The Upper DNS Hierarchy:

Measurements and Laboratory Simulations

Duane Wessels
The Measurement Factory, Inc.
wessels@measurement-factory.com

Marina Fomenko *and* kc claffy
CAIDA, SDSC, UC San Diego
{**marina, kc**}**@caida.org**

Nevil Brownlee
CAIDA, SDSC, UC San Diego *and*
The University of Auckland
nevil@caida.org

PAM2004, Antibes Juan-les-Pins, April 2004

Overview

- Distributing load among nameservers, caching
- Measurements of live DNS traffic
  - distribution of queries among roots
  - A queries to roots and gTLDs
  - query rates when DNS connectivity is lost
- Laboratory simulations
  - network topology, host configuration
  - caches and network configurations
- Cache tests
  - total queries
  - queries sent to roots
  - root/COM nameserver selection
- Conclusion

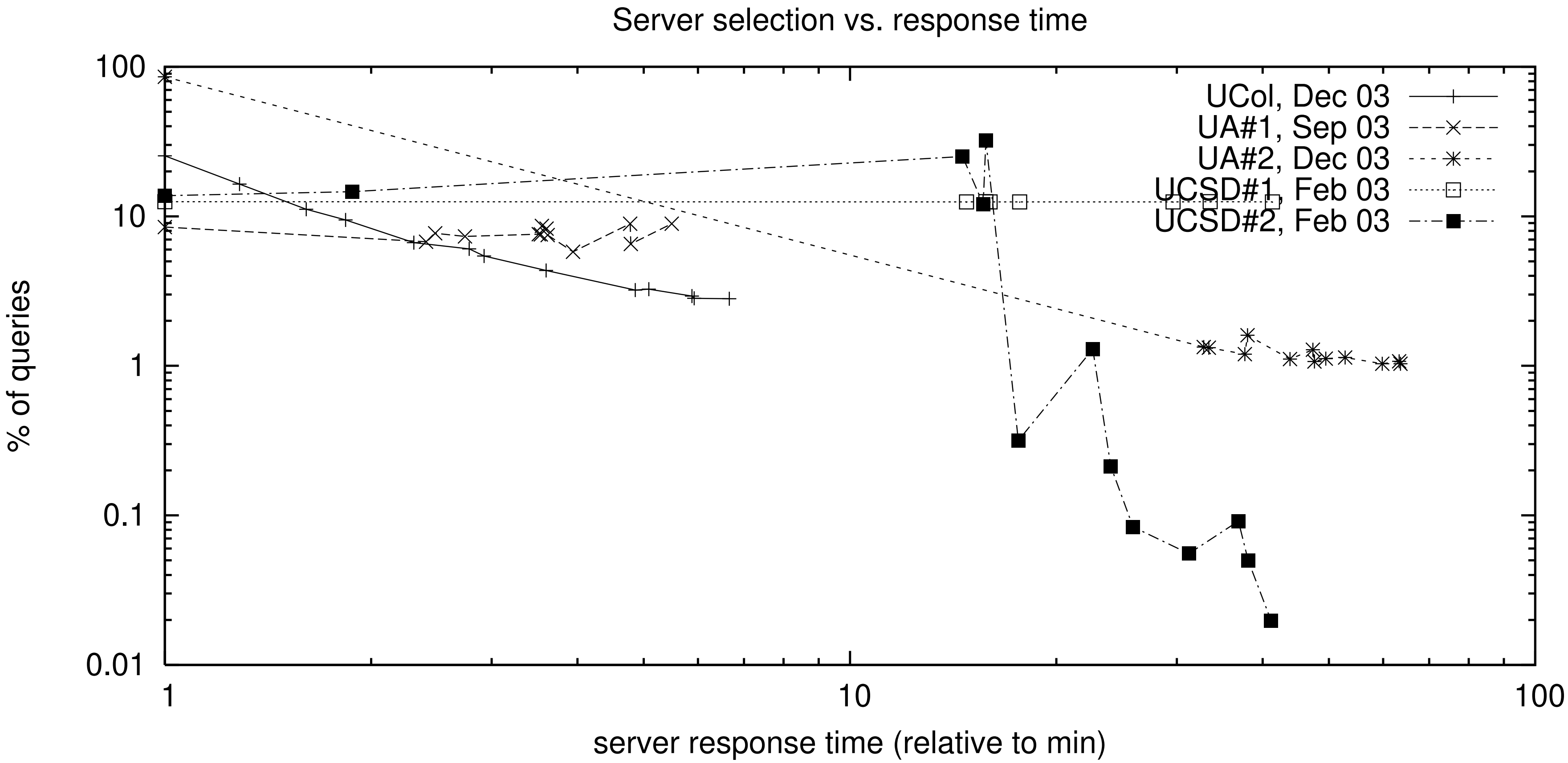
Background

- Top levels of DNS server hierarchy are roots and TLDs
- Multiple servers provide resilience against DoS attacks
  - 13 roots, 13 gTLDs, 2 to ~10 for ccTLDs
  - most roots now have *anycast* copies
- Earlier work (PAM 2003) showed that 98% of F root requests were unneccesary (shouldn't have been made!)
  - root servers are provisioned to handle this load
  - sysadmins should fix misconfigured host systems!
- Caching nameservers reduce the number of requests
  - cache behaviour depends on TTL values
  - different cache behaviour implementations varies

Measurements of live DNS traffic

- NeTraMet meters at UCSD, U Auckland and U Colorado
  - UCSD and Auckland: passive tap on Internet gateway
  - Colorado: SPAN port on link to site's primary caching nameserver
- Expect most requests to come from a few caching nameservers
  - most DNS requests at site come from 3 highest users
  - 3-5 users (nameservers) responsible for 80% traffic
  - 58% of requests are A queries
  - RTTs are heavy-tailed except G, which is bimodal
- Selecting a root ...

Distribution of queries among roots



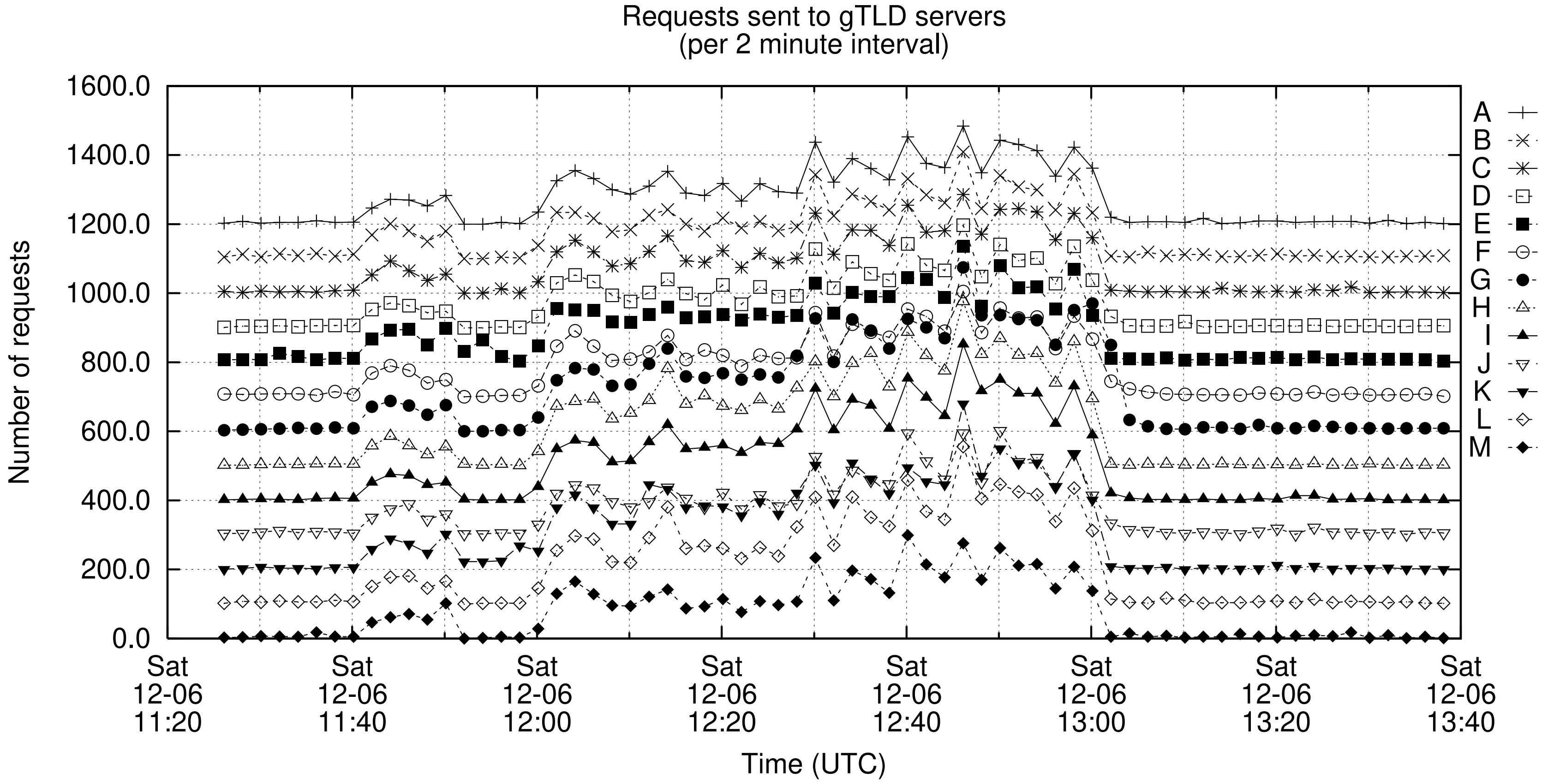
- Nameservers plotted in increasing RTT order
  - UCol and UA#2 show $probablity \approx 1/RTT$
  - UA#1 and UCSD#1 are nearly uniform
  - UCSD#2 is a mixture
- Nameservers: UCol BIND8, UA BIND9, UCSD Microsoft
- *Server choice varies among cache implementations*

A queries to roots and gTLDs

Samples, 6.5 days long	U. Auckland		U. Colorado	
Queries to	roots	gTLDs	roots	gTLDs
positive answers, %	77.7%	94.7%	2.1%	96.3%
av. # of q. per domain	125.4	3.0	6.7	4.4
most frequent	*.net*, 5380	*.iwaynetworks.com*, 2220	*.mil*, 293	*.needsomedealz.com*
negative answers, %	20.6%	1.9%	97.1%	3.4%
most frequent	*.cgs*, 325	*.coltframe.com*, 134	*.dnv*, 13660	*.jclnt.com*, 7694

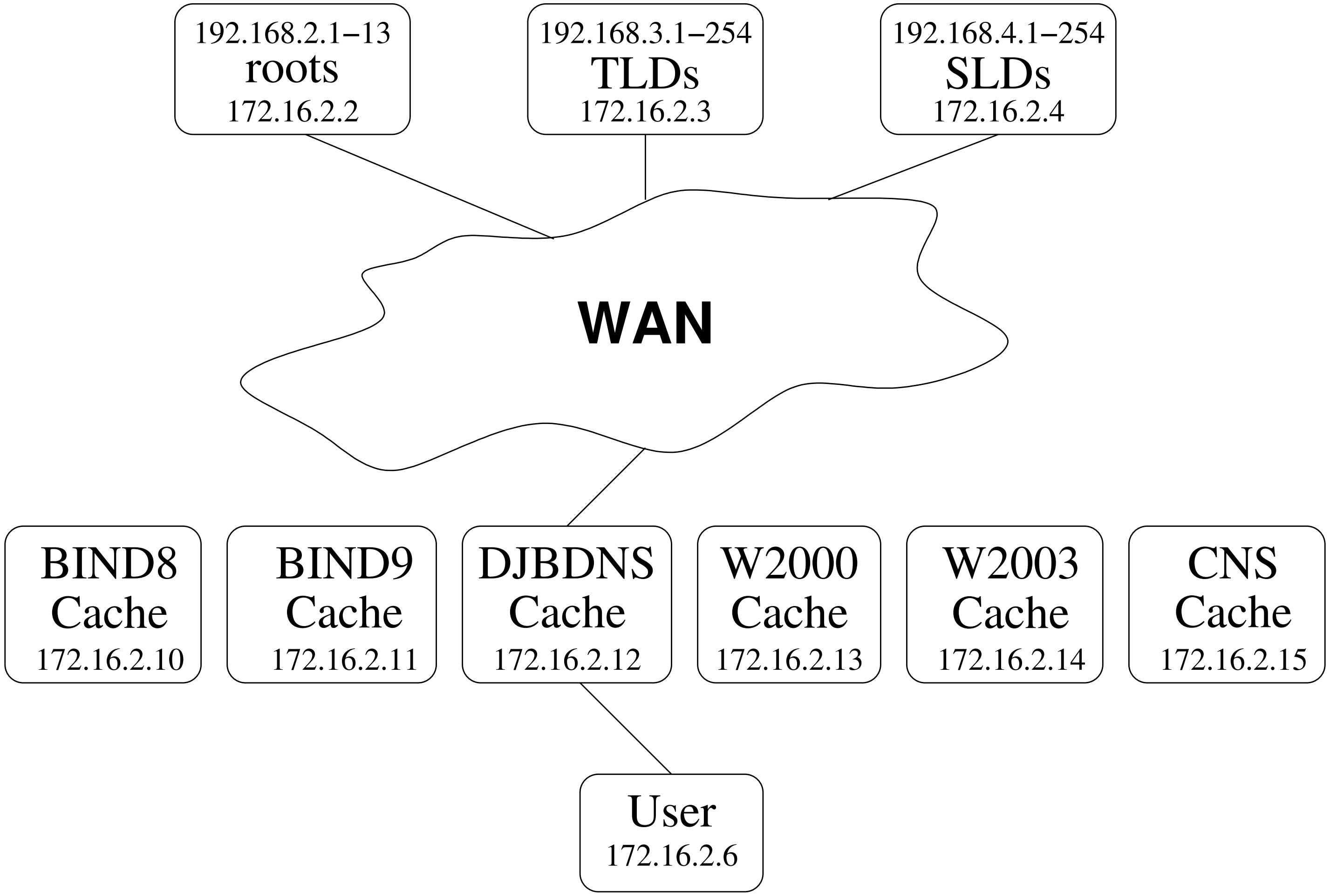
- UCol only got good answers for 2% of queries (PAM 2003 result)
  - Av. queries/domain 6.7 over 6.5 days, i.e. about 1/day
- UA root queries mostly got good answers, but nameserver kept asking (125 queries/root)
- Simulation showed that longer RR TTLs could reduce query rates by a factor of 3

Query rates when DNS connectivity is lost



- gTLD queries from UA#2 data, 6 Dec 03 *(12-06)*
  - plots show queries per 5m interval
  - each subsequent plot is shifted up by 100
- Connectivity lost 1140-1150 and 1200-1300:
  - query rate increases more than 10-fold
  - caches should be less aggressive at such times

Laboratory simulations: network topology



- One PC for each box on the diagram

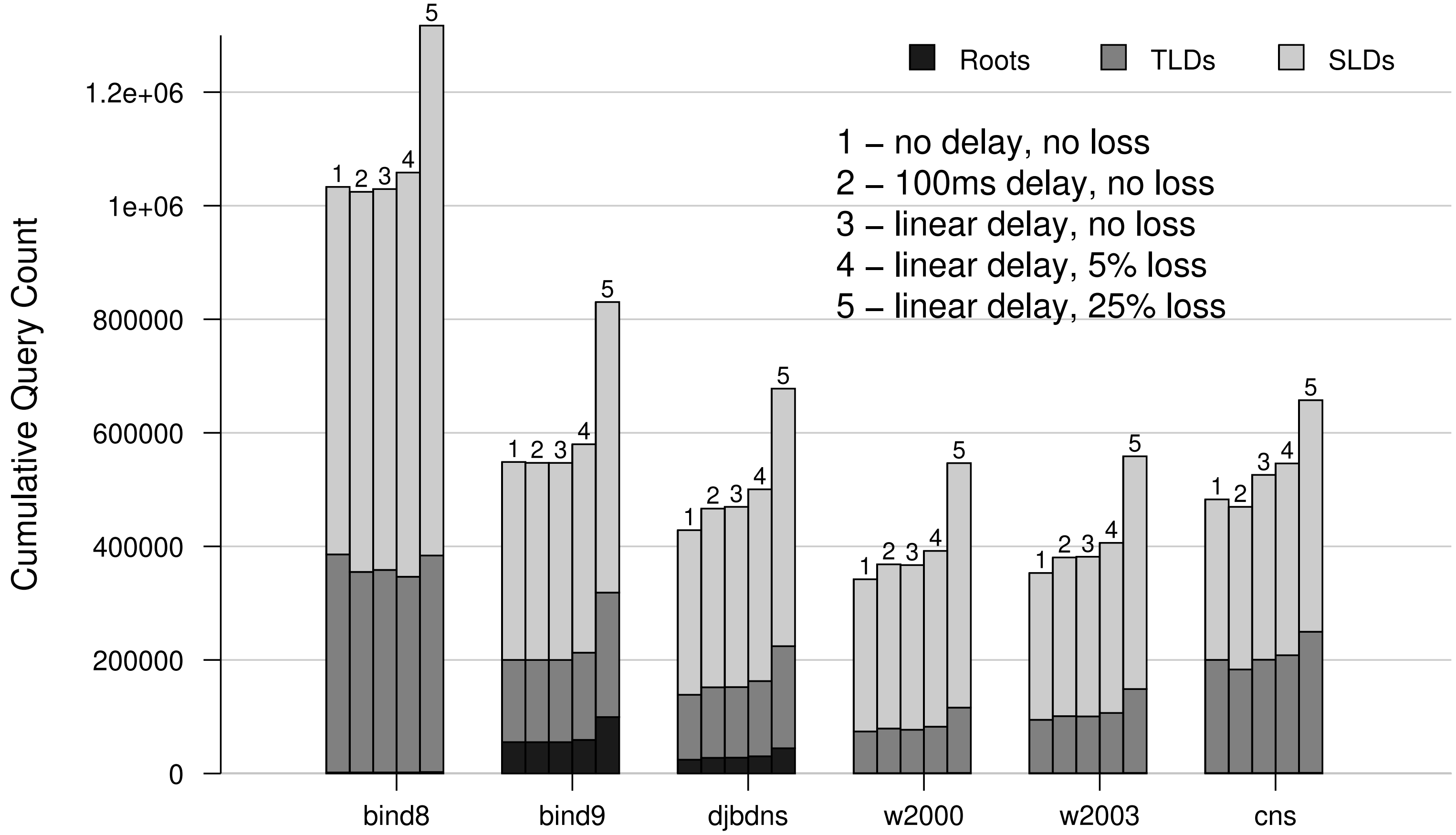
Laboratory simulations: host configuration

- Roots (13), TLDs (254) and SLDs (254)
  - use multiple 192.168/16 addresses bound to the host's loopback interface
  - server host routes packets to them from server's external interface
  - dummynet on host introduces losses and delays
- Trace data from IRCache proxy data
  - timestamps used to set replay rate
  - each test run takes about 12 hours
- Zone files generated from trace file
  - no bogus TLDs (lets us test for negative caching)
- Root and TLD zones are as realistic as possible
  - use random sample of 5000 domains for SLDs, 2 NS records each
  - 35% of names in trace file (and in our zone data) are for CNAME records

Laboratory simulations: caches and network conditions

- Caches: BIND8, BIND9, DBJDNS, W2000, W2003, CNS
- Network configurations:
  1. No delay or loss
  2. 100 ms delay, no loss
  3. linear delay, no loss
  4. linear delay, 5% loss
  5. linear delay, 25% loss
  6. 100% loss
- *Linear* delays proportional to order in nameserver list
  $$\tau = 30 + 10n$$
- *Linear + 5% loss* seems the most realistic set of conditions

Cache tests: total queries



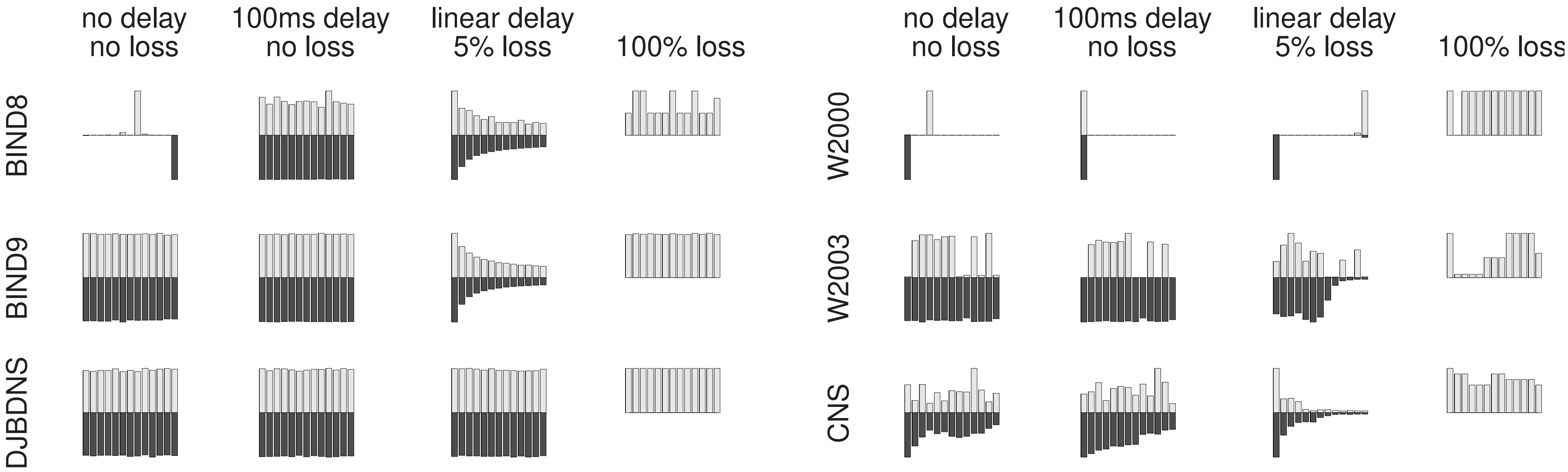
- BIND8 sends A, A6 and AAAA queries to *each* nameserver
- BIND9 always starts at root, sends A and A6 query to *each*
- DJBDNS starts at root, sends A query to *one* nameserver
- Windows sends fewest queries overall
- BIND8 and CNS send fewer queries for 100 ms delay than for 0 ms

Cache tests: queries sent to roots

Delays Pkt Loss	none none	100ms none	linear none	linear 5%	linear 25%	 100%
BIND8	1,826	1,876	1,874	1,899	2,598	37,623,266
BIND9	55,329	55,260	55,256	59,222	99,422	2,564,646
DJBDNS	24,328	27,646	27,985	30,341	44,503	12,155,335
W2000	622	657	663	727	1,020	66,272,276
W2003	693	669	666	709	1,009	39,916,750
CNS	824	831	924	975	1,179	12,456,959

- *Except for BIND9,* 100% loss behaviour is appalling
- W2000 magnifies query stream by an order of magnitude!
  - The trace contains a total of 5.5 million queries
- BIND9 avoids repeat queries for pending answers

Cache tests: root/COM nameserver selection



- BIND selects servers in inverse proportion to their RTTs
- DJBDNS doesn't use RTT or loss rate to select the server (this is a design feature)
- Windows uses poor server selection algorithms
- CNS is also odd, but works well for *linear + 5% loss*

Conclusion

- Different caches use different query distribution algorithms; *Some of them impose much greater query loads on the roots than others*
- BIND8/BIND9 look for IPv6 addresses, most of which don't exist, giving rise to many unanswered queries
- DNS admins: remember that your TTLs affect the DNS system overall. *Use longer TTLs to reduce the global query load*
- Lab tests and live measurements show caches become aggressive when they don't get responses. Nameservers should implement exponential backoff when there's no response, and should also avoid forwarding duplicate queries
- Acknowledgment:
  support from WIDE project and DARPA grant