CS144 – Introduction to Computer Networking

Philip Levis and David Mazières

cs144-staff@scs.stanford.edu

http://cs144.scs.stanford.edu

Networks class

- **Instructors: Philip Levis and David Mazières**
- **TAs: Maria Kazandjieva, Ben Nham, and ...**
- **Goal: Teach the concepts underlying networks**
  - How do networks work? What can one do with them?
  - Give you experience using and writing protocols
  - Give you the tools to understand new protocols & systems
- **What this class will *not* do**
  - Train you on all the latest "hot" technologies
- **Prerequisites:**
  - CS108 (or equiv) – class assumes good programming skills
  - C programming (all the labs will be in C)

Administrivia

- **All assignments are on the web page**
- **Text: Kurose & Ross, *Computer Networking: A Top-Down Approach, 4th edition***
  - 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)
- **All assignment questions to newsgroup (not mail!)**

Grading

- **Quizzes: Midterm & Final**
- **Homework**
  - ~ 5 lab assignments implemented in C
  - Possibly a problem set, or other kind of lab
- **Grading**
  - "Max" policy (sort of)

Topics

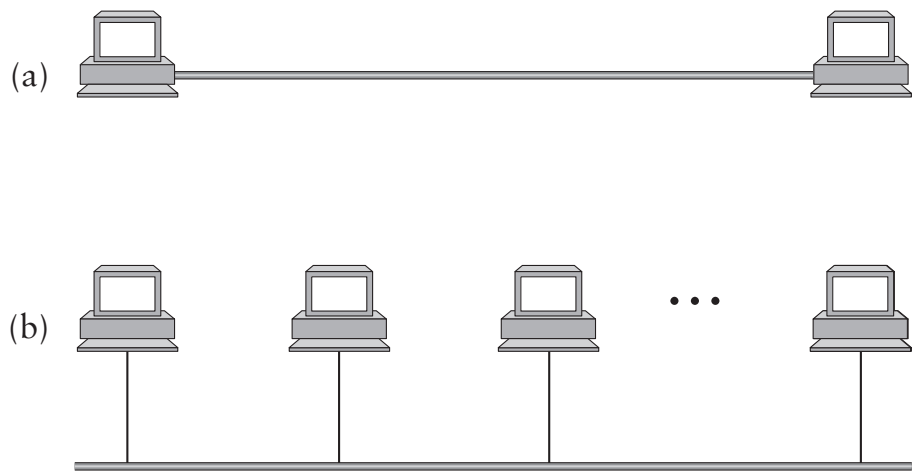
- **Network programming (sockets, RPC)**
- **Network (esp. Internet) architecture**
  - Switching, Routing, Congestion control, TCP/IP, Wireless networks
- **Using the network**
  - Interface hardware & low-level implementation issues, Naming (DNS), Error detection, compression
- **Higher level issues**
  - Encryption and Security, caching & content distribution, Peer-to-peer systems

Networks

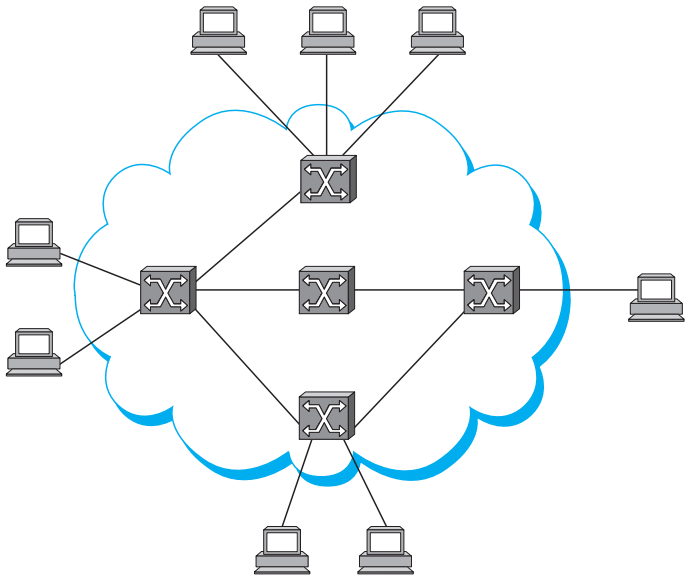
- **What is a network?**
  - A system of lines/channels that interconnect
  - E.g., railroad, highway, plumbing, communication, telephone, computer
- **What is a *computer* network?**
  - A form of communication network—moves information
  - Nodes are general-purpose computers
- **Why study computer 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

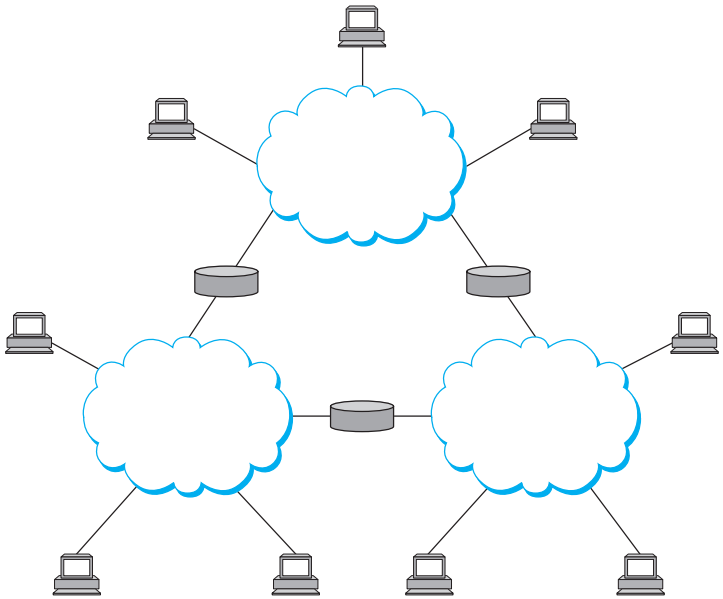
Building blocks

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



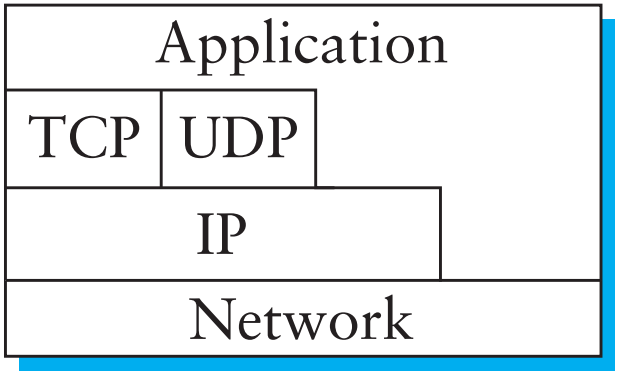
From Links to Networks





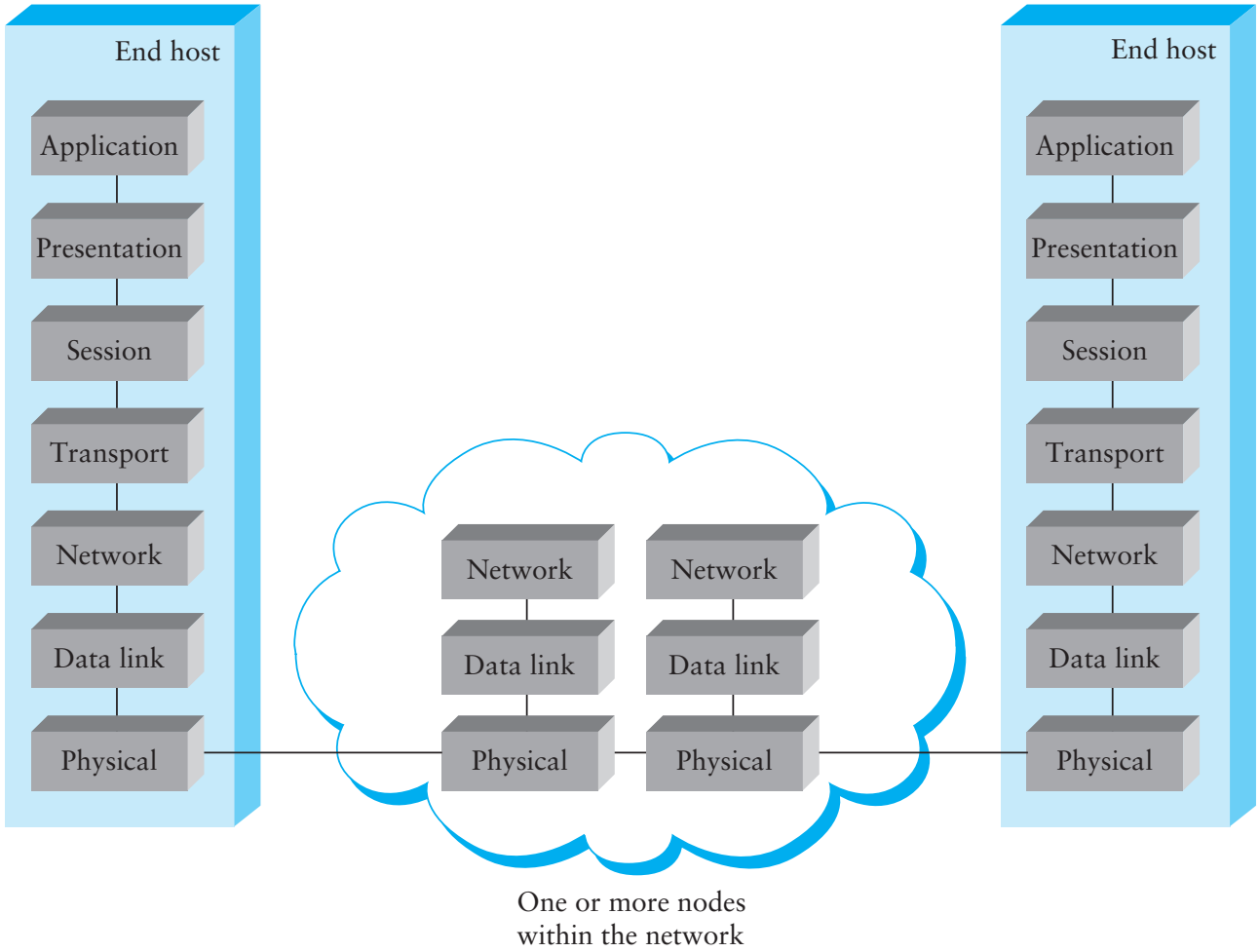
- **To scale to more nodes, use *switching***
  - nodes can connect multiple other nodes, or
  - Recursively, one node can connect multiple networks

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**

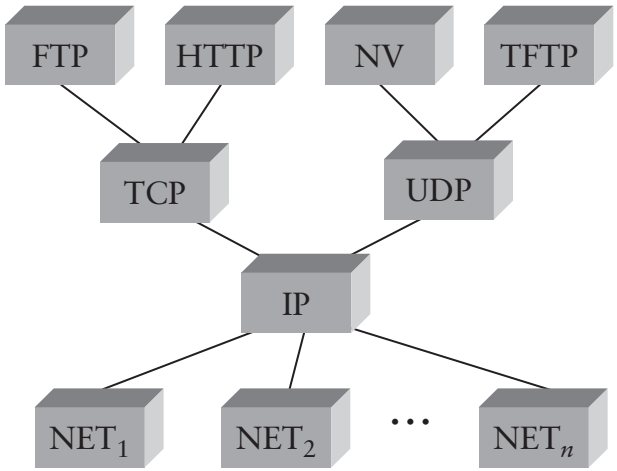
Layers

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

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



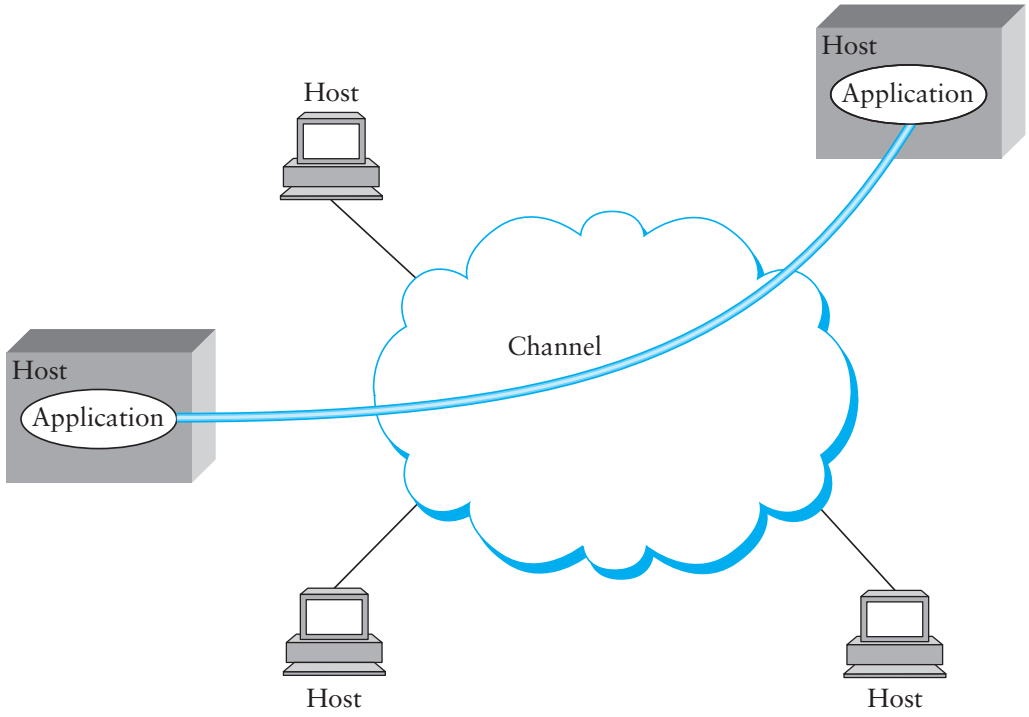
- **Many application protocols over TCP & UDP**
- **IP works over many types of network**
- **This is "Hourglass" philosophy of Internet**
  - Idea: If everybody just supports IP, can use many different applications over many different networks
  - In practice, some claim narrow waist is now network *and* transport layers, due to NAT (lecture 8)

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 → 132.151.6.21
  - Given a node's IP address, the network knows how to route a packet (lectures 3+4)
- **But how do you build something like the web?**
  - Need naming (look up www.ietf.org) – DNS (lecture 7)
  - Need interface for browser & server software (this lecture)
  - Need demultiplexing within a host—E.g., which packets are for web server, which for mail server, etc.? (lecture 5)

[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 5)
  - Handles congestion & flow control (lecture 6)

Uses of TCP

- **Most applications use TCP**
  - Easier interface to program to (reliability)
  - Automatically avoids congestion (don't need to worry about taking down network, lectures 5+6)
- **Servers typically listen on well-known ports**
  - SSH: 22
  - Email: 25
  - Finger: 79
  - Web / HTTP: 80
- **Example: Interacting with www.stanford.edu**

Programming Sockets

- **Book has Java source code**
- **CS144 is in C**
  - 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

System calls

- **Problem: How to access resources other than CPU**
  - Disk, network, terminal, other processes
  - CPU prohibits instructions that would access devices
  - Only privileged OS "kernel" can access devices
- **Applications request I/O operations from kernel**
- **Kernel supplies well-defined *system call* interface**
  - Applications set up syscall arguments and *trap* to kernel
  - Kernel performs operation and returns result
- **Higher-level functions built on syscall interface**
  - `printf`, `scanf`, `gets`, etc. all user-level code

I/O in UNIX

- **OS provides abstraction of a "file descriptor" (fd)**
- **Applications "open" files/devices by name**
  - I/O happens through open files
- `int open(char *path, int flags, ...);`
  - `flags`: `O_RDONLY`, `O_WRONLY`, `O_RDWR`
  - `O_CREAT`: create the file if non-existent
  - `O_EXCL`: (w. `O_CREAT`) create if file exists already
  - `O_TRUNC`: Truncate the file
  - `O_APPEND`: Start writing from end of file
  - `mode`: final argument with `O_CREAT`
- **Returns file descriptor—used for all I/O to file**

Error returns

- **What if** `open` **fails? Returns -1 (invalid fd)**
- **Most system calls return -1 on failure**
  - Specific kind of error in global int `errno`
- `#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`"
- **Always check for errors when you invoke system calls!**

Operations on file descriptors

- `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
- `int close (int fd);`
  - Closes file descriptor, not underlying I/O resource
- `int dup2 (int oldfd, int newfd);`
  - Closes `newfd`, if it was a valid descriptor
  - Makes `newfd` an exact copy of `oldfd`

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`
- **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.

Client interface (no error checking)

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

int s = socket (AF_INET, SOCK_STREAM, 0);
bzero (&sin, sizeof (sin));
sin.sin_family = AF_INET;
sin.sin_port = htons (13);  /* daytime port */
sin.sin_addr.s_addr = htonl (IP_ADDRESS);
connect (s, (sockaddr *) &sin, sizeof (sin));
do_something_with (s);
```

Server interface (no error checking)

```
struct sockaddr_in sin;
int s = socket (AF_INET, SOCK_STREAM, 0);
bzero (&sin, sizeof (sin));
sin.sin_family = AF_INET;
sin.sin_port = htons (9999);
sin.sin_addr.s_addr = htonl (INADDR_ANY);
bind (s, (struct sockaddr *) &sin, sizeof (sin));
listen (s, 5);

for (;;) {
  socklen_t len = sizeof (sin);
  int cfd = accept (s, (struct sockaddr *) &sin, &len);
  /* cfd is new connection; you never read/write s */
  do_something_with (cfd);
  close (cfd);
}
```

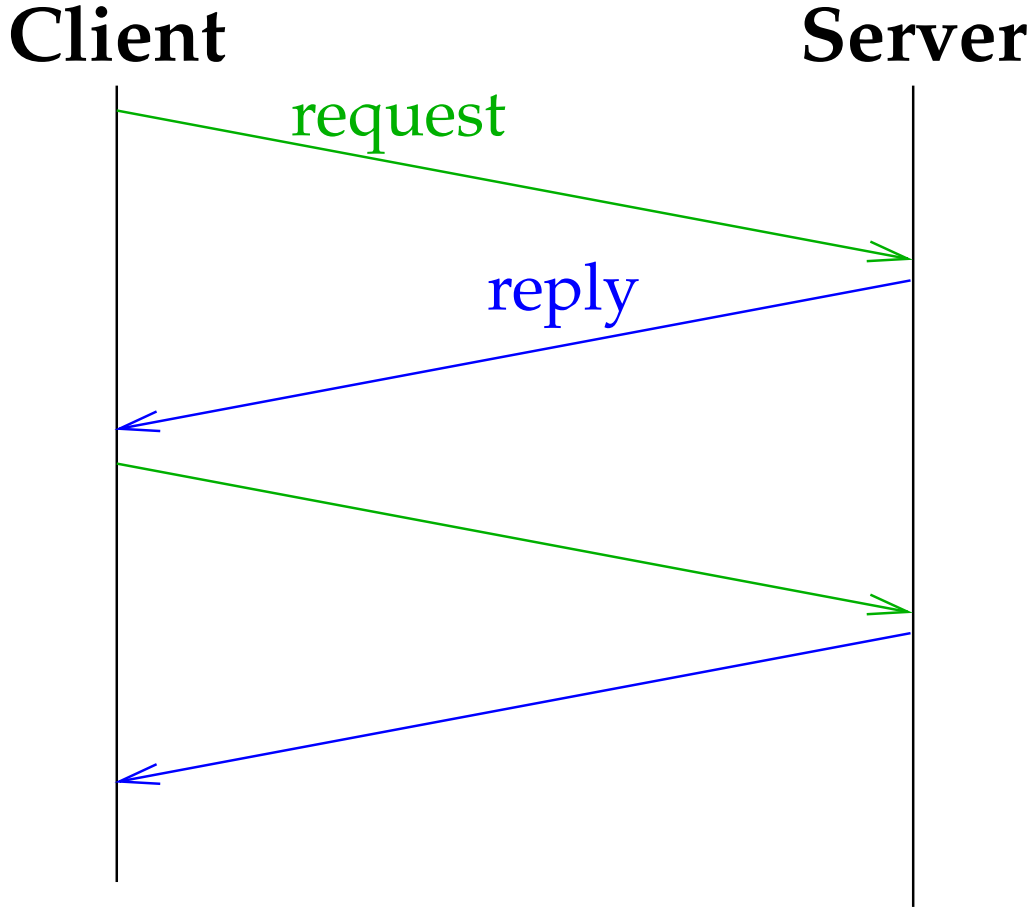
EOF in more detail

- **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

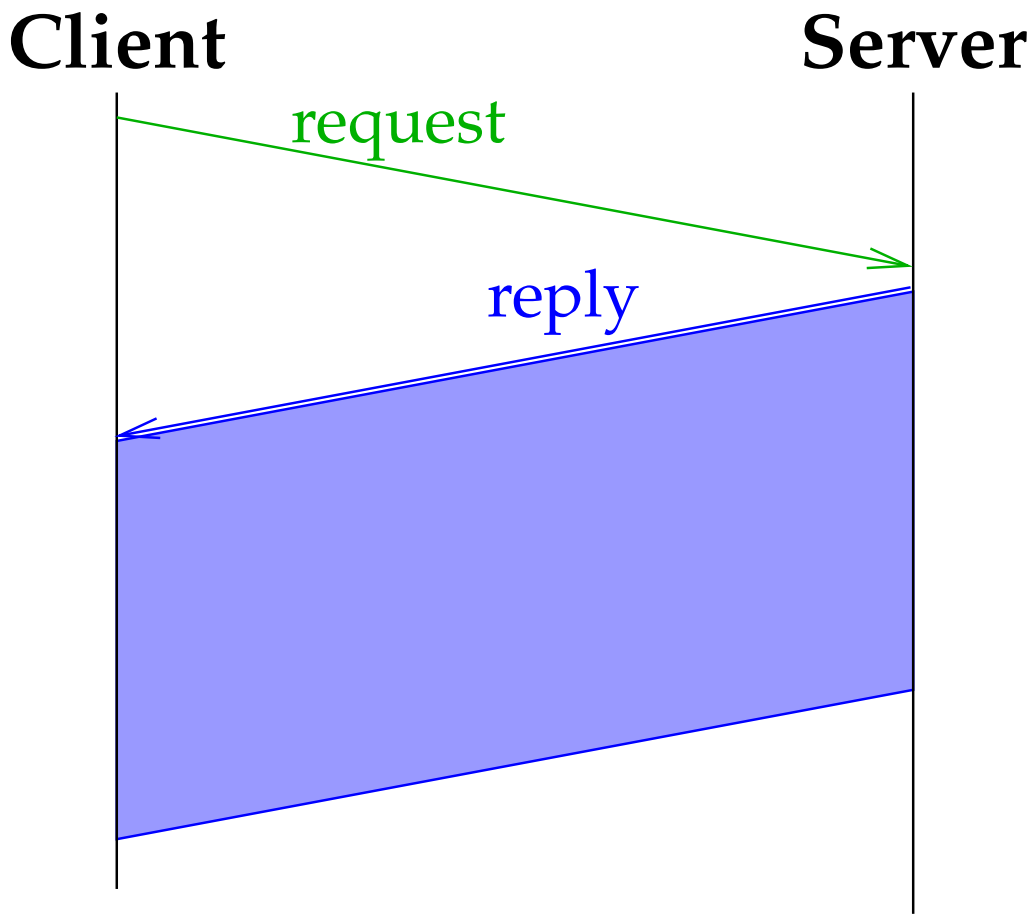
- `int shutdown (int fd, int how);`
  - Shuts down a socket w/o closing file descriptor
  - `how`: 0 = reading, 1 = writing, 2 = both
  - Note: Applies to *socket*, not descriptor—so copies of descriptor (through `dup` or `fork` affected)
  - Note 2: With TCP, can't detect if other side shuts for reading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

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 transmit**
  - 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?**
  - Look at network applications next lecture

Today's Lecture

- **Basic networking abstractions**
  - Protocols
  - OSI layers and the Internet Hourglass
- **Transport protocols: TCP and UDP**
- **Client TCP socket programming**
- **Protocol performance tradeoffs**
- **Next lecture: applications (HTTP, BitTorrent, etc.) and server socket programming**

Structure of Rest of Class

- **IP and above (4 weeks)**
  - Application layers
  - Network layer: IP and routing
  - Transport layer: TCP and congestion control
  - Naming, address translation, and content distribution
- **Below IP (2 weeks)**
  - Link layers and hardware
  - Wireless
- **Advanced topics (2 weeks)**
  - Network coding
  - Security
  - Multimedia