DNS and the Web

- DNS
- CDNs
- Readings
  - DNS Performance and the Effectiveness of Caching
  - Development of the Domain Name System

Naming

- How do we efficiently locate resources?
  - DNS: name → IP address
  - Service location: description → host
- Other issues
  - How do we scale these to the wide area?
  - How to choose among similar services?
Obvious Solutions (1)

Why not centralize DNS?
• Single point of failure
• Traffic volume
• Distant centralized database
• Single point of update
• Doesn’t scale!

Obvious Solutions (2)

Why not use /etc/hosts?
• Original Name to Address Mapping
  • Flat namespace
  • /etc/hosts
  • SRI kept main copy
  • Downloaded regularly
• Count of hosts was increasing: machine per domain → machine per user
  • Many more downloads
  • Many more updates

Domain Name System Goals
• Basically building a wide area distributed database
• Scalability
• Decentralized maintenance
• Robustness
• Global scope
  • Names mean the same thing everywhere
• Don’t need
  • Atomicity
  • Strong consistency

DNS Records

RR format: (class, name, value, type, ttl)
• DB contains tuples called resource records (RRs)
  • Classes = Internet (IN), Chaosnet (CH), etc.
  • Each class defines value associated with type

FOR IN class:
- Type=A
  • name is hostname
  • value is IP address
- Type=NS
  • name is domain (e.g. foo.com)
  • value is name of authoritative name server for this domain
- Type=CNAME
  • name is an alias name for some “canonical” (the real) name
  • value is canonical name
- Type=MX
  • value is hostname of mailserver associated with name
DNS Design: Hierarchy Definitions

- Each node in hierarchy stores a list of names that end with same suffix
  - Suffix = path up tree
- E.g., given this tree, where would following be stored:
  - Fred.com
  - Fred.edu
  - Fred.cmu.edu
  - Fred.cmcl.cs.cmu.edu
  - Fred.cs.mit.edu

DNS Design: Zone Definitions

- Zone = contiguous section of name space
- E.g., Complete tree, single node or subtree
- A zone has an associated set of name servers

DNS Design: Cont.

- Zones are created by convincing owner node to create/delegate a subzone
  - Records within zone stored multiple redundant name servers
  - Primary/master name server updated manually
  - Secondary/redundant servers updated by zone transfer of name space
    - Zone transfer is a bulk transfer of the “configuration” of a DNS server – uses TCP to ensure reliability
- Example:
  - CS.CMU.EDU created by CMU.EDU administrators

Servers/Resolvers

- Each host has a resolver
  - Typically a library that applications can link to
  - Local name servers hand-configured (e.g. /etc/resolv.conf)
- Name servers
  - Either responsible for some zone or…
  - Local servers
    - Do lookup of distant host names for local hosts
    - Typically answer queries about local zone

Servers/Resolvers

- Each host has a resolver
  - Typically a library that applications can link to
  - Local name servers hand-configured (e.g. /etc/resolv.conf)
- Name servers
  - Either responsible for some zone or…
  - Local servers
    - Do lookup of distant host names for local hosts
    - Typically answer queries about local zone
DNS: Root Name Servers

- Responsible for "root" zone
- Approx. dozen root name servers worldwide
  - Currently {a-m}.root-servers.net
- Local name servers contact root servers when they cannot resolve a name
  - Configured with well-known root servers

Typical Resolution

Lookup Methods

Recursive query:
- Server goes out and searches for more info (recursive)
- Only returns final answer or "not found"

Iterative query:
- Server responds with as much as it knows (iterative)
- "I don't know this name, but ask this server"

Workload and Caching

- What workload do you expect for different servers/names?
- Why might this be a problem? How can we solve this problem?
- DNS responses are cached
  - Quick response for repeated translations
  - Other queries may reuse some parts of lookup
    - NS records for domains
- DNS negative queries are cached
  - Don’t have to repeat past mistakes
    - E.g. misspellings, search strings in resolv.conf
- Cached data periodically times out
  - Lifetime (TTL) of data controlled by owner of data
  - TTL passed with every record
**Typical Resolution**

- Client
- Local DNS server
- www.cs.cmu.edu
  - root & edu DNS server
  - ns1.cmu.edu DNS server
  - ns1.cs.cmu.edu DNS server

**Subsequent Lookup Example**

- Client
- Local DNS server
- ftp.cs.cmu.edu
  - cmu.edu DNS server
  - cs.cmu.edu DNS server

**Reliability**

- DNS servers are replicated
  - Name service available if ≥ one replica is up
  - Queries can be load balanced between replicas
- UDP used for queries
  - Need reliability → must implement this on top of UDP!
  - Why not just use TCP?
- Try alternate servers on timeout
  - Exponential backoff when retrying same server
- Same identifier for all queries
  - Don’t care which server responds

**Reverse Name Lookup**

- 128.2.206.138?
  - Lookup 138.206.2.128.in-addr.arpa
  - Why is the address reversed?
  - Happens to be www.intel-iris.net and mammoth.cmcl.cs.cmu.edu → what will reverse lookup return? Both?
    - Should only return name that reflects address allocation mechanism
Prefetching
- Name servers can add additional data to any response
- Typically used for prefetching
  - CNAME/MX/NS typically point to another host name
  - Responses include address of host referred to in “additional section”

Root Zone
- Generic Top Level Domains (gTLD) = .com, .net, .org, etc…
- Country Code Top Level Domain (ccTLD) = .us, .ca, .fi, .uk, etc…
- Root server ({a-m}.root-servers.net) also used to cover gTLD domains
  - Load on root servers was growing quickly!
  - Moving .com, .net, .org off root servers was clearly necessary to reduce load → done Aug 2000

New gTLDs
- .info → general info
- .biz → businesses
- .aero → air-transport industry
- .coop → business cooperatives
- .name → individuals
- .pro → accountants, lawyers, and physicians
- .museum → museums
- Only new one actives so far = .info, .biz, .name

New Registrars
- Network Solutions (NSI) used to handle all registrations, root servers, etc…
  - Clearly not the democratic (Internet) way
  - Large number of registrars that can create new domains → However, NSI still handle root servers
DNS Experience

- 23% of lookups with no answer
  - Retransmit aggressively \(\rightarrow\) most packets in trace for unanswered lookups!
  - Correct answers tend to come back quickly/with few retries
- 10 - 42% negative answers \(\rightarrow\) most = no name exists
  - Inverse lookups and bogus NS records
- Worst 10% lookup latency got much worse
  - Median 85\(\rightarrow\)97, 90\(^{th}\) percentile 447\(\rightarrow\)1176
- Increasing share of low TTL records \(\rightarrow\) what is happening to caching?

Hit rate for DNS = 80% \(\rightarrow\) 1-(#DNS/#connections)

- Most Internet traffic is Web
- What does a typical page look like? \(\rightarrow\) average of 4-5 imbedded objects \(\rightarrow\) needs 4-5 transfers \(\rightarrow\) accounts for 80% hit rate!
- 70% hit rate for NS records \(\rightarrow\) i.e. don’t go to root/gTLD servers
  - NS TTLs are much longer than A TTLs
  - NS record caching is much more important to scalability
- Name distribution = Zipf-like = 1/\(x^a\)
- A records \(\rightarrow\) TTLs = 10 minutes similar to TTLs = infinite
- 10 client hit rate = 1000+ client hit rate

Some Interesting Alternatives

- CoDNS
  - Lookup failures
    - Packet loss
    - LDNS overloading
    - Cron jobs
    - Maintenance problems
  - Cooperative name lookup scheme
  - If local server OK, use local server
  - When failing, ask peers to do lookup
- Push DNS
  - Top of DNS hierarchy is relatively stable
  - Why not replicate much more widely?

Overview

- DNS
  - Server selection and CDNs
CDN
- Replicate content on many servers
- Challenges
  - How to replicate content
  - Where to replicate content
  - How to find replicated content
  - How to choose among known replicas
  - How to direct clients towards replica
    - DNS, HTTP 304 response, anycast, etc.
- Akamai

Server Selection
- Service is replicated in many places in network
- How to direct clients to a particular server?
  - As part of routing ➔ anycast, cluster load balancing
  - As part of application ➔ HTTP redirect
  - As part of naming ➔ DNS
- Which server?
  - Lowest load ➔ to balance load on servers
  - Best performance ➔ to improve client performance
    - Based on Geography? RTT? Throughput? Load?
  - Any alive node ➔ to provide fault tolerance

Routing Based
- Anycast
  - Give service a single IP address
  - Each node implementing service advertises route to address
  - Packets get routed from client to “closest” service node
    - Closest is defined by routing metrics
    - May not mirror performance/application needs
  - What about the stability of routes?

Routing Based
- Cluster load balancing
  - Router in front of cluster of nodes directs packets to server
  - Can only look at global address (L3 switching)
  - Often want to do this on a connection by connection basis – why?
    - Forces router to keep per connection state
    - L4 switching – transport headers, port numbers
  - How to choose server
    - Easiest to decide based on arrival of first packet in exchange
    - Primarily based on local load
    - Can be based on later packets (e.g. HTTP Get request) but makes system more complex (L7 switching)
Application Based

- HTTP supports simple way to indicate that Web page has moved
- Server gets Get request from client
  - Decides which server is best suited for particular client and object
  - Returns HTTP redirect to that server
- Can make informed application specific decision
- May introduce additional overhead → multiple connection setup, name lookups, etc.
- While good solution in general HTTP Redirect has some design flaws – especially with current browsers?

Naming Based

- Client does name lookup for service
- Name server chooses appropriate server address
- What information can it base decision on?
  - Server load/location → must be collected
  - Name service client
    - Typically the local name server for client
- Round-robin
  - Randomly choose replica
  - Avoid hot-spots
- [Semi-]static metrics
  - Geography
  - Route metrics
  - How well would these work?

How Akamai Works

- Clients fetch html document from primary server
  - E.g. fetch index.html from cnn.com
- URLs for replicated content are replaced in html
  - E.g. `<img src="http://cnn.com/af/x.gif">` replaced with `<img src="http://a73.g.akamaitech.net/7/23/cnn.com/af/x.gif">`
- Client is forced to resolve aXYZ.g.akamaitech.net hostname

How Akamai Works

- How is content replicated?
- Akamai only replicates static content
  - Serves about 7% of the Internet traffic!
- Modified name contains original file
- Akamai server is asked for content
  - First checks local cache
  - If not in cache, requests file from primary server and caches file
How Akamai Works

- Root server gives NS record for akamai.net
- Akamai.net name server returns NS record for g.akamaitech.net
  - Name server chosen to be in region of client’s name server
  - TTL is large
- G.akamaitech.net nameserver choses server in region
  - Should try to chose server that has file in cache - How to choose?
  - Uses aXYZ name and consistent hash
  - TTL is small

Hashing

- Advantages
  - Let the CDN nodes are numbered 1..m
  - Client uses a _good_ hash function to map a URL to 1..m
  - Say hash (url) = x, so, client fetches content from node x
  - No duplication – not being fault tolerant.
  - One hop access
  - Any problems?
    - What happens if a node goes down?
    - What happens if a node comes back up?
    - What if different nodes have different views?

Robust hashing

- Let 90 documents, node 1..9, node 10 which was dead is alive again
- % of documents in the wrong node?
  - 10, 19-20, 28-30, 37-40, 46-50, 55-60, 64-70, 73-80, 82-90
  - Disruption coefficient = ½
  - Unacceptable, use consistent hashing – idea behind Akamai!

Consistent Hash

- “view” = subset of all hash buckets that are visible
- Desired features
  - Balanced – in any one view, load is equal across buckets
  - Smoothness – little impact on hash bucket contents when buckets are added/removed
  - Spread – small set of hash buckets that may hold an object regardless of views
  - Load – across all views # of objects assigned to hash bucket is small
### Consistent Hash – Example

- **Construction**
  - Assign each of \( C \) hash buckets to random points on mod \( 2^n \) circle, where, hash key size = \( n \).
  - Map object to random position on circle
  - Hash of object = closest clockwise bucket
- **Smoothness** → addition of bucket does not cause much movement between existing buckets
- **Spread & Load** → small set of buckets that lie near object
- **Balance** → no bucket is responsible for large number of objects

### How Akamai Works

1. **cnn.com (content provider)**
2. **DNS root server**
3. **Akamai server**
4. **End-user**

   - **Get index.html**
   - **Get foo.jpg**

### Akamai – Subsequent Requests

1. **cnn.com (content provider)**
2. **DNS root server**
3. **Akamai server**
4. **End-user**

   - **Get index.html**
   - **Get /cnn.com/foo.jpg**

### Coral: An Open CDN

- **Implement an open CDN**
- **Allow anybody to contribute**
- **Works with unmodified clients**
- **CDN only fetches once from origin server**

### Pool resources to dissipate flash crowds
Using CoralCDN

- Rewrite URLs into “Coralized” URLs

    - Directs clients to Coral, which absorbs load

- Who might “Coralize” URLs?
  - Web server operators Coralize URLs
  - Coralized URLs posted to portals, mailing lists
  - Users explicitly Coralize URLs

Functionality needed

- DNS: Given network location of resolver, return a proxy near the client
  - put (network info, self)
  - get (resolver info) → {proxies}

- HTTP: Given URL, find proxy caching object, preferably one nearby
  - put (URL, self)
  - get (URL) → {proxies}

Use a DHT?

- Supports put/get interface using key-based routing
- Problems with using DHTs as given
  - Lookup latency
  - Transfer latency
  - Hotspots
Coral Contributions

• Self-organizing clusters of nodes
  • NYU and Columbia prefer one another to Germany

• Rate-limiting mechanism
  • Everybody caching and fetching same URL does not overload any node in system

• Decentralized DNS Redirection
  • Works with unmodified clients

No centralized management or \textit{a priori} knowledge of proxies’ locations or network configurations