15-441 Computer Networking

DNS

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?

Overview

- DNS
- Server selection
- DNS experience/trends
- Service location

DNS: Domain Name System

People: many identifiers:
- SSN, name, Passport #

Internet hosts, routers:
- IP address (32 bit) - used for addressing datagrams
- "name", e.g., gaia.cs.umass.edu - used by humans

Q: Map between IP addresses and name ?

Domain Name System:
- *Distributed database* implemented in hierarchy of many *name servers*
- *Application-layer protocol* host, routers, name servers to communicate to *resolve* names (address/name translation)
  - Note: core Internet function implemented as application-layer protocol
  - Complexity at network's "edge"

Obvious Solutions (1)

Why not centralize DNS?
- Single point of failure
- Traffic volume
- Distant centralized database
- 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

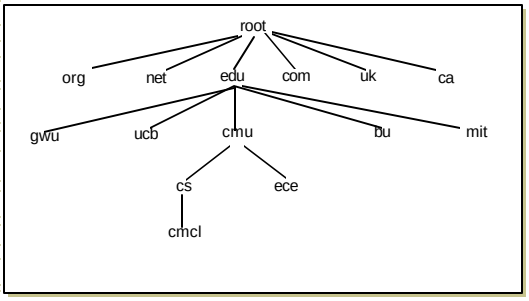
DNS Design

- DB contains tuples called resource records (RRs)
  - RR contains name, type, class and application data
  - Classes = Internet (IN), Chaosnet (CH), etc.
  - Each class defines data associated with type, e.g. for IN:
    - A = IP address, NS = name server, CNAME = canonical name (for aliasing), HINFO = CPU/OS info, MX = mail exchange, PTR = domain name = pointer for reverse mapping of address to name

DNS Design

- Administrative hierarchy
  - "." as separator
  - Zone = contiguous section of name space
    - E.g., Complete tree, single node or subtree

Hierarchical Name Space



DNS Design

- Zones are created by convincing owner node to create/delegate a subzone
  - Records within zone stored multiple redundant 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

DNS Records

DNS: distributed db storing resource records (RR)

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

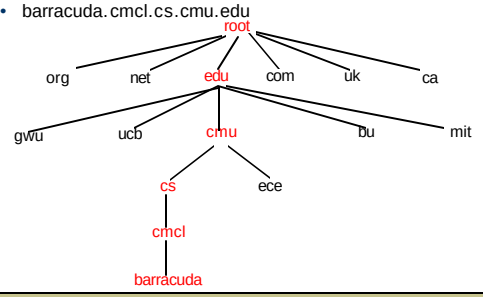
- Type=A
  - **name** is hostname
  - **value** is IP address
- Type=NS
  - **name** is domain (e.g. foo.com)
  - **value** is IP address of authoritative name server for this domain
- Type=CNAME
  - **name** is an alias name for some "canonical" (the real) name
  - **value** is canonical name
- Type=MX
  - **value** is hostname of mailserver associated with **name**

DNS Design

- Host name to address section
  - Top-level domains → edu, gov, ca, us, etc.
  - Sub-domains = subtrees
  - Human readable name = leaf → root path

Hierarchical Name Space

- barracuda.cmcl.cs.cmu.edu

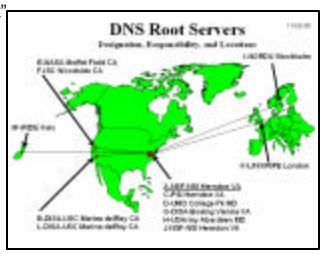


Servers/Resolvers

- 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
  - Typically responsible for some zone
  - 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



DNS Message Format

12 bytes	Identification	Flags
	No. of Questions	No. of Answer RRs
	No. of Authority RRs	No. of Additional RRs
Name, type fields for a query	Questions (variable number of answers)	
RRs in response to query	Answers (variable number of resource records)	
Records for authoritative servers	Authority (variable number of resource records)	
Additional "helpful info that may be used	Additional Info (variable number of resource records	

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

Caching

- 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

Lookup Methods

- Iterative
  - Server responds with as much as it knows (iterative)
- Recursive
  - Server goes out and searches for more info (recursive)
  - Only returns final answer or "not found"
- Impact on caching? workload?
  - Local server typically does recursive
  - Root/distant server does iterative

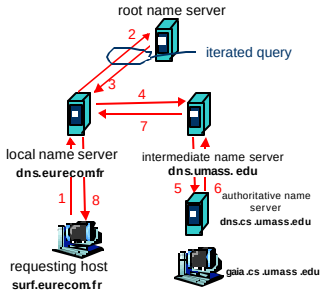
DNS: Iterated Queries

Recursive query:

- Puts burden of name resolution on contacted name server
- Heavy load?

Iterative query:

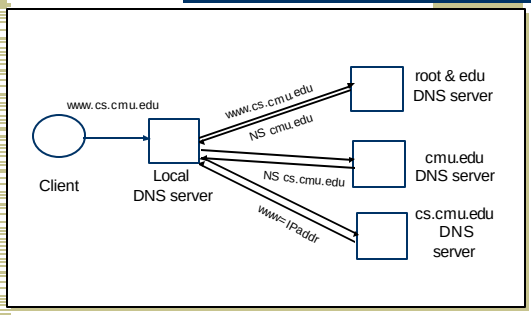
- Contacted server replies with name of server to contact
- "I don't know this name, but ask this server"



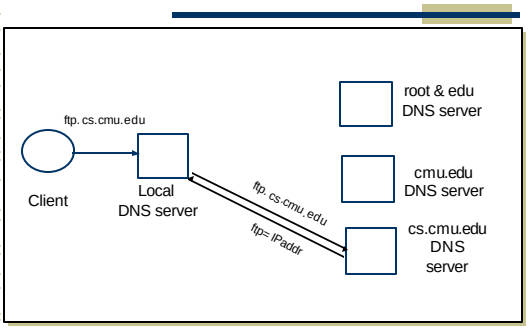
Typical Resolution

- Steps for resolving www.cmu.edu
  - Application calls gethostbyname()
  - Resolver contacts local name server (S_1)
  - S_1 queries root server (S_2) for (www.cmu.edu)
  - S_2 returns NS record for cmu.edu (S_3)
  - What about A record for S_3?
    - This is what the additional information section is for
  - S_1 queries S_3 for www.cmu.edu
  - S_3 returns A record for www.cmu.edu
- Can return multiple A records → what does this mean?

DNS Lookup Example



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 → Why not 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.seshan.org and mammoth.cmcl.cs.cmu.edu → what will reverse lookup return? Both?
    - Should only return primary name

Prefetching

- Name servers can add addition data on 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"

Mail Addresses

- MX records point to mail exchanger for a name
  - E.g. mail.acm.org is MX for acm.org
- 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

Overview

- DNS
- Server selection
- DNS experience/trends
- Service location

Server Selection

- Service is replicated in many places in network
- 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
- How do 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

Routing Based

- Anycast
  - Give service a single IP address
  - Each node implementing service advertises route to address
  - Packets get routed 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
  - Must be done on connection by connection basis – why?
    - Forces router to keep per connection state
  - 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

Application Based

- HTTP support 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
  - A-record returned is "best" one for the client
- What information can name server base decision on?
  - Server load/location → must be collected
  - Information in the name lookup request
    - Name service client → typically the local name server for client

Naming Based

- Round-robin
  - Randomly choose replica
  - Avoid hot-spots
- [Semi-]static metrics
  - Geography
  - Route metrics
  - How well would these work?

Naming Based

- Predicted application performance
  - How to predict?
  - Only have limited info at name resolution
- Multiple techniques
  - Static metrics to get coarse grain answer
  - Current performance among smaller group

Overview

- DNS
- Server selection
- DNS experience/trends
- Service location

DNS Experience

- One of the greatest challenges seemed to be getting good name server implementations
  - Developers were typically happy with "good enough" implementation
  - Challenging, large scale, wide area distributed system
    - Like routing, but easier to have broken implementations that work

DNS Experience

- Common bugs
  - Looped NS/CNAME record handling
  - Poor static configuration (root server list)
  - Lack of exponential backoff
  - 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
- Solution
  - Monitor for misbehaving name servers?

Trends

- DNS is used for server selection more and more
  - Blame Bruce Maggs for this ☺
  - What are reasonable DNS TTLs for this type of use
- Typically want to adapt to load changes
  - Low TTL for A-records → what about NS records?
  - How does this affect caching?
  - What does the first and subsequent lookup do?

Recent Measurements

- Hit rate for DNS = 80%
  - Is this good or bad?
- Most Internet traffic is Web
  - 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

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

New Registrars

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

Overview

- DNS
- Server selection
- DNS experience/trends
- Service location

Service Location

- What if you want to lookup services with more expressive descriptions than DNS names
  - E.g. please find me printers in cs.cmu.edu instead of laserjet1.cs.cmu.edu
- What do descriptions look like?
- How is the searching done?
- How will it be used?
  - Search for particular service?
  - Browse available services?
  - Composing multiple services into new service?

Service Descriptions

- Typically done as hierarchical value-attribute pairs
  - Type = printer → memory = 32MB, lang = PCL
  - Location = CMU → building = WeH
- Hierarchy based on attributes or attributes-values?
  - E.g. Country → state or country=USA → state=PA and country=Canada → province=BC?
- Can be done in something like XML

Service Discovery (Multicast)

- Services listen on well known discovery group address
- Client multicasts query to discovery group
- Services unicast replies to client
- Tradeoffs
  - Not very scalable → effectively broadcast search
  - Requires no dedicated infrastructure or bootstrap
  - Easily adapts to availability/changes
  - Can scope request by multicast scoping and by information in request

Service Discovery (Directory Based)

- Services register with central directory agent
  - Soft state → registrations must be refreshed or the expire
- Clients send query to central directory → replies with list of matches
- Tradeoffs
  - How do you find the central directory service?
    - Typically using multicast based discovery!
    - SLP also allows directory to do periodic advertisements
  - Need dedicated infrastructure

Other Issues

- Dynamic attributes
  - Many queries may be based on attributes such as load, queue length
  - E.g., print to the printer with shortest queue
- Security
  - Don't want others to serve/change queries
  - Also, don't want others to know about existence of services
    - Srini's home SLP server is advertising the $50,000 MP3 stereo system (come steal me!)