CS144 – Introduction to Computer Networking

Instructor: Philip Levis
CAs: Kirian Abraham, Tahir Azim, Arda Kara, Behram Mistree, Harsh Nayyar
Section Leaders: Alex Quach, Pong Eksombatchai
cs144-aut1112-staff@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

- Old school web page: http://cs144.stanford.edu

- We use Piazza for assignments: please send all assignment questions there
  - Link on http://cs144.stanford.edu
  - Piazza allows you to answer questions and rate answers

- Send all staff communication to 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**

- **Programming labs**
  - 5 lab assignments implemented in C
  - If you are not comfortable with C and gdb they will be painful

- **Two writing homework assignments**
  - One question each week, pick 2
  - Opportunity to rewrite and resubmit based on feedback

- **Administrative handout has details on grading, please put questions on Piazza**
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

- Most assignments due at start of Wednesday lecture
  - Late policy: can turn in late until 8:59PM that Friday, grade capped at 90%
  - Lab 1: Due at start of Monday lecture, late is 8:59PM on Wednesday
Section

- Place and time TBA
- Led by TAs and section leaders
- Practical assignment help and guidance
- This week: gdb tutorial
Why You Should Care About the Internet (just 12 months)
The Internet Might Save Main Street

The web is killing mega stores, but it might make room for the Shop Around the Corner.

By PETER FUNT

If Tom Hanks and Meg Ryan ever decide to make a sequel to their 1998 charmer "You've Got Mail," the story line might go like this:

Happily married and still living in Manhattan, the couple has dropped AOL and now communicates exclusively by text, tweet and Skype. But as the film begins, the Fox & Sons mega-bookstore chain is in trouble. Kathleen Kelly-Fox, now executive vice president, tweets: "Sales down 63%. Internet taking best customers. Ebooks killing us."
Main Street Lauds Signing of E-Fairness Legislation in California

WASHINGTON, Sept. 23, 2011 /PRNewswire via COMTEX/ -- The Alliance for Main Street Fairness (AMSF) today released the following statement in response to California Governor Jerry Brown signing into law Assembly Bill 155 or e-fairness legislation:

"Government should not be in the business of picking winners and losers, and now California is not. This should send a message to every other state in the nation that Internet retailers can and should operate by the same rules as everyone else," said Danny Diaz, spokesperson for the Alliance for Main Street Fairness (AMSF). "This also sends an important signal to employers throughout the country that they must demand fairness of their elected officials. One can only hope that federal officials are listening as well."

Press Release

Most Popular
1. SLIDE SHOW
   Top 10 cities with the longest commute
2. JOHNSHINAS TECH INVESTOR
   The day Steve Jobs saved Apple
3. Silver drops 18%, worst in decades; platinum sinks
4. AUTO REVIEW
   Top gas sippers: An overdue look at economy
5. WEEKEND INVESTOR
   5 tips for long-term investing as markets shudder
Iran Protests: Twitter, the Medium of the Movement
by LEV GROSSMAN  Wednesday, Jun. 17, 2009

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.
Political Change 2011

Social Media and the Arab Spring: What Have We Learned?

Nine months have now passed since the tumultuous beginnings of the Arab Spring burst forth in the streets of Tunisia. A rising spirit of protest has since spread like wildfire across the Middle East, communicated primarily through the channels of social media.

For the legions of critics who had previously dismissed platforms like Facebook and Twitter as vapid troughs of celebrity gossip and self-aggrandizement, the toppling of regimes in Tunisia and Egypt suggested that these tools were as effective for organizing protests and revolutions as they were for organizing keg parties. The movements throughout the Arab world appeared to have imbued social media with an irrevocable sense of legitimacy as a tool for fomenting change.

As the ongoing tumult throughout the Middle East enters a sort of adolescence, however, the true role of social media in the revolutions is undergoing a necessary closer inspection.
Iran: Computer Malware Sabotaged Uranium Centrifuges

By Kim Zetter

November 29, 2010 | 4:16 pm | Categories: Cybersecurity

Send us a tip
During the 20th century, the United States experienced two major trends in income distribution. The first, termed the "Great Compression" by economists Claudia Goldin of Harvard and Robert Margo of Boston University, was egalitarian. From 1940 to 1973, incomes became more equal. The share taken by the very richest Americans (i.e., the top 1 percent and the top 0.1 percent) shrank. The second trend, termed the "Great Divergence" by economist Paul Krugman of Princeton (and the New York Times op-ed page), was inequalitarian. From 1979 to the present, incomes have become less equal. The share taken by the very richest Americans increased.

Correction, Sept. 15, 2010: An earlier version of this slide misstated Goldin and Margo’s term as the "Great Convergence."

More Economic Change
Educational Change 2010
Educational Change 2011

Introduction to Databases
Jennifer Widom

COURSE DESCRIPTION

From October-December 2011 this online course will be offered in a structured fashion free of charge to students worldwide. The public offering will include scored assignments and exams, online discussions with the instructor, and a statement of accomplishment for those who complete the course. Please visit the Introduction to Databases website to sign up.

This online course covers database design and the use of database management systems for applications. It includes extensive coverage of the relational model, relational algebra, and SQL -- the standard language for creating, querying, and modifying relational databases. It also covers XML data including DTDs and XML Schema for validation, and the query and transformation languages XPath, XQuery, and XSLT. The course includes database design in UML, and relational design principles based on dependencies and normal forms. Many additional key database topics from the design and application-building perspective are also covered: indexes, views, transactions, authorization, integrity constraints, triggers, on-line analytical processing (OLAP), and emerging "NoSQL" systems. Further topics will be added over time.

The course does not assume prior knowledge of any specific topics, however a solid computer science foundation -- a reasonable amount of programming, as well as knowledge of basic computer science theory -- will make the material more accessible.

Online materials include videos with embedded quizzes, suggested textbook and web readings, extensive written and programming exercises, and instructions for installing any necessary software.

Status update August 2011: All videos for the first phase of core material are online, covering much of what we cover in our ten-week course at Stanford. Extensive SQL and XML query exercises are provided (including quick guides for the software involved), along with written or database-programming exercises for all other major topics. The course materials page is complete for the current set of topics, including suggested readings in several textbooks, pointers to online materials, and all slides and scripts used in the videos.

Scheduled for September 2011: Many more quizzes embedded into the videos
Dominance in Technology Today
Mobile Devices

Mobile Internet Ramping Faster than Desktop Internet Did – Apple Leading Charge

iPhone + iTouch vs. NTT docomo i-mode vs. AOL vs. Netscape Users
First 20 Quarters Since Launch

Mobile Internet
iPhone + iTouch
Launched 6/07

~36MM

Desktop Internet
Netscape*
Launched 12/94

~31MM

~18MM

~8MM

Quarters Since Launch
iPhone + iTouch
NTT docomo i-mode
AOL
Netscape

Morgan Stanley

Note: *AOL subscribers data not available before Q3/94; Netscape users limited to US only. Morgan Stanley Research estimates ~50MM netbooks have shipped in first 10 quarters since launch (10/07). Source: Company Reports, Morgan Stanley Research.
Why You Should Take This Course

• The Internet is driving tremendous changes in the world

• It is continuously growing and evolving
  - Any facts you learn will inevitably be out of date
  - Learn general *principles* of networks

• **Goal: Teach the underlying concepts of 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: basics of reliability 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

<table>
<thead>
<tr>
<th>Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCP</td>
</tr>
<tr>
<td>UDP</td>
</tr>
<tr>
<td>IP</td>
</tr>
<tr>
<td>Link Layer</td>
</tr>
</tbody>
</table>

- Can view network encapsulation as a stack
- A network packet (IP) from A to D must be put in link layer 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
• Layers typically fall into 1 of 7 categories
Layers (and lectures)

- Physical – sends individual bits (9, 13)
- Data link – sends *frames*, handles access control to shared media (e.g., coax) (9, 13)
- Network – delivers packets, using *routing* (4, 6, 7)
- Transport – demultiplexes, provides reliability & flow control (3, 5)
- Session – can tie together multiple streams (e.g., audio & video) (15)
- Presentation – crypto, conversion between representations (15)
- Application – what end user gets, e.g., HTTP (2, 11)
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
• 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 many physical networks

• Every host has a unique 4-byte IP address
  - E.g., www.ietf.org → 132.151.6.21
  - Given an IP address, the network routes a packet (lectures 4, 6, 7)
    - Next-gen IPv6 uses 16-byte host addresses (lecture 12)

• 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 5)
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 5)

- 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 transfer, 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 fails? 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");`
    → “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

<table>
<thead>
<tr>
<th>Client</th>
<th>Server</th>
</tr>
</thead>
<tbody>
<tr>
<td>socket – make socket</td>
<td>socket – make socket</td>
</tr>
<tr>
<td>bind* – assign address</td>
<td>bind – assign address</td>
</tr>
<tr>
<td>connect – connect to listening socket</td>
<td>listen – listen for clients</td>
</tr>
<tr>
<td>accept – accept connection</td>
<td></td>
</tr>
</tbody>
</table>

*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:**

  ```c
  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.:**

  ```c
  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

- But 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

```c
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
", 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