The Internet Domain Name System

Hari Balakrishnan

6.829 Fall 2002

Goals

- DNS architecture
  - How DNS works
- DNS uses
  - Mail
  - Content Distribution Networks (CDNs)
- DNS Performance
  - How well does it work?
  - Why does it work?

Why naming?

- Level(s) of indirection between a resource and its location
- Convenience
  - For apps
  - For humans
  - Autonomous organizational operation (real-world)
- Examples
  - DNS, search engines, intentional names,…
  - Virtual memory, DHTs,…

DNS architecture

- Two major components
  - ❑ Name servers: Information repositories
  - ❑ Resolvers: Interface to client programs
    - Stub resolver as libraries
    - Forwarding name servers that proxy for stubs
- DNS name space
- Resource records
- Database distribution
  - ❑ Zones
  - ❑ Caching
- Datagram-based protocol

DNS name space

- Organized as a variable-depth rooted tree
- Each node in tree has associated label
  - Label = variable-length string of octets
  - Case-insensitive
- DNS name of node = path from node to root
  - E.g., nms.lcs.mit.edu. (“.” separates labels)
  - Joe.Schmoe@lcs.mit.edu. (left of “@” is a single label, to the right are four labels)
- No implicit semantics in tree structure in general
  - Except for IN-ADDR.ARPA domain used for reverse lookups
- Design tuned for administrative delegation of the name space (more on this in a bit)

Resource Records (RRs)

- Data in DNS structured using RRs
- Idea is to help both apps and DNS itself
- Classes are orthogonal to each other
  - IN, ISO, CHAOS, XNS,… (pretty much only IN today!)
- Each class has a set of types; new types can be added, but require standardization
- Example IN types
  - A, NS, MX, PTR, CNAME, …

Example

- dig www.google.com

www.google.com.	162	IN	A	216.239.53.100
google.com.	345579	IN	NS	ns3.google.com.
google.com.	345579	IN	NS	ns4.google.com.
google.com.	345579	IN	NS	ns1.google.com.
google.com.	345579	IN	NS	ns2.google.com.

- dig www.google.com –t MX

www.google.com.	86210	IN	MX	20 smtp2.google.com.

- What are the #s in the second column?
- What's the number next to the MX answer?
- Advantage of one RR per type, versus single RR with multiple values?

Database distribution

- Two distribution mechanisms
  - ❑ Zones
  - ❑ Caching
- Separation invisible to user/application
- Zone = complete description of a contiguous section of the DNS name space
  - ❑ Stores RRs for labels
  - ❑ And pointers to all other contiguous zones
  - ❑ Zone divisions can be made anywhere in the name space

Zone logistics

- Persuade parent organization to *delegate* a subzone consisting of a single node
  - E.g., persuade lcs.mit.edu. to delegate nms.lcs.mit.edu (the delegated node is “nms”)
  - Persuade com. to delegate label “cnn” to me
- New zone can grow to arbitrary size and further delegated *autonomously*

Zone owner's responsibilities

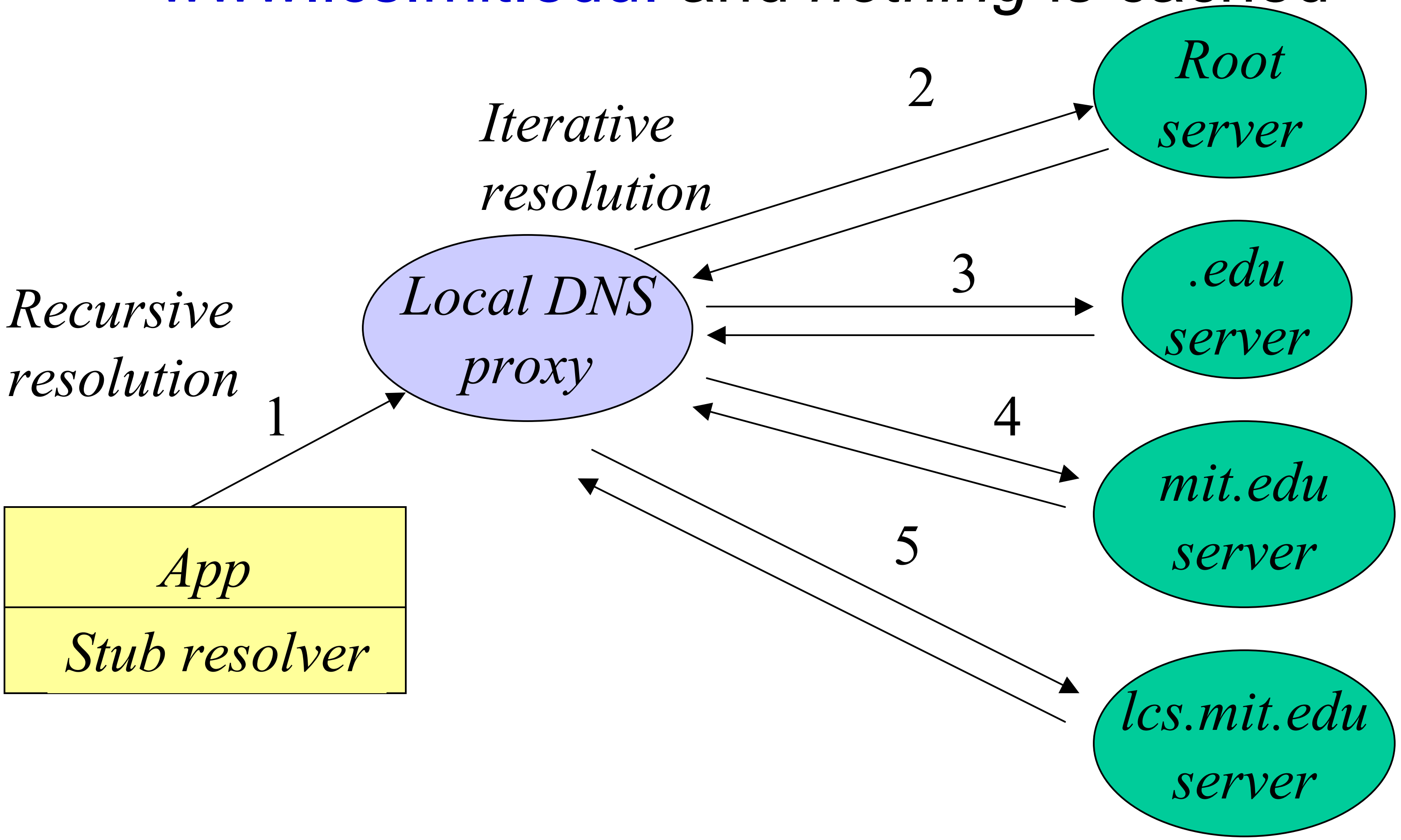
- Authoritatively maintain the zone's data
- Arrange for replicated name servers for the zone
  - Typically, zone data is maintained in a master file and loaded into a primary (master) server
  - Replicated servers use TCP-based zone transfers specified in DNS protocol to refresh their data
- A name server authoritative for a zone *does not* have to be in that zone (great idea)
- A name server can handle any number of zones, which don't have to be contiguous
- Example: dig cnn.com.
  - cnn.com.		600	IN	NS	twdns-02.ns.aol.com

Caching

- Each name server aggressively caches everything it can
- Only control on caching: TTL field
  - ❑ An expired TTL requires a fresh resolution
  - ❑ Each RR has its own TTL
- Low TTL values reduces inconsistencies, allows for dynamic name-to-RR mappings
- Large TTL values reduce network and server load

Example resolution

- Suppose you want to lookup A-record for www.lcs.mit.edu. and *nothing* is cached



Caching

- In reality, one almost never sees the chain of request-response messages of previous slide
- NS records for labels higher up the tree usually have long TTLs
- E.g., the google.com example from before
- But what about cnn.com?

  cnn.com. 600 IN NS twdns-02.ns.aol.com
- Not a problem

  twdns-02.ns.aol.com. 3600 IN A 152.163.239.216

  ns.aol.com. 3553 IN NS dns-02.ns.aol.com.
- Cache not only positive answers, but also stuff that does not exist

Communication protocol

- Normal request response uses a UDP-based datagram protocol with retransmissions
- Retry timer is configurable, typically 4 or 8 seconds
- Often, retries are extremely persistent (many times)
- Use transaction ID field to disambiguate responses
- Key point: App using DNS is typically decoupled from the DNS resolver making recursive queries!
- Zone transfers use TCP (bulk data, rather than RPC-style comm.)

Definitions

- gethostbyname() is a lookup
- Local DNS server makes one or more queries (recursive resolution)
- Each contacted server responds with a response
- A response could be a referral, to go someplace else
- A response that is not a referral is an answer

Performance study motivation

- How well does DNS work today?
  - Scalability
  - Robustness
  - Protocol
- Which of its mechanisms are actually useful?
  - Hierarchy
  - Caching
- DNS is being put to new uses: Is that likely to cause problems?
  - Load-balancing
  - Content Distribution Networks

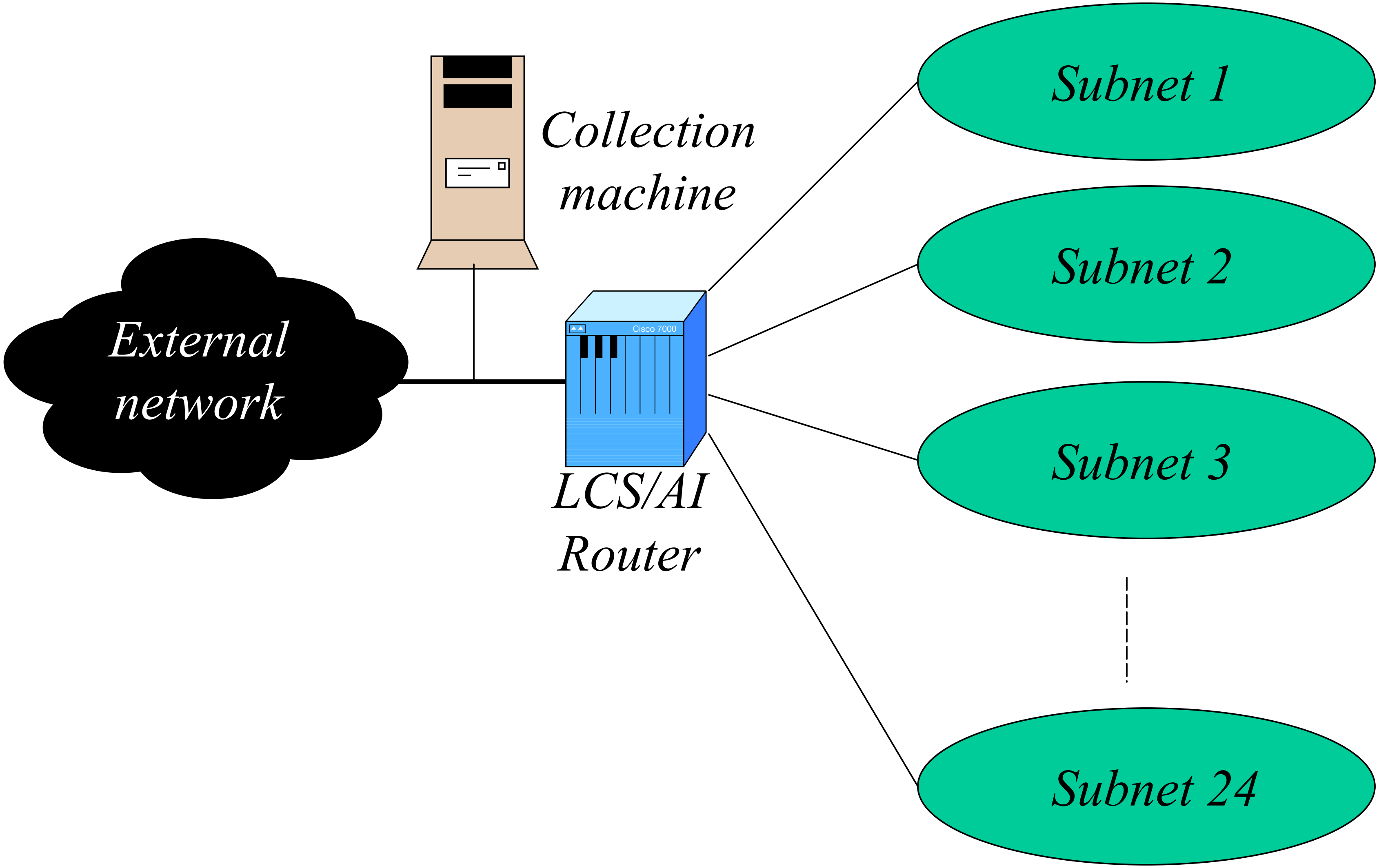
Suspicion

- DNS in WAN traffic traces
  - 14% of all packets (estimate) in Danzig et al. 1990 8% in 1992
  - 5% in NSFNET (1995)
  - 3% in 1997 (MCI traces, 1997)
- But…
  - 18% of all “flows” in 1997
  - 1 out of 5 flows is a DNS flow???
- But yet, the DNS <u>seems</u> to work OK
  - Because of caching is traditional view
- Low-TTL bindings have important benefits
  - Load-balancing
  - Mobility

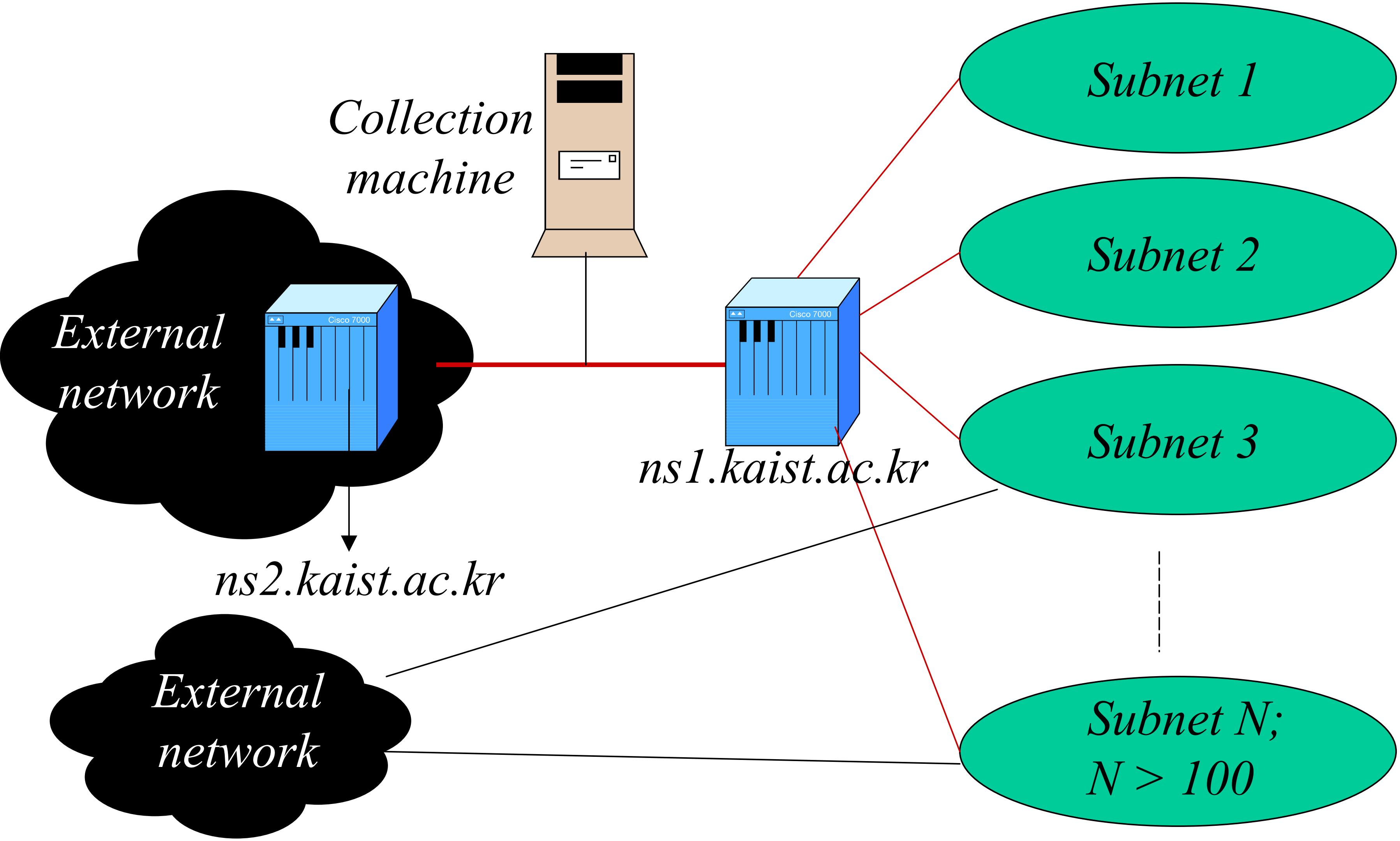
Analysis: Two Data Sets

- MIT: Jan 2000 (mit-jan00) & Dec 2000 (mit-dec00)
  - All DNS traffic at LCS/AI border <u>and</u> all TCP SYN/FIN/RST
  - Protocol analysis & cache simulations
- KAIST, Korea: May 2001 (kaist-may01)
  - All DNS traffic at border and <u>some</u> TCP SYN/FIN/RST
  - Protocol analysis & cache simulations
- Key insight: *Joint analysis* of DNS and its driving workload (TCP connection) can help understand what's going on

MIT LCS/AI Topology



KAIST Topology



Basic Trace Statistics

	mit-jan00	mit-dec00	kaist-may01
Total lookups	2,530,430	4,160,954	4,339,473
Unanswered	**23.5%**	**22.7%**	**20.1%**
Answered with success	**64.3%**	**63.6%**	**36.4%**
Answered with failure	**11.1%**	**13.1%**	**42.2%**
Total query packets	6,039,582	10,617,796	5,326,527
TCP connections	**3,619,173**	**4,623,761**	**6,337,269**
#TCP:#valid "A" answers	7.3	4.9	7.8
Hit rate	86%	80%	87%

Why so many unanswered lookups?
Why so many failures?
Why so many query packets?
Why is hit rate not much higher than 80% and does it matter?

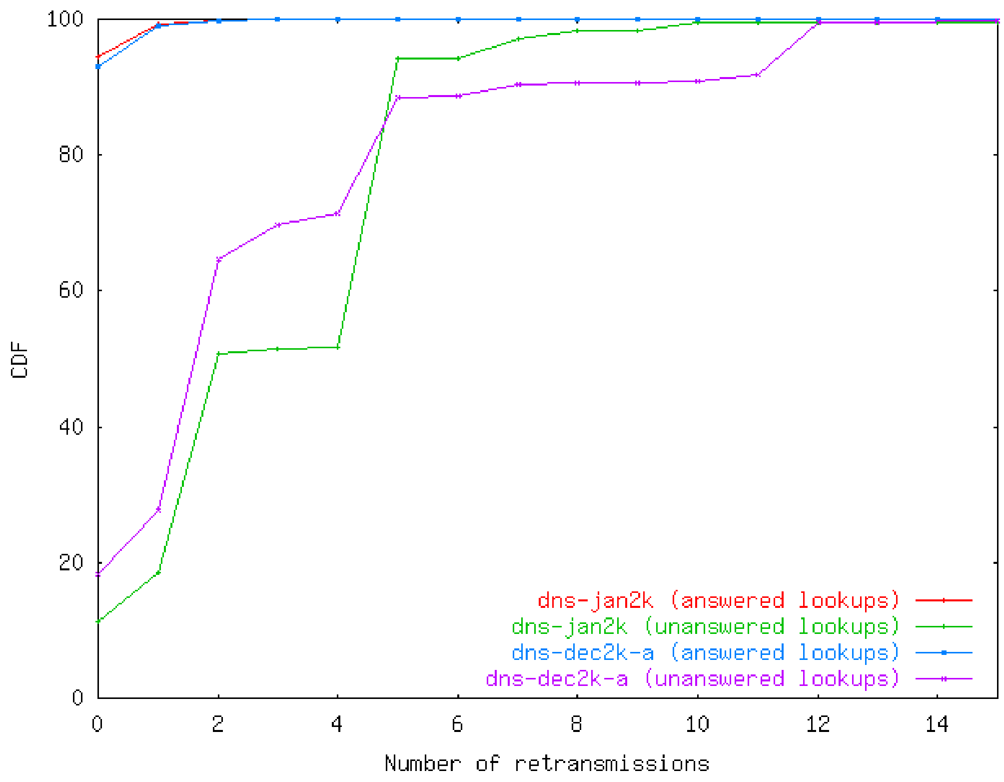
Unanswered lookups

- What's the main reason for this large fraction?

- Three syndromes
  - ❑ Zero referrals (5%-10%)
  - ❑ Non-zero referrals (13%-10%)
  - ❑ Loops (5%-3%)

Reason: Misconfigurations!

Many Lookups Elicit No Response (MIT data)



- About 50% of the wide-area DNS packets are not necessary!

DNS Protocol

- 20-25% of all lookups are unresponded
- Of all answered requests, 99.9% had at most two retransmissions
- Implementations retransmit every 4 or 8 secs
  - And they keep on going and going and going…
  - And becoming worse (more secondaries?)
- But about 20% of the unanswered lookups gave up after ZERO retransmits!
  - More in the KAIST data
- This suggests schizophrenia!
- Solution: tightly bound number of retransmissions

Failure Responses

	mit-jan00	mit-dec00	kaist-may01
Failed lookups	11.1%	13.1%	42.2%

- NXDOMAIN and SERVFAIL are most common reasons
- Most common NXDOMAIN reason: Reverse (PTR) lookups for mappings that don't exist
  - Happens, e.g., because of access control or logging mechanisms in servers
- Other reasons
  - Inappropriate name search paths (`foobar.com.lcs.mit.edu`)
- Invalid queries: `ld`
- Negative caching <u>ought to</u> take care of this

Two Hacks

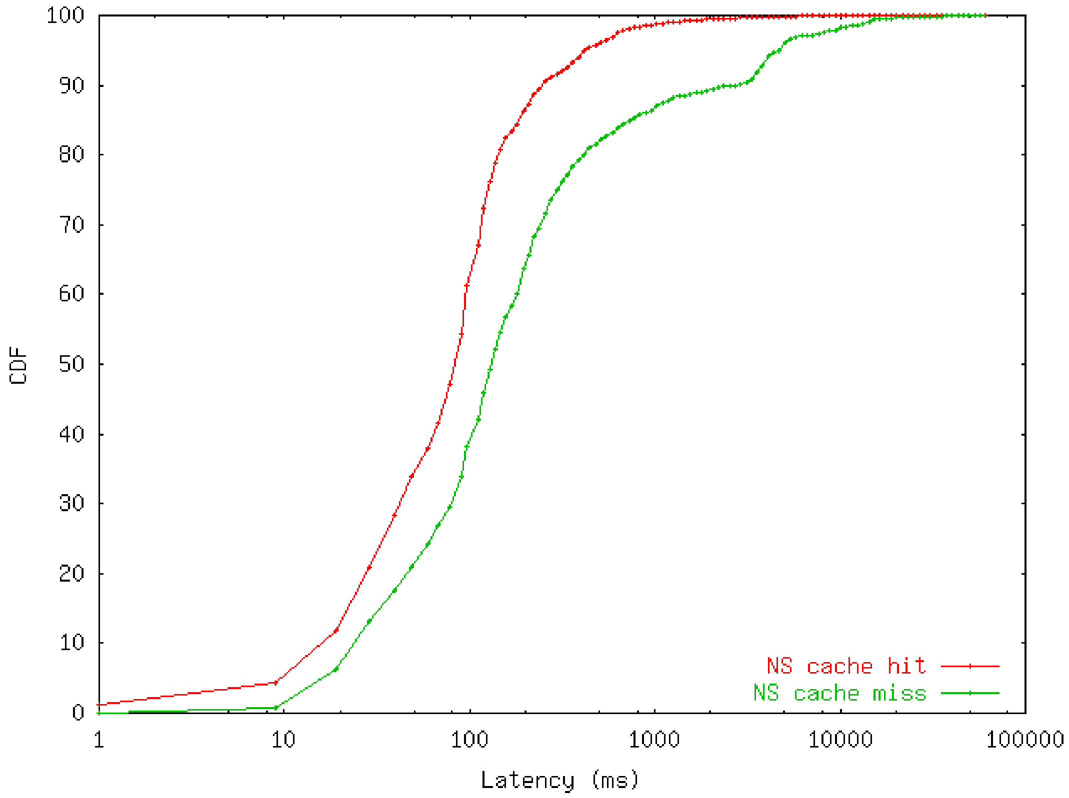
1. Use `dig` option to find BIND version
   - Main result: flood of email from disgruntled administrators
   - Hint: set up reverse DNS with a txt message explaining what you're doing
2. Send back-to-back `a.b.c.com` to name servers
   - First one with recursion-desired bit, second not
   - With –ve caching, second query would respond with `NXDOMAIN` and not a referral

- Result: 90% of name servers appear to implement negative caching
- NXDOMAIN lookups are heavy-tailed too!
  - Many for non-existent TLDs: **`loopback, workgroup, cow`**

DNS Scalability Reasons

- DNS scales because of good NS-record caching, which <u>partitions</u> the database
  - Alleviates load on root/gTLD servers
- Hierarchy is NOT the reasons for DNS scalability
  - The namespace is essentially flat in practice
- A-record caching is, to first-order, a non-contributor to scalability
  - Make 'em all 5 minutes (or less!) and things will be just fine
  - Large-scale sharing doesn't improve hit-rates

NS-record caching is critical



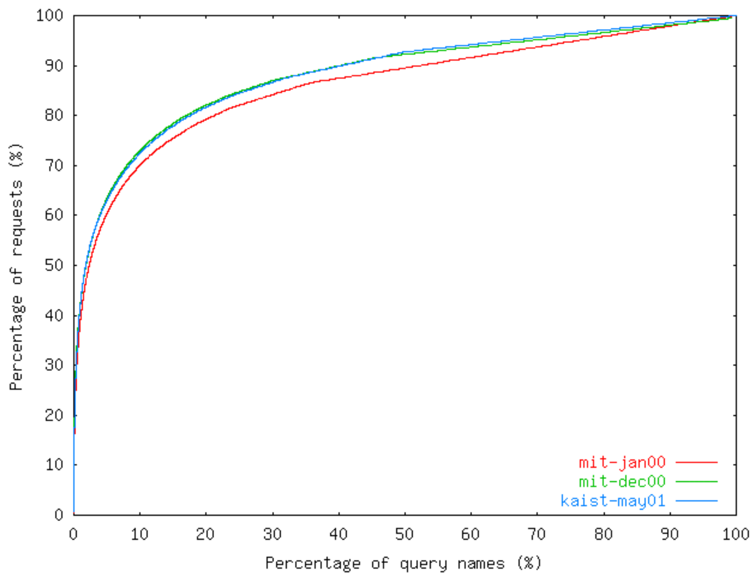
- Substantially reduces DNS lookup latency
- Reduces root load by about 4-5X

Effectiveness of A-record Caching

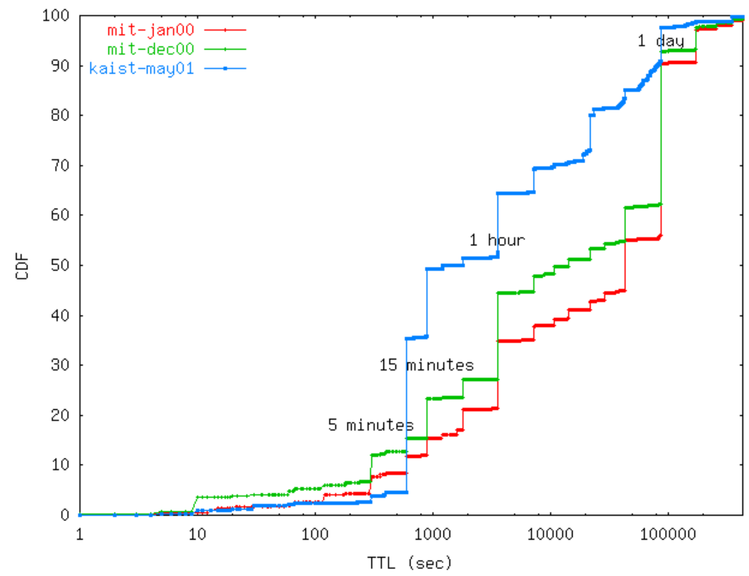
- Cache sharing amongst clients
  - How much aggregation is really needed?
- Impact of TTL on caching effectiveness?
  - Is the move to low TTLs bad for caching?
- What does the cache hit rate depend on?
  - Name popularity distribution
  - Name TTL distribution
  - Inter-arrival distribution
- Methodology
  - Trace-driven simulation

DNS Caching: Locality of References

Name popularity



TTL distribution



- The top 10% account for more than 68% of total answers
- A long tail: 9.0% unique names
  - Root queries regardless of caching scheme

- Shorter TTL names are more frequently accessed
- The fraction of accesses to short TTLs has greatly increased
  - Indicating increased deployment of DNS-based server selection

Trace-driven Simulation

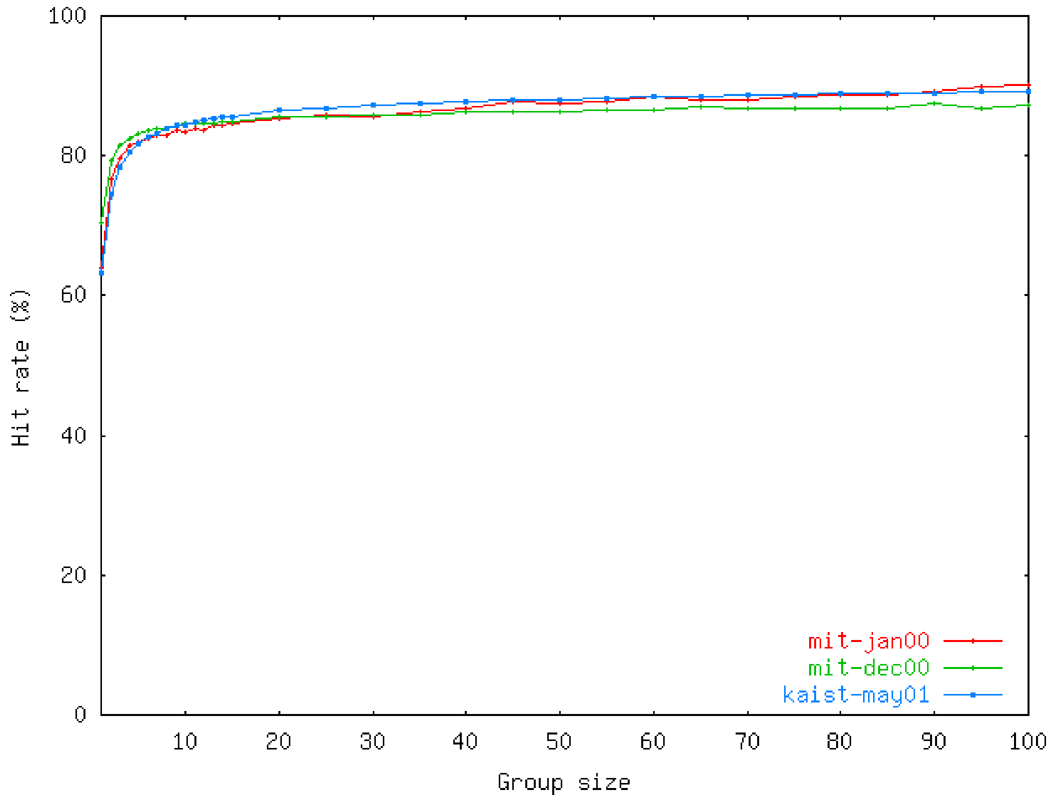
- Key insight: correlate DNS traffic with driving TCP workload
- Parse traces to get:
  - ❑ Outgoing TCP SYNs per client to external addresses
  - ❑ Databases containing
    - IP-to-Name bindings
    - Name-to-TTL bindings per simulated cache

Algorithm

1. Randomly divide the TCP clients into groups of size *S*. Give each group a shared cache.
2. For each new TCP connection in the trace, determine the group *G* and look for a name *N* in the cache of group *G*.
3. If *N* exists and the cached TTL has not expired, record a hit. Otherwise record a miss.
4. On a miss, make an entry in *G*'s cache for *N*, and copy the TTL from the TTL DB to *N*'s cache entry

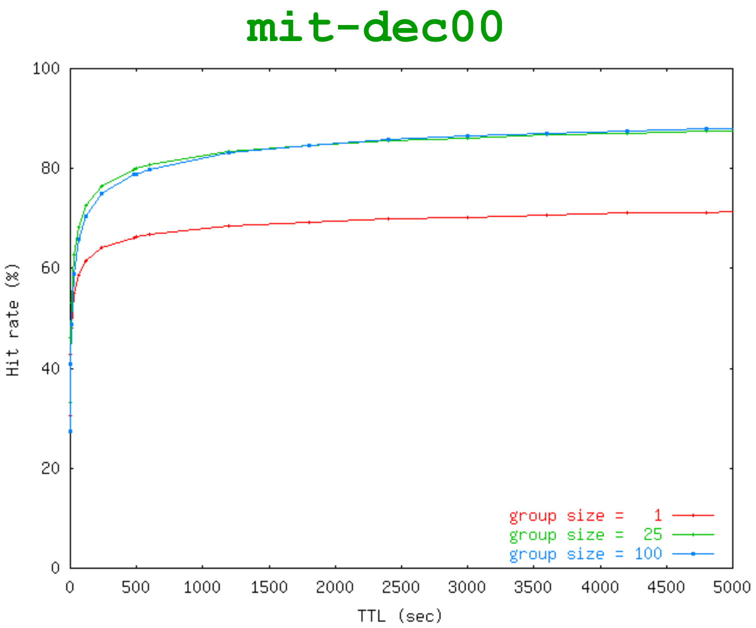
- Same name may have many IPs (handled)
- Same IP may have many names (ignored)

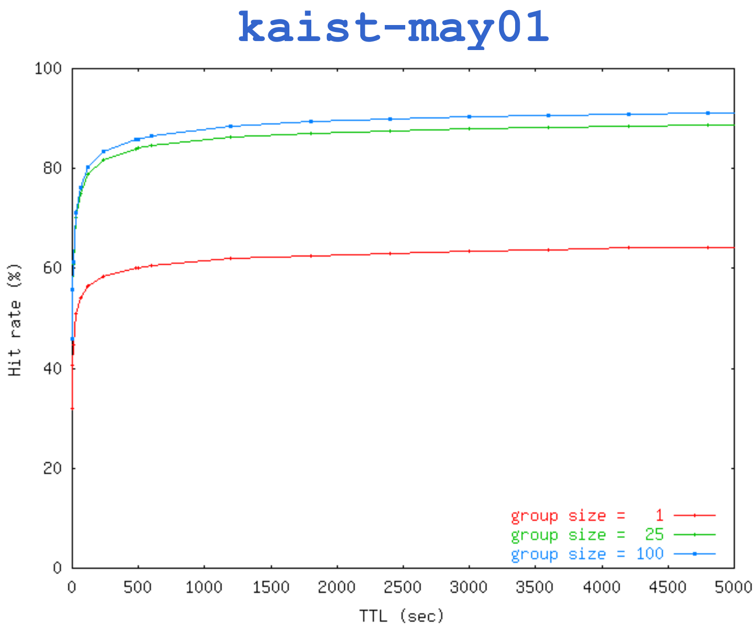
Effect of Sharing on Hit Rate



- 64% (s = 1) vs. 91% ($s \rightarrow 1000$)
- Small s (10 or 20 clients per cache) are enough
  - Small # of very popular names
  - Each remaining name is of interest to only a tiny fraction of clients

Impact of TTL on Hit Rate





- Peg TTL to some value *T* in each simulation run; vary *T*
- TTL of even 200s gives most of the benefit of caching, showing that long-TTL A-record caching is not critical

Bottom line

- The importance of TTL-based caching may have been greatly exaggerated
  - ❑ NS-record caching is critical: reduces root & WAN load
  - ❑ Large TTLs for A-records aren't critical to hit rates
    - 10-min TTLs don't add extra root or WAN load
    - 0 TTL with client caching would only increase load by 2X
- The importance of hierarchy may have been greatly exaggerated
  - ❑ Most of the name space is flat; resolved within 2 referrals
- What matters is <u>partitioning</u> of the distributed database
- The DNS protocol would work better without all that retransmit persistence

Other issues

- How does reverse name lookup work?
  - Trie data structure of numeric IP addresses treated as part of the in-addr.arpa zone
- Dynamic updates?
  - DNS update spec standard now, in BIND 9
- Secure updates?
  - DNS updates need authentication (also std now)
- Attacks on DNS?
  - PS 3 question!