Storage Management and Caching in PAST, a Large-scale, Persistent Peer-to-peer Storage Utility

Presented by Haiming Jin

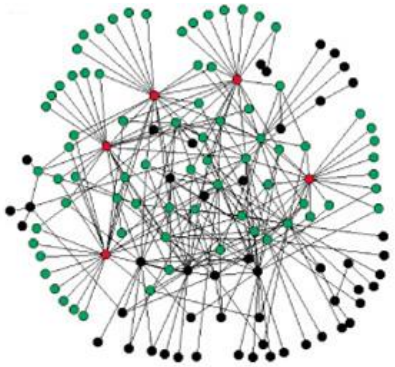
2013-03-07

Background

- P2P applications emerges as mainstream applications
  - 53.3% of upstream internet traffic (2010)
  - Scalability, robustness to failures, information availability, etc.
  - P2P file sharing, VoP2P, P2PTV, etc.

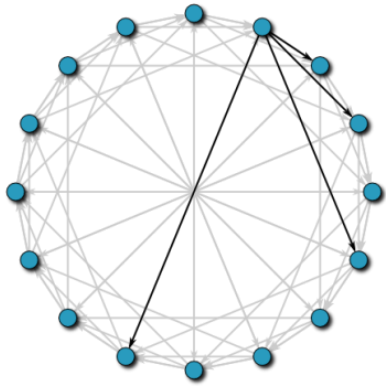
BitTorrent

Overlay Structures



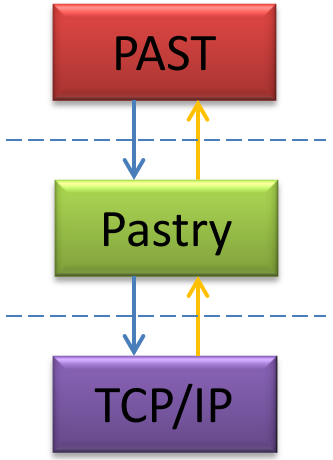
- Unstructured overlays
  - Napster, Gnutella, FastTrack, Freenet, etc.
  - Random graph, power-law graph, etc.
  - Random walk, flooding, etc.

- Structured overlays
  - Chord, Pastry, Tapestry, P-Grid, etc.
  - Ring overlay, etc.
  - Distributed Hash Table (DHT)



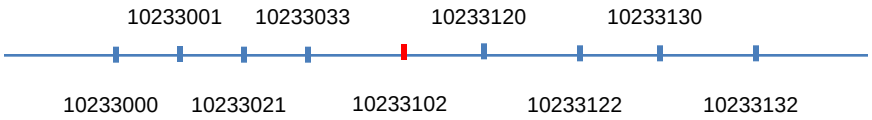
PAST Overview

- Internet-based, peer-to-peer global storage utility (archival storage system)
  - Persistence, availability, scalability, security and load balancing
  - Semantically different from a conventional file system
    - Insert, Lookup and Reclaim
    - No searching, directory lookup or key distribution
    - Immutable (read-only) files
  - Built on top of Pastry
    - Logarithmic complexity for routing message exchange
    - Locality
  - Whole file replication (block-based file-replication?)



Pastry-Routing

- Leaf set
  - l numerically closest nodes



- Routing table
  - $\lceil \log_{2^b} N \rceil \times (2^b - 1)$ entries
  - Prefix matching and proximity metric based
- Neighborhood set
  - l closest nodes with respect to proximity metric
  - Scalar metric, e.g. number of IP hops, geographical distance, etc.

Level 2

NodeId 10233102

Leaf set	SMALLER	LARGER	
10233033	10233021	10233120	10233122
10233001	10233000	10233230	10233232

Routing table			
-0-2212102	1	-2-2301203	-3-1203203
0	1-1-301233	1-2-230203	1-3-021022
10-0-31203	10-1-32102	2	10-3-23302
102-0-0230	102-1-1302	102-2-2302	3
1023-0-322	1023-1-000	1023-2-121	3
10233-0-01	1	10233-2-32	
0		102331-2-0	
		2	

Neighborhood set			
13021022	10200230	11301233	31301233
02212102	22301203	31203203	33213321

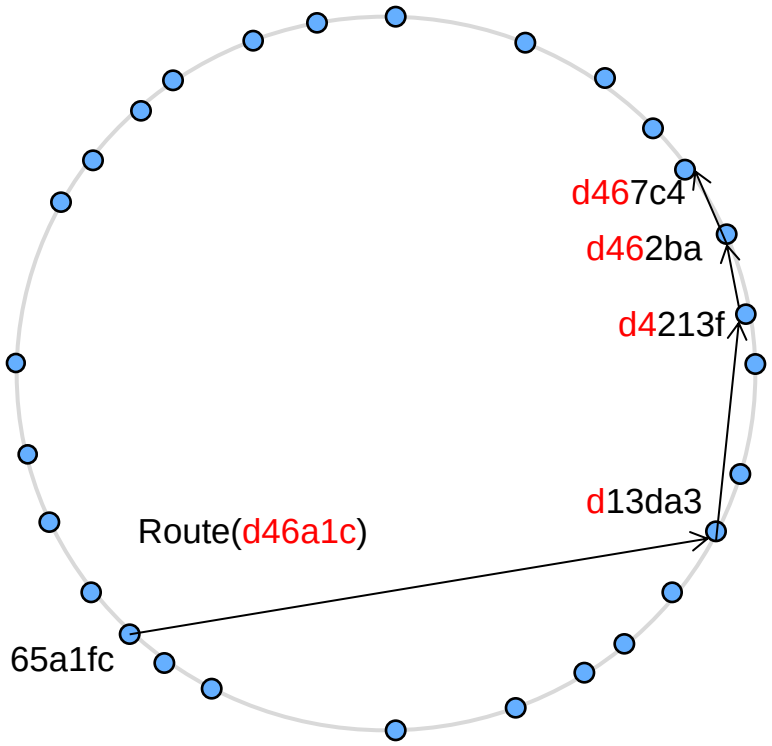
State of Pastry Node with NodeId 10233102, b=2 and l=8

Pastry-Routing

- Routing algorithm

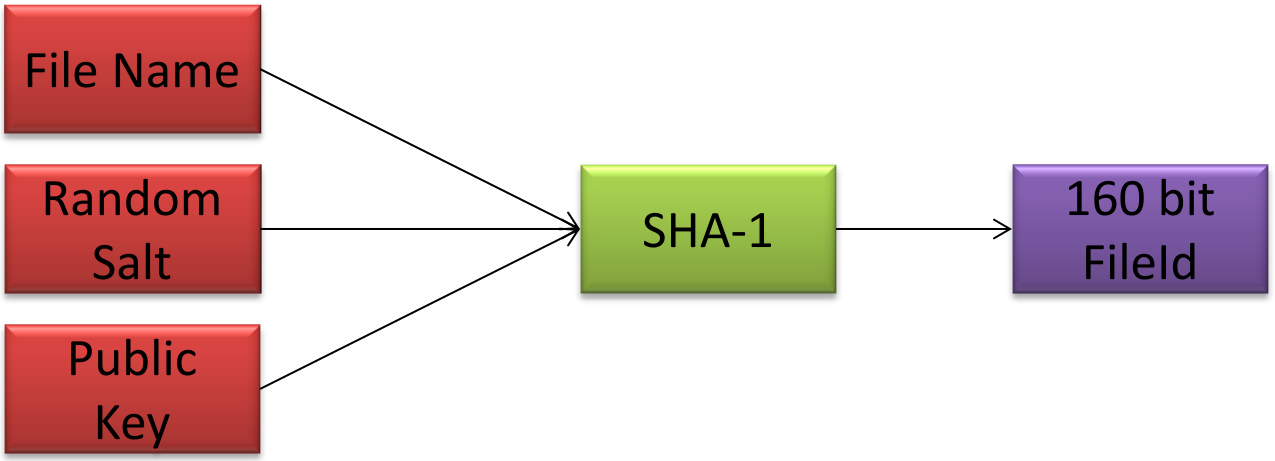
```
(1)  if (L_{-⌊|L|/2⌋} ≤ D ≤ L_{⌊|L|/2⌋}) {
(2)      // D is within range of our leaf set
(3)      forward to L_i, s.th. |D − L_i| is minimal;
(4)  } else {
(5)      // use the routing table
(6)      Let l = shl(D, A);
(7)      if (R_l^{D_l} ≠ null) {
(8)          forward to R_l^{D_l};
(9)      }
(10)     else {
(11)         // rare case
(12)         forward to T ∈ L ∪ R ∪ M, s.th.
(13)             shl(T, D) ≥ l,
(14)             |T − D| < |A − D|
(15)     }
(16) }
```

- Example



PAST-Operations

- File insertion
  - fileId=Insert(name, owner-credentials, k, file)
  - Route file and certificate via Pastry with destination fileId
    - Certificate=fileId+SHA-1(file content)+k+salt+date+metadata
  - *Ack* with *store receipts* routed back when all *k* nodes receive the file



PAST-Operations

- File lookup
  - file=Lookup(fileId)
  - Route request message using fileId as destination
  - Likely to retrieve content within proximity of the client

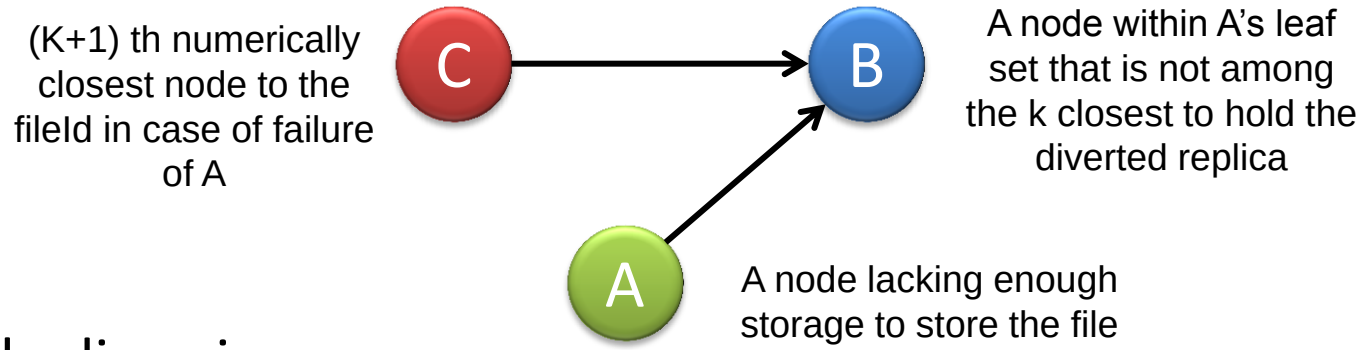
- File reclamation
  - Reclaim(fileId, owner-credentials)
  - No longer guarantee successful lookup for file with fileId
  - Similar to file insertion
    - Reclaim certificate and reclaim receipt routing

PAST-Storage Management

- Responsibilities of storage management
  - Load balancing among PAST nodes
    - Statistical variation in NodeId assignment, file size distribution, heterogeneous node storage capacity
  - Maintain that copies of each file are maintained by k nodes with nodeIds closest to the fileId
- Ways of storage management
  - Replica diversion
    - Load balancing within leaf set
  - File diversion
    - Load balancing among different storage portions

PAST-Storage Management

- Replica diversion
  - Load balancing within leaf set
  - Replica diversion policy
    - A node *N* rejects file *D* if $\frac{S_D}{F_N} > t \left(t_{pri} > t_{div}\right)$



- File diversion
  - Load balancing among different portions of PAST storage
  - On failure of file insertion, a different salt is chosen to divert the file to another storage space

PAST-Caching

- Cache insertion policy
  - Cache copies are inserted to a node along the routing of lookup or insert
  - $File\ Size < c \times Node\ Current\ Cache\ Size$
- Cache replacement policy
  - GreedyDual-Size Policy
  - Maintain weight for each file, $H_d = \frac{c(d)}{s(d)}$
    - Pick the file with minimum weight, H_v to be evicted
    - Subtract , $H_v$from the H values of all cached files
    - Cache hit rate is maximized if $c(d)$ is set to 1

Experimental Results

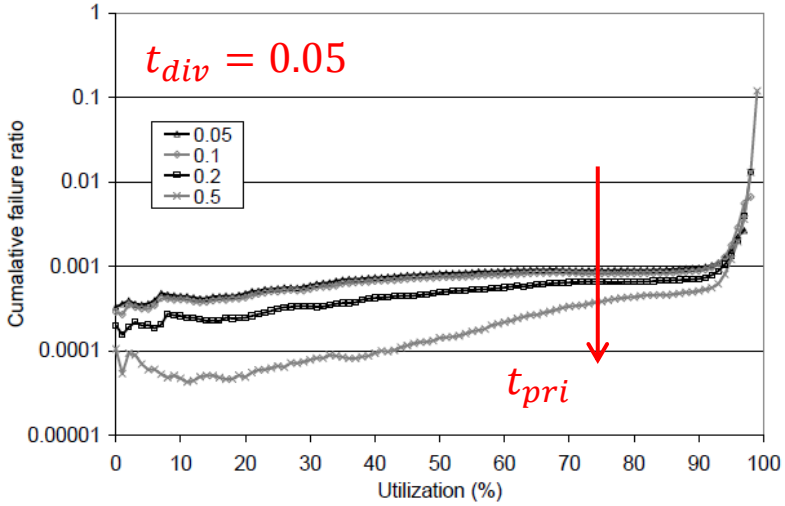
- 2250 nodes
- Necessity of storage management
  - Fail ratio=51.1%, Storage utilization=60.8% without storage management

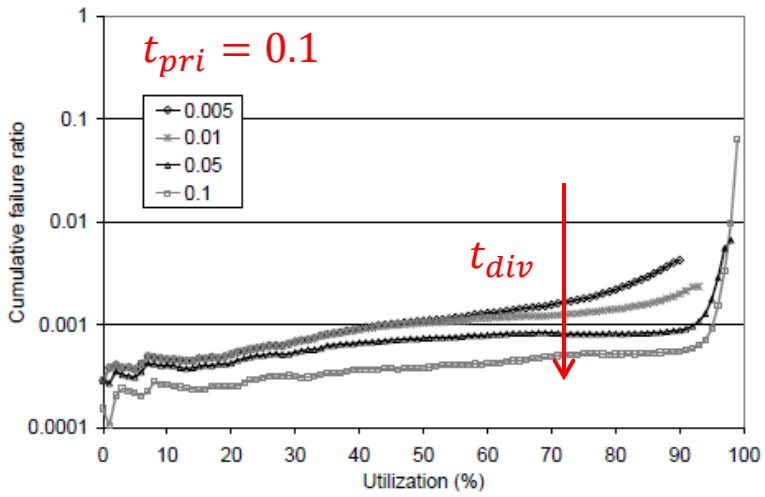
Dist. Name	Succeed	Fail	File diversion	Replica diversion	Util.
$l = 16$					
d_1	97.6%	2.4%	8.4%	14.8%	94.9%
d_2	97.8%	2.2%	8.0%	13.7%	94.8%
d_3	96.9%	3.1%	8.2%	17.7%	94.0%
d_4	94.5%	5.5%	10.2%	22.2%	94.1%
$l = 32$					
d_1	99.3%	0.7%	3.5%	16.1%	98.2%
d_2	99.4%	0.6%	3.3%	15.0%	98.1%
d_3	99.4%	0.6%	3.1%	18.5%	98.1%
d_4	97.9%	2.1%	4.1%	23.3%	99.3%

	Median	Mean	Max	Min	Number of files
NLANR	1,312B	10,517B	138MB	0	10,517
File system	4,578B	88.233B	2.7GB	0	2,027,908

Experimental Results

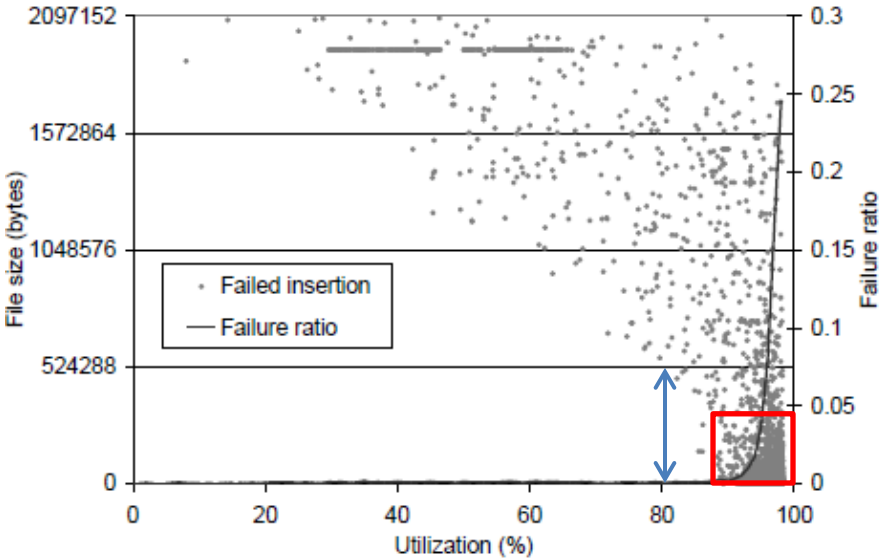
- Impact of t_{pri} and t_{div}
  - Cumulative failure ratio of file insertion v.s. Storage utilization ratio
    - **Reminder:** if $\frac{S_D}{F_N} > t$, the file insertion is rejected.



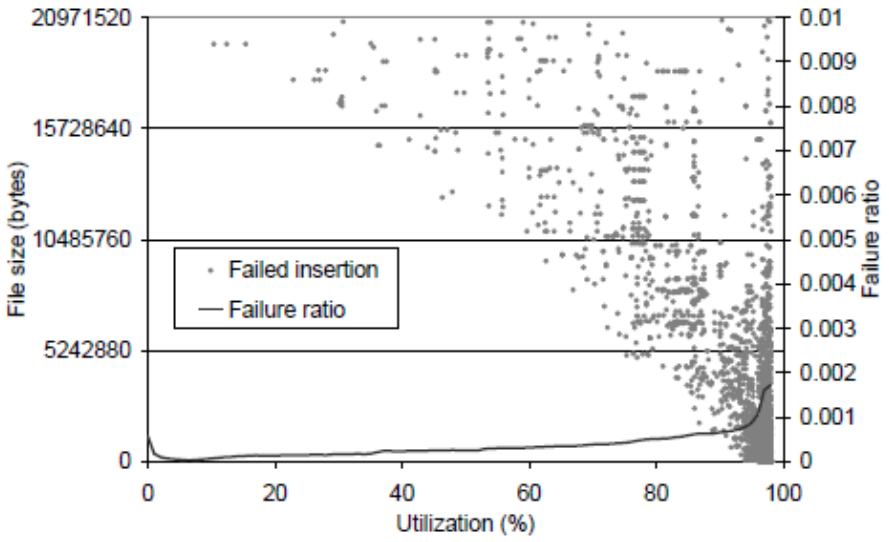


Experimental Results

- Rejected file sizes v.s. utilization



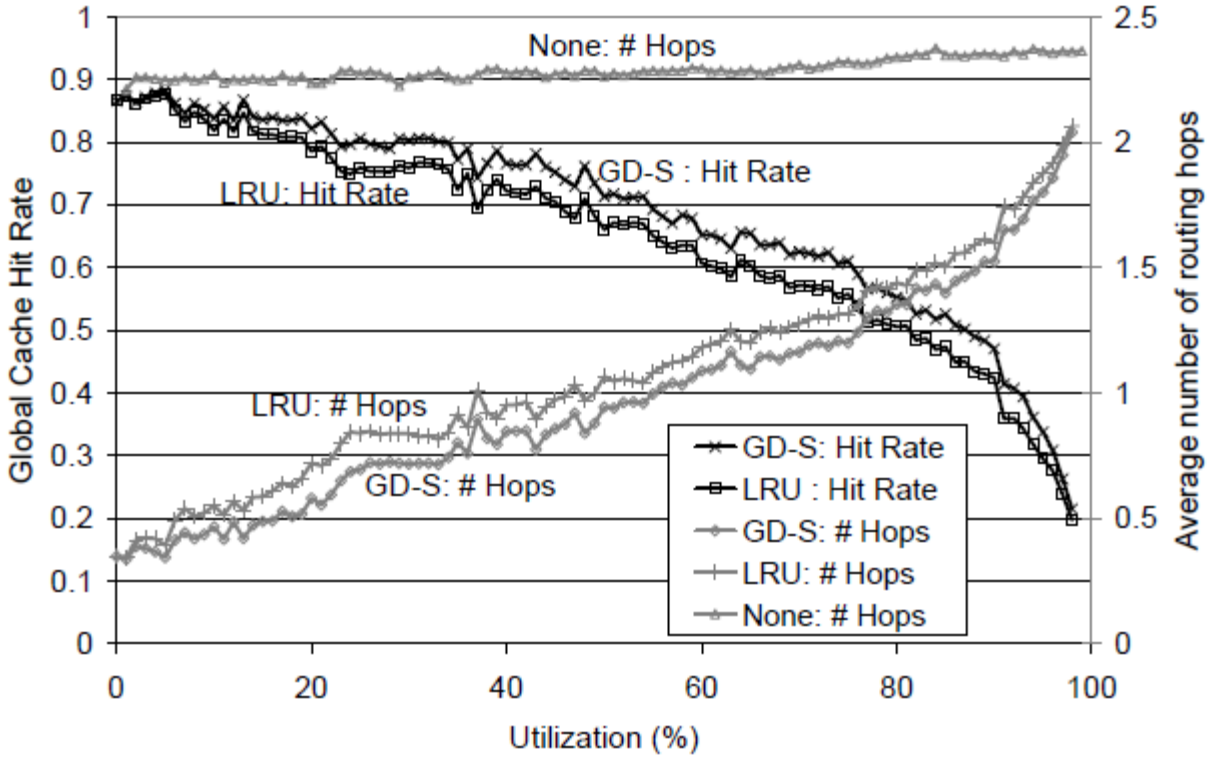
MLANR trace



File system trace

Experimental Results

- Impact of caching
  - GD-S v.s. LRU v.s. No caching



Discussions

- Any methods to optimally decide replication factor *k*?
- Whole file storage (PAST) v.s. file fragmentation (CFS)?
  - Trade-off?
- Semantics:
  - Read-only operations
  - Directory lookup, delete, key distribution, etc.
- Concurrent joining of nodes?

- **Discussions from piazza:**
  - Pitfalls of invariant based system?
  - Stability when there are frequent node removals and additions?
  - Applicability in real scenarios?

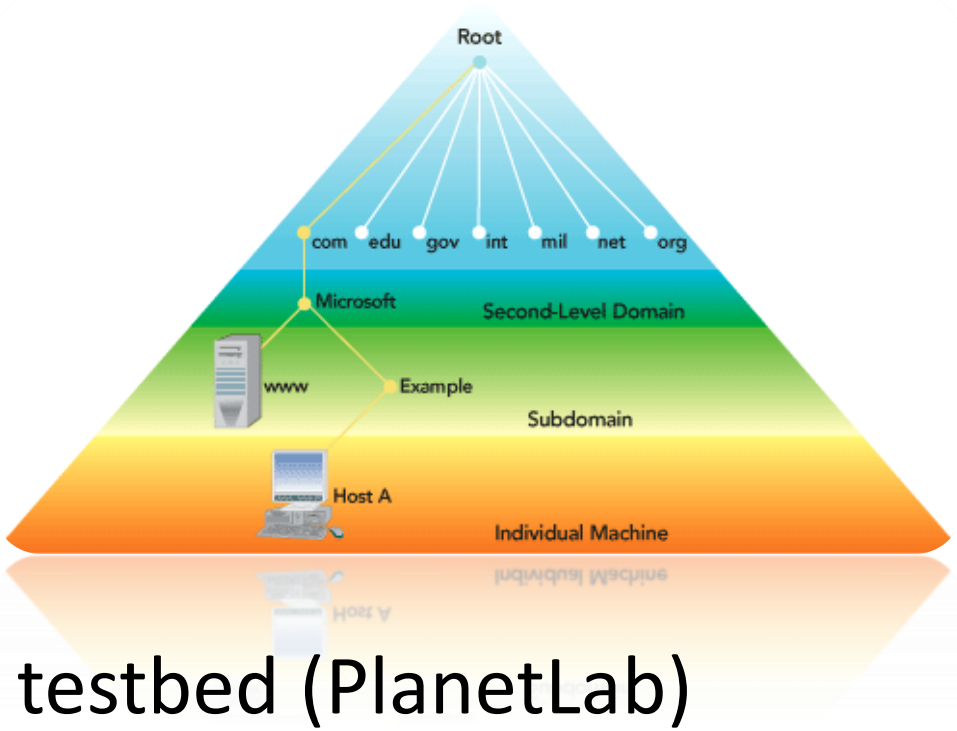
CoDNS: Masking DNS Delays via Cooperative Lookups

Presented by Zhenhuan Gao

03/07/2013

Introduction

- Domain Name System
  - Effectiveness, human-friendliness, scalability
  - Convert domain to IP
  - Multiple levels
  - Local nameserver



- Wide-area distributed testbed (PlanetLab)
  - Diagnosing “failures”
  - Providing a cooperative lookup scheme to mask the failure-induced local delays

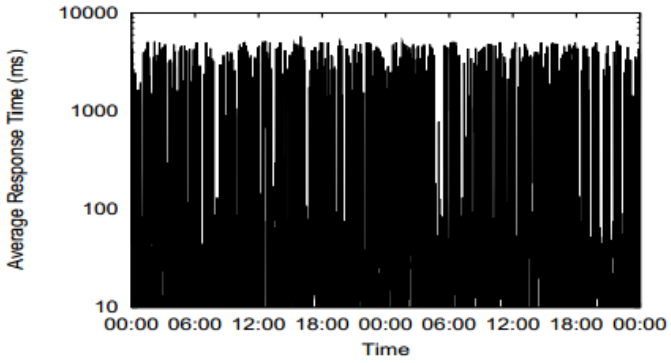
Background and Analysis

- CoDeeN content distribution network (CDN)
  - Consists of a network of Web proxy servers that include custom code to control request forwarding between nodes.
  - When forward requests to the origin server, it performs a DNS lookup to convert the server's name into an IP address in a timely manner.
  - Desire to have a standard for comparison across all CoDeeN nodes.

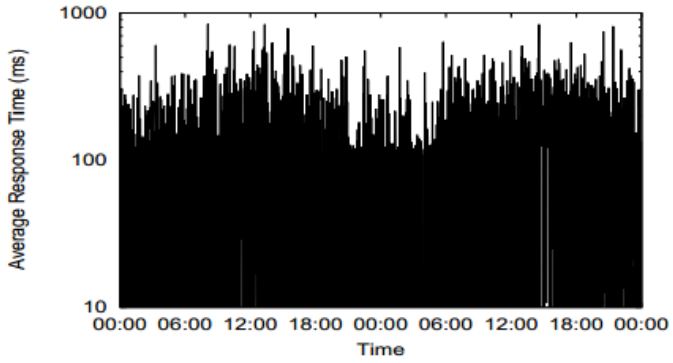
CoDeeN

Background and Analysis

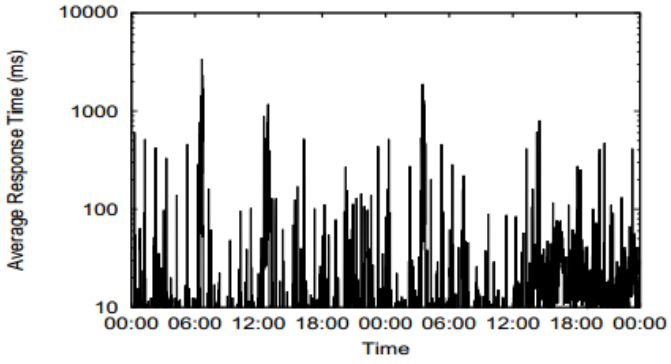
- Name Lookups of CoDeeN Nodes (10% CodeeN)



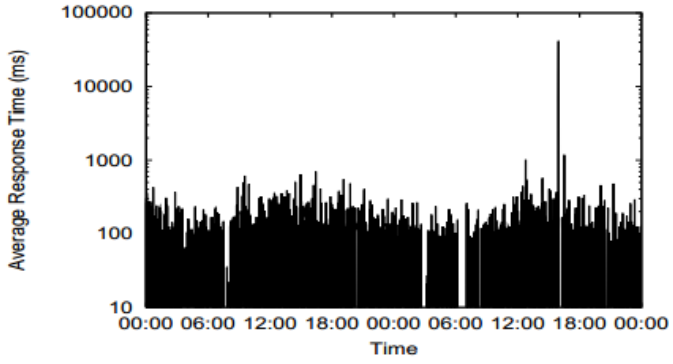
(a) planetlab1.cs.cornell.edu



(b) lefthand.eecs.harvard.edu



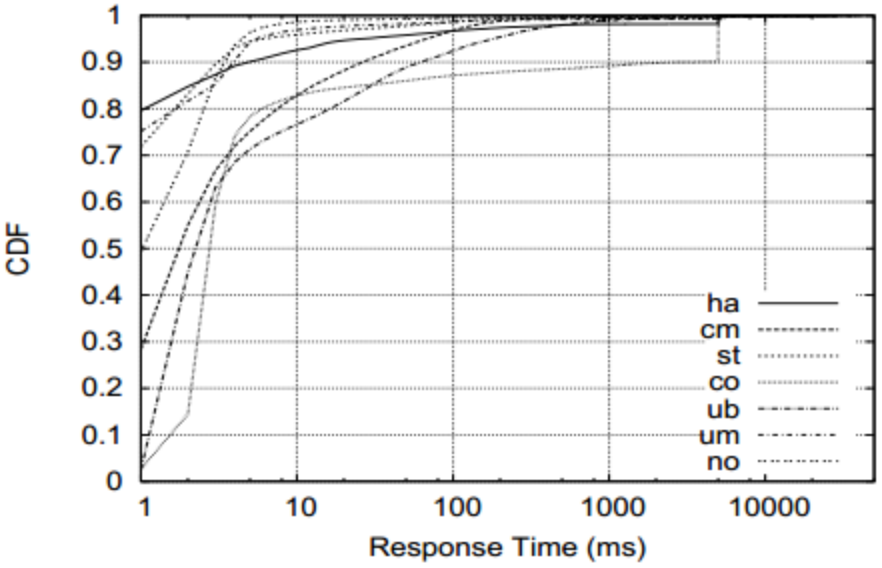
(c) planetlab-1.cmcl.cs.cmu.edu



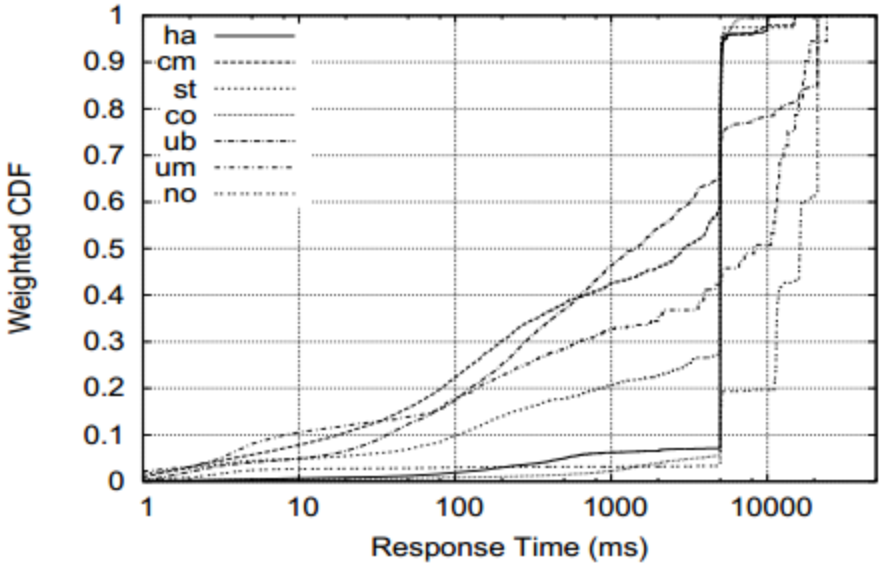
(d) kupl1.ittc.ku.edu

Background and Analysis

- Name Lookups of CoDeeN Nodes
  - The number of requests which fail is small
  - However, figure (b) indicates **a small percentage of failure cases dominates the totall time!**



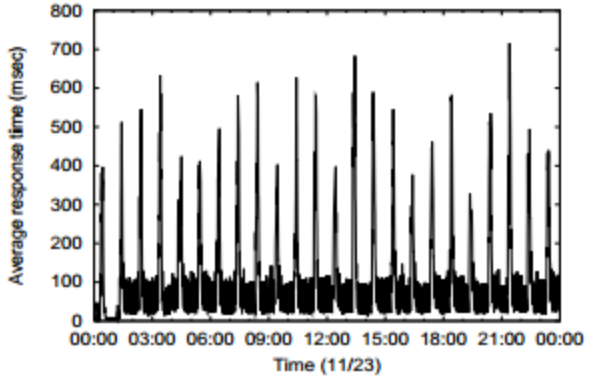
(a) Percentage of lookups taking < x ms



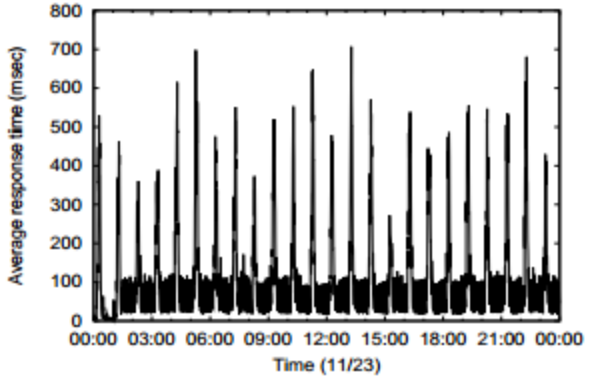
(b) Percentage of the sum of lookups taking < x ms

Background and Analysis

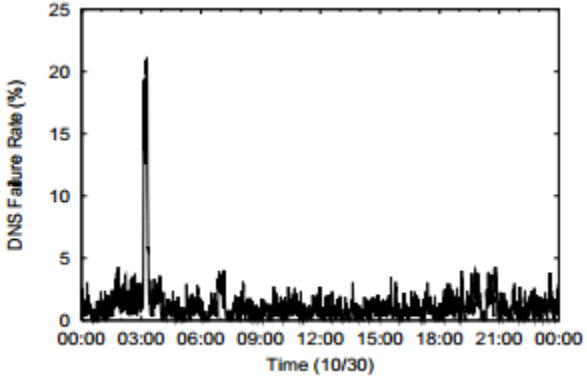
- The poor responsiveness stems from the node performing the measurement? No, because,



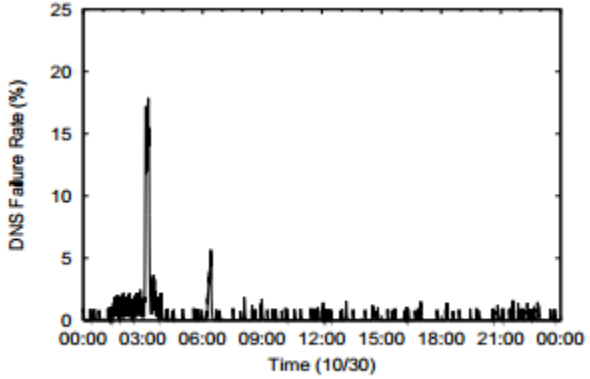
(a) purdue1



(b) purdue2



(a) harvard1



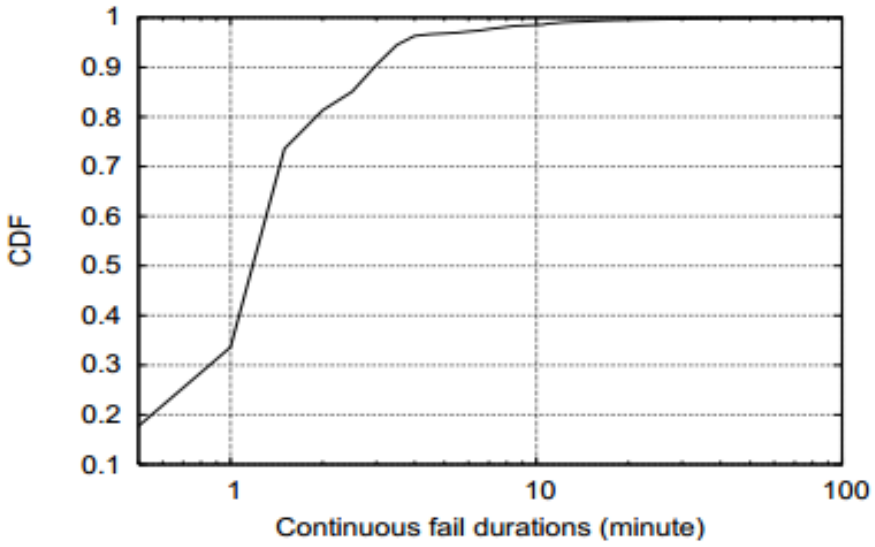
(b) harvard2

Background and Analysis

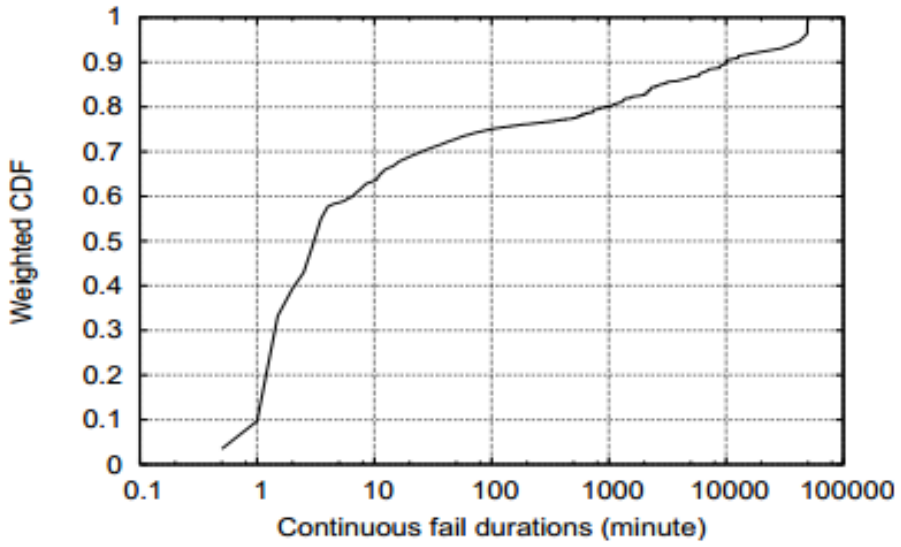
- Failure Characterization
  - Periodic failures
    - Cron jobs running on the local nameserver.
  - Long lasting continuous failures
    - Local nameserver malfunctioning or extended overloading.
  - Sporadic short failures:
    - Temporary overloading of the local name server.

Background and Analysis

- Failure Characterization
  - How long the failures typically last?



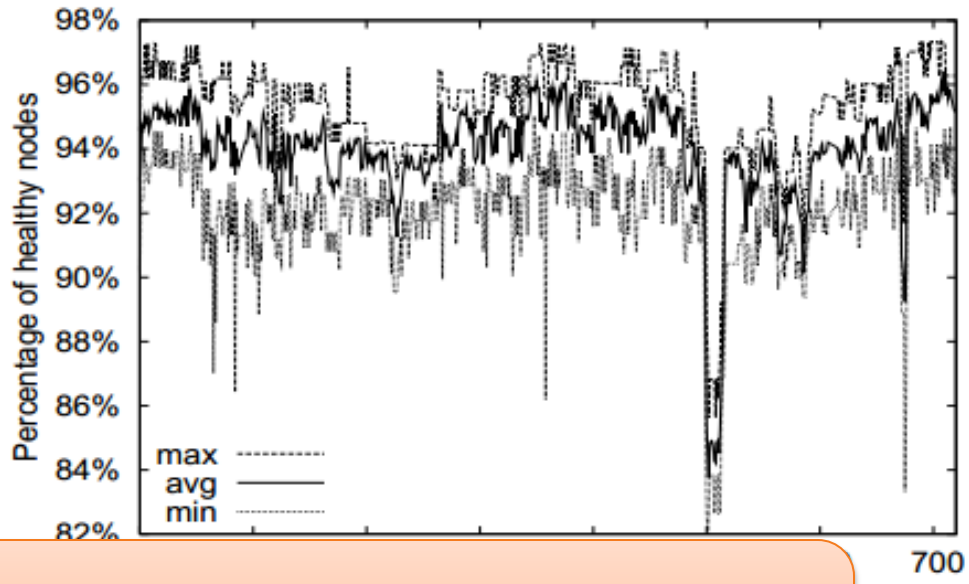
(a) CDF



(b) Weighted CDF

Background and Analysis

- Correlation of the DNS lookup failures
  - "Healthy" servers
    - Failure rate < 1%
    - Less than 1.25x global failure rate
    - Avoiding failure for some DNS sites
  - Healthy server > 90%



As long as there is a reasonable number of healthy nameservers, they can be used to mask locally-observed delays

good NS

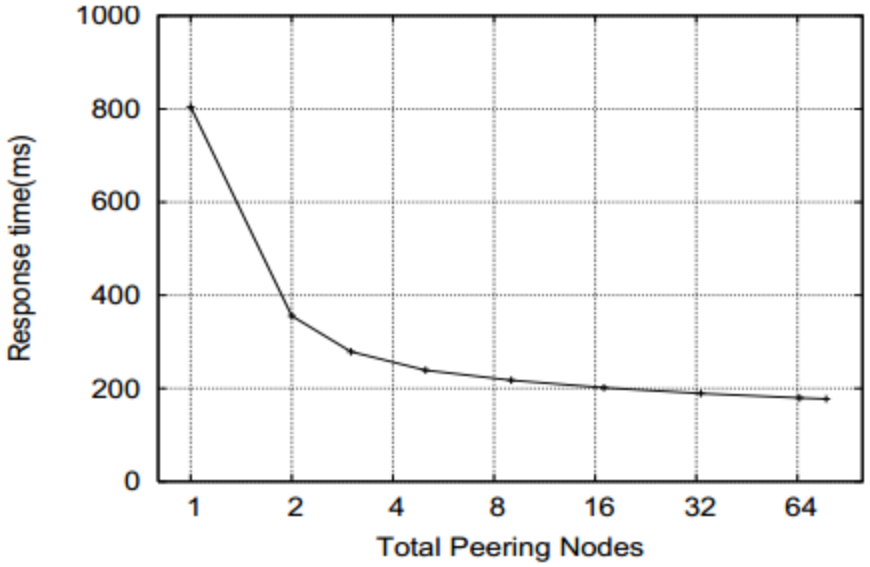
Design

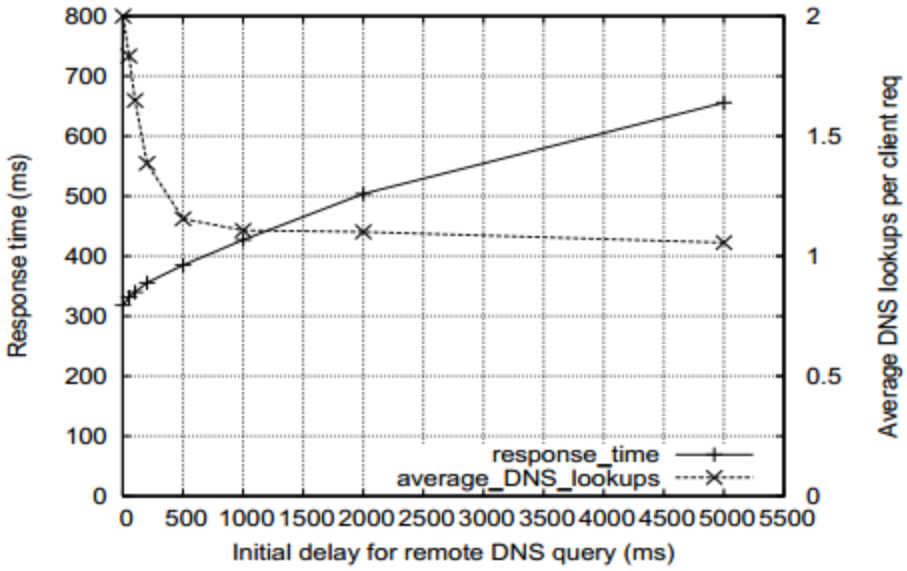
- CoDNS
  - Forward name lookup queries to peer nodes when the local name service is experiencing a problem
  - When to send remote queries?
    - Most name lookups are fast in the local nameserver.
    - Spreading the requests to peers might generate additional traffic.
  - Proximity and Locality
    - Trivial

When to using remote servers and how many to involve?

Design

- CoDNS
  - Experiment
    - Relationship between CoDNS response time and peers involved
    - Extra DNS overhead





Design

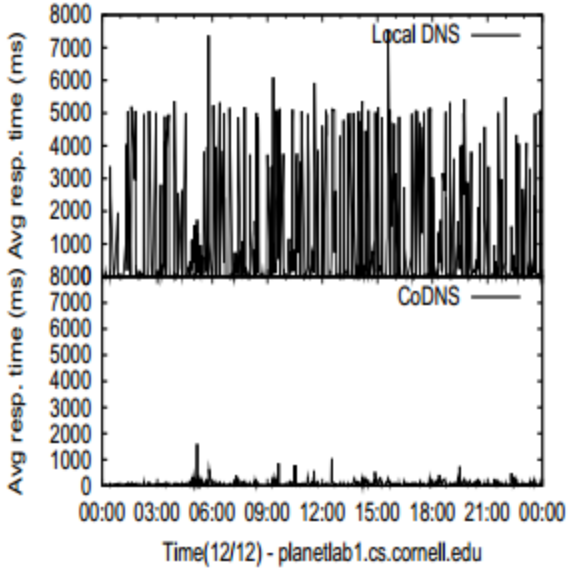
- Other Approaches
  - The recursive DNS query ability into local node
    - Reduces the caching effectiveness
    - Increases the configuration efforts and also causes extra management problems
    - More resources on each node
  - making the resolver library on the local node act more aggressively
    - Many failures observed are caused by overload rather than network packet loss
    - Second nameserver will be overloaded as a result
    - The problems are local, not global

Implementation

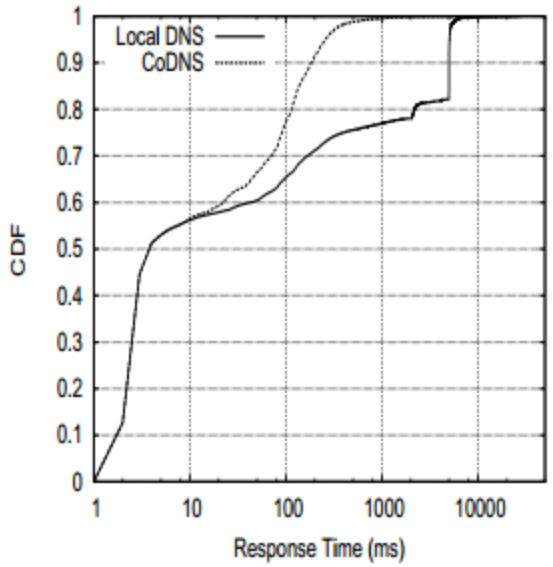
- Remote query initiation
  - The initial delay would be dynamically adjusted
- Proximity, Locality and Availability
  - Each CoDNS node gathers a set of eligible neighbors
  - Liveness is periodically checked
  - Heartbeat to neighbors every 30s
  - Periodically update dead nodes with fresh ones

Results

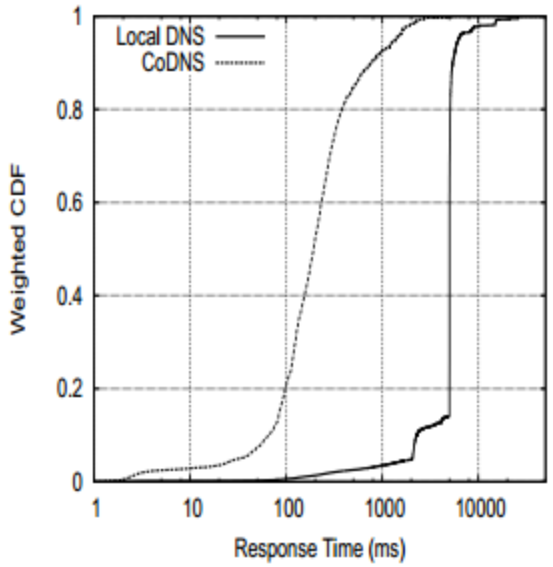
- Local DNS vs. CoDNS



(a) Average Response Time



(b) Percentage of lookups taking < x ms

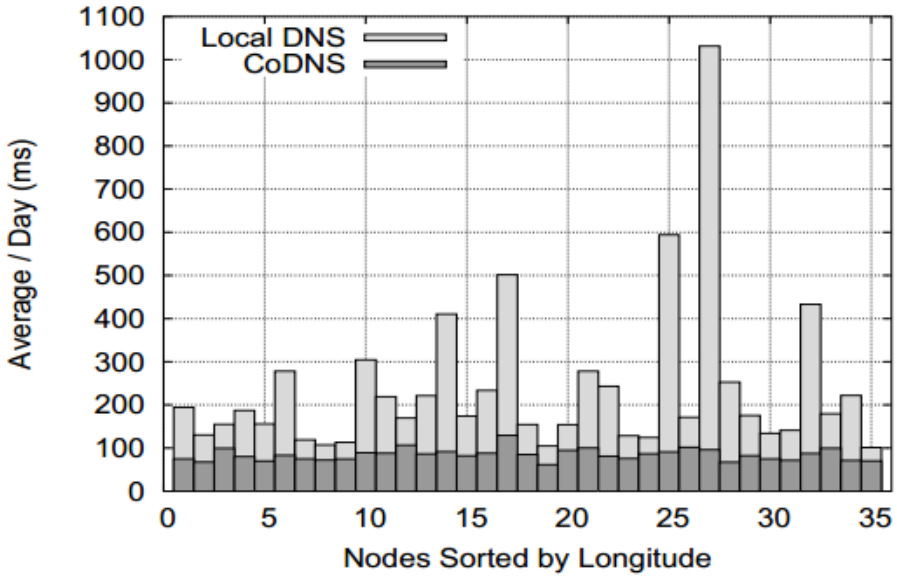


(c) Percentage of the sum of all lookups taking < x ms

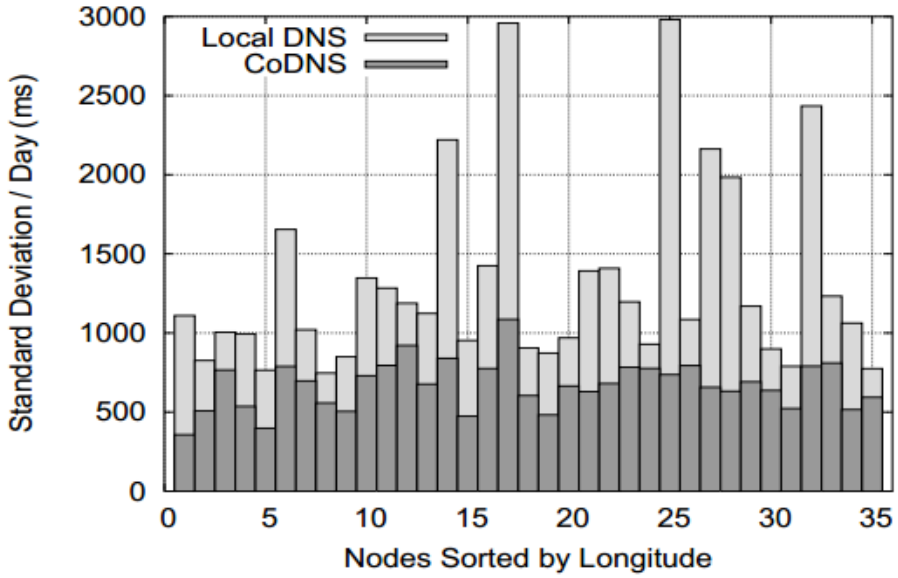
Non-existent name → fail at first phase → network problem → CoDNS ↓

Results

- Local DNS vs. CoDNS
  - Average response time
  - Standard deviation



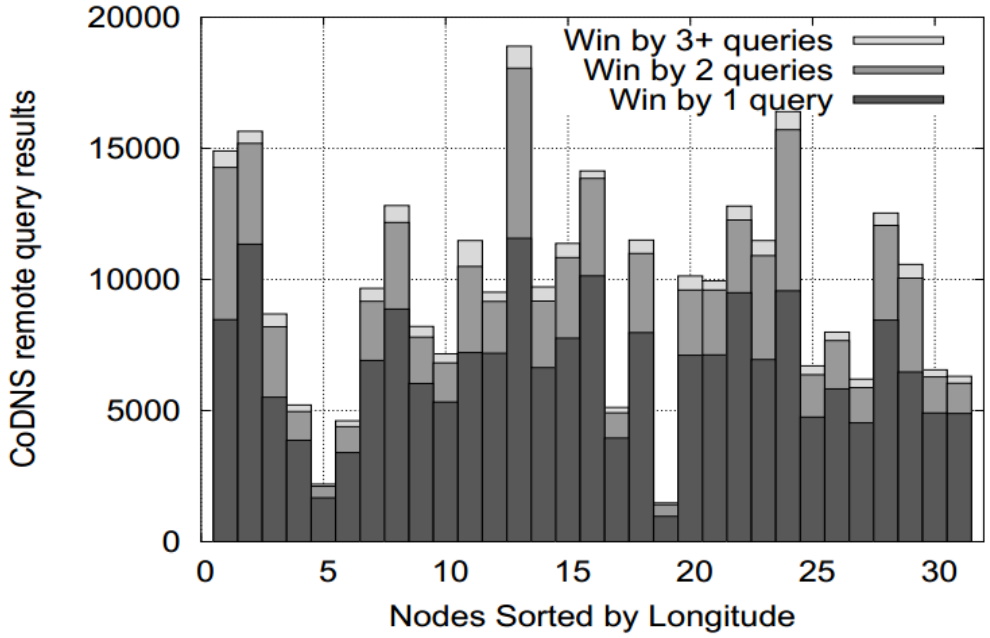
(a) Average Response Time over 12/11



(b) Standard Deviation over 12/11

Results

- Analysis
  - 18.9% of all the lookups using remote peers
  - 34.6% of the remote queries “win”
  - The effect of multiple querying



Discussion

- Locality and proximity?
- privacy Issue
- Trust build with peer nodes
- Failure in master nameserver

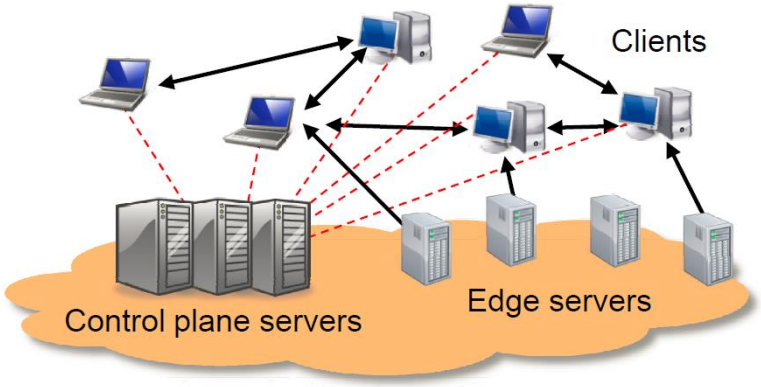
Reliable Client Accounting for P2P-Infrastructure Hybrids

Presented by Haiming Jin

2013-03-07

Background

- Hybrid CDN-P2P architecture
  - **P2P:** Scalability, infrastructure independent, etc.
  - **Infrastructure:** Predictable QoS, etc.
  - **Commercial hybrid systems:** Net Session, Livesky, etc.



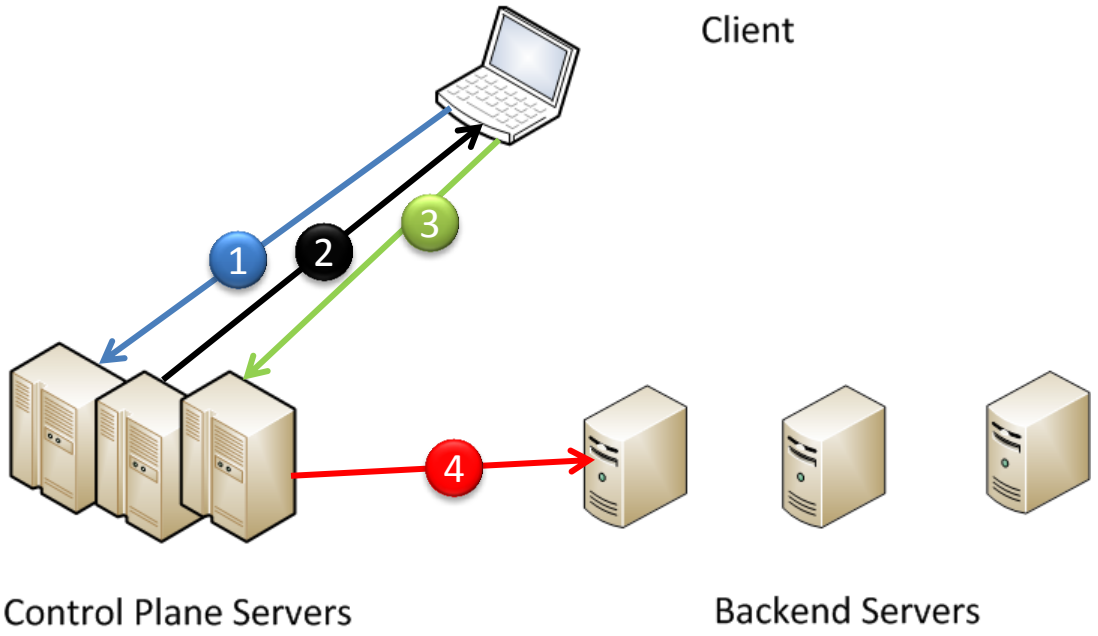
- Accounting reliability?

Threat Models and Countermeasures

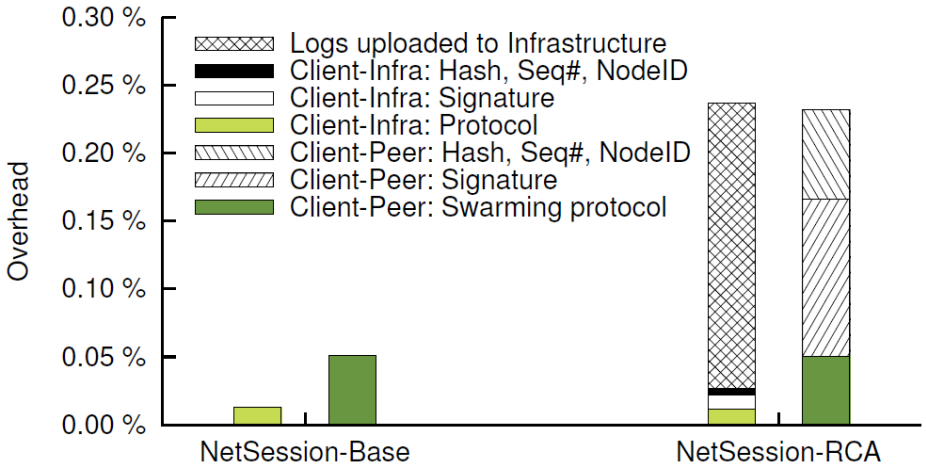
Threat models	Countermeasures
Fail to log exact set of messages sent or acknowledged	Message commitment
Fail to log consistent sequence of messages	Log consistency checking
Execute illegal, or fail to execute required protocol action	Log plausibility checking
Faulty peers collude to report fictitious exchanges	Client paring control and anomalous client quarantine
Render poor service to peers	Anomalous client quarantine
Nefarious user requests	Suspicious user behavior throttling/flagging
Sybil attack	Resource limits enforcement

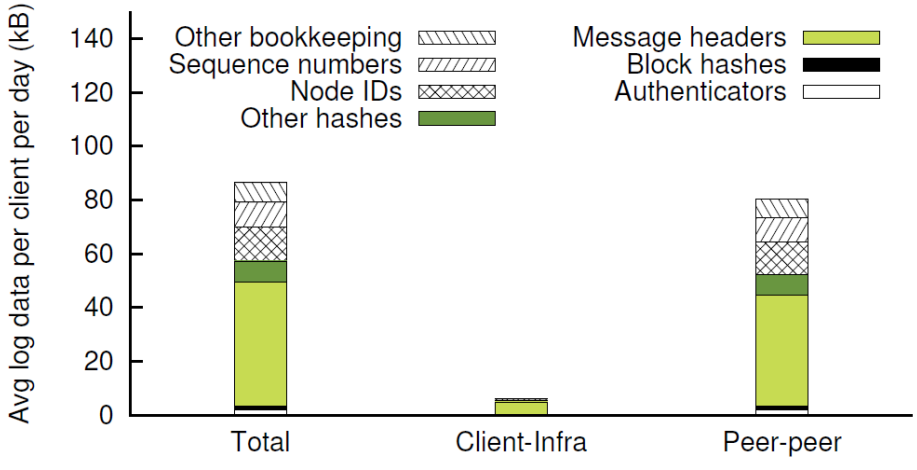
Application to NetSession-RCA System

- RCA workflow
  - 1. The client uploads a short file to demonstrate its link capacity
  - 2. Private key σ_i, public key π_i and certificate Γ_i
  - 3. Periodically uploading of temper-evident log
  - 4. Forwarding of temper-evident log to backend servers



Performance Evaluation





Discussions

- Infrastructure resource consumption in quarantining clients?
- Applicability to other P2P hybrid systems?
- Plausibility of adversary model?
- Scalability of the scheme?
- Overhead in storage space, network traffic, etc.?