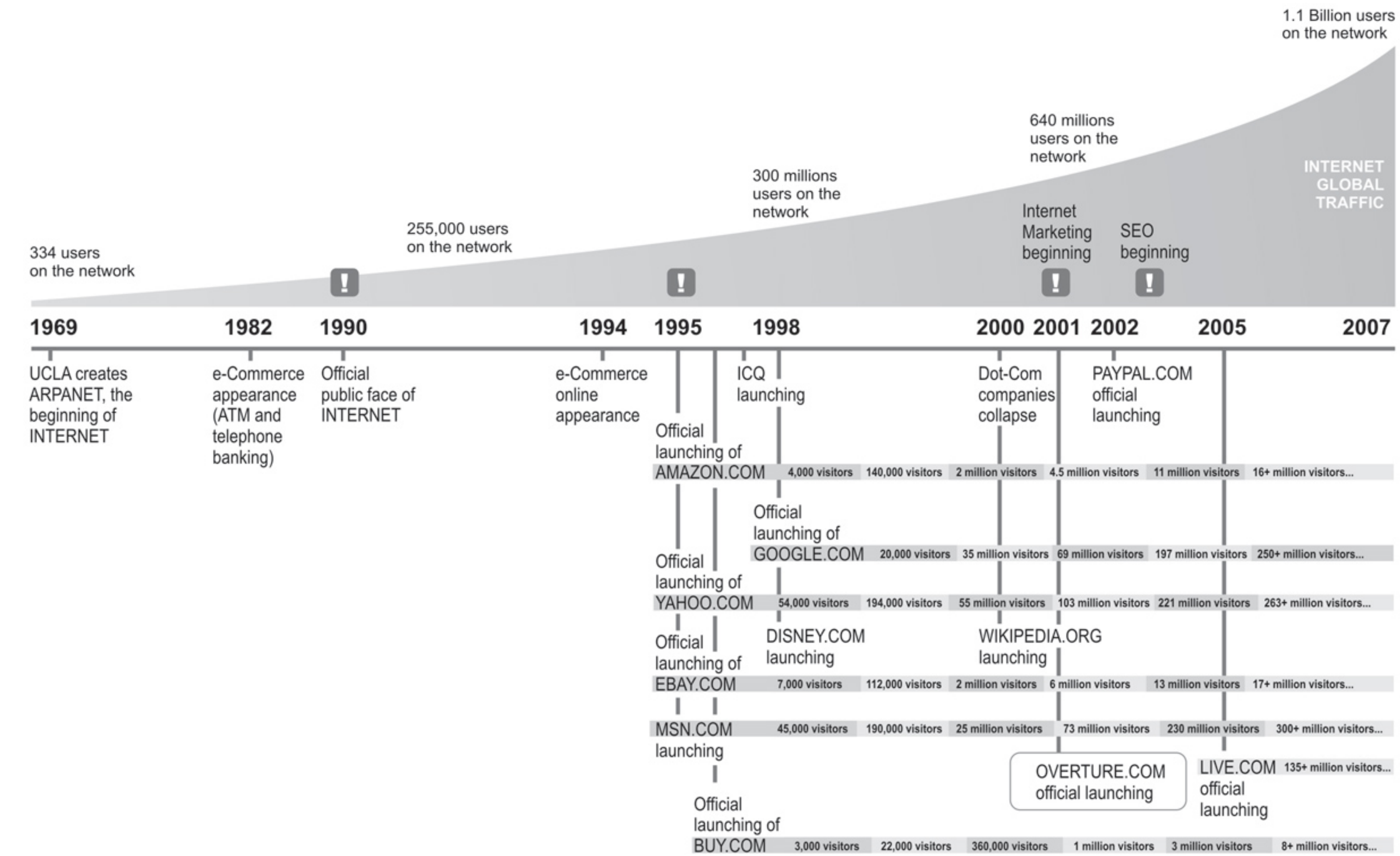
Educational Change



Dominance in Technology Today



It Hasn't Stopped



* User traffic calculation per day

Why You Should Take This Course

- **The Internet is driving tremendous change in the world**
- **It is continuously changing and evolving**
  - Any facts you learn will inevitably be out of date
  - Learn general *principles* of networks
- **Goal: Teach the concepts underlying networks**
  - How do networks work? What can one do with them?
  - Give you a basic understanding of the Internet
  - Give you experience using and writing protocols
  - Give you tools to understand new protocols & applications

Today's Lecture

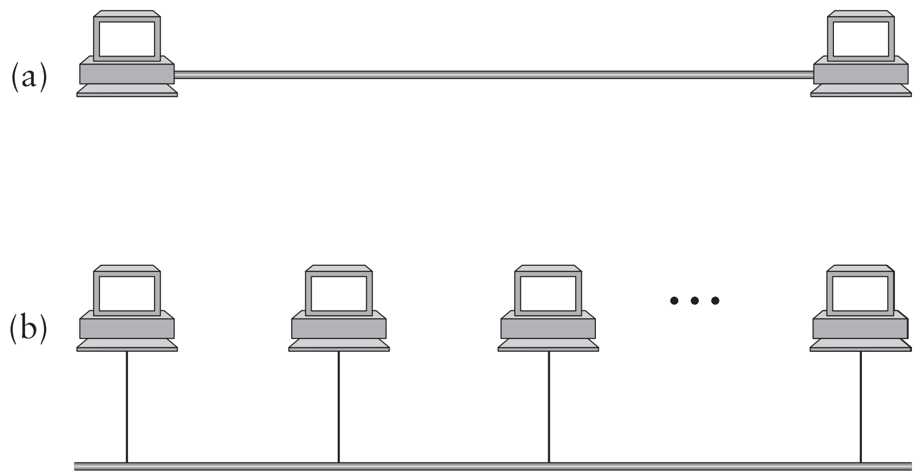
- **Basic networking abstractions**
  - Protocols
  - OSI layers and the Internet Hourglass
- **Transport protocols: TCP and UDP**
- **Protocol performance tradeoffs**
- **Programming refresher for lab 1+2**
  - Review of file descriptors
  - Some functions from the socket API
- **Next lecture: applications (HTTP, BitTorrent, etc.) and server socket programming**

Networks

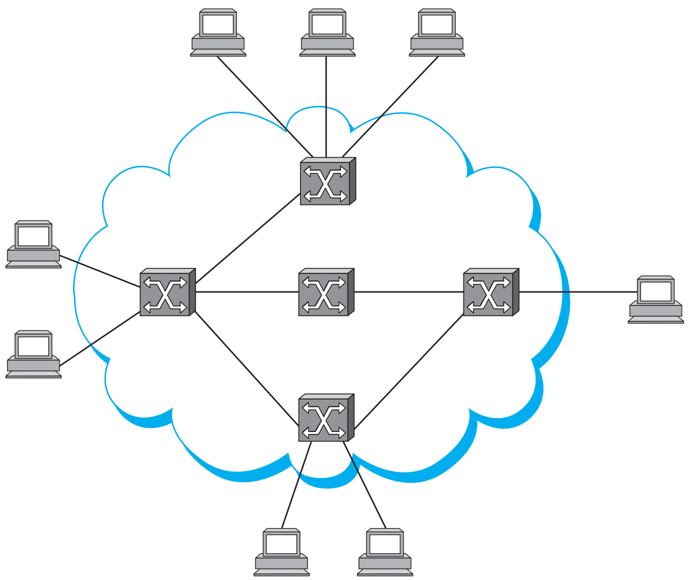
- **What is a network?**
  - A system of lines/channels that interconnect
  - E.g., railroad, highway, plumbing, communication, telephone, computer
- **What is a *computer* network?**
  - A form of communication network—moves information
  - Nodes are general-purpose computers
- **Why study computer networks?**
  - Many nodes are general-purpose computers
  - *You* can program the nodes
  - Very easy to innovate and develop new uses of network
  - Contrast: Old PSTN – all logic is in the core

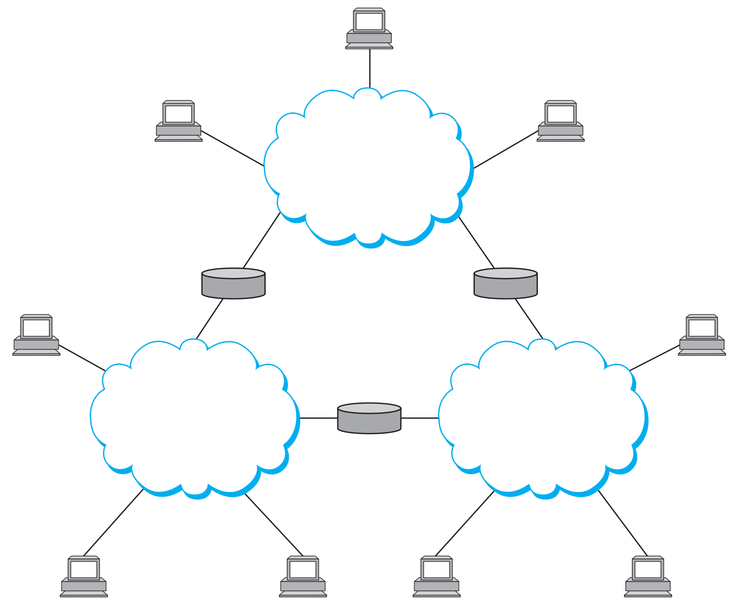
Building blocks

- **Nodes: Computers, dedicated routers, . . .**
- **Links: Coax, twisted pair, fibers, radio . . .**
  (a) point-to-point
  (b) multiple access – every node sees every packet



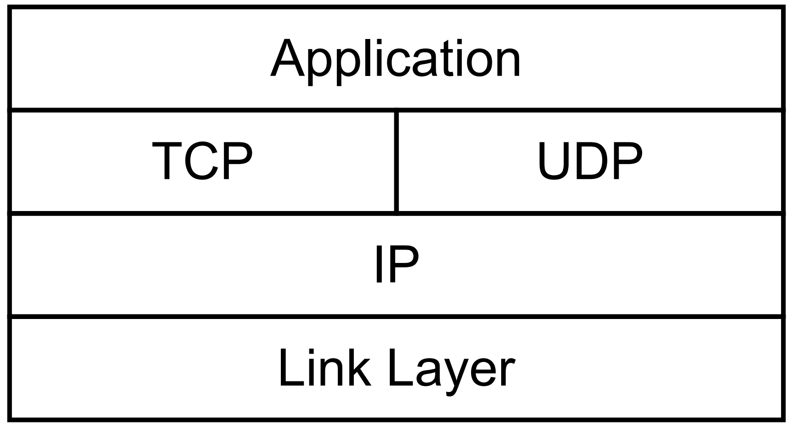
From Links to Networks





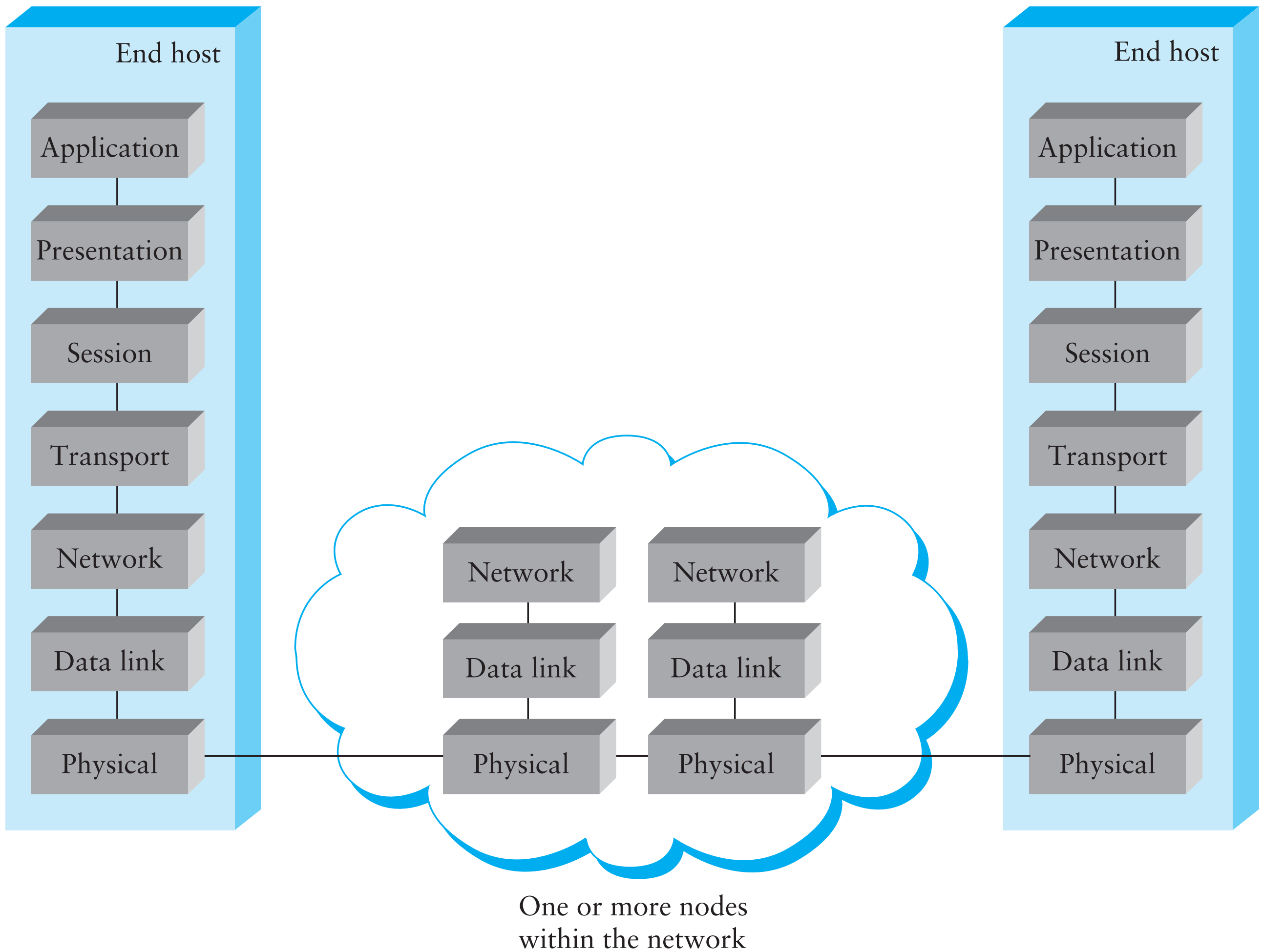
- **To scale to more nodes, use *switching***
  - nodes can connect multiple other nodes, or
  - Recursively, one node can connect multiple networks

Protocol layering



- **Can view network encapsulation as a stack**
- **A network packet from A to D must be put in link packets A to B, B to C, and C to D**
  - Each layer produces packets that become the payload of the lower-layer's packets
  - This is *almost* correct, but TCP/UDP "cheat" to detect certain errors in IP-level information like address

OSI layers



- **Layers typically fall into 1 of 7 categories**

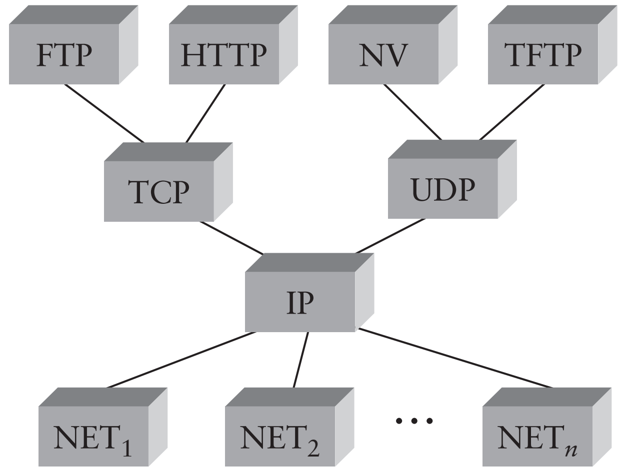
Layers (and lectures)

- **Physical – sends individual bits (8, 11)**
- **Data link – sends *frames*, handles access control to shared media (e.g., coax) (8, 11)**
- **Network – delivers packets, using *routing* (5-8)**
- **Transport – demultiplexes, provides reliability & flow control (3, 4)**
- **Session – can tie together multiple streams (e.g., audio & video)**
- **Presentation – crypto, conversion between representations (16)**
- **Application – what end user gets, e.g., HTTP (2, 9)**

Addressing

- **Each node typically has unique *address***
  - (or at least is made to think it does when there is shortage)
- **Each layer can have its own addressing**
  - Link layer: e.g., 48-bit Ethernet address (interface)
  - Network layer: 32-bit IP address (node)
  - Transport layer: 16-bit TCP port (service)
- ***Routing* is process of delivering data to destination across multiple link hops**
- **Special addresses can exist for broadcast/multicast**

Hourglass



- **Many application protocols over TCP & UDP**
- **IP works over many types of network**
- **This is "Hourglass" philosophy of Internet**
  - Idea: If everybody just supports IP, can use many different applications over many different networks
  - In practice, some claim narrow waist is now network *and* transport layers, due to NAT (lecture 10)

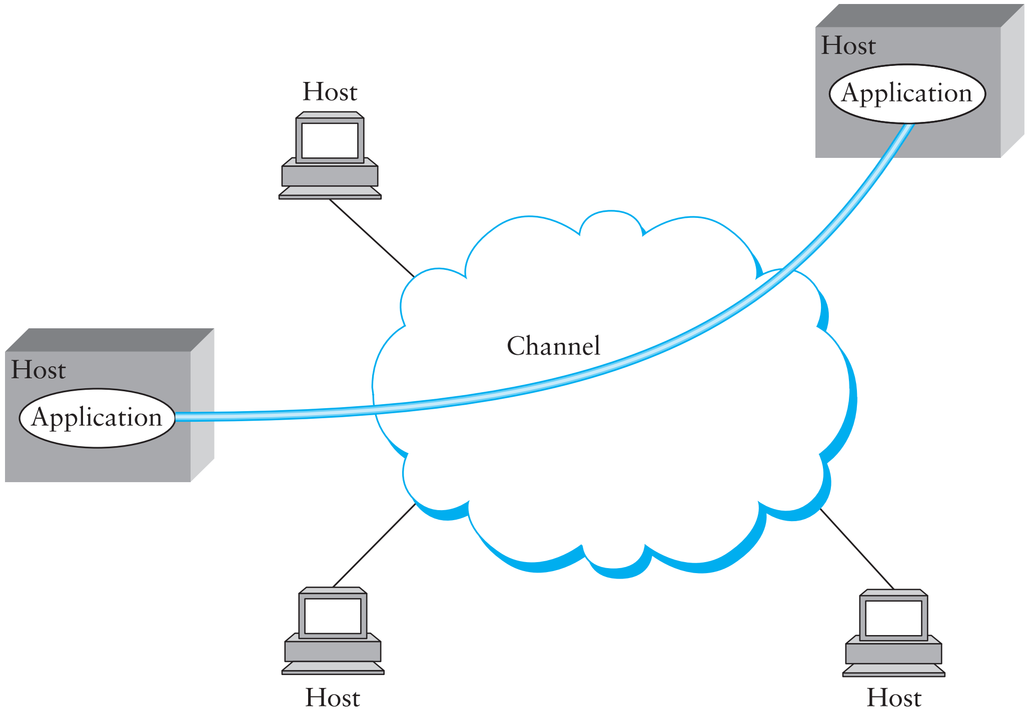
Internet protocol

- **Most computer nets connected by Internet protocol**
  - Runs over a variety of physical networks, so can connect Ethernet, Wireless, people behind modem lines, etc.
- **Every host has[a] a unique 4-byte IP address**
  - E.g., `www.ietf.org` $\rightarrow$ 132.151.6.21
  - Given a node's IP address, the network knows how to route a packet (lectures 5-7)
  - Next generation IPv6 uses 16-byte host addresses
- **But how do you build something like the web?**
  - Need naming (look up www.ietf.org) – DNS (lecture 8)
  - Need API for browser, server (CS110/this lecture)
  - Need demultiplexing within a host—E.g., which packets are for web server, which for mail server, etc.? (lecture 2)

---

[a] or thinks it has

Inter-process communication



- **Want abstraction of inter-process (not just inter-node) communication**
- **Solution: *Encapsulate* another protocol within IP**

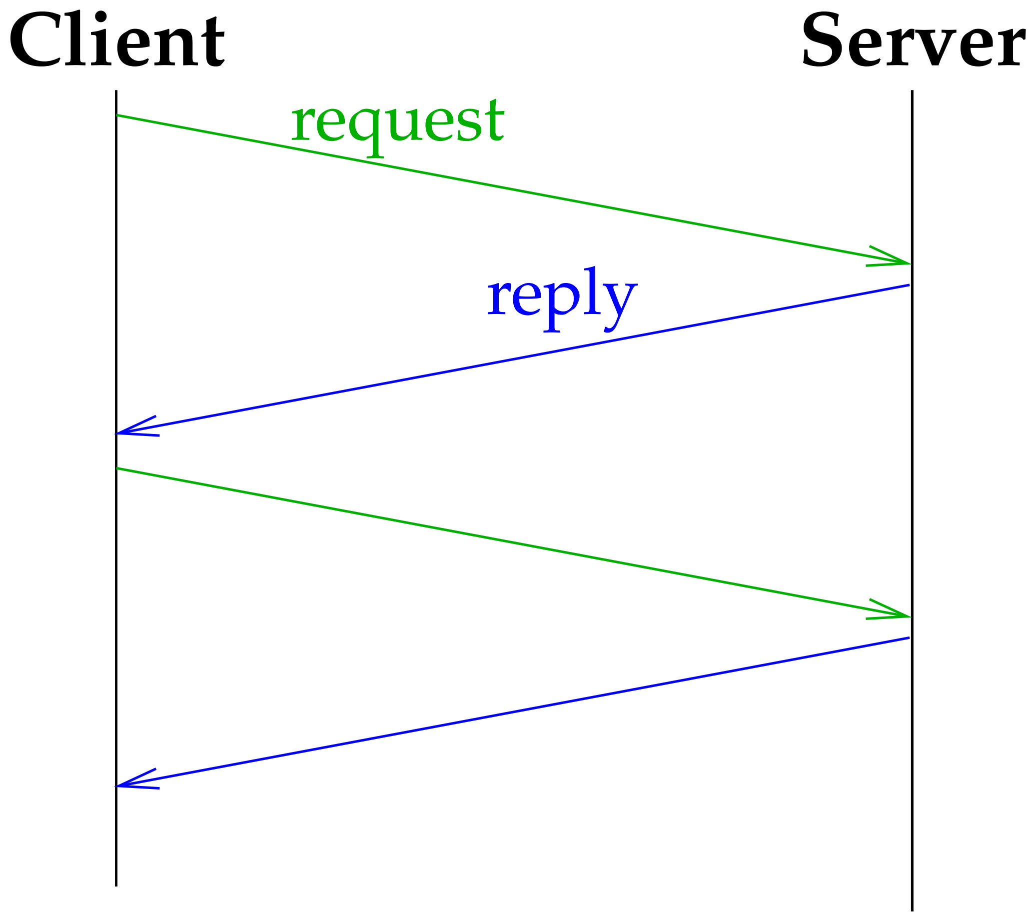
UDP and TCP

- **UDP and TCP most popular protocols on IP**
  - Both use 16-bit *port* number as well as 32-bit IP address
  - Applications *bind* a port & receive traffic to that port
- **UDP – unreliable datagram protocol**
  - Exposes packet-switched nature of Internet
  - Sent packets may be dropped, reordered, even duplicated (but generally not corrupted)
- **TCP – transmission control protocol**
  - Provides illusion of a reliable "pipe" between to processes on two different machines (lecture 3)
  - Handles congestion & flow control (lecture 4)

Uses of TCP

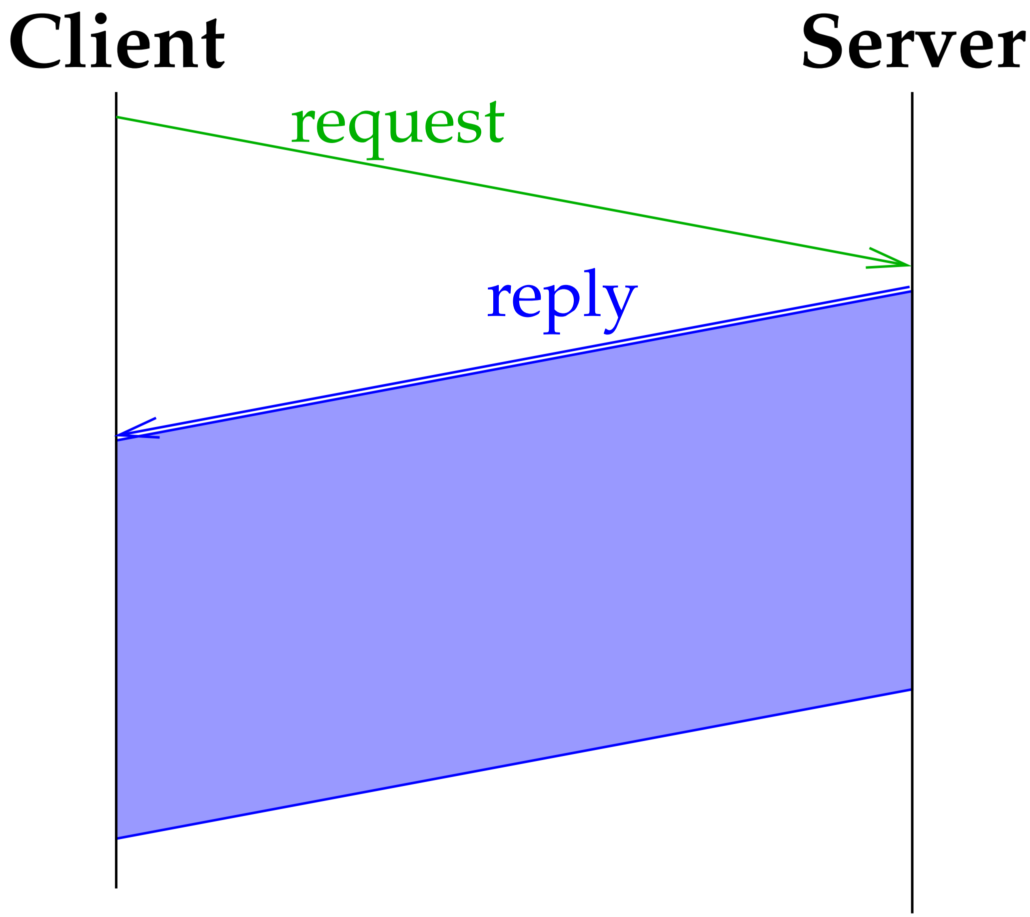
- **Most applications use TCP**
  - Easier interface to program to (reliability, lecture 3)
  - Automatically avoids congestion (don't need to worry about taking down network, lecture 4)
- **Servers typically listen on well-known ports**
  - SSH: 22
  - Email: 25
  - Finger: 79
  - Web / HTTP: 80
- **Example: Interacting with www.stanford.edu**

Small request/reply protocol



- **Small message protocols typically dominated by latency**

Large reply protocol



- **For bulk tranfer, throughput is most important**

Performance definitions

- **Throughput – Number of bits/time you can sustain at the receiver**
  - Improves with technology
- **Latency – How long for message to cross network**
  - Propagation + Transmit + Queue
  - We are stuck with speed of light…
    10s of milliseconds to cross country
- **Goodput – TransferSize/Latency**
- **Jitter – Variation in latency**
- **What matters most for your application?**
  - We'll look at network applications next lecture

Programming Sockets

- **Book has Java source code**
- **CS144 is in C**
  - C is the language of choice for low-level systems
  - Many books and internet tutorials
- **Berkeley sockets API**
  - Bottom-level OS interface to networking
  - Important to know and do once
  - Higher-level APIs build on them

Quick CS110 review: System calls

- **System calls invoke code in the OS kernel**
  - Kernel runs in a more privileged mode than application
  - Can execute special instructions that application cannot
  - Can interact directly with devices such as network card
- **Higher-level functions built on syscall interface**
  - `printf`, `scanf`, `gets`, etc. all user-level code

File descriptors

- **Most IO done on file descriptors**
  - Small integers referencing per-process table in the kernel
- **Examples of system calls with file descriptors:**
  - `int open(char *path, int flags, ...);`
    - Returns new file descriptor bound to file `path`
  - `int read (int fd, void *buf, int nbytes);`
    - Returns number of bytes read
    - Returns 0 bytes at end of file, or -1 on error
  - `int write (int fd, void *buf, int nbytes);`
    - Returns number of bytes written, -1 on error
    - (Never returns 0 if `nbytes` $>$ 0)
  - `int close (int fd);`
    - Deallocates file descriptor (not underlying I/O resource)

Error returns

- **What if syscall failes? E.g.** `open` **non-existent file?**
  - Returns -1 (invalid fd number)
- **Most system calls return -1 on failure**
  - Always check for errors when invoking system calls
  - Specific kind of error in global int `errno`
    (But `errno` will be unchanged if syscall did not return -1)
- `#include <sys/errno.h>` **for possible values**
  - 2 = `ENOENT` "No such file or directory"
  - 13 = `EACCES` "Permission Denied"
- `perror` **function prints human-readable message**
  - `perror ("initfile");`
    $\rightarrow$ "`initfile: No such file or directory`"

Sockets: Communication between machines

- **Network sockets are file descriptors too**
- **Datagram sockets: Unreliable message delivery**
  - With IP, gives you UDP
  - Send atomic messages, which may be reordered or lost
  - Special system calls to read/write: `send`/`recv`, `sendto`/`recvfrom`, and `sendmsg`/`recvmsg` (most general)
- **Stream sockets: Bi-directional pipes**
  - With IP, gives you TCP
  - Bytes written on one end read on the other
  - Reads may not return full amount requested—must re-read

Socket naming

- **Recall how TCP & UDP name communication endpoints**
  - 32-bit IP address specifies machine
  - 16-bit TCP/UDP port number demultiplexes within host
  - Well-known services “listen” on standard ports: finger—79, HTTP—80, mail—25, ssh—22
  - Clients connect from arbitrary ports to well known ports
- **A *connection* can be named by 5 components**
  - Protocol (TCP), local IP, local port, remote IP, remote port
  - TCP requires connected sockets, but not UDP

System calls for using TCP

Client	Server
	`socket` – make socket
	`bind` – assign address
	`listen` – listen for clients
`socket` – make socket	
`bind`* – assign address	
`connect` – connect to listening socket	
	`accept` – accept connection

*This call to `bind` is optional; `connect` can choose address & port.

Socket address structures

- **Socket interface supports multiple network types**
- **Most calls take a generic `sockaddr`:**

```
struct sockaddr {
  uint16_t  sa_family;   /* address family */
  char      sa_data[14]; /* protocol-specific address */
};                       /* (may be longer than this) */

int connect(int fd, const struct sockaddr *, socklen_t);
```

- **Cast `sockaddr *` from protocol-specific struct, e.g.:**

```
struct sockaddr_in {
  short   sin_family;        /* = AF_INET */
  u_short sin_port;          /* = htons (PORT) */
  struct  in_addr sin_addr; /* 32-bit IPv4 address */
  char    sin_zero[8];
};
```

Dealing with address types [RFC 3493]

- **All values in network byte order (big endian)**
  - `htonl` converts 32-bit value from host to network order
  - `ntohl` converts 32-bit value from network to host order
  - `ntohs`/`htons` same for 16-bit values
- **All address types begin with family**
  - `sa_family` in `sockaddr` tells you actual type
- **Unfortunately, not all address types are the same size**
  - E.g., `struct sockaddr_in6` is typically 28 bytes, yet generic `struct sockaddr` is only 16 bytes
  - So most calls require passing around socket length
  - Can simplify code with new generic `sockaddr_storage` big

enough for all types (but have to cast between *3* types now)

Looking up a socket address w. `getaddrinfo`

```
struct addrinfo hints, *ai;
int err;

memset (&hints, 0, sizeof (hints));
hints.ai_family = AF_UNSPEC;     /* or AF_INET or AF_INET6 */
hints.ai_socktype = SOCK_STREAM; /* or SOCK_DGRAM for UDP */

err = getaddrinfo ("www.stanford.edu", "http", &hints, &ai);
if (err)
  fprintf (stderr, "%s\n", gia_strerror (err));
else {
  /* ai->ai_family  = address type (AF_INET or AF_INET6) */
  /* ai->ai_addr    = actual address cast to (sockaddr *) */
  /* ai->ai_addrlen = length of actual address */
  freeaddrinfo (ai); /* must free when done! */
}
```

Address lookup details

- `getaddrinfo` **notes:**
  - Can specify port as service name or number (e.g., `"80"` or `"http"`, allows possibility of dynamically looking up port)
  - May return multiple addresses (chained with `ai_next` field)
  - You must free structure with `freeaddrinfo`
- **Other useful functions to know about**
  - `getnameinfo` – Lookup hostname based on address
  - `inet_ntop` – convert IPv4 or 6 address to printable form
  - `inet_pton` – convert string to IPv4 or 6 address

EOF in more detail

- **Simple client-server application**
  - Client sends request
  - Server reads request, sends response
  - Client reads response
- **What happens when you're done?**
  - Client wants server to read EOF to say request is done
  - But still needs to be able to read server reply – fd is not closed!

shutdown

- `int shutdown (int fd, int how);`
  - Shuts down a socket w/o closing file descriptor
  - `how`: 0 = reading, 1 = writing, 2 = both
  - Note: Applies to *socket*, not descriptor—so copies of descriptor (through `dup` or `fork` affected)
  - Note 2: With TCP, can't detect if other side shuts for reading
- **Many network applications detect & use EOF**
  - Common error: "leaking" file descriptor via `fork`, so not closed (and no EOF) when you exit

Today's Lecture

- **Basic networking abstractions**
  - Protocols
  - OSI layers and the Internet Hourglass
- **Transport protocols: TCP and UDP**
- **Protocol performance tradeoffs**
- **Programming refresher for lab 1+2**
  - Review of file descriptors
  - Some functions from the socket API
- **Next lecture: applications (HTTP, BitTorrent, etc.) and server socket programming**

Structure of Rest of Class

- **IP and above (5 weeks)**
  - Application layers
  - Network layer: IP and routing, multicast
  - Transport layer: TCP and congestion control
  - Naming, address translation, and content distribution
- **Below IP (2 weeks)**
  - Network address translation (NAT)
  - Link and physical layers
- **Advanced topics (2 weeks)**
  - Multimedia
  - Network coding
  - Security