Naming

- DNS
- Assigned reading
  - [MD88] P. Mockapetris and K. Dunlap, Development of the Domain Name System
  - [JSBM01] Jaeyeon Jung, Emil Sit, Hari Balakrishnan, and Robert Morris, DNS Performance and the Effectiveness of Caching

Overview

- DNS
- Service location

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 (DNS) 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 type, class and application data
  • Classes = Internet (IN), Chaosnet (CH), etc.
  • Each class defines types, e.g. for IN:
    • A = address, NS = name server, CNAME = canonical name (for aliasing), HINFO = CPU/OS info, MX = mail exchange, PTR = pointer for reverse mapping of address to name
• Administrative hierarchy
  • “.” as separator
  • Zone = contiguous section of name space
    • Complete tree, single node or subtree

DNS Design

• Zones are created by convincing owner node to create/delegate a subzone
  • Each zone contains multiple redundant servers
  • Primary/master name server updated manually
  • Secondary/redundant servers updated by zone transfer of name space
• 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
  • Local name servers hand-configured (e.g. /etc/resolv.conf)
• Name servers
  • Configured with well-known root servers
    • Currently (a-m).root-servers.net
  • Local servers
    • Do recursive lookup of distant host names for local hosts
    • Typically answer queries about local zone

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
• 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

Typical Resolution

- Find name based on knowledge of root of name space
- Steps for resolving www.cmu.edu
  - Application calls gethostbyname()
  - Resolver contacts local name server (S₁)
  - S₁ queries root server (S₂) for www.cmu.edu
  - S₂ returns NS record for cmu.edu (S₃)
  - S₁ queries S₃ for www.cmu.edu
  - S₃ returns A record for www.cmu.edu
- Can return multiple addresses → what does this mean?

DNS Lookup Example

- Client
- Local DNS server
- Root & edu DNS server
- cmu.edu DNS server
- www.cs.cmu.edu DNS server
- NS cmu.edu
- Ftp.cs.cmu.edu DNS server
- Ftp.cs.cmu.edu DNS server
- Ftp.cs.cmu.edu DNS server

Subsequent Lookup Example

- Client
- Local DNS server
- Root & edu DNS server
- cs.cmu.edu DNS server
- cmu.edu DNS server
- www.cs.cmu.edu DNS server
- Ftp.cs.cmu.edu

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?
  - Why is it that forward lookup can have multiple answers but not reverse?
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

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 on caching local server
    - Why is this a problem?
  - Solution
    - Monitor for misbehaving name servers?

DNS Experience - Client performance

- 23% of lookups with no answer
  - Inverse lookups and bogus NS records
- 13% error response → most = no name exists
- Retransmit aggressively → most packets in trace for unanswered lookups
- Increasing share of low TTL records
- Worst 10% lookup latency got much worse

DNS Experience - Cache effectiveness

- Hit rates = 80 – 86%
- Name distribution = Zipf-like = 1/x^a
- A records → TTLs = 10 minutes similar to TTLs = infinite
- 10 client hit rate = 1000+ client hit rate
- Only 20% of requests go to root/gTLD servers
- NS record caching is much more important to scalability
Overview

• DNS

• 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 \(\rightarrow\) memory = 32MB, lang = PCL
  • Location = CMU \(\rightarrow\) building = WeH

• Hierarchy based on attributes or attributes-values?
  • E.g. Country \(\rightarrow\) state or country=USA \(\rightarrow\) state=PA and country=Canada \(\rightarrow\) 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 \(\rightarrow\) 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 \(\rightarrow\) registrations must be refreshed or the expire
  • Clients send query to central directory \(\rightarrow\) 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
  • How do directory agents interact with each other?
  • Well suited for browsing and composition \(\rightarrow\) knows full list of services

• How do you find the central directory service?
  • Typically using multicast based discovery!
  • SLP also allows directory to do periodic advertisements
  • Need dedicated infrastructure
  • How do directory agents interact with each other?
  • Well suited for browsing and composition \(\rightarrow\) knows full list of services

Service Discovery (Routing Based)

• Client issues query to overlay network
  • Query can include both service description and actual request for service
  • Overlay network routes query to desired service[s]
  • If query only description, subsequent interactions can be outside overlay (early-binding)
  • If query includes request, client can send subsequent queries via overlay (late-binding)
  • Subsequent requests may go to different services agents
  • Enables easy fail-over/mobility of service

• Tradeoffs
  • Routing on complex parameters can be difficult/expensive
  • Can work especially well in ad-hoc networks
  • Can late-binding really be used in many applications?
Wide Area Scaling

- How do we scale discovery to wide area?
  - Hierarchy?
  - Hierarchy must be based on attribute of services
    - All services must have this attribute
    - All queries must include (implicitly or explicitly) this attribute
  - Tradeoffs
    - What attribute? Administrative (like DNS)? Geographic? Network Topologic?
    - Should we have multiple hierarchies?
    - Do we really need hierarchy? Search engines seem to work fine!

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
    - Srin’s home SLP server is advertising the $50,000 MP3 stereo system (come steal me!)

Next Lecture: Midterm

- Closed book
- Up through fair queuing
- Last year’s exam posted on Web page