DNS and the Web

15-744
David Andersen

DNS

- Purpose:
  - Map from a human-readable name to a (human-unfriendly) IP address
- Let's look at a bit of history first...

HOSTS.TXT

- In the beginning, there was hosts.txt
- Every host on the Internet downloaded it periodically from SRI-NIC or from a friend
- As the Internet grew,
  - so did the file
  - so did the # of people who had to download the file
  - so did the # of updates to a central service
  - so did the pain.

Centralized service?

- The usual motherhood & apple pie problems:
- Single point of failure
- Traffic volume
- Poor locality
- *Scaling*

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
  - (Note how very important this is! CAP tradeoff...)

DNS Records

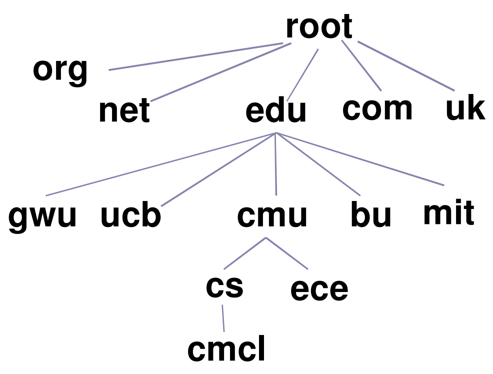
RR format: **(class, name, value, type, ttl)**

- 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**

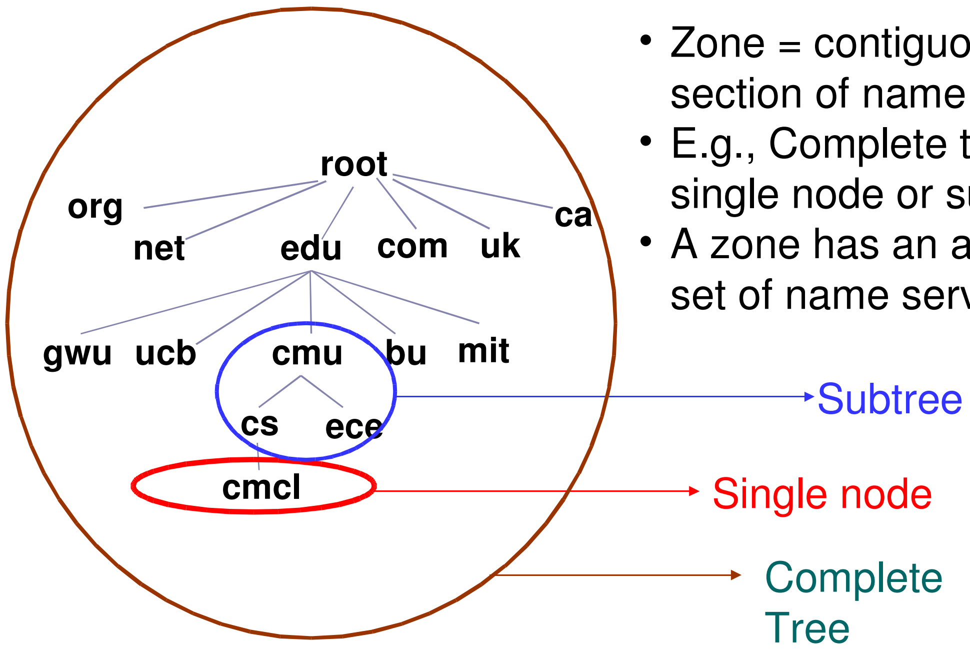
DNS Design: Hierarchy Definitions



- 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.

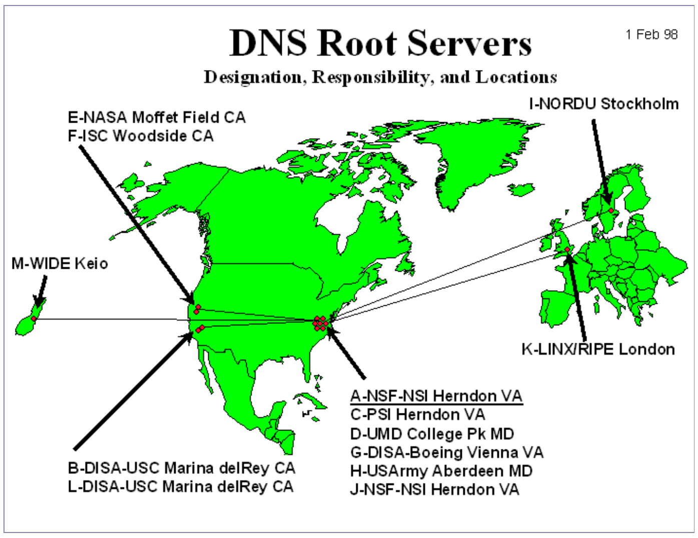
- Zones are created by convincing owner node to create/delegate a subzone
  - Records within zone stored multiple redundant name servers
  - Primary/master name server updated manually
  - Secondary/redundant servers updated by zone transfer of name space
    - Zone transfer is a bulk transfer of the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

Servers/Resolvers

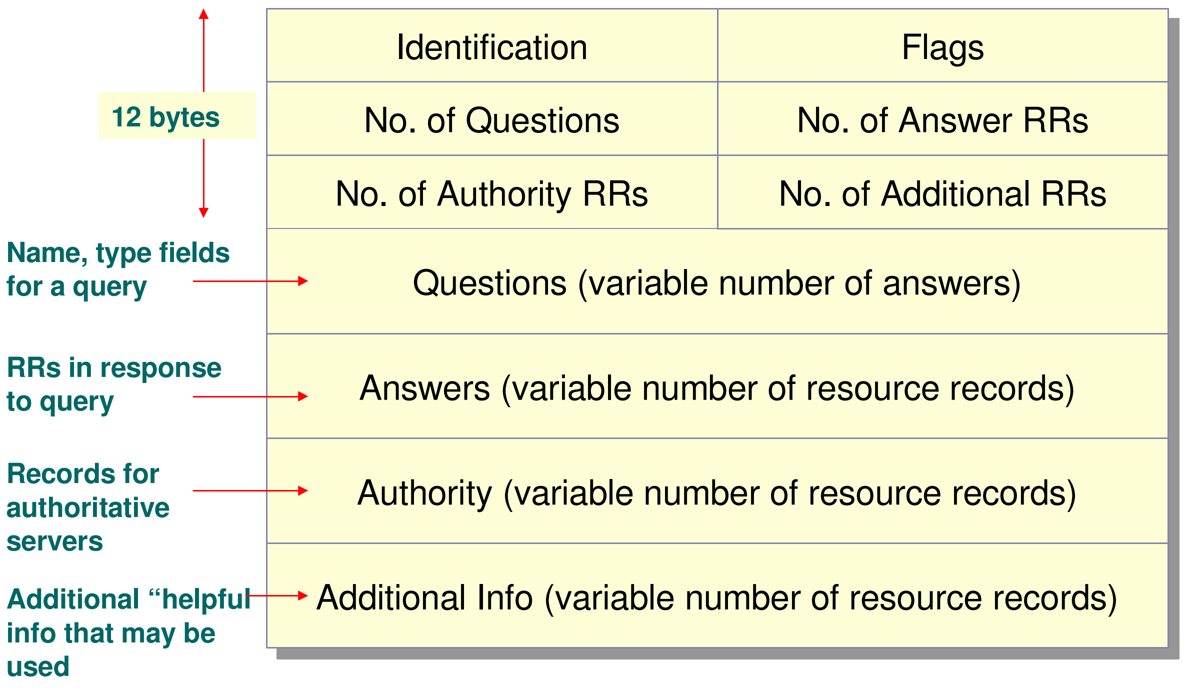
- Each host has a resolver
  - Typically a library that applications can link to
  - Local name servers hand-configured (e.g. /etc/resolv.conf)
- Name servers
  - Either responsible for some zone or…
  - Local servers
    - Do lookup of distant host names for local hosts
    - Typically answer queries 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



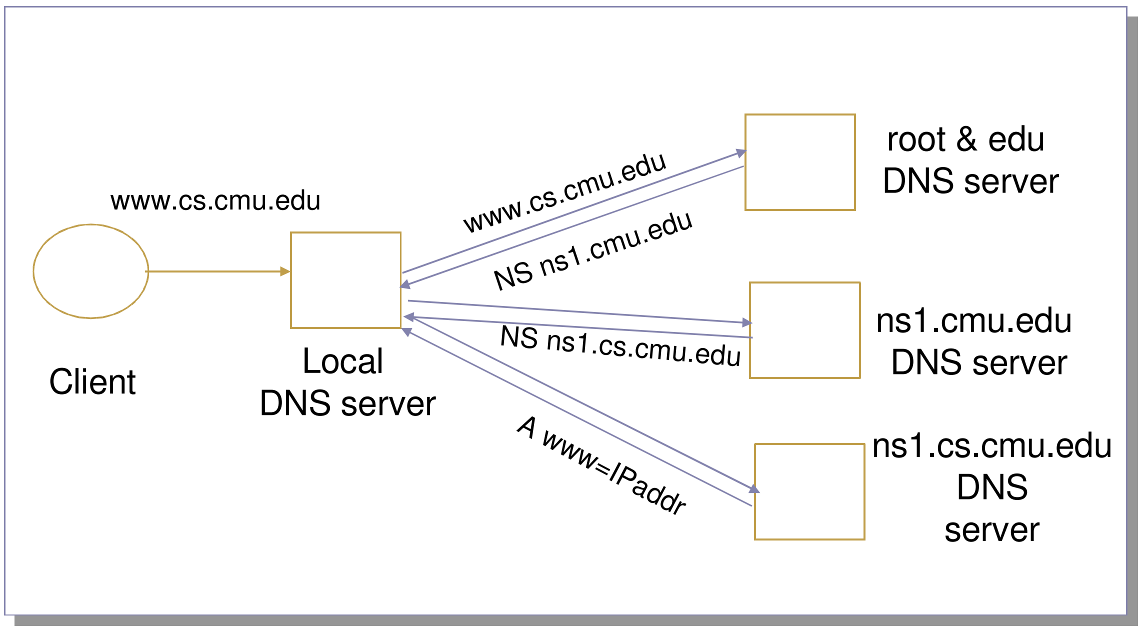
DNS Message Format


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

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:

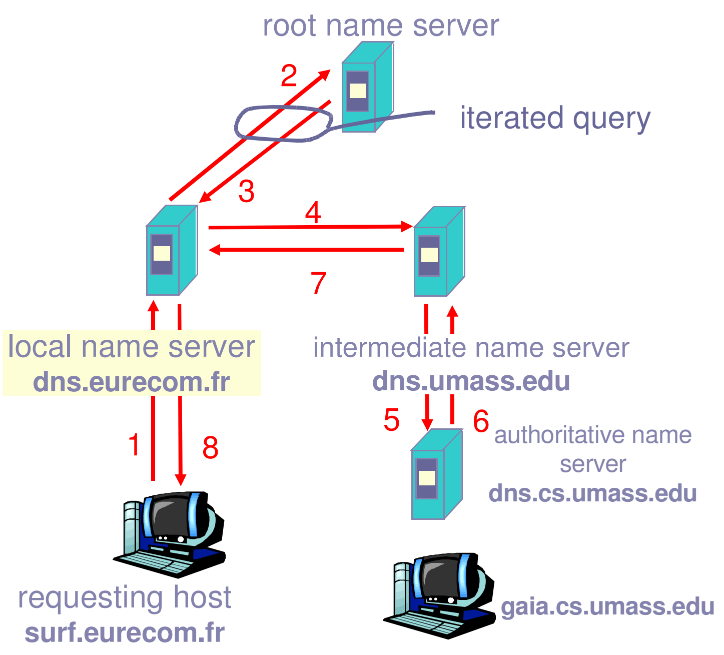
- Server goes out and searches for more info (recursive)
- Only returns final answer or "not found"

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?

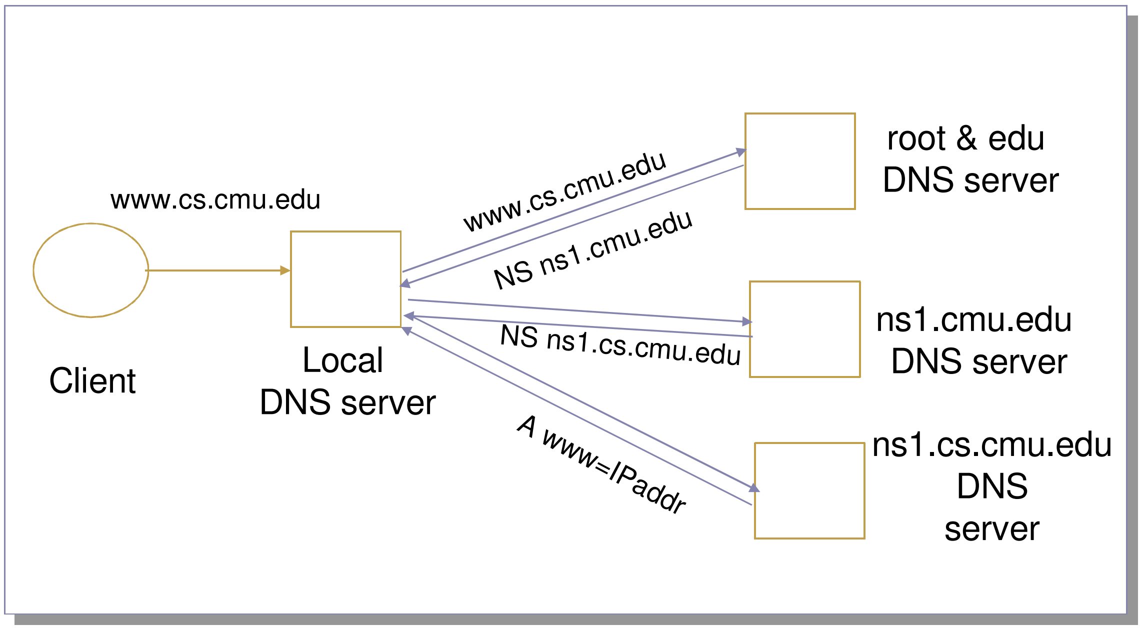
- Local server typically does recursive
- Root/distant server does iterative



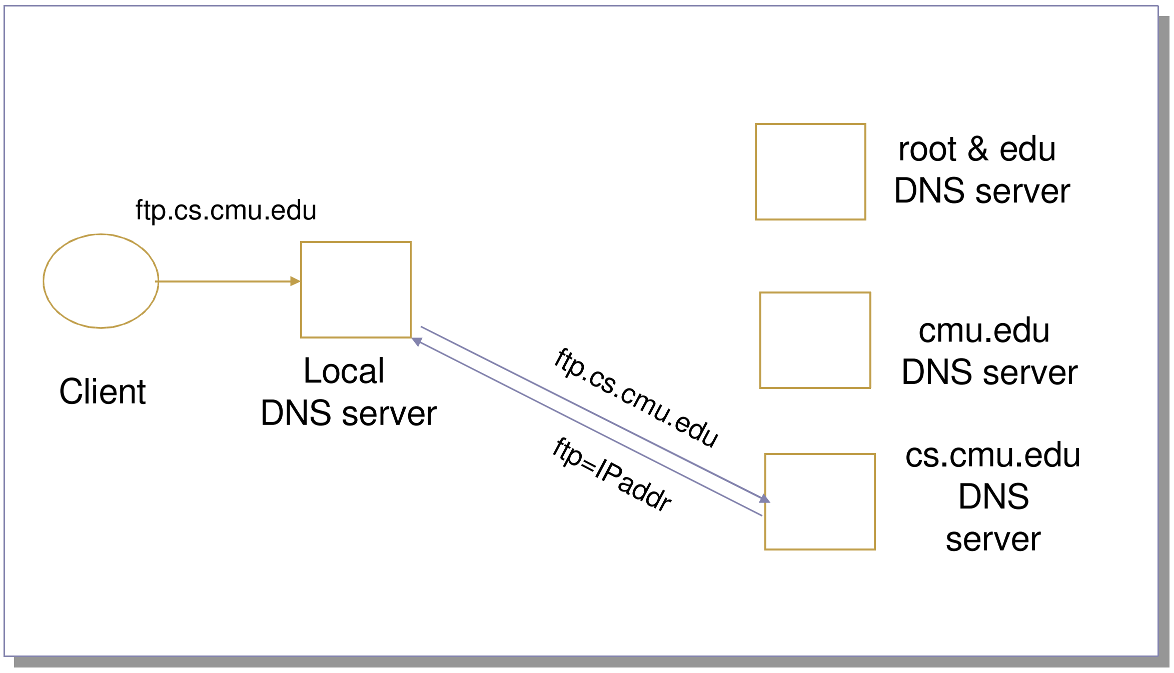
Workload and Caching

- 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



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

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

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”

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, .name

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

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

The Web

- Assuming this is review...
- HTTP
  - Stateless request/response protocol
  - Almost always carried over TCP
  -
  - GET /foo/bar/index.html HTTP/1.0
    header: value
    header2: value
  -

Stateless?

- Yup! No state maintained about client between requests
  - Might keep connection open (persistent connections; performance improvement), but no state.
  - Cookies or other parameters used to communicate state

Caching the Web

- Browsers cache things
- Can interpose a *proxy cache*
  - Send GET http://www.example.com/ HTTP/1.1
- Cache management?
  - Expires header
  - GET-IF-MODIFIED-SINCE <date>
  - Etags
    - Entity tags; identify unique content
      - (Since it might vary with cookies/user ID/etc)

Content Delivery Networks

- Client caches help clients and are optional
- CDNs are typically server-driven – clients have no choice
  - Server directs client to a particular replica
  - Client retrieves content from replica, not original server
- Major benefit of CDNs: Handling load and popularity spikes
  - Sub-benefit: Reduced page load times

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

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

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?

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)

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?

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?

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

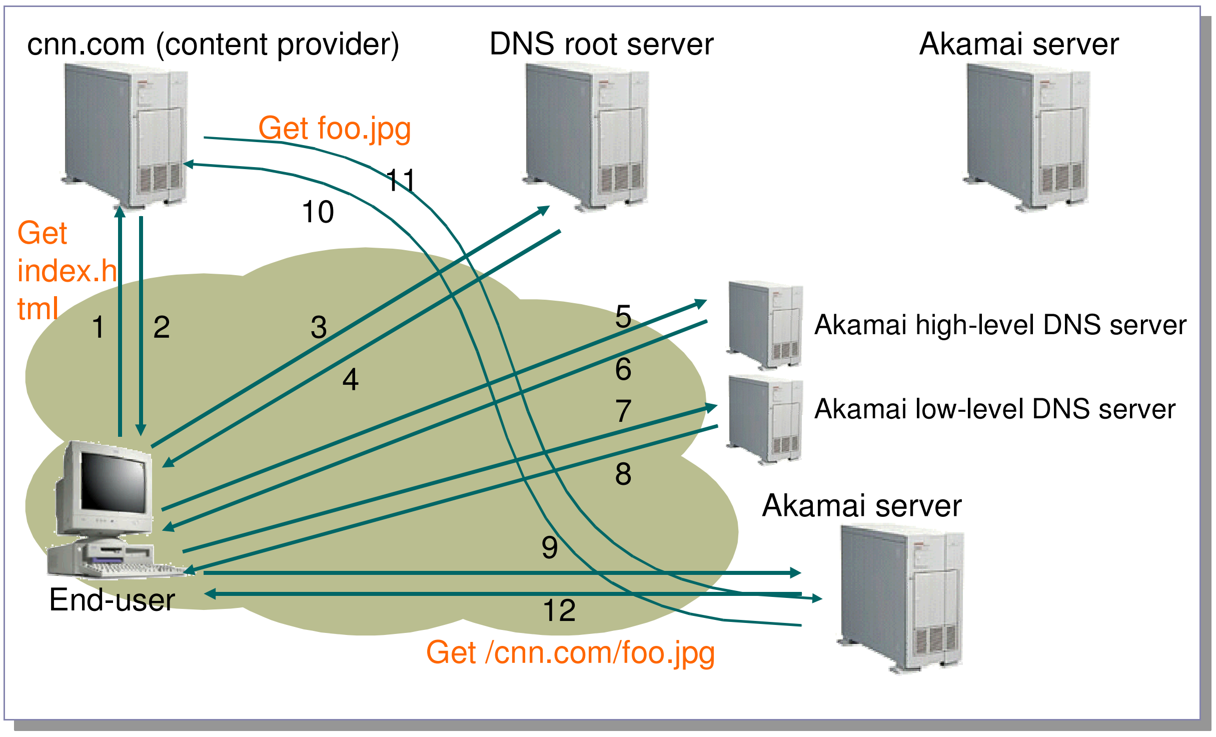
How Akamai Works

- How is content replicated?
- Akamai only replicates static content
  - Serves about 7% of the Internet traffic !
- 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

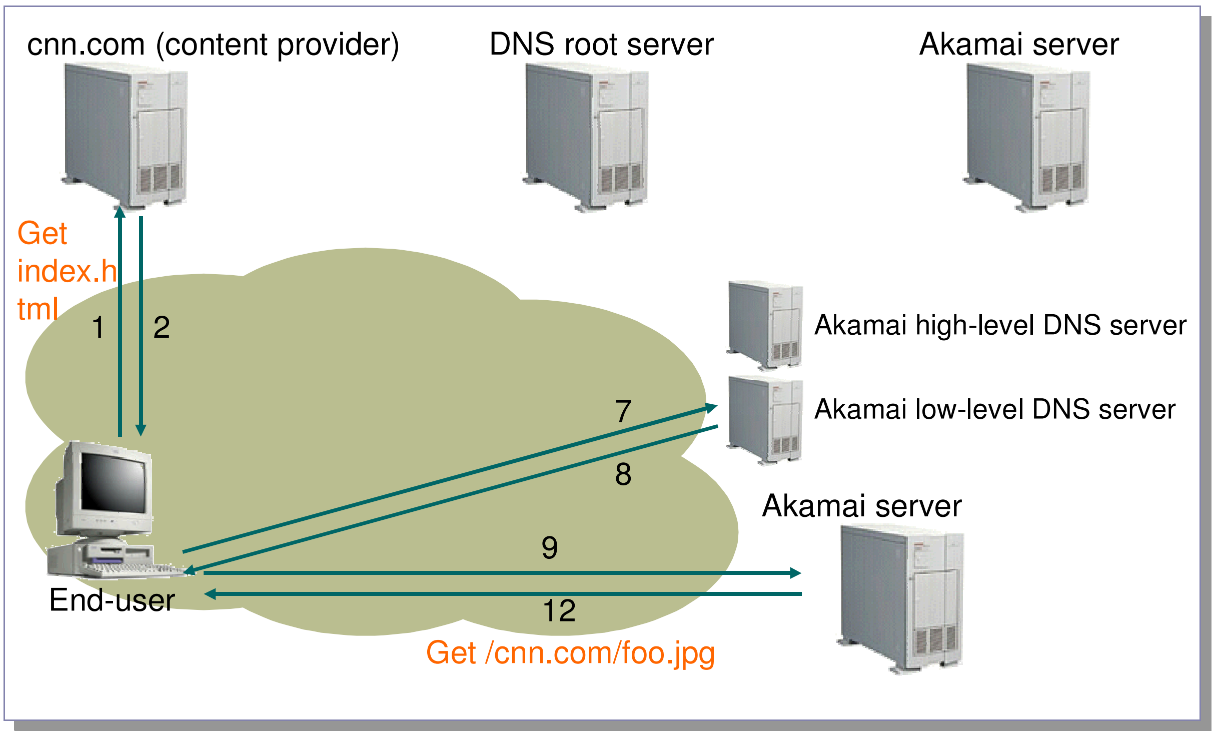
How Akamai Works

- 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

How Akamai Works



Akamai – Subsequent Requests



cnn.com (content provider)
DNS root server
Akamai server
Get index.html
1
2
Akamai high-level DNS server
Akamai low-level DNS server
7
8
Akamai server
9
End-user
12
Get /cnn.com/foo.jpg