What is DNS?

- DNS (Domain Name Service) is primarily used to translate human readable names into machine usable addresses, e.g., IP addresses.

- DNS goal:
  - Efficiently locate resources.
  - E.g., Map name → IP address
  - Scale to many users over a large area
  - Scale to many updates
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
• Mid 80’s this became untenable. Why?
• Count of hosts was increasing: machine per domain → machine per user
  - Many more downloads
  - Many more updates

Domain Name System Goals
• Basically a wide-area distributed database
  (The biggest in the world!)
• Scalability
• Decentralized maintenance
• Robustness
• Global scope
  - Names mean the same thing everywhere
• Don’t need all of ACID
  - Atomicity
  - Strong consistency
• Do need: distributed update/query & Performance

Programmer’s View of DNS
• Conceptually, programmers can view the DNS database as a collection of millions of host entry structures.

```c
/* DNS host entry structure */
struct hostent {
    char *h_name; /* official domain name of host */
    char **h_aliases; /* null-terminated array of domain names */
    int h_addrtype; /* host address type (AF_INET) */
    int h_length; /* length of an address, in bytes */
    char **h_addr_list; /* null-terminated array of in_addr structs */
};
```

- in_addr is a struct consisting of 4-byte IP addr
• Functions for retrieving host entries from DNS:
  - gethostbyname: query key is a DNS host name.
  - gethostbyaddr: query key is an IP address.
DNS Message Format

<table>
<thead>
<tr>
<th>Identification</th>
<th>Flags</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. of Questions</td>
<td>No. of Answer RRs</td>
</tr>
<tr>
<td>No. of Authority RRs</td>
<td>No. of Additional RRs</td>
</tr>
</tbody>
</table>

12 bytes

Name, type fields for a query
RRs in response to query
Records for authoritative servers
Additional “helpful info that may be used

DNS Header Fields

- **Identification**
  - Used to match up request/response
- **Flags**
  - 1-bit to mark query or response
  - 1-bit to mark authoritative or not
  - 1-bit to request recursive resolution
  - 1-bit to indicate support for recursive resolution

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” name
  - value is canonical name

- **Type=MX**
  - value is hostname of mailserver associated with name

Properties of DNS Host Entries

Different kinds of mappings are possible:

- 1-1 mapping between domain name and IP addr:
  - provolone.crcl.cs.cmu.edu maps to 128.2.218.81

- Multiple domain names maps to the same IP addr:
  - www.scs.cmu.edu and www.cs.cmu.edu both map to 128.2.203.164

- Single domain name maps to multiple IP addresses:
  - aol.com and www.aol.com map to multiple IP addrs.

- Some valid domain names don’t map to any IP addr:
  - crcl.cs.cmu.edu doesn’t have a host
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.crcl.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
  - Must store list of names and tree links

DNS Design: Cont.

- Zones are created by convincing owner node to create/delegate a subzone
  - Records within zone stored in 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 admins
  - Who creates CMU.EDU or .EDU?

DNS: Root Name Servers

- Responsible for “root” zone
  - 13 root name servers
    - Currently {a-m}.root-servers.net
  - Local name servers contact root servers when they cannot resolve a name
- Why 13?
Not really 13!

Check out anycast)

So Far

- Database structure
  - Hierarchy of labels x.y.z
  - Organized into zones
  - Zones have nameservers (notice plural!)
- Database layout
  - Records which map names→names, names→ip, etc.
- Programmer API: gethostbyname, ...

Servers/Resolvers

- Each host has a resolver
  - Typically a library that applications can link to
  - Local name servers hand-configured (or DHCP)
  (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

Typical Resolution

Hmm: Notice root server returned NS ns1.cmu.edu
Typical Resolution

- Steps for resolving www.cmu.edu
  - Application calls gethostbyname() (RESOLVER)
  - Resolver contacts local name server (S₁)
  - S₁ queries root server (S₂) for (www.cmu.edu)
  - S₂ returns NS record for cmu.edu (S₃)
  - What about A record for S₃?
    - This is what the additional info section is for (PREFETCHING)
  - S₁ queries S₃ for www.cmu.edu
  - S₃ returns A record for www.cmu.edu

- Can return multiple A records →
  What does this mean?

Lookup Methods

Recursive query:
- Server goes out and searches for more info
- Only returns final answer or “not found”

Iterative query:
- Server responds with as much as it knows.
- “I don’t know this name, but ask this server”

Workload impact on choice?
- Root/distant server does iterative
- Local server typically does recursive

How to manage workload?

- Does root nameserver do recursive lookups?
- What about other zones?
- What about imbalance in popularity?
  - .com versus .dj
  - google.com versus bleu.crcl.cs.cmu.edu?
- How do we scale query workload?

Workload and Caching

- DNS responses are cached
  - Quick response for repeated translations
  - Other queries may reuse some parts of lookup
    - E.g., NS records for domains
- DNS negative queries are cached
  - Don’t have to repeat past mistakes
    - E.g., misspellings, search strings in resolv.conf
- How do you handle updates?
Workload and Caching

- DNS responses are cached
  - Quick response for repeated translations
  - Other queries may reuse some parts of lookup
    - E.g., 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

Subsequent Lookup Example

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
So far

- Hierarchial name space

Reverse DNS

Arpa: backronym \rightarrow Address and Routing Parameter Area

- Task
  - Given IP address, find its name
- Method
  - Maintain separate hierarchy based on IP names
  - Write 128.2.204.27 as 27.204.2.128.in-addr.arpa
  - Why is the address reversed?
- Managing
  - Authority manages IP addresses assigned to it
  - E.g., CMU manages name space 2.128.in-addr.arpa

.arpa Name Server Hierarchy

- At each level of hierarchy, have group of servers that are authorized to handle that region of hierarchy

Prefetching

- Name servers can add additional data to response
- Why would they?
Prefetching

- Name servers can add additional data to response
- Why would they?
- Typically used for prefetching
  - CNAME/MX/NS typically point to another host name
  - Responses include address of host referred to in “additional section”

Mail Addresses

- MX records point to mail exchanger for a name
  - E.g.
    - cmu.edu. 2590 IN MX 10 CMU-MX4.ANDREW.cmu.edu.
    - cmu.edu. 2590 IN MX 10 CMU-MX5.ANDREW.cmu.edu.

- Addition of MX record type proved to be a challenge
  - How to get mail programs to lookup MX record for mail delivery?
  - Needed critical mass of such mailers
  - Could we add a new one now?

Outline

- DNS Design
- DNS Today

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
  - How significant an effect would this have?
    - On load?
    - On performance?
gTLDs

- Un-sponsored
  - .com, .edu, .gov, .mil, .net, .org
  - .biz → businesses
  - .info → general info
  - .name → individuals
- Sponsored (controlled by a particular association)
  - .aero → air-transport industry
  - .cat → catalan related
  - .coop → business cooperatives
  - .jobs → job announcements
  - .museum → museums
  - .pro → accountants, lawyers, and physicians
  - .travel → travel industry
- Starting up
  - .mobi → mobile phone targeted domains
  - .post → postal
  - .tel → telephone related
- Proposed
  - .asia, .cym, .geo, .kid, .mail, .sco, .web, .xxx
  - Whatever you want!

Is there anything special about .com?

What about adding .goldstein as a gTLD?

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 handles A root server

Measurements of DNS

- No centralized caching per site
  - Each machine runs own caching local server
  - Why is this a problem?
  - How many hosts do we need to share cache? → recent studies suggest 10-20 hosts
  - "Hit rate for DNS:1 - (#DNS/#connections) → 80%"
  - Is this good or bad?

- Lower TTLs for A records does not affect performance
- DNS performance really relies more on NS-record caching

- "Hit rate for DNS:1 - (#DNS/#connections) → 80%"
  - Is this good or bad?
  - Most Internet traffic was Web with HTTP 1.0
    - What does a typical page look like? → average of 4-5 imbedded objects → needs 4-5 transfers
    - This alone accounts for 80% hit rate!

- Lower TTLs for A records does not affect performance
- DNS performance really relies more on NS-record caching
Tracing Hierarchy (1)

- Dig Program
  - Allows querying of DNS system
  - Use flags to find name server (NS)
  - Disable recursion so that operates one step at a time

```
unix> dig +norecurse @a.root-servers.net NS kittyhawk.cmcl.cs.cmu.edu
```

- All .edu names handled by set of servers

---

Tracing Hierarchy (2)

- 3 servers handle CMU names

```
unix> dig +norecurse @e3.nstld.com NS kittyhawk.cmcl.cs.cmu.edu
```

---

Tracing Hierarchy (3 & 4)

- 4 servers handle CMU CS names

```
unix> dig +norecurse @t-ns1.net.cmu.edu NS kittyhawk.cmcl.cs.cmu.edu
```

- Quasar is master NS for this zone

```
unix> dig +norecurse @blueberry.srv.cs.cmu.edu NS kittyhawk.cmcl.cs.cmu.edu
```

---

DNS (Summary)

- Motivations → large distributed database
  - Scalability
  - Independent update
  - Robustness
- Hierarchical database structure
  - Zones
  - How lookups are done
- Caching/prefetching and TTLs
- Reverse name lookup
- What are the steps to creating your own domain?