15-744: Computer Networking

L-17 DNS and the Web

DNS and the Web

- DNS
- CDNs
- Readings
  - DNS Performance and the Effectiveness of Caching
  - Development of the Domain Name System

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 and CDNs

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
- 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 Records

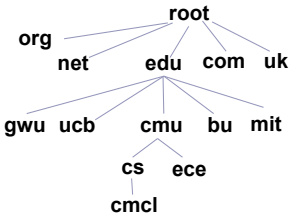
RR format: **(class, name, value, type, ttl)**

- DB contains tuples called resource records (RRs)
  - Classes = Internet (IN), Chaosnet (CH), etc.
  - Each class defines value associated with type

FOR IN class:

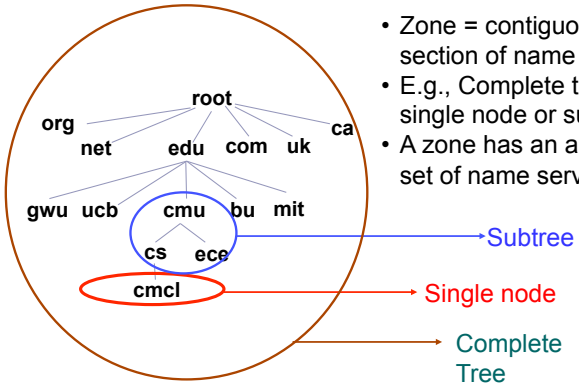
- Type=A
  - **name** is hostname
  - **value** is IP address
- Type=NS
  - **name** is domain (e.g. foo.com)
  - **value** is name of authoritative name server for this domain
- Type=CNAME
  - **name** is an alias name for some "canonical" (the real) name
  - **value** is canonical name
- Type=MX
 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

DNS Design: 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.cmcl.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

DNS Design: Cont.

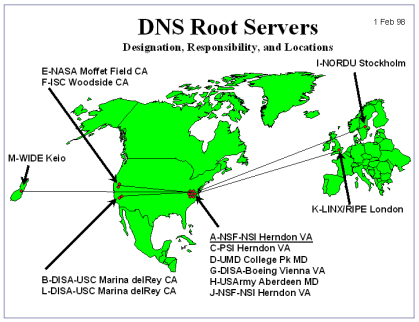
- Zones are created by convincing owner node to create/delegate a subzone
  - Records within zone stored multiple redundant name servers
  - Primary/master name server updated manually
  - Secondary/redundant servers updated by zone transfer of name space
    - Zone transfer is a bulk transfer of the "configuration" of a DNS server – uses TCP to ensure reliability
- Example:
  - CS.CMU.EDU created by CMU.EDU administrators

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
  - Either responsible for some zone or…
  - 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



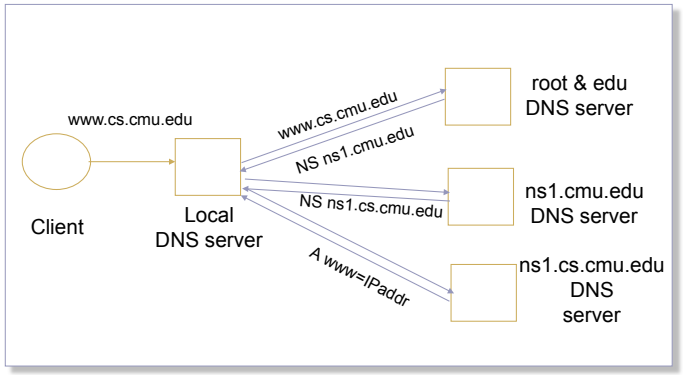
DNS Message Format

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

Typical Resolution



Typical Resolution

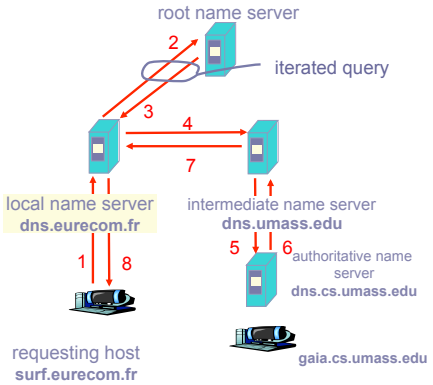
- Steps for resolving www.cmu.edu
  - Application calls gethostbyname() (RESOLVER)
  - Resolver contacts local name server (S_1)
  - S_1 queries root server (S_2) for (www.cmu.edu)
  - S_2 returns NS record for cmu.edu (S_3)
  - What about A record for S_3?
    - This is what the additional information section is for (PREFETCHING)
  - 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?

Lookup Methods

Recursive query:
- Server goes out and searches for more info (recursive)
- Only returns final answer or "not found"

Iterative query:
- Server responds with as much as it knows (iterative)
- "I don't know this name, but ask this server"

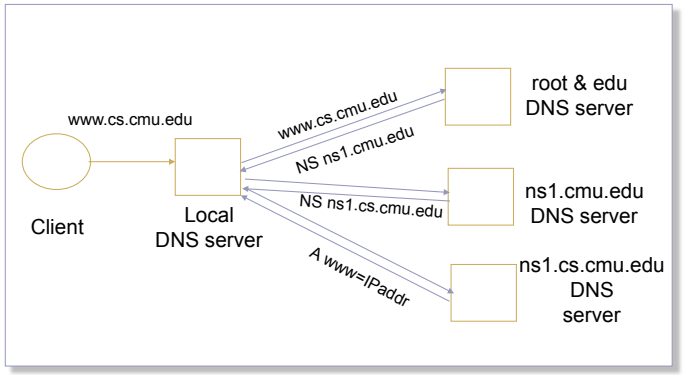
Workload impact on choice?
- Local server typically does recursive
- Root/distant server does iterative



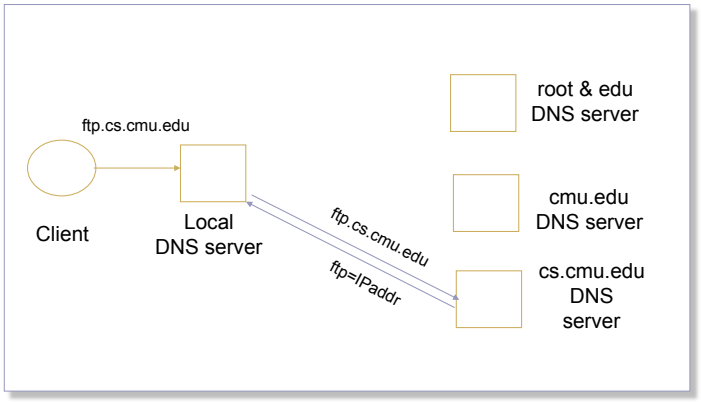
Workload and Caching

- What workload do you expect for different servers/names?
  - 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



Subsequent Lookup Example



21

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

22

Reverse Name Lookup

- 128.2.206.138?
  - Lookup 138.206.2.128.in-addr.arpa
  - Why is the address reversed?
  - Happens to be www.intel-iris.net and mammoth.cmcl.cs.cmu.edu → what will reverse lookup return? Both?
    - Should only return name that reflects address allocation mechanism

23

Prefetching

- Name servers can add additional data to 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"

24

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

25

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, .name

26

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 handle root servers

27

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)
- What services might this break?

28

Protecting the Root Nameservers

Attack On Internet Called Largest Ever

By David McGuire and Brian Krebs
washingtonpost.com Staff Writers
Tuesday, October 22, 2002; 5:40 PM

The heart of the Internet sustained its largest and most sophisticated attack ever, starting late Monday, according to officials at key online backbone organizations.

Around 5:00 p.m. EDT on Monday, a "distributed denial of service" (DDOS) attack struck the 13 "root servers" that provide the primary roadmap for almost all Internet communications. Despite the scale of the attack, which lasted about an hour, Internet users worldwide were largely unaffected, experts said.

Sophisticated?
Why did nobody notice?

seshan.org. 13759 IN NS www.seshan.org.

Defense Mechanisms

- Redundancy: 13 root nameservers
- IP Anycast for root DNS servers {c,f,i,j,k}.root-servers.net
  - RFC 3258
  - Most *physical* nameservers lie outside of the US

29

Defense: Replication and Caching

Letter	Old name	Operator	Location
A	ns.internic.net	VeriSign	Dulles, Virginia, USA
B	ns1.isi.edu	ISI	Marina Del Rey, California, USA
C	c.psi.net	Cogent Communications	distributed using anycast
D	terp.umd.edu	University of Maryland	College Park, Maryland, USA
E	ns.nasa.gov	NASA	Mountain View, California, USA
F	ns.isc.org	ISC	distributed using anycast
G	ns.nic.ddn.mil	U.S. DoD NIC	Columbus, Ohio, USA
H	aos.arl.army.mil	U.S. Army Research Lab	Aberdeen Proving Ground, Maryland, USA
I	nic.nordu.net	Autonomica	distributed using anycast
J		VeriSign	distributed using anycast
K		RIPE NCC	distributed using anycast
L		ICANN	Los Angeles, California, USA
M		WIDE Project	distributed using anycast

source: wikipedia

30

DNS Hack #1: Load Balance

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

31

DNS Hack #3: 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"

32

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?

33

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

34

Some Interesting Alternatives

- CoDNS
  - Lookup failures
    - Packet loss
    - LDNS overloading
    - Cron jobs
    - Maintenance problems
  - Cooperative name lookup scheme
  - If local server OK, use local server
  - When failing, ask peers to do lookup
- Push DNS
  - Top of DNS hierarchy is relatively stable
  - Why not replicate much more widely?

35

Overview

- DNS
- Server selection and CDNs

36

CDN

- 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
    - DNS, HTTP 304 response, anycast, etc.
- Akamai

37

Server Selection

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

38

Routing Based

- Anycast
  - Give service a single IP address
  - Each node implementing service advertises route to address
  - Packets get 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?

39

Routing Based

- Cluster load balancing
  - Router in front of cluster of nodes directs packets to server
  - Can only look at global address (L3 switching)
  - Often want to do this on a connection by connection basis – why?
    - Forces router to keep per connection state
    - L4 switching – transport headers, port numbers
  - 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 (L7 switching)

40

Application Based

- HTTP supports 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?

41

Naming Based

- Client does name lookup for service
- Name server chooses appropriate server address
- What information can it base decision on?
  - Server load/location → must be collected
  - Name service client
    - Typically the local name server for client
- Round-robin
  - Randomly choose replica
  - Avoid hot-spots
- [Semi-]static metrics
  - Geography
  - Route metrics
  - How well would these work?

42

How Akamai Works

- 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

43

How Akamai Works

- How is content replicated?
- Akamai only replicates static content
  - Serves about 7% of the Internet traffic ! (in 2003)
- Modified name contains original file
- Akamai server is asked for content
  - First checks local cache
  - If not in cache, requests file from primary server and caches file

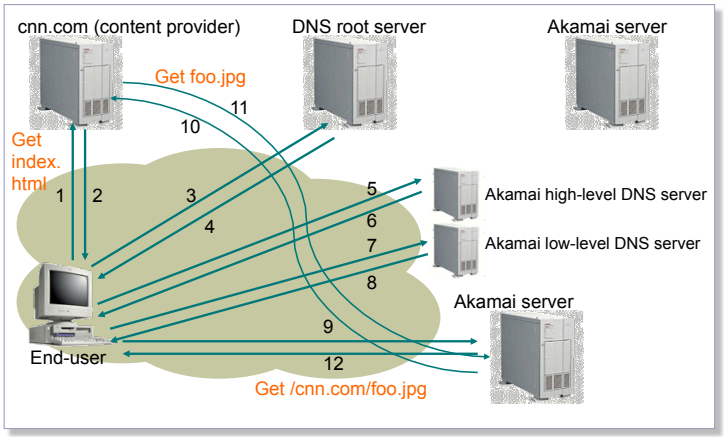
44

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

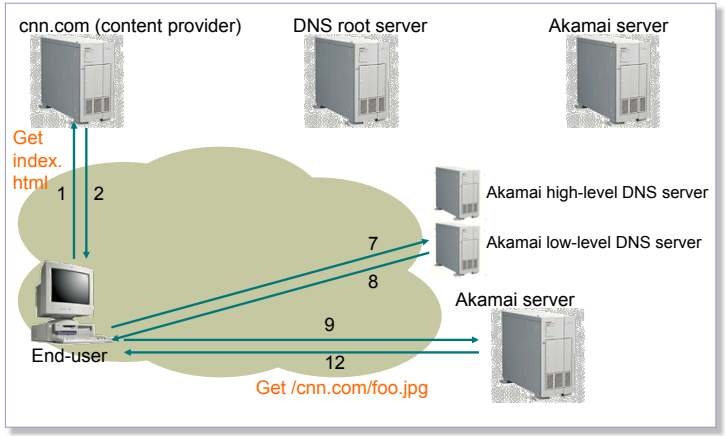
45

How Akamai Works



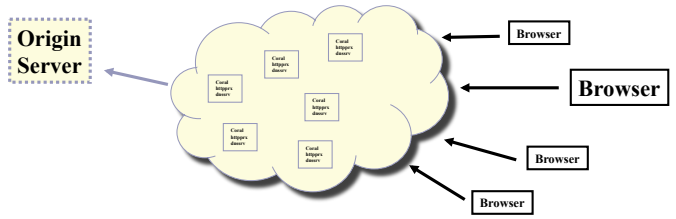
46

Akamai – Subsequent Requests



47

Coral: An Open CDN



Pool resources to dissipate flash crowds

- Implement an open CDN
- Allow anybody to contribute
- Works with unmodified clients
- CDN only fetches once from origin server

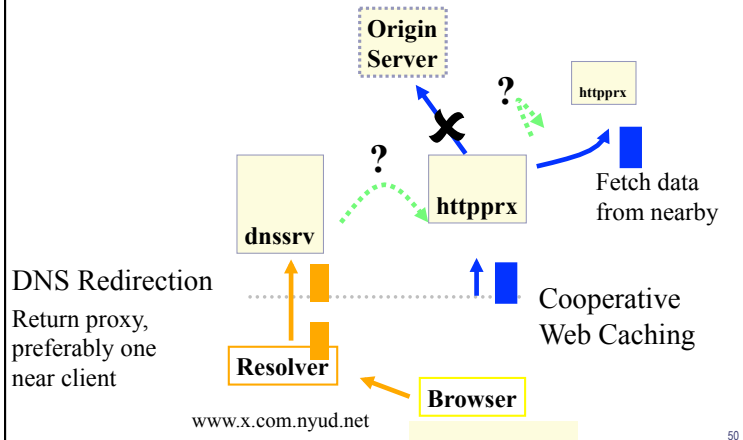
48

Using CoralCDN

- Rewrite URLs into “Coralized” URLs
- www.x.com → www.x.com.nyud.net:8090
  - Directs clients to Coral, which absorbs load
- Who might “Coralize” URLs?
  - Web server operators Coralize URLs
  - Coralized URLs posted to portals, mailing lists
  - Users explicitly Coralize URLs

49

CoralCDN components



50

Functionality needed

- DNS: Given network location of resolver, return a proxy near the client

  put (network info, self)
  get (resolver info) → {proxies}

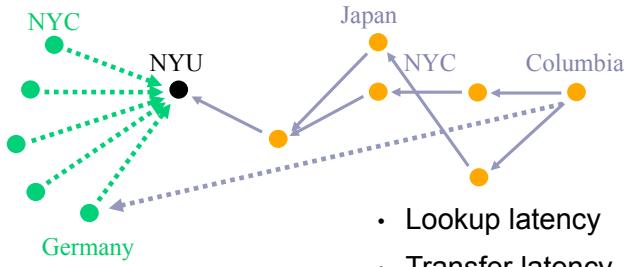
- HTTP: Given URL, find proxy caching object, preferably one nearby

  put (URL, self)
  get (URL) → {proxies}

51

Use a DHT?

- Supports put/get interface using key-based routing
- Problems with using DHTs as given



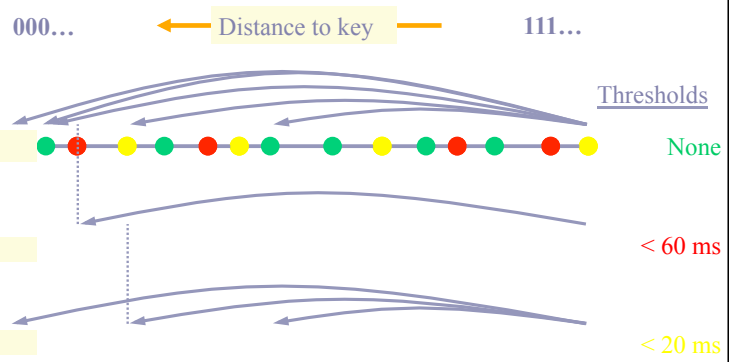
- Lookup latency
- Transfer latency
- Hotspots

52

Coral distributed index

- Insight: Don't need hash table semantics
  - Just need one well-located proxy
- put (key, value, ttl)
  - Avoid hotspots
- get (key)
  - Retrieves some subset of values put under key
  - Prefer values put by nodes near requestor
- Hierarchical clustering groups nearby nodes
  - Expose hierarchy to applications
- Rate-limiting mechanism distributes puts

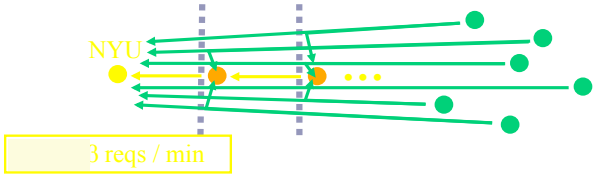
Key-based XOR routing



- Minimizes lookup latency
- Prefer values stored by nodes within faster clusters

Prevent insertion hotspots

- Store value once in each level cluster
  - Always storing at closest node causes hotspot



- Halt put routing at full and loaded node
  - Full → M vals/key with TTL > ½ insertion TTL
  - Loaded → β puts traverse node in past minute
- Store at furthest, non-full node seen

Coral Contributions

- Self-organizing clusters of nodes
  - NYU and Columbia prefer one another to Germany
- Rate-limiting mechanism
  - Everybody caching and fetching same URL does not overload any node in system
- Decentralized DNS Redirection
  - Works with unmodified clients

No centralized management or *a priori* knowledge of proxies' locations or network configurations

56

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 → 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
  - How do directory agents interact with each other?
  - Well suited for browsing and composition → knows full list of services

61

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?

62

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!

63

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!)

64

Hashing

- Advantages
  - Let the CDN nodes are 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.
  - One hop access
  - Any problems?
    - What happens if a node goes down?
    - What happens if a node comes back up?
    - What if different nodes have different views?

65

Robust 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
  - Unacceptable, use consistent hashing – idea behind Akamai!

66

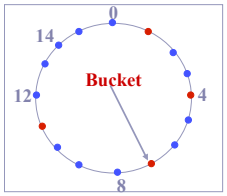
Consistent Hash

- "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

67

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

68