Module 3: Dealing with (Internet) Scale

- Dealing with scale, at the application layer
- The Internet’s Domain Name System
- Content Distribution Networks (CDN)
- CDN Business model

Domain Name Service (DNS): translate host names like “www.umass.edu” to 32-bit IP address 128.119.103.148
- Internet-wide service
- ~1 trillion request day
- Netflix uses ~33% of all downstream bandwidth to homes in US during peak hours

*Neither DNS nor Netflix could be implemented using a single “massive” server!*
Module 3: Dealing with (Internet) Scale

- Distributed system: implement service using many Internet-connected servers
- More resources:
  - more server -> more aggregate bandwidth
  - more servers -> more processing
- No single point of failure
- Any drawbacks of distributed implementation?
  - lots of coordination, messaging
  - recovery from failure

DNS: domain name system

people: many identifiers:
- SSN, name, passport #

Internet hosts, routers:
- IP address (32 bit) - used for addressing datagrams
- “name”, e.g., www.yahoo.com - used by humans

Q: how to map between IP address and name, and vice versa?

Domain Name System:
- distributed database
  implemented in hierarchy of many name servers
- application-layer protocol: hosts, name servers communicate to resolve names (address/name translation)
  - note: core Internet function, implemented as application-layer protocol
  - complexity at network’s “edge”
DNS: services, structure

**DNS services**
- hostname to IP address translation
- host aliasing
  - canonical, alias names
- mail server aliasing
- load distribution
  - replicated Web servers: many IP addresses correspond to one name

**why not centralize DNS?**
- single point of failure
- traffic volume
- distant centralized database
- maintenance

*A: doesn’t scale!*

---

DNS: a distributed, hierarchical database

Root DNS Servers

- com DNS servers
  - yahoo.com DNS servers
  - amazon.com DNS servers

- org DNS servers
  - pbs.org DNS servers

- edu DNS servers
  - poly.edu DNS servers
  - umass.edu DNS servers

*client wants IP for www.amazon.com; 1st approx:*
- client queries root server to find com DNS server
- client queries .com DNS server to get amazon.com DNS server
- client queries amazon.com DNS server to get IP address for www.amazon.com
DNS: root name servers

- contacted by local name server that can not resolve name
- root name server:
  - contacts authoritative name server if name mapping not known
  - gets mapping
  - returns mapping to local name server

13 root name “servers” worldwide

TLD, authoritative servers

top-level domain (TLD) servers:
- responsible for com, org, net, edu, aero, jobs, museums, and all top-level country domains, e.g.: uk, fr, ca, jp
- Network Solutions maintains servers for .com TLD
- Educause for .edu TLD

authoritative DNS servers:
- organization’s own DNS server(s), providing authoritative hostname to IP mappings for organization’s named hosts
- can be maintained by organization or service provider
Local DNS name server

- does not strictly belong to hierarchy
- each ISP (residential ISP, company, university) has one
  - also called “default name server”
- when host makes DNS query, query is sent to its local DNS server
  - has local cache of recent name-to-address translation pairs (but may be out of date!)
  - acts as proxy, forwards query into hierarchy

DNS name resolution example

- host at cis.poly.edu wants IP address for gaia.cs.umass.edu

iterated query:
- contacted server replies with name of server to contact
- “I don’t know this name, but ask this server”
**DNS name resolution example**

*recursive query:*
- puts burden of name resolution on contacted name server
- heavy load at upper levels of hierarchy!

---

**DNS: caching, updating records**

- once (any) name server learns mapping, it *caches* mapping
  - cache entries timeout (disappear) after some time (TTL)
  - TLD servers typically cached in local name servers
    - thus root name servers not often visited
- cached entries may be *out-of-date* (best effort name-to-address translation!)
  - if name host changes IP address, may not be known Internet-wide until all TTLs expire
- update/notify mechanisms proposed IETF standard
  - RFC 2136
**DNS records**

**DNS:** distributed db storing resource records (RR)

**RR format:** `(name, value, type, ttl)``

- **type=A**
  - `name` is hostname
  - `value` is IP address

- **type=NS**
  - `name` is domain (e.g., `foo.com`)
  - `value` is hostname of authoritative name server for this domain

- **type=CNAME**
  - `name` is alias name for some “canonical” (the real) name
  - `value` is canonical name

- **type=MX**
  - `value` is name of mailserver associated with name

**DNS protocol, messages**

- **query** and **reply** messages, both with same **message format**

**msg header**

- **identification:** 16 bit # for query, reply to query uses same #
- **flags:**
  - query or reply
  - recursion desired
  - recursion available
  - reply is authoritative
DNS protocol, messages

<table>
<thead>
<tr>
<th>identification</th>
<th>flags</th>
</tr>
</thead>
<tbody>
<tr>
<td># questions</td>
<td># answer RRs</td>
</tr>
<tr>
<td># authority RRs</td>
<td># additional RRs</td>
</tr>
<tr>
<td>questions (variable # of questions)</td>
<td>answers (variable # of RRs)</td>
</tr>
<tr>
<td>authority (variable # of RRs)</td>
<td>additional info (variable # of RRs)</td>
</tr>
</tbody>
</table>

name, type fields for a query
RRs in response to query
records for authoritative servers
additional “helpful” info that may be used

Attacking DNS

DDoS attacks
- Bombard root servers with traffic
  - Not successful to date
  - Traffic Filtering
  - Local DNS servers cache IPs of TLD servers, allowing root server bypass
- Bombard TLD servers
  - Potentially more dangerous

Redirect attacks
- Man-in-middle
  - Intercept queries
- DNS poisoning
  - Send bogus relies to DNS server, which caches

Exploit DNS for DDoS
- Send queries with spoofed source address: target IP
- Requires amplification
Dealing with (Internet) Scale: CDNs

Recall: one single “mega-server” can’t possibly handle all requests for popular service

- DNS
- not enough bandwidth
  - Netflix video streaming at 2 Mbps per connection
  - only 5000 connections over fastest possible (10Gbs) connection to Internet at one server
  - 30 Million Netflix customers
- too far from some users: halfway around the globe to someone
- reliability: single point of failure

A single server doesn’t “scale”

Content distribution networks

- challenge: how to stream content (selected from millions of videos) to hundreds of thousands of simultaneous users?

- option 1: single, large “mega-server”
  - single point of failure
  - point of network congestion
  - long path to distant clients
  - multiple copies of video sent over outgoing link

....quite simply: this solution doesn’t scale
Content distribution networks

- **challenge**: how to stream content (selected from millions of videos) to hundreds of thousands of simultaneous users?

- **option 2**: store/serve multiple copies of videos at multiple geographically distributed sites (**CDN**)
  - **enter deep**: push CDN servers deep into many access networks
    - close to users
    - used by Akamai, 1700 locations
  - **bring home**: smaller number (10’s) of larger clusters in POPs near (but not within) access networks
    - used by Limelight

---

CDN: “simple” content access scenario

**Bob** (client) requests video http://netcinema.com/6Y7B23V
- video stored in CDN at http://KingCDN.com/NetC6y&B23V

2. resolve http://netcinema.com/6Y7B23V via Bob’s local DNS
3. netcinema’s DNS returns URL http://KingCDN.com/NetC6y&B23V
4. 4&5. Resolve http://KingCDN.com/NetC6y&B23 via KingCDN’s authoritative DNS, which returns IP address of KingCDN server with video
5. request video from KINGCDN server, streamed via HTTP
CDN cluster selection strategy

- **challenge:** how does CDN DNS select “good” CDN node to stream to client
  - pick CDN node geographically closest to client
  - pick CDN node with shortest delay (or min # hops) to client (CDN nodes periodically ping access ISPs, reporting results to CDN DNS)
- **alternative:** let client decide - give client a list of several CDN servers
  - client pings servers, picks “best”

---

Akamai CDN: quickie

- pioneered creation of CDNs circa 2000
- **now:** 61,000 servers in 1,000 networks in 70 countries
- delivers est 15-20% of all Internet traffic
- runs its own DNS service (alternative to public root, TLD, hierarchy)
- hundreds of billions of Internet interactions daily
- more shortly….
Case study: Netflix

- 30% downstream US traffic in 2013
- $1B quarterly revenue
- 2B hours viewing streamed video
- owns very little infrastructure, uses 3rd party services:
  - own registration, payment servers
  - Amazon (3rd party) cloud services:
    - Netflix uploads studio master to Amazon cloud
    - create multiple version of movie (different encodings) in cloud, move to CDNs
    - Cloud hosts Netflix web pages for user browsing

http://www.statisticbrain.com/netflix-statistics/

---

Case study: Netflix

- uses three 3rd party CDNs to store, host, stream Netflix content:
  - Akamai, Limelight, Level-3
  - can play each of the CDNs against each other in terms of customer experience
  - developing its own CDN (2012)

<table>
<thead>
<tr>
<th>Rank</th>
<th>Application</th>
<th>Share</th>
<th>Application</th>
<th>Share</th>
<th>Application</th>
<th>Share</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>BitTorrent</td>
<td>52.01%</td>
<td>Netflix</td>
<td>29.70%</td>
<td>Netflix</td>
<td>24.71%</td>
</tr>
<tr>
<td>2</td>
<td>HTTP</td>
<td>8.31%</td>
<td>HTTP</td>
<td>18.36%</td>
<td>BitTorrent</td>
<td>17.23%</td>
</tr>
<tr>
<td>3</td>
<td>Skype</td>
<td>3.81%</td>
<td>YouTube</td>
<td>11.04%</td>
<td>HTTP</td>
<td>17.18%</td>
</tr>
<tr>
<td>4</td>
<td>Netflix</td>
<td>3.99%</td>
<td>BitTorrent</td>
<td>10.32%</td>
<td>YouTube</td>
<td>9.80%</td>
</tr>
<tr>
<td>5</td>
<td>P2Pstream</td>
<td>2.92%</td>
<td>Flash Video</td>
<td>4.88%</td>
<td>Flash Video</td>
<td>3.62%</td>
</tr>
</tbody>
</table>

Table 1 - North America - Top Applications by Bytes (Peak Period, Fixed Access)
Case study: Netflix

1. Bob manages Netflix account
   Netflix registration, accounting servers

2. Bob browses Netflix video

3. Manifest file returned for requested video

4. DASH streaming
   upload copies of multiple versions of video to CDNs

- Akamai CDN
- Limelight CDN
- Level-3 CDN

Streaming multimedia: DASH

- **DASH**: Dynamic, Adaptive Streaming over HTTP
- **Server**:
  - divides video file into multiple chunks
  - each chunk stored, encoded at different rates
  - **manifest file**: provides URLs for different chunks
- **Client**:
  - periodically measures server-to-client bandwidth
  - consulting manifest, requests one chunk at a time
    - chooses maximum coding rate sustainable given current bandwidth
    - can choose different coding rates at different points in time (depending on available bandwidth at time)
Streaming multimedia: DASH

- **manifest file**: provides URLs for different chunks

---

**Streaming multimedia: DASH**

- **DASH**: Dynamic, Adaptive Streaming over HTTP
- **“intelligence” at client**: client determines
  - *when* to request chunk (so that buffer starvation, or overflow does not occur)
  - *what encoding rate* to request (higher quality when more bandwidth available)
  - *where* to request chunk (can request from URL server that is “close” to client or has high available bandwidth)
Let’s check out Netflix:

- use wireshark to capture and view packets being sent to/from my laptop
  - you’re not responsible for knowing how to use it, but it’s a lot of fun
- logging in (authorization), content selection
  - main webpage hosted by amazon cloud?
  - use whois to see who server is
- select video and begin playing
  - use whois to see who is serving video
  - how far away is server (use traceroute, no info?)