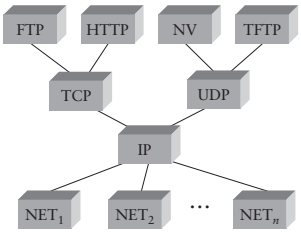
Administrivia

- **Important correction about 3 vs. 4 units:**
  - All students should take CS144 for 4 credits if possible
  - Undergraduates *must* enroll for 4 credits
  - Graduate students *may* enroll for 3 credits if need be to stay under unit cap
  - Workload is identical for 3 vs. 4 credits
- **Section tomorrow: 11:00-11:50 AM in Huang 018**
- **Lab 1 on web page (updated this morning)**
  - Due exactly one week from now. Free extension to 12AM if you attend lecture (or, for SCPD, watch before 12AM)
  - Else, up to 90% credit if you turn in by Saturday night 10/2
  - Recall Lab 1 grade is $\max(\text{Lab 1}, \text{Lab 2})$

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/process demultiplexing
  - Link layer below for a single hop



- **Sockets are system call interface for communication**

Implementing reliability

- **Lab 1 requires implementing a reliable transport**
- **Must overcome several challenges**
  - Network can drop, duplicate, corrupt, and re-order packets
- **Techniques you will employ**
  - Checksums expose bit errors (more in future lectures)
  - Acknowledgments, timeouts, and retransmissions correct for dropped packets
  - Sequence numbers reveal duplicate/re-ordered packets
- **Next: A few slides to get you started**

Acknowledgements and Timeouts

- **Stop and wait approach to reliability**
  - Send message, wait
  - Receive message, send ACK
  - Receive ACK, send next message
  - Don't receive ACK, timeout and retransmit
- **Example: Air traffic control**
  - Pilots acknowledge messages by repeating them
- **Okay if messages are never re-ordered/duplicated**
  - True for air traffic control, but not for Internet
  - For Internet, also need a sequence number in packets...

Reliable transport and EOF

- **End-of-file (EOF) condition means no more input**
  - Detect EOF when `read` returns 0 bytes
- **Consider simple TCP client-server application**
  - Client sends request followed by EOF
  - Server reads request, sees EOF, sends response
- **What happens at end of request message?**
  - Client needs server to see EOF as end-of-request marker
  - One way to send EOF: call `close` on socket fd
  - But still need to read server reply after sending EOF
- **Many network applications detect & use EOF**
  - Common error: "leaking" file descriptor via `fork`, so not closed (and no EOF) when you exit

`shutdown` system call

- `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, so in practice 1 is only useful `how` argument
- **Play with `uc` from Lab 1 to see this**
  - Pressing Ctrl-D at beginning of line sends EOF
  - `uc` reads EOF from standard input, uses `shutdown` to write EOF to TCP socket

Applications

- **Simple socket abstraction is very powerful**
  - Applications can use them in many ways
  - We'll go through four examples
- **"the intelligence is end to end rather than hidden in the network."**
  – **[RFC 1958]** (architectural principles doc)
- Two basic questions for an application
  - 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/translate requests
  - Respond with cached data
  - Secure communications
  - Migrate connections across networks

End-to-End Model

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 line of reasoning… "the end-to-end argument."

– Saltzer, Reed, and Clark, 1984

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?**

Example: File Transfer

- **What can go wrong?**
  - File might not arrive at receiver
  - File could be corrupted in transit
- **How can we be sure a file arrives successfully?**
  - Send acknowledgment to ensure file arrived at destination
  - Use checksum to ensure data has not been corrupted
- **Why do something in the middle?**
  - Network often drops packets because of congestion, sometimes corrupts packets because of transmission errors
  - Cheaper to resend one packet than whole file
  - …or to re-send over one link than through whole network

Link-by-link reliability

- **Link-by-link reliability is not sufficient**
  - MIT network did link-by-link ack and checksum
  - Thought transmission was most likely source or errors
  - But problem arose from bad RAM in router
  - Ended up corrupting a lot of MULTICS source code
- **Principle applies even above the network layer**
  - E.g., Google learned disk error-correction insufficient… GFS keeps its own application-layer checkums

End-to-end & net neutrality

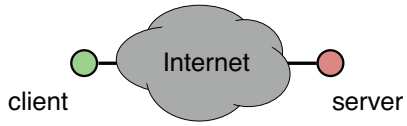
- **"Dumb" network clearly beneficial…**
  - But what if ISPs *want* to do complex things?
- **Require ISPs to treat everyone's traffic equally?**
- **Pro: prevent extortion from last-mile duopoly**
  - Create a hugely profitable business (next Google)?
  - Seeing $$$s, ISPs could charge you to reach their customers
- **Con: reduces ISP flexibility to cope with change**
  - E.g., new applications that clog the network
- **Current FCC strongly in favor of Net neutrality**

FCC net neutrality history

- **2002: Classifies cable modems as information services**
  - Much less regulated than telephone or TV
- **2005: FCC issues Internet Policy Statement:**
  - Users should be able to 1) access any lawful content, 2) run any applications, 3) use any devices, and 4) enjoy competition between network, application, and content providers.
- **2007: Comcast forges RSTs to BitTorrent**
- **2008: FCC finds Comcast in violation of policy**
- **2009: FCC proposes 2 more network guidelines**
  - ISPs should 5) treat content/applications/services in nondiscriminatory manner, and 6) provide transparency
- **2010: Court finds FCC lacks authority to regulate ISPs**

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 for performance?**

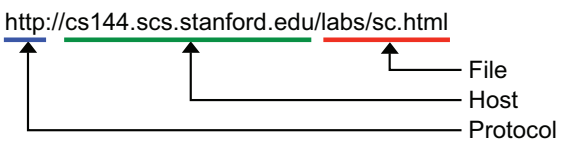


2-minute stretch

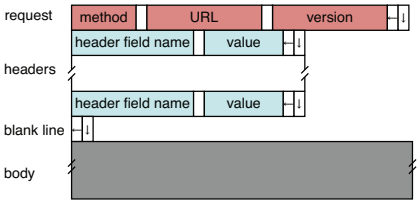
Web/HTTP

- **Server listens 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



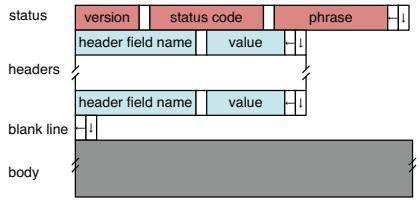
- **Request types: GET, POST, HEAD, PUT, DELETE**
- **A URL given to browser:** `http://localhost:8000/`
- **Resulting request:** `GET / HTTP/1.1`
  - Someday, requests will contain the full URL not just path

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
```

[LiveHTTPHeaders for Firefox is good way to see this]

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 for performance?**
- **Depends on types of requests**
  - Lots of small requests (loading a page)
  - Big request (fetching a download)
- **Require different solutions**

Small Requests

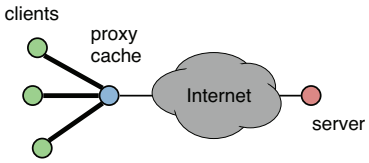
- **Latency matters**
- **Governed by round-trip-time (RTT) between hosts**
- **Two major causes:**
  - Opening a TCP connection
    (TCP requires 7 messages even with no data, including one round-trip even before you can send request)
  - Actually sending request and receiving response
- **Mitigate first cost with 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, 02 Apr 2008 08:00:00 GMT
```

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 12)
  - Slashdotting/flash crowds
- **Peer-to-peer: get a bunch of end-hosts to distribute content collaboratively**
- **A common peer-to-peer challenge is finding whom to collaborate with**

BitTorrent

- **Torrent file (`.torrent`) describes file to download**
  - Names *tracker*, server tracking who is participating
  - File length, piece length, SHA1 hashes of pieces
  - Additional metadata (who created torrent, etc.)
  - Also specifies tracker
- **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**
  - Hash computes a short summary of a piece of data
  - Cryptographically strong hashes: hard to find collisions or data that produces a hash (more in Security lectures)

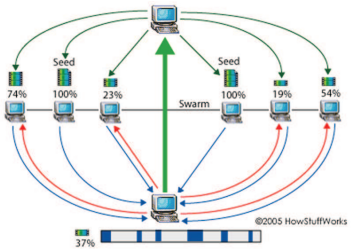
Whom To Talk To?

- 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 [Piatek]

- 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!

BitTorrent Today: Research (SIGCOMM/NSDI/OSDI)

- Can a coordinator improve swarm performance by load-balancing peers?
- What do swarms look like in terms of communication patterns and size?
- How does one swarm streaming video?

BitTorrent Today: Engineering (IETF)

- Locality can improve performance (lower RTT)
- Hard for clients to know who might be local
- Can ISPs collaborate to reduce transit costs and improve performance?
- Issues of contributory infringement...
- Application Layer Transport Optimization (alto)
- Low Extra Delay Background Transport (ledbat)

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

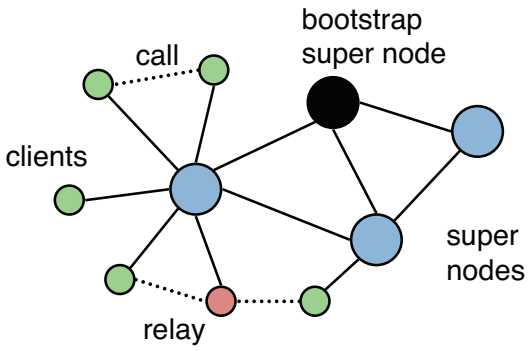
Finding a User

- **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, 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**