Topic 6 – Applications

• Overview

• Infrastructure Services (DNS)

• Traditional Applications (web)

• Multimedia Applications (SIP)

• P2P Networks
Client-server architecture

server:
- always-on host
- permanent IP address
- server farms for scaling

clients:
- communicate with server
- may be intermittently connected
- may have dynamic IP addresses
- do not communicate directly with each other
Pure P2P architecture

- *no* always-on server
- arbitrary end systems directly communicate
- peers are intermittently connected and change IP addresses

Highly scalable but difficult to manage
Hybrid of client-server and P2P

Skype
– voice-over-IP P2P application
– centralized server: finding address of remote party:
– client-client connection: direct (not through server)

Instant messaging
– chatting between two users is P2P
– centralized service: client presence detection/location
  • user registers its IP address with central server when it comes online
  • user contacts central server to find IP addresses of buddies
Addressing processes

- to receive messages, process must have **identifier**
- host device has unique 32-bit IP address
- **Q:** does IP address of host on which process runs suffice for identifying the process?
  - **A:** No, *many* processes can be running on same host
- **identifier** includes both IP address and port numbers associated with process on host.
- Example port numbers:
  - HTTP server: 80
  - Mail server: 25
- to send HTTP message to yuba.stanford.edu web server:
  - IP address: 171.64.74.58
  - Port number: 80
- more shortly...
Recall: Multiplexing is a service provided by (each) layer too!

Multiplexing

Demultiplexing

Application: one web-server multiple sets of content
Host: one machine multiple services
Network: one physical box multiple addresses (like vns.cl.cam.ac.uk)

UNIX: /etc/protocols = examples of different transport-protocols on top of IP

UNIX: /etc/services = examples of different (TCP/UDP) services – by port

(These files are an example of a (static) approach to name services)
App-layer protocol defines

• Types of messages exchanged,
  – e.g., request, response
• Message syntax:
  – what fields in messages & how fields are delineated
• Message semantics
  – meaning of information in fields
• Rules for when and how processes send & respond to messages

Public-domain protocols:
• defined in RFCs
• allows for interoperability
  • e.g., HTTP, SMTP

Proprietary protocols:
• e.g., Skype
What transport service does an app need?

Data loss

- some apps (e.g., audio) can tolerate some loss
- other apps (e.g., file transfer, telnet) require 100% reliable data transfer

Timing

- some apps (e.g., Internet telephony, interactive games) require low delay to be “effective”

Throughput

- some apps (e.g., multimedia) require minimum amount of throughput to be “effective”
- other apps (“elastic apps”) make use of whatever throughput they get

Security

- Encryption, data integrity, ...

Mysterious secret of *Transport*

- There is more than sort of *transport* layer

Shocked?

  I seriously doubt it...

Recall the two most common TCP and UDP
Naming

• Internet has one global system of addressing: IP
  – By explicit design

• And one global system of naming: DNS
  – Almost by accident

• At the time, only items worth naming were hosts
  – A mistake that causes many painful workarounds

• Everything is now named relative to a host
  – Content is most notable example (URL structure)
Logical Steps in Using Internet

• Human has name of entity she wants to access
  – Content, host, etc.

• Invokes an application to perform relevant task
  – Using that name

• App invokes DNS to translate name to address

• App invokes transport protocol to contact host
  – Using address as destination
Addresses vs Names

• Scope of relevance:
  – App/user is primarily concerned with names
  – Network is primarily concerned with addresses

• Timescales:
  – Name lookup once (or get from cache)
  – Address lookup on each packet

• When moving a host to a different subnet:
  – The address changes
  – The name does not change

• When moving content to a differently named host
  – Name and address both change!
Relationship Between Names & Addresses

• Addresses can change underneath
  – Move www.bbc.co.uk to 212.58.246.92
  – Humans/Apps should be unaffected

• Name could map to multiple IP addresses
  – www.bbc.co.uk to multiple replicas of the Web site
  – Enables
    • Load-balancing
    • Reducing latency by picking nearby servers

• Multiple names for the same address
  – E.g., aliases like www.bbc.co.uk and bbc.co.uk
  – Mnemonic stable name, and dynamic canonical name
    • Canonical name = actual name of host
Mapping from Names to Addresses

• Originally: per-host file /etc/hosts
  – SRI (Menlo Park) kept master copy
  – Downloaded regularly
  – Flat namespace

• Single server not resilient, doesn’t scale
  – Adopted a distributed hierarchical system

• Two intertwined hierarchies:
  – Infrastructure: hierarchy of DNS servers
  – Naming structure: www.bbc.co.uk
Domain Name System (DNS)

• Top of hierarchy: Root
  – Location hardwired into other servers

• Next Level: Top-level domain (TLD) servers
  – .com, .edu, etc.
  – .uk, .au, .to, etc.
  – Managed professionally

• Bottom Level: Authoritative DNS servers
  – Actually do the mapping
  – Can be maintained locally or by a service provider
Distributed Hierarchical Database

Top-Level Domains (TLDs)

- com
- edu
- org
- ac
- uk
- zw
- arpa
- bar
- west
- east
- foo
- my
- ac
- cam
- cl
- in-addr

my.east.bar.edu
cl.cam.ac.uk

generic domains
country domains
DNS Root

- Located in Virginia, USA
- How do we make the root scale?
DNS Root Servers

- 13 root servers (see http://www.root-servers.org/)
  - Labeled A through M
- Does this scale?

A Verisign, Dulles, VA
B USC-ISI Marina del Rey, CA
C Cogent, Herndon, VA
D U Maryland College Park, MD
E NASA Mt View, CA
F Internet Software Consortium
  Palo Alto, CA
G US DoD Vienna, VA
H ARL Aberdeen, MD
I Autonomica, Stockholm
J Verisign
K RIPE London
L ICANN Los Angeles, CA
M WIDE Tokyo
DNS Root Servers

• 13 root servers (see http://www.root-servers.org/)
  – Labeled A through M
• Replication via any-casting (localized routing for addresses)

A Verisign, Dulles, VA
C Cogent, Herndon, VA (also Los Angeles, NY, Chicago)
D U Maryland College Park, MD
G US DoD Vienna, VA
H ARL Aberdeen, MD
J Verisign (21 locations)

K RIPE London (plus 16 other locations)
I Autonomica, Stockholm (plus 29 other locations)

E NASA Mt View, CA
F Internet Software Consortium, Palo Alto, CA (and 37 other locations)

B USC-ISI Marina del Rey, CA
L ICANN Los Angeles, CA

M WIDE Tokyo plus Seoul, Paris, San Francisco
Using DNS

• Two components
  – Local DNS servers
  – Resolver software on hosts

• Local DNS server (“default name server”)
  – Usually near the endhosts that use it
  – Local hosts configured with local server (e.g., /etc/resolv.conf) or learn server via DHCP

• Client application
  – Extract server name (e.g., from the URL)
  – Do gethostbyname() to trigger resolver code
How Does Resolution Happen?

**(Iterative example)**

Host at cl.cam.ac.uk wants IP address for www.stanford.edu.

- Iterated query:
  - Host enquiry is delegated to local DNS server.
  - Consider transactions 2–7 only.
  - Contacted server replies with name of next server to contact.
  - "I don’t know this name, but ask this server.”

Diagram:

- 2. Host enquiry is delegated to local DNS server dns.cam.ac.uk.
- 3. Local DNS server contacts root DNS server.
- 4. Root DNS server helps by delegating to TLD DNS server.
- 5. TLD DNS server contacts authoritative DNS server.
- 6. Authoritative DNS server responds with name of next server to contact.
- 8. www.stanford.edu is contacted by requesting host cl.cam.ac.uk.
DNS name resolution **recursive** example

**recursive query:**
- puts burden of name resolution on contacted name server
- heavy load?

Diagram:
- Root DNS server
- Local DNS server: `dns.cam.ac.uk`
- Authoritative DNS server: `dns.stanford.edu`
- TLD DNS server
- Requesting host: `cl.cam.ac.uk`
- www.stanford.edu
Recursive and Iterative Queries - **Hybrid** case

- **Recursive** query
  - Ask server to get answer for you
  - E.g., requests 1,2 and responses 9,10

- **Iterative** query
  - Ask server who to ask next
  - E.g., all other request-response pairs

---

Recursive query:
- Ask server to get answer for you
- E.g., requests 1,2 and responses 9,10

Iterative query:
- Ask server who to ask next
- E.g., all other request-response pairs

Diagram:
- **Requesting host**: `my-host.cl.cam.ac.uk`
- **Authoritative DNS server**: `dns.stanford.edu`
- **TLD DNS server**: `dns.cam.ac.uk`
- **Site DNS server**: `dns.cam.ac.uk`
- **Root DNS server**: `root DNS server`
DNS Caching

• Performing all these queries takes time
  – And all this before actual communication takes place
  – E.g., 1-second latency before starting Web download

• **Caching** can greatly reduce overhead
  – The top-level servers very rarely change
  – Popular sites (e.g., www.bbc.co.uk) visited often
  – Local DNS server often has the information cached

• How DNS caching works
  – DNS servers cache responses to queries
  – Responses include a “time to live” (TTL) field
  – Server deletes cached entry after TTL expires
Negative Caching

• Remember things that don’t work
  – Misspellings like bbcc.co.uk and www.bbc.com.uk
  – These can take a long time to fail the first time
  – Good to remember that they don’t work
  – ... so the failure takes less time the next time around

• But: negative caching is optional
  – And not widely implemented
Reliability

• DNS servers are replicated (primary/secondary)
  – Name service available if at least one replica is up
  – Queries can be load-balanced between replicas
• Usually, UDP used for queries
  – Need reliability: must implement this on top of UDP
  – Spec supports TCP too, but not always implemented
• Try alternate servers on timeout
  – Exponential backoff when retrying same server
• Same identifier for all queries
  – Don’t care which server responds
DNS Measurements (MIT data from 2000)

- What is being looked up?
  - ~60% requests for A records
  - ~25% for PTR records
  - ~5% for MX records
  - ~6% for ANY records

- How long does it take?
  - Median ~100msec (but 90th percentile ~500msec)
  - 80% have no referrals; 99.9% have fewer than four

- Query packets per lookup: ~2.4
  - But this is misleading....
DNS Measurements (MIT data from 2000)

• Does DNS give answers?
  – ~23% of lookups fail to elicit an answer!
  – ~13% of lookups result in NXDOMAIN (or similar)
    • Mostly reverse lookups
  – Only ~64% of queries are successful!
    • *How come the web seems to work so well?*

• ~63% of DNS packets in unanswered queries!
  – Failing queries are frequently retransmitted
  – 99.9% successful queries have ≤2 retransmissions
DNS Measurements (MIT data from 2000)

• Top 10% of names accounted for ~70% of lookups
  – Caching should really help!

• 9% of lookups are unique
  – Cache hit rate can never exceed 91%

• Cache hit rates ~ 75%
  – But caching for more than 10 hosts doesn’t add much
A Common Pattern…..

• Distributions of various metrics (file lengths, access patterns, etc.) often have two properties:
  – Large fraction of total metric in the top 10%
  – Sizable fraction (~10%) of total fraction in low values

• Not an exponential distribution
  – Large fraction is in top 10%
  – But low values have very little of overall total

• Lesson: have to pay attention to both ends of dist.
• Here: caching helps, but not a panacea
Moral of the Story

• If you design a highly resilient system, many things can be going wrong without you noticing it!

and this is a good thing
Cache Poisoning, an old badness example

- Suppose you are a Bad Guy and you control the name server for foobar.com. You receive a request to resolve www.foobar.com and reply:

  ;; QUESTION SECTION:
  ;www.foobar.com. IN A

  ;; ANSWER SECTION:
  www.foobar.com. 300 IN A 212.44.9.144

  ;; AUTHORITY SECTION:
  foobar.com. 600 IN NS google.com.

  ;; ADDITIONAL SECTION:
  google.com. 5 IN A 212.44.9.155

Evidence of the attack disappears 5 seconds later!

A foobar.com machine, not google.com
DNS and Security

• No way to verify answers
  – Opens up DNS to many potential attacks
  – DNSSEC fixes this

• Most obvious vulnerability: recursive resolution
  – Using recursive resolution, host must trust DNS server
  – When at Starbucks, server is under their control
  – And can return whatever values it wants

• More subtle attack: Cache poisoning
  – Those “additional” records can be anything!
Why is the web so successful?

- What do the web, youtube, facebook, tumblr, twitter, flickr, ..... have in common?
  - The ability to self-publish

- Self-publishing that is easy, independent, *free*

- No interest in collaborative and idealistic endeavor
  - People aren’t looking for Nirvana (or even Xanadu)
  - People also aren’t looking for technical perfection

- Want to make their mark, and find something neat
  - Two sides of the same coin, creates synergy
  - “Performance” more important than dialogue....
Web Components

• Infrastructure:
  – Clients
  – Servers
  – Proxies

• Content:
  – Individual objects (files, etc.)
  – Web sites (coherent collection of objects)

• Implementation
  – HTML: formatting content
  – URL: naming content
  – HTTP: protocol for exchanging content
    Any content not just HTML!
HTML: HyperText Markup Language

• A Web page has:
  – Base HTML file
  – Referenced objects (e.g., images)

• HTML has several functions:
  – Format text
  – Reference images
  – Embed hyperlinks (HREF)
### URL Syntax

\[ \text{protocol} : // \text{hostname} [ : \text{port} ] / \text{directorypath} / \text{resource} \]

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>protocol</strong></td>
<td>http, ftp, https, smtp, rtsp, etc.</td>
</tr>
<tr>
<td><strong>hostname</strong></td>
<td>DNS name, IP address</td>
</tr>
<tr>
<td><strong>port</strong></td>
<td>Defaults to protocol’s standard port</td>
</tr>
<tr>
<td></td>
<td><em>e.g.</em> http: 80, https: 443</td>
</tr>
<tr>
<td><strong>directory path</strong></td>
<td>Hierarchical, reflecting file system</td>
</tr>
<tr>
<td><strong>resource</strong></td>
<td>Identifies the desired resource</td>
</tr>
</tbody>
</table>

Can also extend to program executions:

```
http://us.f413.mail.yahoo.com/ym/ShowLetter?box=%40Bulk&MsgId=2604_1744106_29699_1123_1261_0_289_17_3552_1289957100&Search=&Nhead=f&YY=31454&order=down&sort=date&pos=0&view=a&head=b
```
HyperText Transfer Protocol (HTTP)

- Request-response protocol
- Reliance on a global namespace
- Resource metadata
- Stateless
- ASCII format

```bash
$ telnet www.cl.cam.ac.uk 80
GET /~awm22/win HTTP/1.0
<blank line, i.e., CRLF>
```
Steps in HTTP Request

• HTTP Client initiates TCP connection to server
  – SYN
  – SYNACK
  – ACK
• Client sends HTTP request to server
  – Can be piggybacked on TCP’s ACK
• HTTP Server responds to request
• Client receives the request, terminates connection
• TCP connection termination exchange

How many RTTs for a single request?
Client-Server Communication

- two types of HTTP messages: request, response
- HTTP request message: (GET POST HEAD ....)

GET /somadir/page.html HTTP/1.1
Host: www.someschool.edu
User-agent: Mozilla/4.0
Connection: close
Accept-language: fr

(extra carriage return, line feed)

HTTP response message

HTTP/1.1 200 OK
Connection close
Date: Thu, 06 Aug 1998 12:00:15 GMT
Server: Apache/1.3.0 (Unix)
Last-Modified: Mon, 22 Jun 1998 ....
Content-Length: 6821
Content-Type: text/html

(data, e.g., requested HTML file)

data data data data data data ...
Different Forms of Server Response

• Return a file
  – URL matches a file (*e.g.*, `/www/index.html`)  
  – Server returns file as the response  
  – Server generates appropriate response header

• Generate response dynamically  
  – URL triggers a program on the server  
  – Server runs program and sends output to client

• Return meta-data with no body
HTTP Resource Meta-Data

• Meta-data
  – Info about a resource, stored as a separate entity

• Examples:
  – Size of resource, last modification time, type of content

• Usage example: Conditional GET Request
  – Client requests object “If-modified-since”
  – If unchanged, “HTTP/1.1 304 Not Modified”
  – No body in the server’s response, only a header
HTTP is *Stateless*

- Each request-response treated independently
  - Servers *not* required to retain state

- **Good**: Improves scalability on the server-side
  - Failure handling is easier
  - Can handle higher rate of requests
  - Order of requests doesn’t matter

- **Bad**: Some applications *need* persistent state
  - Need to uniquely identify user or store temporary info
  - *e.g.*, Shopping cart, user profiles, usage tracking, ...
State in a Stateless Protocol:

**Cookies**

- *Client-side* state maintenance
  - Client stores small state on behalf of server
  - Client sends state in future requests to the server
- Can provide authentication
HTTP Performance

• Most Web pages have multiple objects
  – e.g., HTML file and a bunch of embedded images

• How do you retrieve those objects (naively)?
  – One item at a time

• Put stuff in the optimal place?
  – Where is that precisely?
    • Enter the Web cache and the CDN
Fetch HTTP Items: Stop & Wait

Start fetching page → Request item 1 → Transfer item 1 → Request item 2 → Transfer item 2 → Request item 3 → Transfer item 3 → Finish; display page

≥2 RTTs per object
Improving HTTP Performance:
Concurrent Requests & Responses

- Use multiple connections *in parallel*
- Does not necessarily maintain order of responses

- Client = 😊
- Server = 😊
- Network = 😞 Why?
Improving HTTP Performance: Pipelined Requests & Responses

• **Batch** requests and responses
  – Reduce connection overhead
  – Multiple requests sent in a single batch
  – Maintains order of responses
  – Item 1 always arrives before item 2

• How is this different from concurrent requests/responses?
  – Single TCP connection
Improving HTTP Performance:
Persistent Connections

• Enables multiple transfers per connection
  – Maintain TCP connection across multiple requests
  – Including transfers subsequent to current page
  – Client or server can tear down connection

• Performance advantages:
  – Avoid overhead of connection set-up and tear-down
  – Allow TCP to learn more accurate RTT estimate
  – Allow TCP congestion window to increase
  – i.e., leverage previously discovered bandwidth

• Default in HTTP/1.1
HTTP evolution

• 1.0 – one object per TCP: simple but slow
• Parallel connections - multiple TCP, one object each: wastes b/w, may be svr limited, out of order
• 1.1 pipelining – aggregate retrieval time: ordered, multiple objects sharing single TCP
• 1.1 persistent – aggregate TCP overhead: lower overhead in time, increase overhead at ends (e.g., when should/do you close the connection?)
Scorecard: Getting n Small Objects

*Time dominated by latency*

- One-at-a-time: $\sim 2n \text{ RTT}$
- Persistent: $\sim (n+1) \text{ RTT}$
- M concurrent: $\sim 2 \frac{n}{m} \text{ RTT}$
- Pipelined: $\sim 2 \text{ RTT}$
- Pipelined/Persistent: $\sim 2 \text{ RTT}$ first time, RTT later
Scorecard: Getting n Large Objects

*Time dominated by bandwidth*

- One-at-a-time: $\sim nF/B$
- M concurrent: $\sim [n/m] F/B$
  - assuming shared with large population of users
- Pipelined and/or persistent: $\sim nF/B$
  - The only thing that helps is getting more bandwidth..
Improving HTTP Performance: Caching

• Many clients transfer **same information**
  – Generates **redundant** server and network load
  – Clients experience **unnecessary** latency
Improving HTTP Performance:
Caching: How

• Modifier to GET requests:
  – If-modified-since – returns “not modified” if resource not modified since specified time

• Response header:
  – Expires – how long it’s safe to cache the resource
  – No-cache – ignore all caches; always get resource directly from server
Improving HTTP Performance:

**Caching: Why**

- **Motive for placing content closer to client:**
  - User gets better response time
  - Content providers get happier users
    - Time is money, really!
  - Network gets reduced load

- **Why does caching work?**
  - Exploits *locality of reference*

- **How well does caching work?**
  - Very well, up to a limit
  - Large overlap in content
  - But many unique requests
Improving HTTP Performance:
Caching on the Client

Example: Conditional GET Request
• Return resource only if it has changed at the server
  – Save server resources!

Request from client to server:

GET /~awm22/win HTTP/1.1
Host: www.cl.cam.ac.uk
User-Agent: Mozilla/4.03
If-Modified-Since: Sun, 27 Aug 2006 22:25:50 GMT
<CRLF>

• HOW?
  – Client specifies “if-modified-since” time in request
  – Server compares this against “last modified” time of desired resource
  – Server returns “304 Not Modified” if resource has not changed
  – .... or a “200 OK” with the latest version otherwise
Improving HTTP Performance: Caching with Reverse Proxies

Cache documents close to server
→ decrease server load
• Typically done by content providers

• Only works for static(* content

(*) static can also be snapshots of dynamic content
Improving HTTP Performance:
Caching with Forward Proxies

Cache documents close to clients
→ reduce network traffic and decrease latency

• Typically done by ISPs or corporate LANs
Improving HTTP Performance:
Caching w/ Content Distribution Networks

• Integrate forward and reverse caching functionality
  – One overlay network (usually) administered by one entity
  – e.g., Akamai

• Provide document caching
  – Pull: Direct result of clients’ requests
  – Push: Expectation of high access rate

• Also do some processing
  – Handle dynamic web pages
  – Transcoding
  – Maybe do some security function – watermark IP
Improving HTTP Performance: Caching with CDNs (cont.)

Diagram showing the flow of data between clients, forward proxies, ISP, Backbone ISP, CDN, and the server.
Improving HTTP Performance:

CDN Example – Akamai

• Akamai creates new domain names for each client content provider.
  – e.g., a128.g.akamai.net

• The CDN’s DNS servers are authoritative for the new domains

• The client content provider modifies its content so that embedded URLs reference the new domains.
  – “Akamaize” content
  – e.g.: http://www.bbc.co.uk/popular-image.jpg becomes http://a128.g.akamai.net/popular-image.jpg

• Requests now sent to CDN’s infrastructure...
Hosting: Multiple Sites Per Machine

• Multiple Web sites on a single machine
  – Hosting company runs the Web server on behalf of multiple sites (e.g., www.foo.com and www.bar.com)

• Problem: GET /index.html

• Solutions:
  – Multiple server processes on the same machine
    • Have a separate IP address (or port) for each server
  – Include site name in HTTP request
    • Single Web server process with a single IP address
    • Client includes “Host” header (e.g., Host: www.foo.com)
    • Required header with HTTP/1.1
Hosting: Multiple Machines Per Site

• Replicate popular Web site across many machines
  – Helps to handle the load
  – Places content closer to clients

• Helps when content isn’t cacheable

• Problem: Want to direct client to particular replica
  – Balance load across server replicas
  – Pair clients with nearby servers
Multi-Hosting at Single Location

- Single IP address, multiple machines
  - Run multiple machines behind a single IP address
  - Ensure all packets from a single TCP connection go to the same replica
Multi-Hosting at Several Locations

- Multiple addresses, multiple machines
  - Same name but different addresses for all of the replicas
  - Configure DNS server to return *closest* address
CDN examples round-up

• CDN using DNS
  DNS has information on loading/distribution/location

• CDN using anycast
  same address from DNS name but local routes

• CDN based on rewriting HTML URLs
  (akami example just covered – akami uses DNS too)
SIP – Session Initiation Protocol

Session?

Anyone smell an OSI / ISO standards document burning?
SIP - VoIP

Establishing communication through SIP proxies.
SIP?

• SIP – bringing the fun/complexity of telephony to the Internet
  – User location
  – User availability
  – User capabilities
  – Session setup
  – Session management
    • (e.g. “call forwarding”)
H.323 – ITU

• Why have one standard when there are at least two....

• The full H.323 is hundreds of pages
  – The protocol is known for its complexity – an ITU hallmark

• SIP is not much better
  – IETF grew up and became the ITU....
Multimedia Applications

Message flow for a basic SIP session
The (still?) missing piece:
Resource Allocation for Multimedia Applications

I can ‘differentiate’ VoIP from data but...
I can only control data going into the Internet
Multimedia Applications

• Resource Allocation for Multimedia Applications

Admission control using session control protocol.
Resource Allocation for Multimedia Applications

So where does it happen?
Inside single institutions or *domains of control*.....
*Universities, Hospitals, big corp...*  

Who are we kidding??

Co-ordination of SIP signaling and resource reservation.

What about my aDSL/CABLE/etc it combines voice and data?
Phone company **controls** the multiplexing on the line and throughout their own network too.....
P2P – efficient network use that annoys the ISP
Pure P2P architecture

- *no* always-on server
- arbitrary end systems directly communicate
- peers are intermittently connected and change IP addresses

- **Three topics:**
  - File distribution
  - Searching for information
  - Case Study: Skype
File Distribution: Server-Client vs P2P

**Question**: How much time to distribute file from one server to \( N \) peers?

\[ u_s: \text{server upload bandwidth} \]
\[ u_i: \text{peer } i \text{ upload bandwidth} \]
\[ d_i: \text{peer } i \text{ download bandwidth} \]

Network (with abundant bandwidth)
File distribution time: server-client

- server sequentially sends N copies:
  - $NF/u_s$ time
- client $i$ takes $F/d_i$ time to download

Time to distribute $F$ to $N$ clients using client/server approach:

$$d_{cs} = \max \left\{ NF/u_s, \frac{F}{\min(d_i)} \right\}$$

increases linearly in $N$ (for large $N$)
File distribution time: P2P

- server must send one copy: $F/u_s$ time
- client $i$ takes $F/d_i$ time to download
- NF bits must be downloaded (aggregate)
  - fastest possible upload rate: $u_s + \sum u_i$

$$d_{P2P} = \max \left\{ \frac{F}{u_s}, \frac{F}{\min(d_i)}, \frac{NF}{u_s + \sum u_i} \right\}$$
Server-client vs. P2P: example

Client upload rate = $u$, $F/u = 1$ hour, $u_s = 10u$, $d_{\text{min}} \geq u_s$
File distribution: BitTorrent*

*rather old BitTorrent

- P2P file distribution

**tracker:** tracks peers participating in torrent

**torrent:** group of peers exchanging chunks of a file

Peer obtains list of peers

Trading chunks
BitTorrent (1)

- file divided into 256KB *chunks*.
- peer joining torrent:
  - has no chunks, but will accumulate them over time
  - registers with tracker to get list of peers, connects to subset of peers ("neighbors")
- while downloading, peer uploads chunks to other peers.
- peers may come and go
- once peer has entire file, it may (selfishly) leave or (altruistically) remain
BitTorrent (2)

Pulling Chunks

• at any given time, different peers have different subsets of file chunks
• periodically, a peer (Alice) asks each neighbor for list of chunks that they have.
• Alice sends requests for her missing chunks
  – rarest first

Sending Chunks: tit-for-tat

❖ Alice sends chunks to four neighbors currently sending her chunks at the highest rate
  ❖ re-evaluate top 4 every 10 secs
❖ every 30 secs: randomly select another peer, starts sending chunks
  ❖ newly chosen peer may join top 4
  ❖ “optimistically unchoke”
BitTorrent: Tit-for-tat

(1) Alice “optimistically unchokes” Bob
(2) Alice becomes one of Bob’s top-four providers; Bob reciprocates
(3) Bob becomes one of Alice’s top-four providers

With higher upload rate, can find better trading partners & get file faster!
Distributed Hash Table (DHT)

• DHT = distributed P2P database
• Database has (key, value) pairs;
  – key: ss number; value: human name
  – key: content type; value: IP address
• Peers query DB with key
  – DB returns values that match the key
• Peers can also insert (key, value) peers
Distributed Hash Table (DHT)

- DHT = distributed P2P database
- Database has *(key, value)* pairs;
  - key: ss number; value: human name
  - key: content type; value: IP address
- Peers **query** DB with key
  - DB returns values that match the key
- Peers can also **insert** *(key, value)* peers
DHT Identifiers

• Assign integer identifier to each peer in range \([0,2^n-1]\).
  – Each identifier can be represented by \(n\) bits.
• Require each key to be an integer in \textcolor{red}{\text{same range}}.
• To get integer keys, hash original key.
  – eg, \(key = h(“Game of Thrones season 4”)\)
  – This is why they call it a distributed “hash” table
How to assign keys to peers?

- Central issue: Assigning (key, value) pairs to peers.
- Rule: assign key to the peer that has the closest ID.
- Convention in lecture: closest is the immediate successor of the key.
- Ex: n=4; peers: 1,3,4,5,8,10,12,14;
  - key = 13, then successor peer = 14
  - key = 15, then successor peer = 1
• Each peer *only* aware of immediate successor and predecessor.

• “Overlay network”
O(N) messages on avg to resolve query, when there are N peers

Define closest as closest successor

Circle DHT (2)

Who’s resp for key 1110?
Circular DHT with Shortcuts

- Each peer keeps track of IP addresses of predecessor, successor, short cuts.
- Reduced from 6 to 2 messages.
- Possible to design shortcuts so $O(\log N)$ neighbors, $O(\log N)$ messages in query

Who’s resp for key 1110?
Peer Churn

- Peer 5 abruptly leaves
- Peer 4 detects; makes 8 its immediate successor; asks 8 who its immediate successor is; makes 8’s immediate successor its second successor.
- What if peer 13 wants to join?

• To handle peer churn, require each peer to know the IP address of its two successors.
• Each peer periodically pings its two successors to see if they are still alive.
P2P Case study: Skype (pre-Microsoft)

- inherently P2P: pairs of users communicate.
- proprietary application-layer protocol (inferred via reverse engineering)
- hierarchical overlay with SNs
- Index maps usernames to IP addresses; distributed over SNs
Peers as relays

• Problem when both Alice and Bob are behind “NATs”.
  – NAT prevents an outside peer from initiating a call to insider peer

• Solution:
  – Using Alice’s and Bob’s SNs, Relay is chosen
  – Each peer initiates session with relay.
  – Peers can now communicate through NATs via relay
Summary.

• Apps need protocols too

• We covered examples from
  – Traditional Applications (web)
  – Scaling and Speeding the web (CDN/Cache tricks)

• Infrastructure Services (DNS)
  – Cache and Hierarchy

• Multimedia Applications (SIP)
  – Extremely hard to do better than worst-effort

• P2P Network examples