15-440 Distributed Systems

Lecture 19 – Naming and Hashing

Names

- Names are associated with objects
  - Enables passing of references to objects
  - Indirection
  - Deferring decision on meaning/binding
- Examples
  - Registers → R5
  - Memory → 0xdeadbeef
  - Host names → srini.com
  - User names → sseshan
  - Email → srini@cmu.edu
  - File name → /usr/srini/foo.txt
  - URLs → http://www.srini.com/index.html
  - Ethernet → f8:e4:fb:bf:3d:a6

Name Lookup Styles

- Table lookup
  - Simple, table per context
- Recursive
  - Names consist of context + name
  - E.g. path + filename, hostname + domain name
  - Context name must also be resolved
    - Need special context such as “root” built into resolver
- Multiple lookup
  - Try multiple contexts to resolve name → search paths

Name Discovery

- Well-known name
  - www.google.com, port 80…
- Broadcast
  - Advertise name → e.g. 802.11 Beacons
- Query
  - Use google
- Broadcast query
  - Ethernet ARP
- Use another naming system
  - DNS returns IP addresses
- Introductions
  - Web page hyperlinks
- Physical rendezvous
  - Exchange info in the real world

Naming Model

- 3 key elements

1. Name space
   - Alphabet of symbols + syntax that specify names
2. Name-mapping
   - Associates each name to some value in…
3. Universe of values
   - Typically an object or another name from original name space (or another name space)

- Name-to-value mapping is called a “binding” i.e. name is bound to value

Names

- Uniqueness
  - One-to-one mapping
  - One-to-many or many-to-one (name-to-value) mappings
  - Context sensitive resolution
- Stable binding
  - Names that are never reused
  - Values that can only have one name
  - E.g. using MD5 of file contents, bank account numbers
- Reverse lookup support

Name Mapping

- Names are mapped to values within some context
  - E.g., different lookup tables for names in different settings
- Two sources for context
  - Resolver can supply default context
  - Name can specify an explicit context to use → qualified name
    - “cd /users/srini/440/midterm” vs “cd 440/midterm”

Context

- Common problem → what context to use for names without context
- Consider email from CMU
  - To: srini, yuvraj@gmail.com
  - What happens when yuvraj replies to all?
    - What context will he email srini
  - Solutions:
    - Sendmail converts all address to qualified names
      - Not in body of message
    - Provide context information in email header
      - E.g. like base element in HTML

Overview

- DNS Review
- DNS Details
- Hashing Tricks

DNS Records

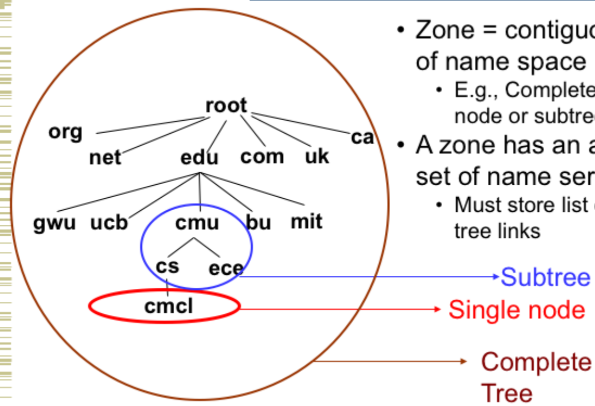
RR format: (class, name, value, type, ttl)

- 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: Zone Definitions

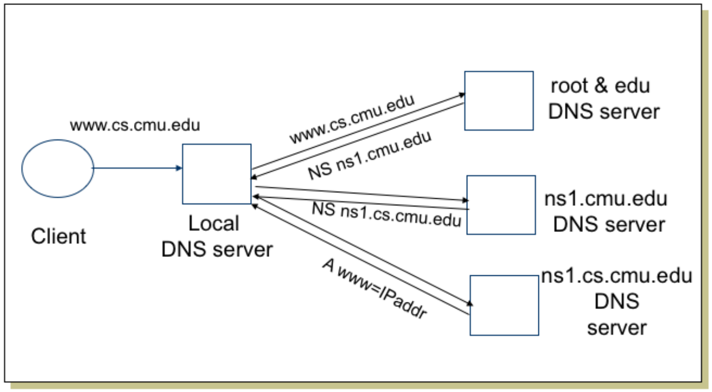


- Zone = contiguous section of name space
  - E.g., Complete tree, single node or subtree
- A zone has an associated set of name servers
  - Must store list of names and tree links

Workload and Caching

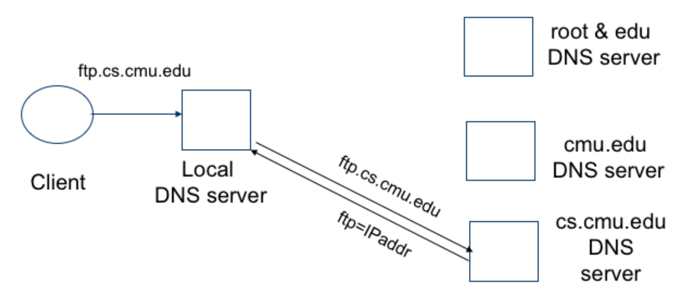
- Are all servers/names likely to be equally popular?
  - Why might this be a problem? How can we solve this problem?
- DNS responses are cached
  - Quick response for repeated translations
  - Other queries may reuse some parts of lookup
    - NS records for domains
- DNS negative queries are cached
  - 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



13

Subsequent Lookup Example

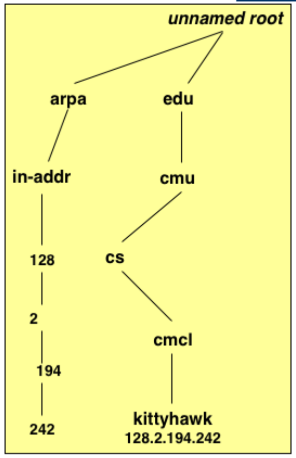


14

Overview

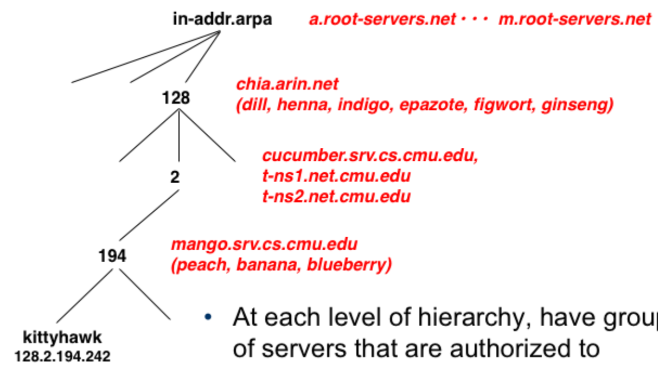
- DNS Review
- DNS Details
- Hashing Tricks

Reverse DNS



- Task
  - Given IP address, find its name
- Method
  - Maintain separate hierarchy based on IP names
  - Write 128.2.194.242 as 242.194.2.128.in-addr.arpa
    - Why is the address reversed?
- Managing
  - Authority manages IP addresses assigned to it
  - E.g., CMU manages name space 128.2.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

Tracing Hierarchy (1)

- Dig Program
  - 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
greatwhite.ics.cs.cmu.edu

;; ADDITIONAL SECTION:
a.edu-servers.net    172800  IN    A       192.5.6.30
c.edu-servers.net.   172800  IN    A       192.26.92.30
d.edu-servers.net.   172800  IN    A       192.31.80.30
f.edu-servers.net.   172800  IN    A       192.35.51.30
g.edu-servers.net.   172800  IN    A       192.42.93.30
g.edu-servers.net.   172800  IN    AAAA    2001:503:cc2c::2:36
l.edu-servers.net.   172800  IN    A       192.41.162.30
```

IP v6 address

  - All .edu names handled by set of servers

Prefetching

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

Tracing Hierarchy (2)

- 3 servers handle CMU names

```
unix> dig +norecurse @g.edu-servers.net NS
greatwhite.ics.cs.cmu.edu

;; AUTHORITY SECTION:
cmu.edu.         172800  IN      NS      ny-server-03.net.cmu.edu.
cmu.edu.         172800  IN      NS      nsauth1.net.cmu.edu.
cmu.edu.         172800  IN      NS      nsauth2.net.cmu.edu.
```

Tracing Hierarchy (3 & 4)

- 3 servers handle CMU CS names

```
unix> dig +norecurse @nsauth1.net.cmu.edu NS
greatwhite.ics.cs.cmu.edu

;; AUTHORITY SECTION:
cs.cmu.edu.     600     IN      NS      AC-DDNS-2.NET.cs.cmu.edu.
cs.cmu.edu.     600     IN      NS      AC-DDNS-1.NET.cs.cmu.edu.
cs.cmu.edu.     600     IN      NS      AC-DDNS-3.NET.cs.cmu.edu.
```

```
unix>dig +norecurse @AC-DDNS-2.NET.cs.cmu.edu NS
greatwhite.ics.cs.cmu.edu

;; AUTHORITY SECTION:
cs.cmu.edu.     300     IN      SOA   PLANISPHERE.FAC.cs.cmu.edu.
```

21

DNS Hack #1

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

- Load Balance
  - Server sends out multiple A records
  - Order of these records changes per-client

Server Balancing Example

- DNS Tricks

```
unix1> dig www.google.com

;; ANSWER SECTION:
www.google.com.        87775   IN      CNAME   www.l.google.com.
www.l.google.com.      81      IN      A       72.14.204.104
www.l.google.com.      81      IN      A       72.14.204.105
www.l.google.com.      81      IN      A       72.14.204.147
www.l.google.com.      81      IN      A       72.14.204.99
www.l.google.com.      81      IN      A       72.14.204.103
```

```
unix2> dig www.google.com

;; ANSWER SECTION:
www.google.com.        603997  IN      CNAME   www.l.google.com.
www.l.google.com.      145     IN      A       72.14.204.99
www.l.google.com.      145     IN      A       72.14.204.103
www.l.google.com.      145     IN      A       72.14.204.104
www.l.google.com.      145     IN      A       72.14.204.105
www.l.google.com.      145     IN      A       72.14.204.147
```

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

gTLDs

- Unsponsored
  - .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

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

26

Do you trust the TLD operators?

- Wildcard DNS record for all .com and .net domain names not yet registered by others
  - September 15 – October 4, 2003
  - February 2004: Verisign sues ICANN
- Redirection for these domain names to Verisign web portal (SiteFinder)

27

Overview

- DNS Refresh
- DNS Details
- Hashing Tricks

DNS (Summary)

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

Hashing

Two uses of hashing that are becoming wildly popular in distributed systems:

- Content-based naming
- Consistent Hashing of various forms

Example systems that use them

- BitTorrent & many other modern p2p systems use content-based naming
- Content distribution networks such as Akamai use consistent hashing to place content on servers
- Amazon, Linkedin, etc., all have built very large-scale key-value storage systems (databases--) using consistent hashing

Dividing items onto storage servers

- Option 1: Static partition (items a-c go there, d-f go there, ...)
  - If you used the server name, what if "cowpatties.com" had 1000000 pages, but "zebras.com" had only 10? → Load imbalance
  - Could fill up the bins as they arrive → Requires tracking the location of every object at the front-end.

Hashing 1

- Let nodes be 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.
- Any other problems?
  - What happens if a node goes down?
  - What happens if a node comes back up?
  - What if different nodes have different views?

Option 2: Conventional Hashing

- bucket = hash(item) % num_buckets
- Sweet! Now the server we use is a deterministic function of the item, e.g., sha1(URL) → 160 bit ID % 20 → a server ID
- But what happens if we want to add or remove a server?

Option 2: Conventional 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 ☹

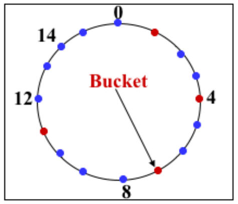
Consistent Hash

- "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

36

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

Detail - “virtual” nodes

- The way we outlined it results in moderate load imbalance between buckets (remember balls and bins analysis of hashing?)
- To reduce imbalance, systems often represent each physical node as k different buckets, sometimes called “virtual nodes” (but really, it’s just multiple buckets).
- log n buckets gets you a very pleasing load balance - O(#items/n) with high probability, if #items large and uniformly distributed

Hashing 2: For naming

- Many file systems split files into blocks and store each block on a disk.
- Several levels of naming:
  - Pathname to list of blocks
  - Block #s are addresses where you can find the data stored therein. (But in practice, they're logical block #s – the disk can change the location at which it stores a particular block... so they're actually more like names and need a lookup to location :)

A problem to solve...

- Imagine you're creating a backup server
- It stores the full data for 1000 CMU users' laptops
- Each user has a 100GB disk.
- That's 100TB and lots of $$$
- How can we reduce the storage requirements?

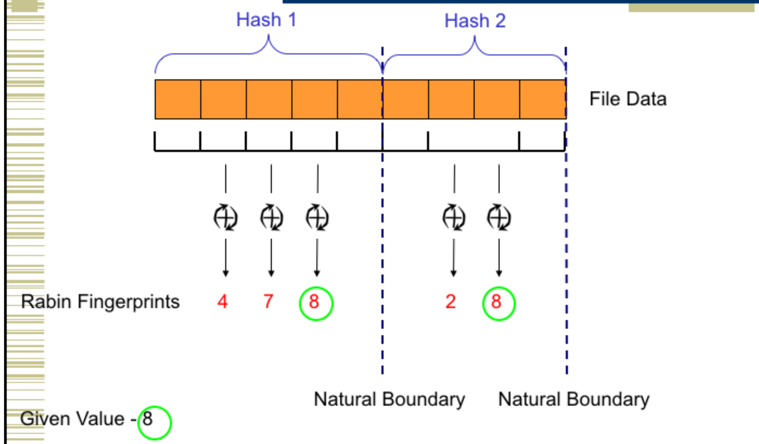
"Deduplication"

- A common goal in big archival storage systems. Those 1000 users probably have a lot of data in common -- the OS, copies of binaries, maybe even the same music or movies
- How can we detect those duplicates and coalesce them?
- One way: Content-based naming, also called content-addressable foo (storage, memory, networks, etc.)
- A fancy name for...

Name items by their hash

- Imagine that your filesystem had a layer of indirection:
  - pathname → hash(data)
  - hash(data) → list of blocks
- For example:
  - /src/foo.c -> 0xfff32f2fa11d00f0
  - 0xfff32f2fa11d00f0 -> [5623, 5624, 5625, 8993]
- If there were two identical copies of foo.c on disk ... We'd only have to store it once!
  - Name of second copy can be different

Name boundaries by their hash: Rabin Fingerprinting



43

Self-Certifying Names

- Several p2p systems operate something like:
- Search for “national anthem”, find a particular file name (starspangled.mp3).
- Identify the files by the hash of their content (0x2fab4f001...)
- Request to download a file whose hash matches the one you want
- Advantage? You can verify what you got, even if you got it from an untrusted source (like some dude on a p2p network)

Self-certifying Names

- Use a name that helps validate the data associated with the name
  - Seems like a circular argument but…
  - Traditional name → Declaration of Independence
  - Self-certifying name →
    SHA1(Declaration of Independence contents)
    - SHA1 → cryptographic hash

Self-Certifying Names

- Can also create names using public key crypto
  - Name = Hash(pubkey, salt)
  - Value = <pubkey, salt, data, signature>
  - Signature == [cryptohash(data)] encrypt with prvkey
  - Can verify name related to pubkey and pubkey signed data
- Benefits
  - Can verify contents after receiving file
  - Can fetch file from untrustworthy sources
- Weaknesses
  - No longer human readible

46

Hash functions

- Given a universe of possible objects U, map N objects from U to an M-bit hash.
- Typically, $|U| >>> 2^M$.
  - This means that there can be collisions: Multiple objects map to the same M-bit representation.
- Likelihood of collision depends on hash function, M, and N.
  - Birthday paradox → roughly 50% collision with $2^{M/2}$ objects for a well designed hash function

Desirable Properties (Cryptographic Hashes)

- Compression: Maps a variable-length input to a fixed-length output
- Ease of computation: A relative metric...
- Pre-image resistance:
  - Given a hash value h it should be difficult to find any message m such that h = hash(m)
- 2nd pre-image resistance:
  - Given an input m_1 it should be difficult to find different input m_2 such that $hash(m_1) = hash(m_2)$
- collision resistance:
  - difficult to find two different messages m_1 and m_2 such that $hash(m_1) = hash(m_2)$

Longevity

- “Computationally infeasible” means different things in 1970 and 2012.
  - Moore’s law
  - Some day, maybe, perhaps, sorta, kinda: Quantum computing.
- Hash functions are not an exact science yet.
  - They get broken by advances in crypto.

Real hash functions

Name	Introduced	Weakened	Broken	Lifetime	Replacement
MD4	1990	1991	1995	1-5y	MD5
MD5	1992	1994	2004	8-10y	SHA-1
MD2	1992	1995	abandoned	3y	SHA-1
RIPEMD	1992	1997	2004	5-12y	RIPEMD-160
HAVAL-128	1992	-	2004	12y	SHA-1
SHA-0	1993	1998	2004	5-11y	SHA-1
SHA-1	1995	2004	not quite yet	9+	SHA-2 & 3
SHA-2 (256, 384, 512)	2001	still good			
SHA-3	2012	brand new			

Using them

- How long does the hash need to have the desired properties (preimage resistance, etc)?
  - rsync: For the duration of the sync;
  - dedup: Until a (probably major) software update;
  - store-by-hash: Until you replace the storage system
- What is the adversarial model?
  - Protecting against bit flips vs. an adversary who can try 1B hashes/second?

Final pointer

- Hashing forms the basis for MACs - message authentication codes
  - Basically, a hash function with a secret key.
  - H(key, data) - can only create or verify the hash given the key.
  - Very, very useful building block

Summary

- Hashes used for:
  - Splitting up work/storage in a distributed fashion
  - Naming objects with self-certifying properties
- Key applications
  - Key-value storage
  - P2P content lookup
  - Deduplication
  - MAC
- Many types of naming
  - DNS names, IP addresses, Ethernet addresses, content-based addresses
  - Make sure you understand differences

Overview

- BigTable
- Spanner
- Naming Overview
- Hashing Tricks