Lecture 2: Applications and Application Programming

Administrivia

- **We're in a new room**
- **If you are a Masters student, you need to enroll for 3, not 4 credits**
- **Prof. Levis will not have regular office hours next week (will be in Redmond)**

Overview

- **End-to-end model**
- **Applications: a lot of different ways to use TCP**
  - Telnet
  - Web/HTTP
  - BitTorrent
  - Skype
- **Application programming**

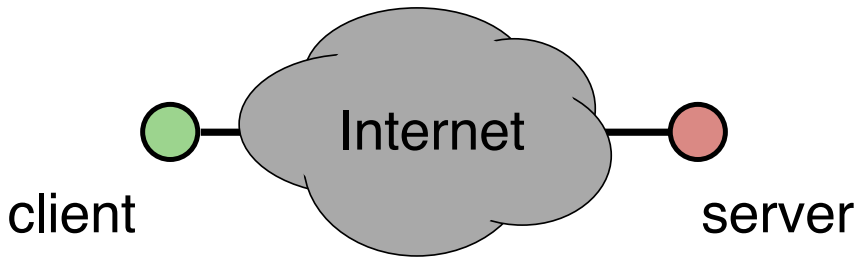
Review

- **The Internet is a network of networks**
- **Hourglass / narrow waist: host-host delivery**
  - IP protocol to deliver packets to hosts
  - Transport above for service/program demultiplexing
  - Link layer below for a single hop
- **Sockets are system call interface for communication**

Applications

- **How services use sockets to provide useful functionality**
- **“the intelligence is end to end rather than hidden in the network.” (RFC 1958)**
- **Two basic questions**
  - What do you send
  - Whom do you send it to

An Application View of the World



- **Why doesn't the network help more?**
  - Compress data
  - Reformat/improve requests
  - Respond with cached data

End-to-End Model

- Saltzer, Reed, and Clark, 1984
- "The function in question can completely and correctly be implemented only with the knowledge and help of the application standing at the end points of the communication system. Therefore, providing that questioned function as a feature of the communication system itself is not possible. (Sometimes an incomplete version of the function provided by the communication system may be useful as a performance enhancement.). We call this ... 'the end-to-end argument."'
- Example: file transfer

File Transfer

- **What can go wrong?**
- **How can we be sure a file arrives successfully?**
- **Why do something in the middle?**

Strong End-to-End

- **"The network's job is to transmit datagrams as efficiently and flexibly as possible. Everything else should be done at the fringes.." (RFC 1958)**
- **Why?**

Net Neutrality

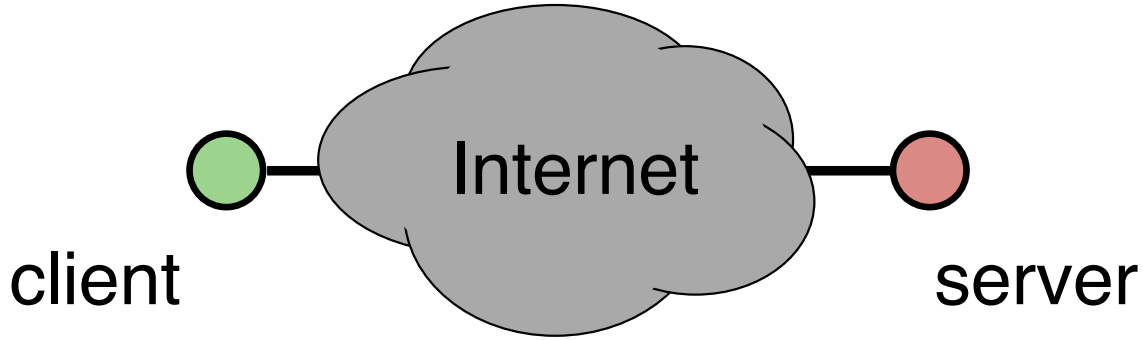
- **Problem: Preferential treatment of some traffic over others**
- **Allows IP service providers to control applications**
- **Provider goals are not the same as end user's**
- **Comcast spurious BitTorrent RSTs (now fixed)**
- **April 17, FCC is holding a hearing here at Stanford!**

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?**

Telnet

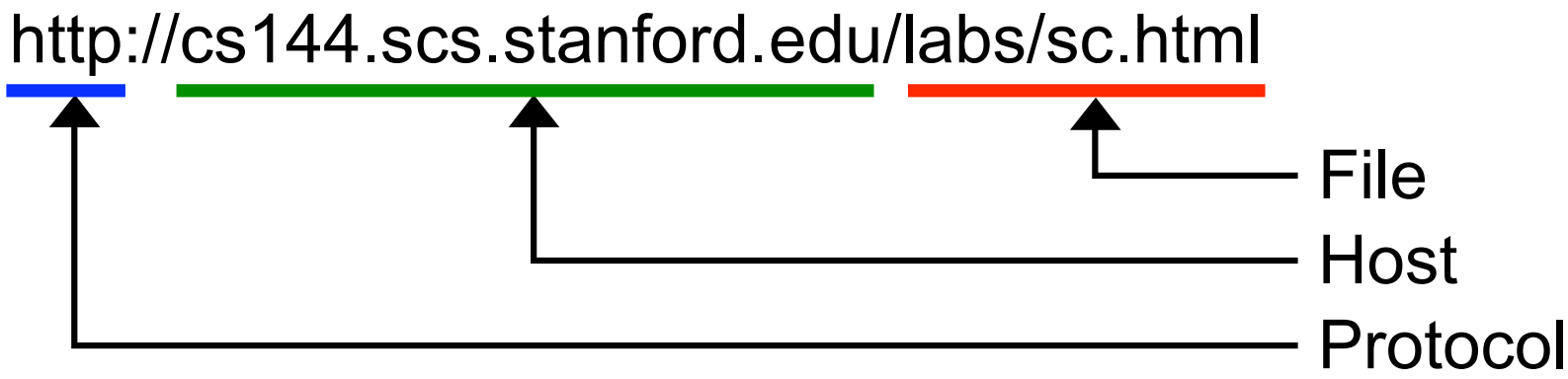
- **Server listens on port 23**
- **On connection, provides a login prompt**
- **Allows insecure remote logins**
- **Telnet client is a text interface**
  - Allows session management
  - Handles terminal control signals, etc.
- **What matters most?**



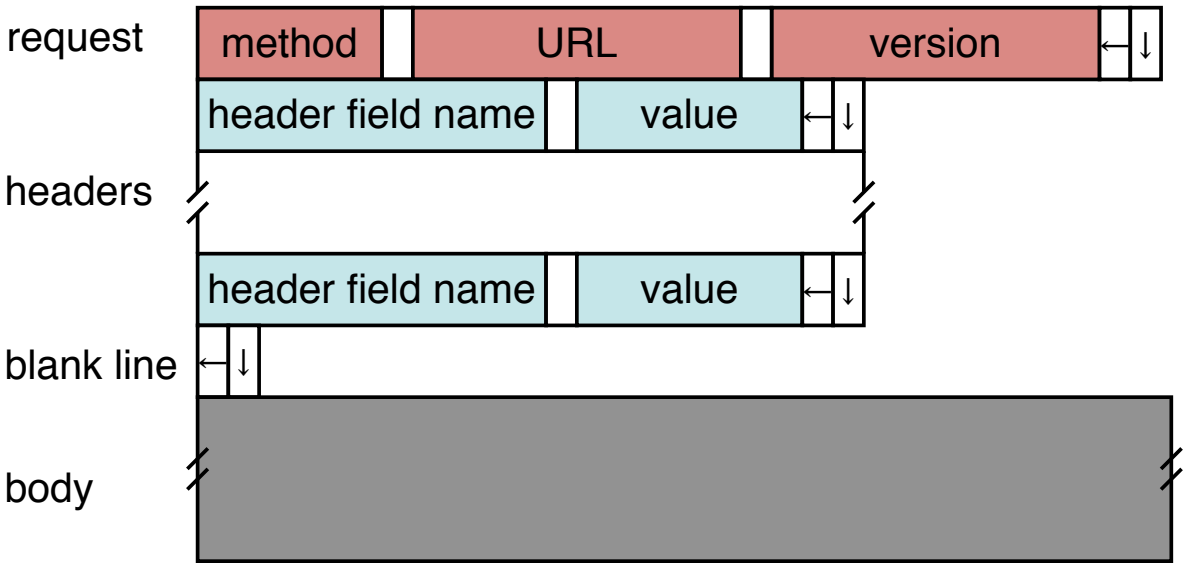
Web/HTTP

- Server lists on port 80
- On connection, waits for a request
- Protocol (but not data...) is in ASCII
- Sends response, maybe closes connection (client can ask it to stay open)

Parsing a URL



HTTP Request Format

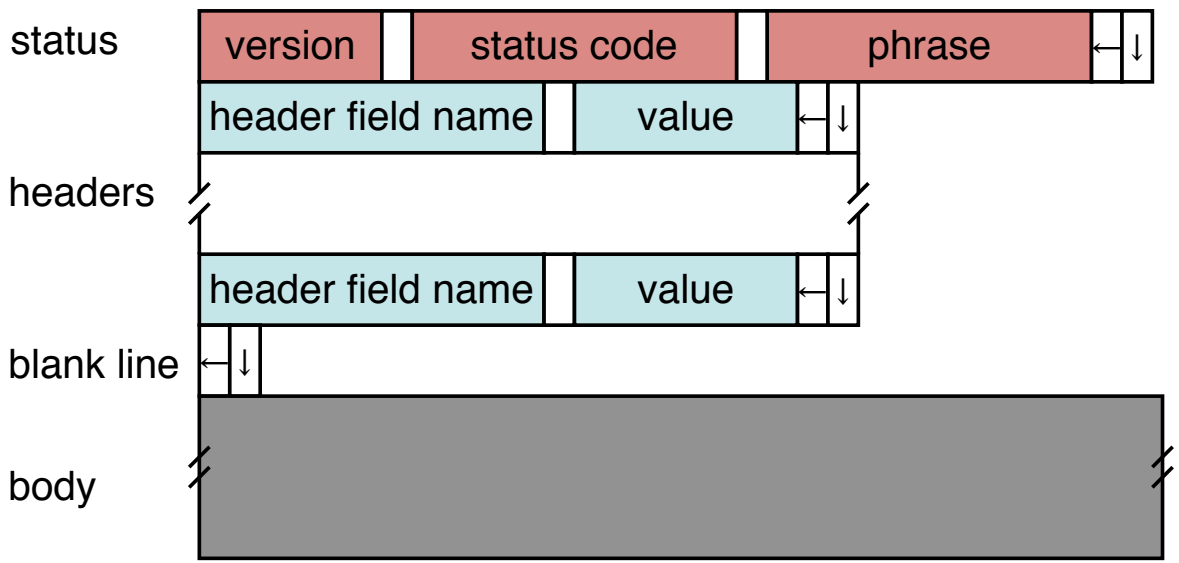


- **Request types: GET, PUT, POST, HEAD, PUT, DELETE**
- **A trivial request:** `GET / HTTP/1.0`
- **A browser request:** `http://localhost:8000`

A Browser Request

```
GET / HTTP/1.1
Host: localhost:8000
User-Agent: Mozilla/5.0 (Macinto ...
Accept: text/xml,application/xm ...
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
```

HTTP Response Format



- **1xx codes: Informational**
- **2xx codes: Successes**
- **3xx codes: Redirection**
- **4xx codes: Client Error, 5xx codes: Server Error**

www.scs.stanford.edu Response

```
HTTP/1.1 200 OK
Date: Thu, 03 Apr 2008 21:09:57 GMT
Server: Apache
Last-Modified: Thu, 03 Jan 2008 01:05:54 GMT
ETag: "a577678157a762e3d46afd4038d9a35bf2ae9386"
Accept-Ranges: bytes
Content-Length: 3586
Connection: close
Content-Type: text/html

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
          "http://www.w3.org/TR/html4/strict.dtd">
...
```

HTTP Performance

- **What matters most?**
- **Different kinds of requests**
  - Lots of small requests (loading a page)
  - Big request (fetching a download)
- **Require different solutions**

Small Requests

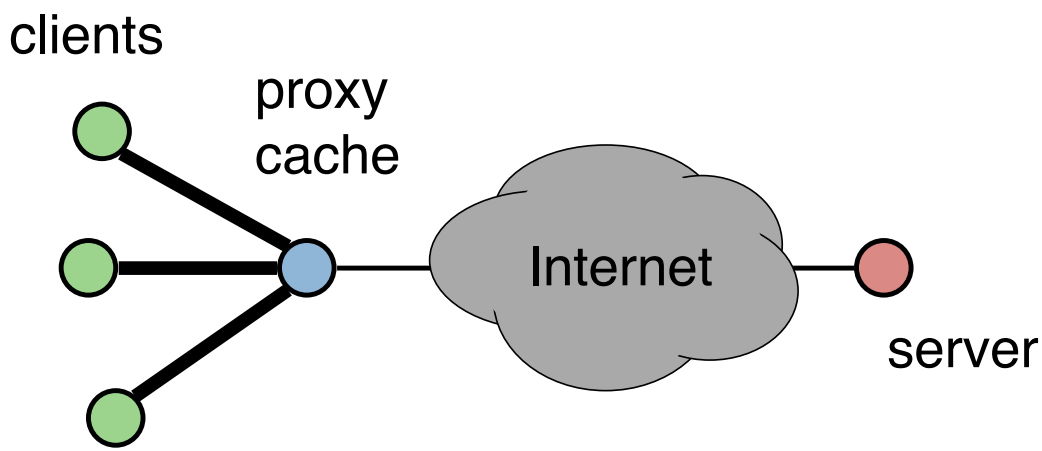
- **Latency matters**
- **Governed by round-trip-time (RTT) between hosts**
- **Two major causes:**
  - Opening a TCP connection
  - Data response-request
- **Solution: persistent connections**

Browser Request, Revisited

```
GET / HTTP/1.1
Host: localhost:8000
User-Agent: Mozilla/5.0 (Macinto ...
Accept: text/xml,application/xm ...
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
```

Big Requests

- **Problem is throughput on edge link**
- **Use an *HTTP proxy cache***
  - Can also improve latency!



Stale Caches

- **Items in the cache can go stale (you don't want to read yesterday's paper)**
- **Cache needs a way to conditionally ask for a document**
- **Issue a conditional GET (`If-modified-since` header)**
  - Server can reply with a `304 Not Modified`

```
GET / HTTP/1.1
Host: cs144.scs.stanford.edu
If-modified-since: Wed, 2 April 2008 08:00:00
```

Client-Server vs. Peer-to-Peer

- **Server can be a bottleneck**
  - Download time can scale $O(n)$ with n clients
  - Scaling up server bandwidth can be expensive (CDNs, lecture 9)
  - Slashdotting/flash crowds
- **Peer-to-peer: get a bunch of end-hosts to collaboratively distribute content**
- **A common peer-to-peer challenge is finding whom to collaborate with**

BitTorrent

- **Torrent file (`.torrent`) describes file to download**
  - Names *tracker*, URL that describes who is participating
  - File length, piece length, SHA1 hashes of pieces
  - Additional metadata (who created torrent, etc.)
- **Client contacts tracker, starts communicating with peers**


```
d8:announce47:http://vip.tracker.thepiratebay.org:80/announce7:comment20:
THANKS FOR SEEDING!10:created by13:uTorrent/177013:creation
datei1207365453e8:encoding5:UTF-84:infod6:lengthi839833600e4:
name18:XXX XXXXXXXXX.avi12:piece lengthi1048576e6:pieces16020:U ....
```

Pieces and Sub-pieces

- **BitTorrent breaks up a file into N pieces**
  - For throughput, pieces are large: 256KB-1MB
  - For latency, broken into subpieces
- **Hashes of pieces in torrent provide end-to-end integrity (HBO/Rome)**
  - Hashes computes a short summary of a piece of data
  - Cryptographically strong hashes: hard to find collisions or data that produces a hash (more in lectures 15+16)

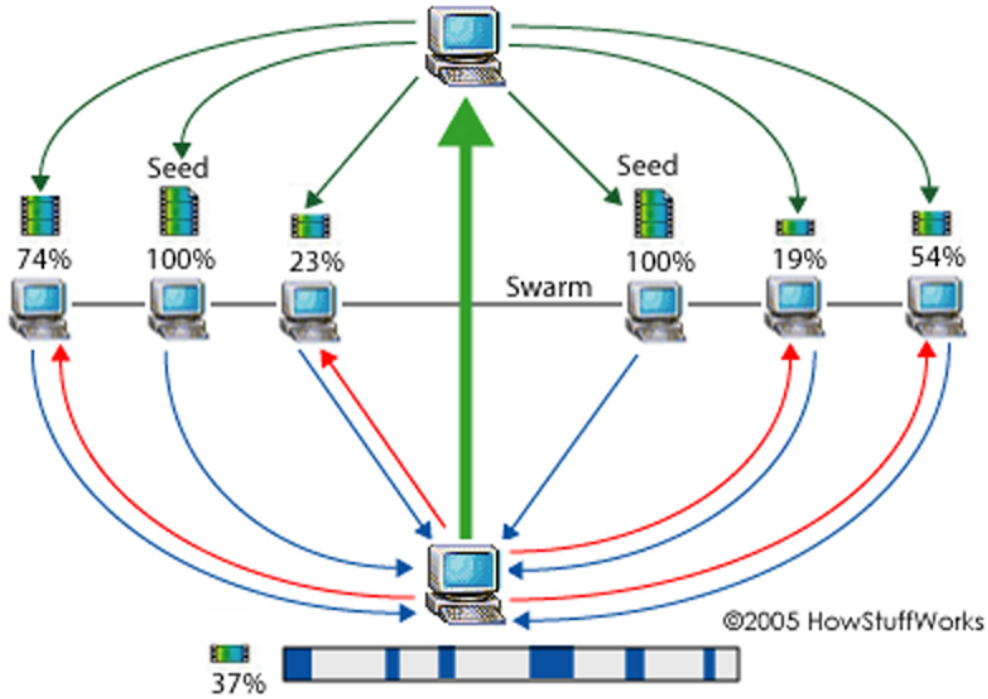
Whom To Talk To?

- **Uses a *Tit-for-Tat* (TFT) policy: upload only to those who give you data**
- **Most peers are "choked" and get no data**
- **Order unchoked peers by download rate, choke all but P best (e.g., 4, $\sqrt{C}$)**
- **Occasionally unchoke a random peer (might find its way into P best)**

What to Say?

- **Peers exchange metadata on what pieces they have**
- **Download rarest pieces**
- **When down to the last few pieces, ask for them from multiple peers**

BitTorrent Communication



BitTyrant

- **Optional reading: 2007 research paper**
- **Take advantage of altriusm**
  - Many peers give more than they take
  - Rate-limit yourself just enough to be unchoked
  - Connect to more peers instead of giving each peer more
- **Leads to a median 70% performance gain!**

2-minute stretch

Skype

- **Real-time communication (voice, IM, etc.)**
- **Two major challenges**
  - Finding what host a user is on
  - Being able to communicate with arbitrary hosts
- **All Skype traffic is encrypted**
  - Researchers have mostly reverse-engineered the protocol
  - A few details are still unknown

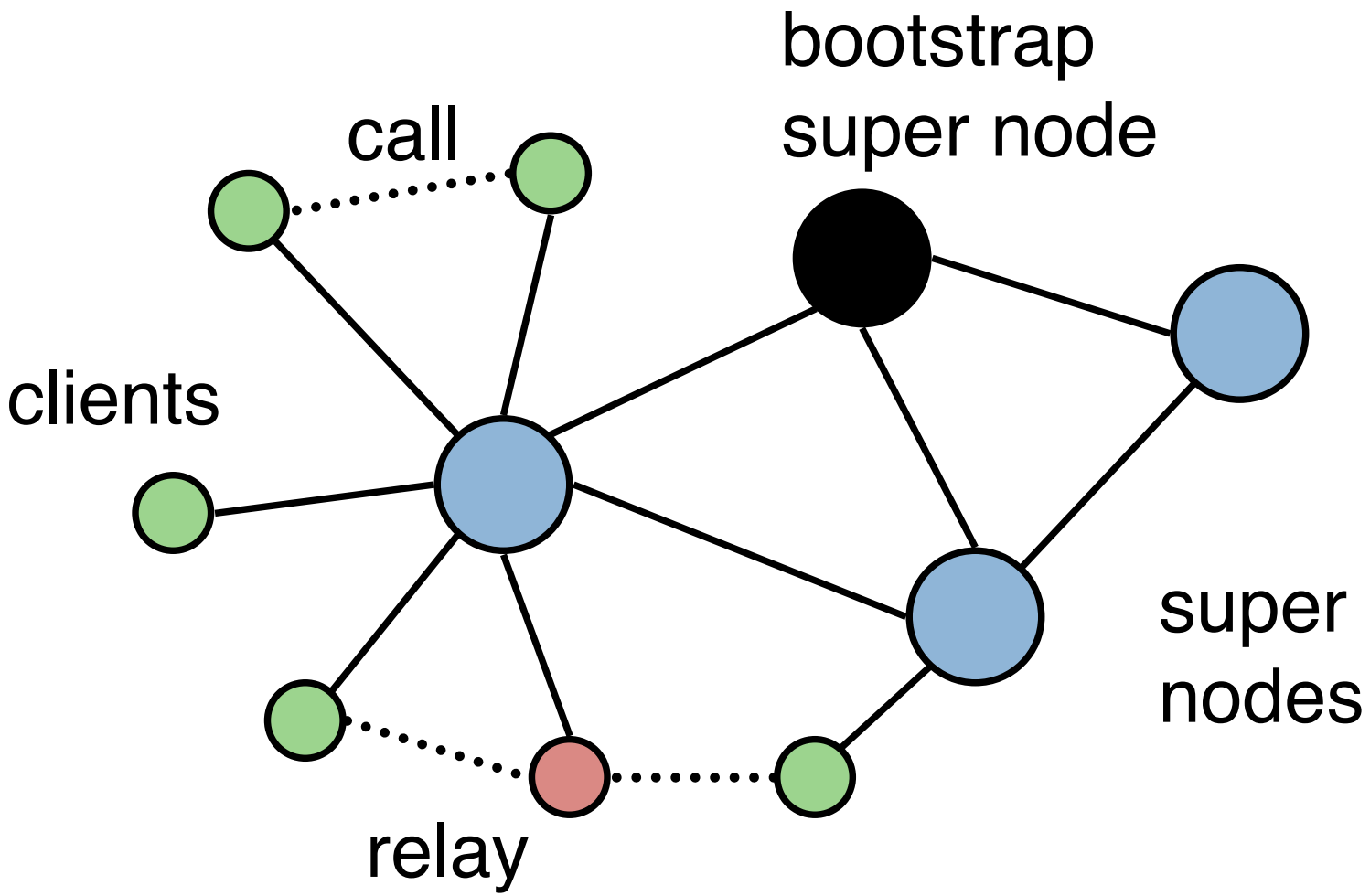
Finding a User

- **Distributed index across *super-peers***
  - Super-peers cannot be behind a NAT
  - There are 7 bootstrap super-peers: `66.235.180.9`, `66.235.181.9`, `212.72.49.143`, `195.215.8.145`, `64.246.49.60`, `64.246.49.61`, `64.246.48.24`
- **Uncertain exactly how this index is organized**
- **As number of peers grows, so does number of super-peers maintaining the index (or Skype, Inc. itself can add them)**

Talking to End-Hosts

- **What if neither host allows incoming connections?**
  - This is a common issue with NATs
  - Skype is more fragile to this that BitTorrent, because BitTorrent can connect to any peer, while Skype wants to connect to a specific peer
- **Skype uses *relays***

Skype Communication Architecture



Skype Summary

- **Uses a distributed index to find end-hosts**
- **Clients communicate directly, use relays when NATs are a problem**

Application Programming

Creating processes

- `int fork (void);`
  - Create new process that is exact copy of current one
  - Returns *process ID* of new proc. in "parent"
  - Returns 0 in "child"
- `int waitpid (int pid, int *stat, int opt);`
  - `pid` – process to wait for, or -1 for any
  - `stat` – will contain exit value, or signal
  - `opt` – usually 0 or `WNOHANG`
  - Returns process ID or -1 on error

Deleting processes

- `void exit (int status);`
  - Current process ceases to exist
  - `status` shows up in `waitpid` (shifted)
  - By convention, `status` of 0 is success, non-zero error
- `int kill (int pid, int sig);`
  - Sends signal `sig` to process `pid`
  - `SIGTERM` most common value, kills process by default (but application can catch it for "cleanup")
  - `SIGKILL` stronger, kills process always

Running programs

- `int execve (char *prog, char **argv, char **envp);`
  - `prog` – full pathname of program to run
  - `argv` – argument vector that gets passed to `main`
  - `envp` – environment variables, e.g., `PATH`, `HOME`
- **Generally called through a wrapper functions**
- `int execvp (char *prog, char **argv);`
  - Search `PATH` for prog
  - Use current environment
- `int execlp (char *prog, char *arg, ...);`
  - List arguments one at a time, finish with `NULL`
- **Example:** `minish.c`

File descriptor numbers

- **File descriptors are inherited by processes**
  - When one process spawns another, same fds by default
- **Descriptors 0, 1, and 2 have special meaning**
  - 0 – "standard input" (`stdin` in ANSI C)
  - 1 – "standard output" (`stdout, printf` in ANSI C)
  - 2 – "standard error" (`stderr, perror` in ANSI C)
  - Normally all three attached to terminal
- **Example:** `type.c`
- **Can use** `dup2` **to make a program's stdin/stdout/stderr be other I/O channels than terminals (e.g., a socket)**

Pipes

- `int pipe (int fds[2]);`
  - Returns two file descriptors in `fds[0]` and `fds[1]`
  - Writes to `fds[1]` will be read on `fds[0]`
  - When last copy of `fds[1]` closed, `fds[0]` will return EOF
  - Returns 0 on success, -1 on error
- **Operations on pipes**
  - `read/write/close` – as with files
  - When `fds[1]` closed, `read(fds[0])` returns 0 bytes
  - When `fds[0]` closed, `write(fds[1])`:
    - Kills process with `SIGPIPE`, or if blocked
    - Fails with EPIPE
- **For example code, see `pipesh.c` on web site**