15-440 Distributed Systems

Lecture 20 – Hashing and CDNs

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

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) DIV num_buckets
- Sweet! Now the server we use is a deterministic function of the item, e.g., sha1(URL) → 160 bit ID DIV 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 ☹

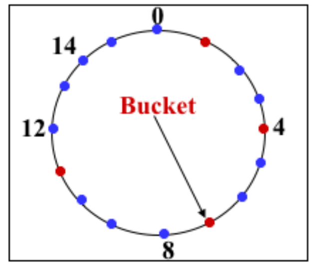
6

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

8

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?

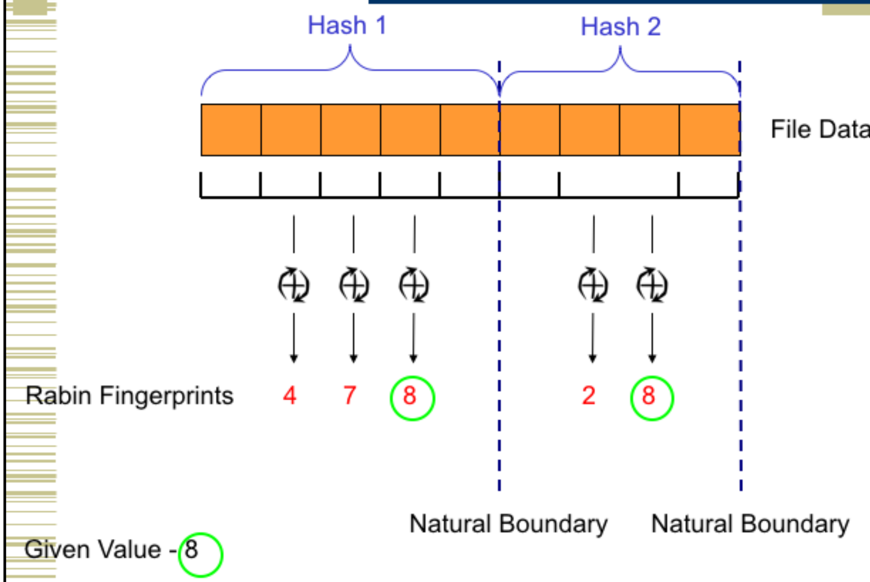
“Deduplication”

- 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



14

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

16

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

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

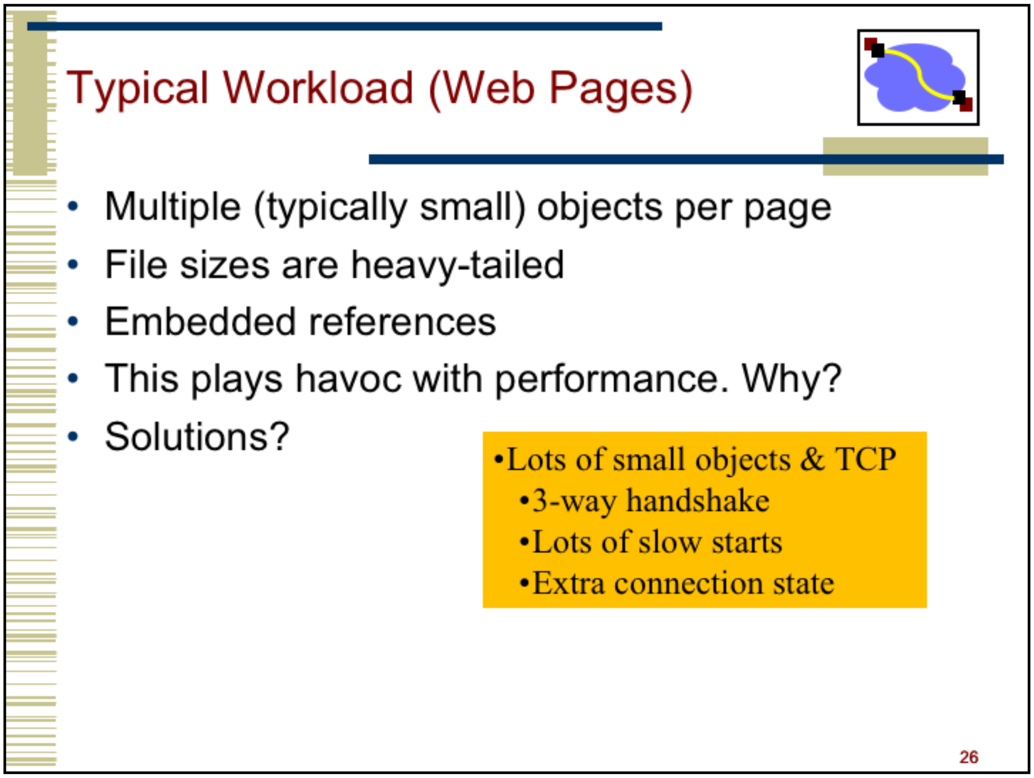
Summary

- 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

24

Outline

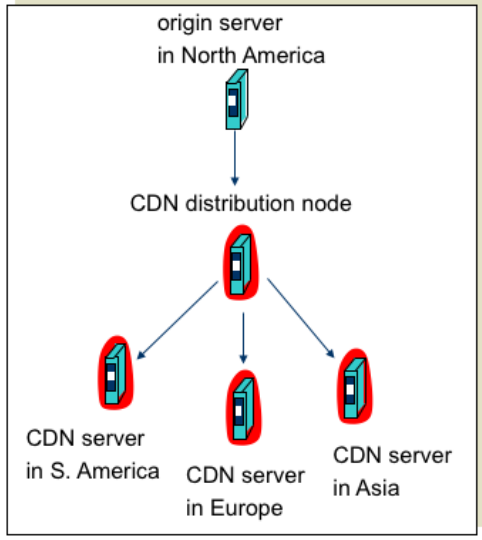
- Hashing
- Content Distribution Networks



3 way handshake, multiple slow starts

Content Distribution Networks (CDNs)

- The content providers are the CDN customers.
- Content replication
- CDN company installs hundreds of CDN servers throughout Internet
  - Close to users
- CDN replicates its customers' content in CDN servers. When provider updates content, CDN updates servers



27

Slashdot NEWS FOR NERDS. STUFF THAT MATTERS.

▶ Log In | Create Account | Help | Subscribe | Firehose

▼ Sections

- Main
- Apple
- AskSlashdot
- Books
- Developers
- Games
- Hardware
- IT
- Idle
- Index
- Interviews

Political Sites Scale Up For Election Traffic

Posted by timothy **on Tuesday November 04, @12:15PM**
from the **hearken-are-those-trumpets** dept.

miller60 writes

"News sites and political blogs are expecting extraordinary traffic tonight as Americans track results of the Presidential election, and are scaling their infrastructure to meet the challenge. Yahoo anticipates its Election Night traffic may be three times the volume seen in 2004, when it had 80 million page views on Election Day and 142 million more visits the following day. Hosting companies say customers have been ordering extra servers and load balancing services, while content delivery networks are also expecting a busy night. Will traffic approach record levels? Akamai's Net Usage Index, which tracks traffic to its customer news sites, is one metric to watch."

http://www.akamai.com/html/technology/nui/news/index.html

Content Distribution Networks & Server Selection

- 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

29

Server Selection

- 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 to direct clients to a particular server?
  - As part of routing → anycast, cluster load balancing
    - Not covered ☹
  - As part of application → HTTP redirect
  - As part of naming → DNS

30

Application Based

- HTTP supports simple way to indicate that Web page has moved (30X responses)
- Server receives 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, but…
  - HTTP Redirect has some design flaws – especially with current browsers

31

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

How Akamai Used to Work

- 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

33

How Akamai Works

- Clients delegate domain to akamai
  - ibm.com. 172800 IN NS usw2.akam.net.
- CNAME records eventually lead to
  - Something like e2874.x.akamaiedge.net. For IBM
  - Or a1686.q.akamai.net for IKEA….
- Client is forced to resolve eXYZ.x.akamaiedge.net. hostname

34

How Akamai Works

- How is content replicated?
- Akamai only replicates static content (*)
- Modified name contains original file name
- Akamai server is asked for content
  - First checks local cache
  - If not in cache, requests file from primary server and caches file

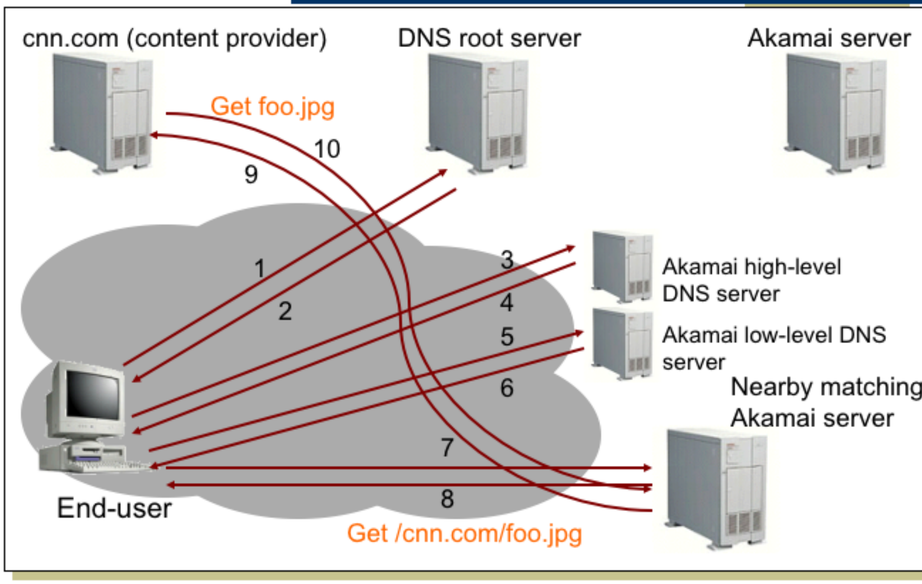
* (At least, the version we're talking about today. Akamai actually lets sites write code that can run on Akamai's servers, but that's a pretty different beast)

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 chooses server in region
  - Should try to chose server that has file in cache - How to choose?
  - Uses aXYZ name and hash
  - TTL is small → why?

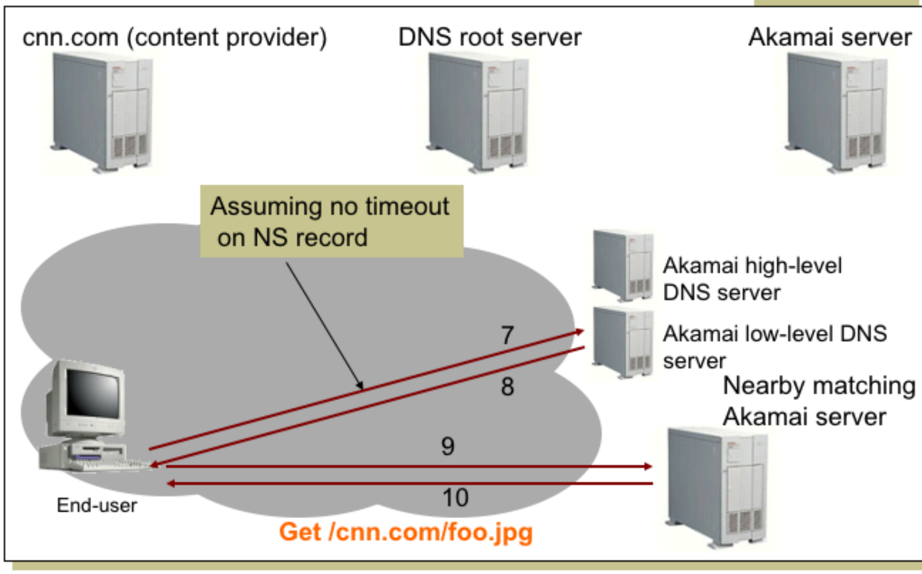
36

How Akamai Works



37

Akamai – Subsequent Requests



38

Consistent Hashing Reminder…

- Finding a nearby server for an object in a CDN uses centralized knowledge.
- Consistent hashing can also be used in a distributed setting
- Consistent Hashing to the rescue.

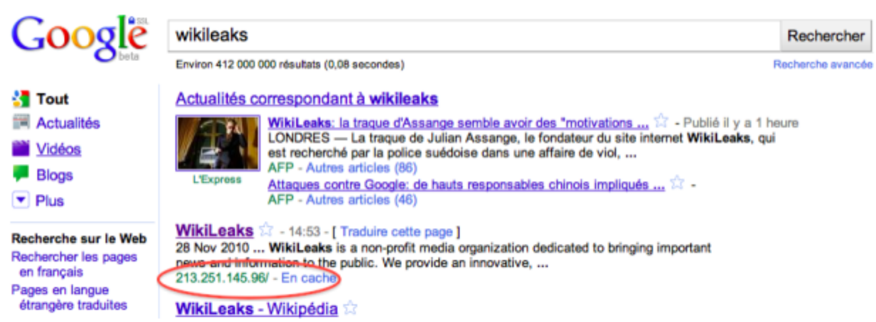
Summary

- DNS
- Content Delivery Networks move data closer to user, maintain consistency, balance load
  - Consistent Caching maps keys AND buckets into the same space
  - Consistent caching can be fully distributed, useful in P2P systems using structured overlays

40

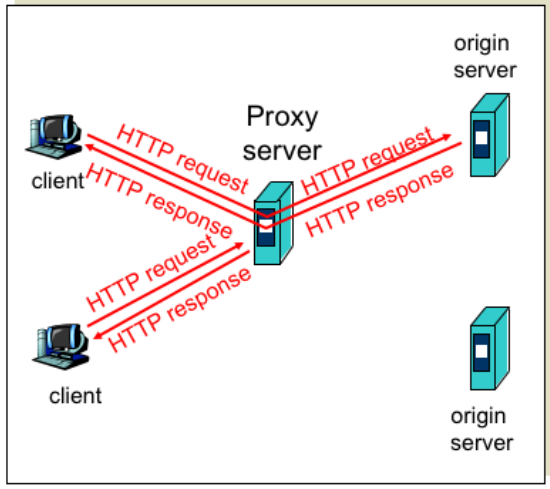
A rose by any other name….

- "DNS Is Sexy: Making Things Go While Making It Fun" (http://dyn.com/dns-is-sexy/)
  - If you can convince yourself that something like DNS is sexy, then Dyn must be a great place to work
- TheGoodThingAboutDomainNameJokesISThatAllTheGoodShortOnesHaveBeenTold.com unless you're being creati.ve



Web Proxy Caches

- User configures browser: Web accesses via cache
- Browser sends all HTTP requests to cache
  - Object in cache: cache returns object
  - Else cache requests object from origin server, then returns object to client



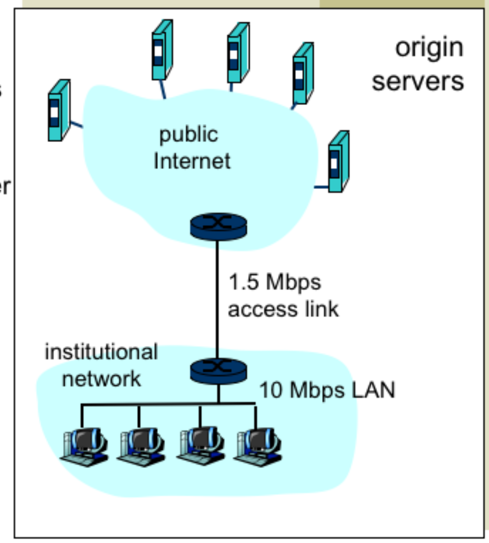
42

No Caching Example (1)

Assumptions
- Average object size = 100,000 bits
- Avg. request rate from institution's browser to origin servers = 15/sec
- Delay from institutional router to any origin server and back to router = 2 sec

Consequences
- Utilization on LAN = 15%
- Utilization on access link = 100%
- Total delay = Internet delay + access delay + LAN delay

= 2 sec + minutes + milliseconds



No Caching Example (2)

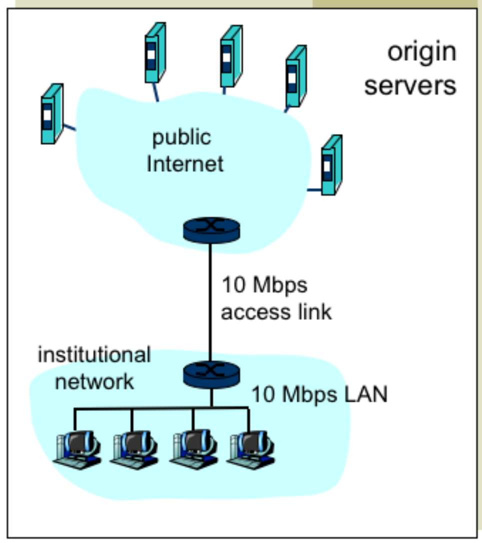
Possible solution

- Increase bandwidth of access link to, say, 10 Mbps
- Often a costly upgrade

Consequences

- Utilization on LAN = 15%
- Utilization on access link = 15%
- Total delay = Internet delay + access delay + LAN delay

= 2 sec + msecs + msecs



44

Problems

- Over 50% of all HTTP objects are uncacheable – why?
- Not easily solvable
  - Dynamic data → stock prices, scores, web cams
  - CGI scripts → results based on passed parameters
- Obvious fixes
  - SSL → encrypted data is not cacheable
    - Most web clients don't handle mixed pages well →many generic objects transferred with SSL
  - Cookies → results may be based on passed data
  - Hit metering → owner wants to measure # of hits for revenue, etc.

45

Caching Proxies – Sources for Misses

- Capacity
  - How large a cache is necessary or equivalent to infinite
  - On disk vs. in memory → typically on disk
- Compulsory
  - First time access to document
  - Non-cacheable documents
    - CGI-scripts
    - Personalized documents (cookies, etc)
    - Encrypted data (SSL)
- Consistency
  - Document has been updated/expired before reuse

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 = 80% → 1 - (#DNS/#connections)
  - 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

DNS Experience

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

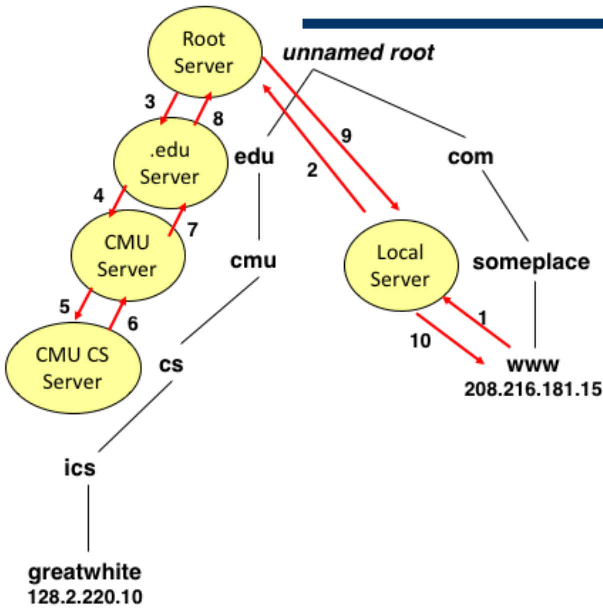
DNS Experience

- Hit rate for DNS = 80% → 1-(#DNS/#connections)
  - Most Internet traffic is Web
  - What does a typical page look like? → average of 4-5 imbedded objects → needs 4-5 transfers → accounts for 80% hit rate!
- 70% hit rate for NS records → 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 → TTLs = 10 minutes similar to TTLs = infinite
- 10 client hit rate = 1000+ client hit rate

Mail Addresses

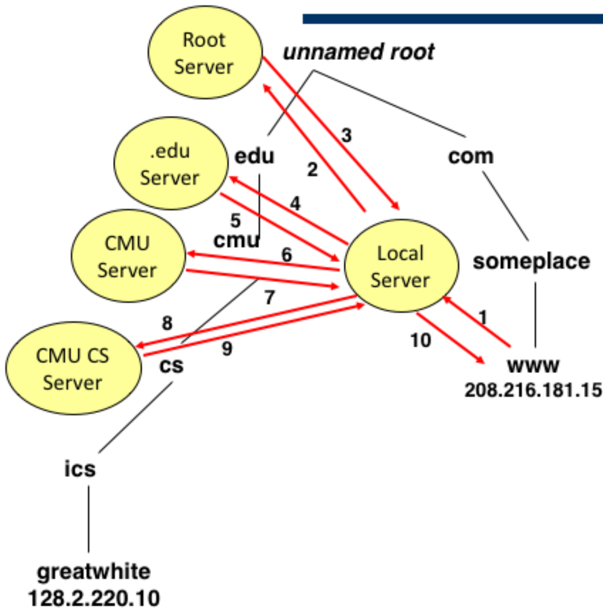
- MX records point to mail exchanger for a name
  - E.g. mail.acm.org is MX for acm.org
- Addition of MX record type proved to be a challenge
  - How to get mail programs to lookup MX record for mail delivery?
  - Needed critical mass of such mailers

Recursive DNS Name Resolution



- Nonlocal Lookup
  - Recursively from root server downward
  - Results passed up
- Caching
  - Results stored in caches along each hop
  - Can shortcircuit lookup when cached entry present

Iterative DNS Name Resolution



- Nonlocal Lookup
  - At each step, server returns name of next server down
  - Local server directly queries each successive server
- Caching
  - Local server builds up cache of intermediate translations
  - Helps in resolving names xxx.cs.cmu.edu, yy.cmu.edu, and z.edu

52

DNS Hack #2: Blackhole Lists

- First: Mail Abuse Prevention System (MAPS)
  - Paul Vixie, 1997
- Today: Spamhaus, spamcop, dnsrbl.org, etc.

Different addresses refer to different reasons for blocking

% dig 91.53.195.211.bl.spamcop.net

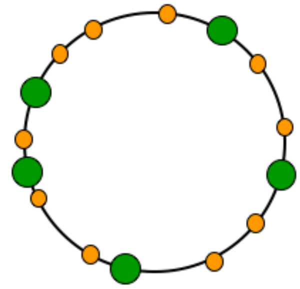
```
;; ANSWER SECTION:
91.53.195.211.bl.spamcop.net. 2100 IN   A       127.0.0.2

;; ANSWER SECTION:
91.53.195.211.bl.spamcop.net. 1799 IN   TXT     "Blocked - see
http://www.spamcop.net/bl.shtml?211.195.53.91"
```

53

Consistent Hashing

- **Main idea:**
  - map both keys and nodes to the same (metric) identifier space
  - find a "rule" how to assign keys to nodes

Ring is one option.



54

Consistent Hashing

- The consistent hash function assigns each node and key an *m*-bit identifier using SHA-1 as a base hash function
- **Node identifier:** SHA-1 hash of IP address
- **Key identifier:** SHA-1 hash of key

Identifiers

- *m* bit identifier space for both keys and nodes
- **Key identifier:** SHA-1(key)

  Key="LetItBe" —SHA-1→ ID=60

- **Node identifier:** SHA-1(IP address)

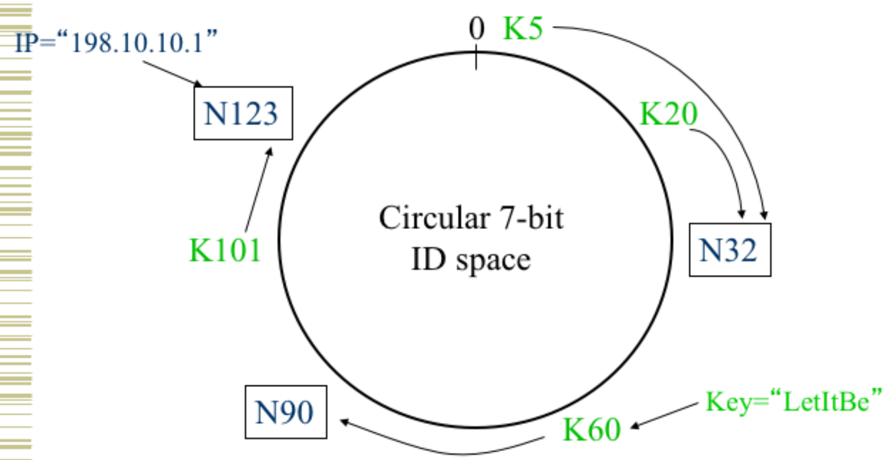
  IP="198.10.10.1" —SHA-1→ ID=123

•How to map key IDs to node IDs?

56

Consistent Hashing Example

Rule: A key is stored at its **successor**: node with next higher or equal ID



57

Consistent Hashing Properties

- **Load balance:** all nodes receive roughly the same number of keys

- For *N* nodes and *K* keys, with high probability
  - each node holds at most $(1+\varepsilon)K/N$ keys
  - (provided that K is large enough compared to N)

Load Balance

- Redirector knows all CDN server Ids
- Can track approximate load (or delay)
- To balance load:
  - W_i = Hash(URL, ip of s_i) for all i
  - Sort W_i from high to low
  - find first server with low enough load
- Benefits?
- How should “load” be measured?

59