OpenDHT: A Public DHT Service and Its Uses

Sean Rhea, Brighten Godfrey, Brad Karp, John Kubiatowicz,
Sylvia Ratnasamy, Scott Shenker, Ion Stoica, and Harlan Yu

UC Berkeley and Intel Research

SIGCOMM 2005

Presented by: Igors Svecs

Outline

- Tradeoffs in DHT design
- Related work
  - Dynamo
  - i3
  - Cooperative DNS and Beehive
- OpenDHT

Tradeoffs: Distributed Systems

Distributed systems often involve tradeoffs:

Failure detection:

- Completeness (detect 100% failures)
- Accuracy (no false positives)

Distributed objects:

- Reliability
- Availability
- Consistency
- Performance

Tradeoffs: DHTs

DHT interface:

- Simplicity: ease-of-use
- Generality: more control

Interfaces:

- Routing: general access to any DHT node along the path
- Lookup: general access to DHT node responsible for the key
- Storage: only supports put(key, value) and get(key) operations

(extensions: *context* in Dynamo)

Visibility:

- Public: security and fairness concerns
- Private: improved performance

Tradeoffs: Overlays

Gnutella

- Makes no assumptions about topology
- Queries are flooded out
- Simple to maintain, but has O(n) lookup time

Chord

- O(log n) lookup time by imposing structure (logical ring topology)
- Must perform stabilization to maintain consistency

O(1) and pseudo-O(1) overlays (Dynamo, Beehive)

- Not scalable
- Can perform only basic *map* interface (get, put)

Dynamo: Amazon's Highly Available Key-value Store

Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall and Werner Vogels

Amazon.com

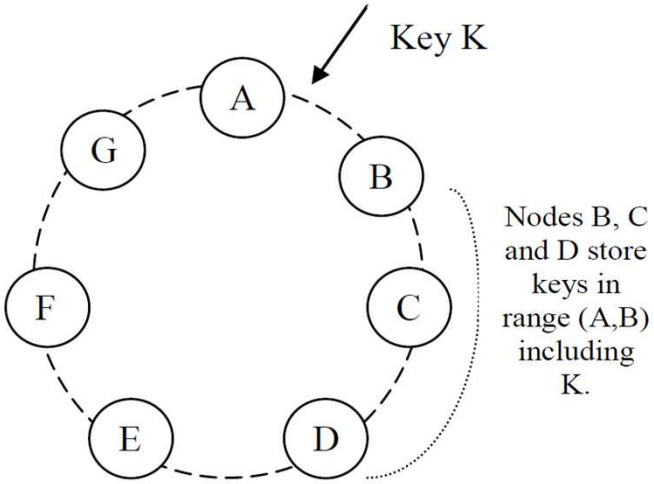
SOSP 2007

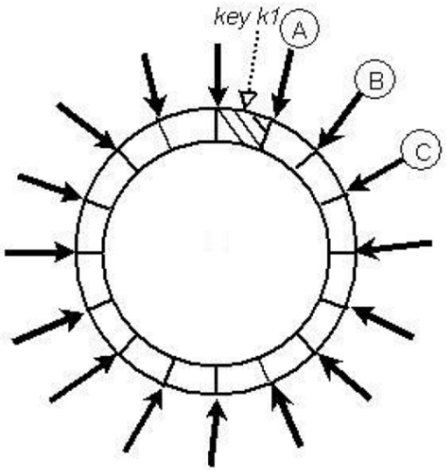
Dynamo

Observations:

- Split workload into many atomic services → scalability is not important
- SLAs demand high availability and guaranteed performance
- Best reconciliation strategy depends on application

Consistent hashing distributes keys equally along the ring





Dynamo

Maintaining consistency: **Quorum** system

- N nodes
- Choose parameters $R <= N$, $W <= N$, such that $R + W > N$
- R votes required for successful read
- W votes required for successful write
- Increasing R yields the same effect as decreasing W:
  Improving consistency at the cost of availability

Client-driven requests

- One hop instead of two
- Reduces latency by a factor of two

Consider other DHT applications. When are client-driven requests undesirable, and why?

Internet Indirection Infrastructure

Ion Stoica, Daniel Adkins, Shelley Zhuang, Scott Shenker, Sonesh Surana

University of California, Berkeley

SIGCOMM ‘02

i3

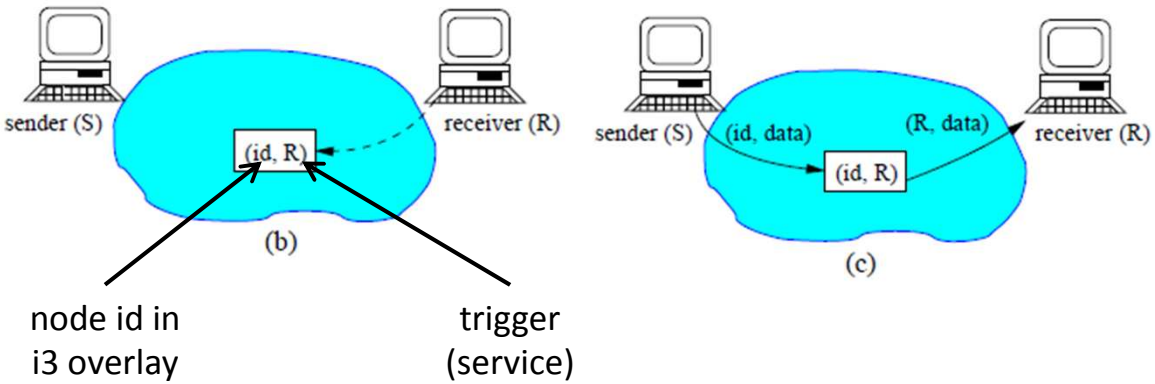
Need to generalize Internet point-to-point communication infrastructure (recall Layered Naming paper)

- Mobility, Multicast, Anycast

Rendezvous-based communication abstraction

i3's Application Programming Interface (API)	
$sendPacket(p)$	send packet
$insertTrigger(t)$	insert trigger
$removeTrigger(t)$	remove trigger

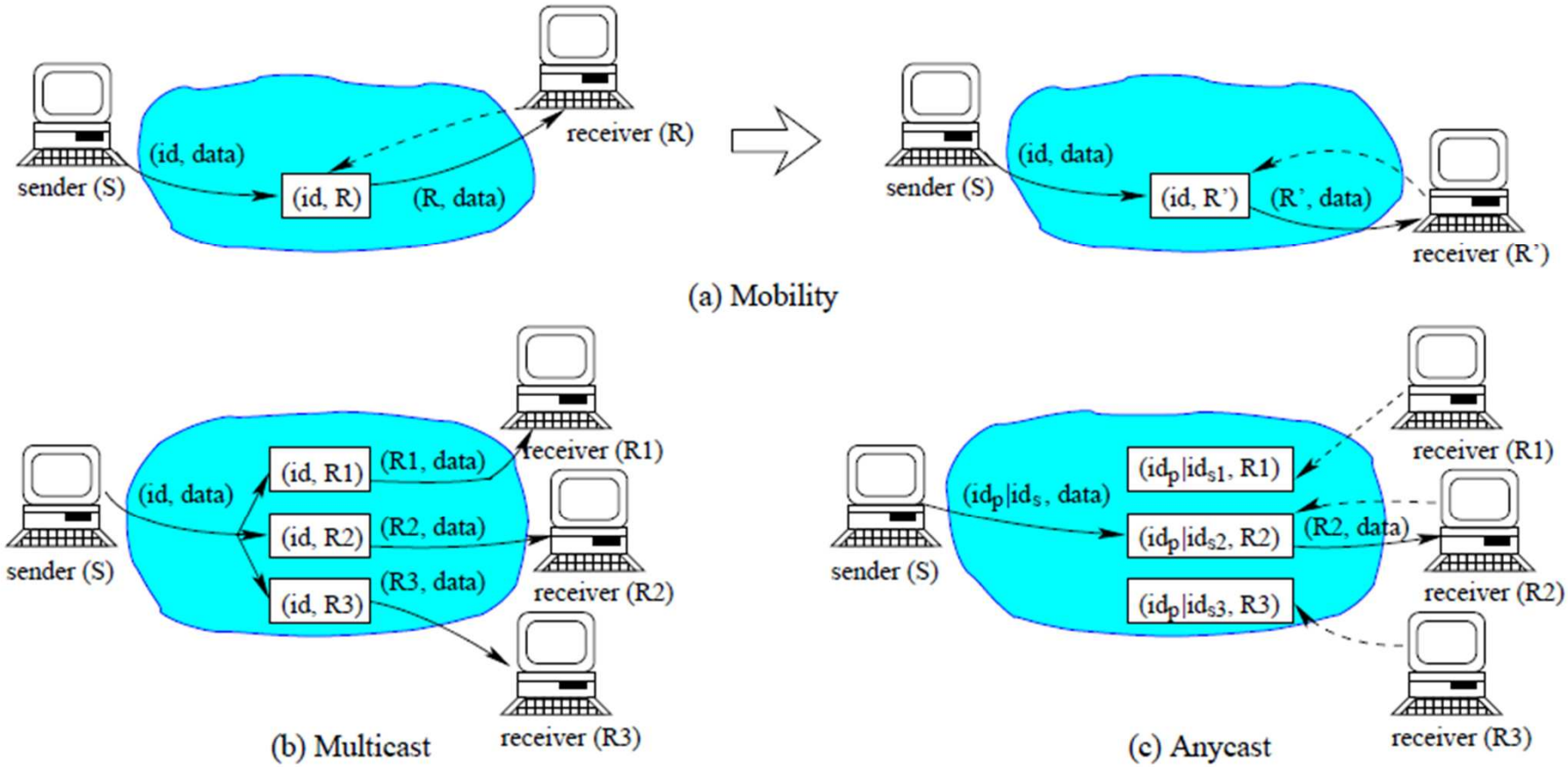
(a)



i3 node id 1000101001111001011101 01

k significant bits

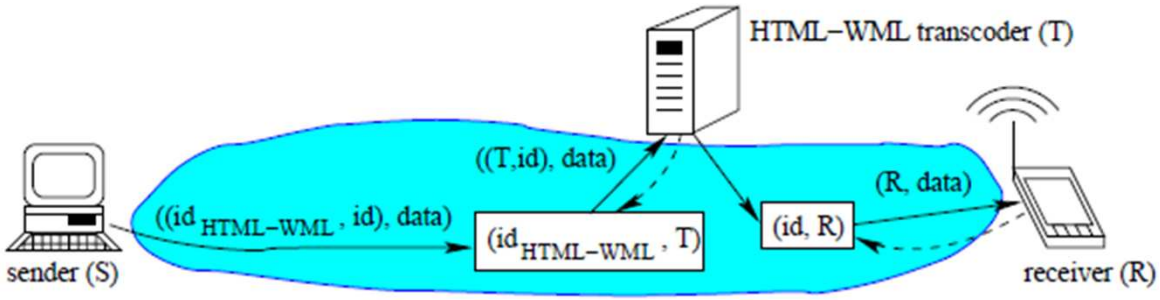
Abstractions



Stack of Identifiers – more indirection!

Identifiers are replaced by a stack → service composition

- Packet $p = (id_{stack}, data)$
- Trigger $t = (id, id_{stack})$

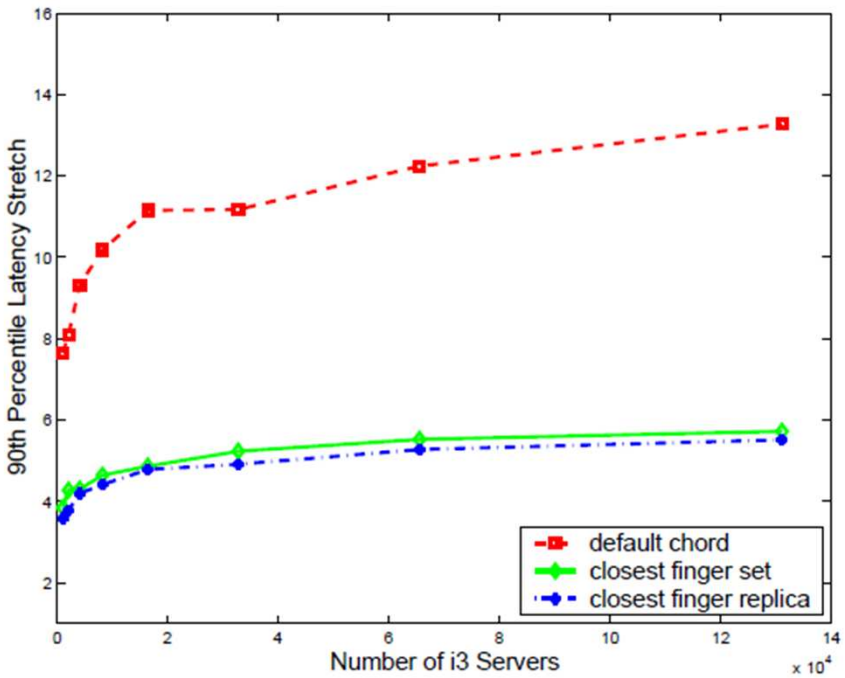


(a) Service composition

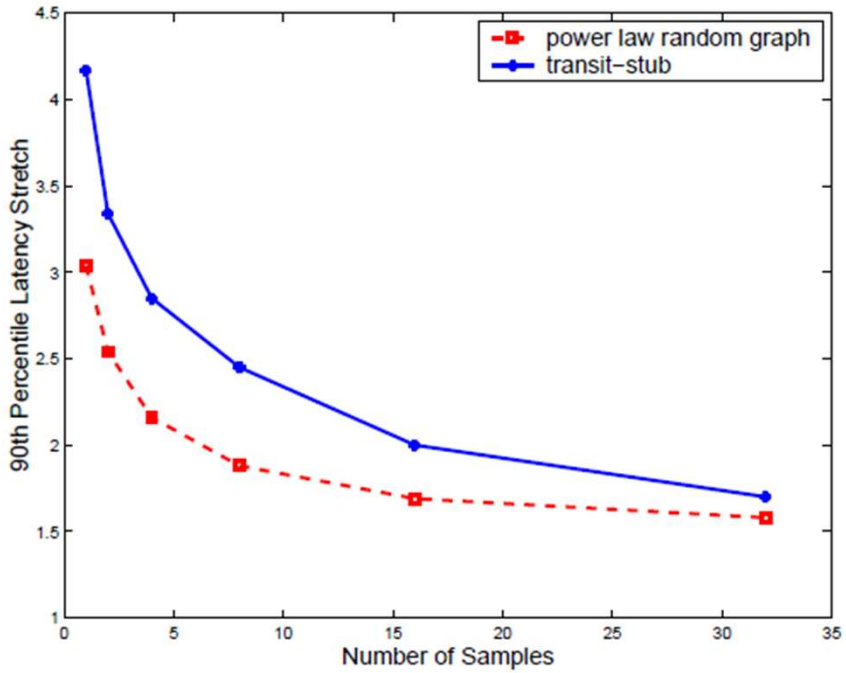
Public and private triggers: Create a pair of public triggers for the duration of the flow to improve routing performance and security

Latency

Stretch = (latency over i3) / (latency over direct route)



first packet



later packets

The Design and Implementation of a Next Generation Name Service for the Internet

Venugopalan Ramasubramanian, Emin Gun Sirer

Dept. of Computer Science, Cornell University

SIGCOMM ‘04

CoDoNS

Problems with “legacy” DNS:

- Failure resilience – bottlenecks
  - In 79% cases, there are only 2 DNS servers
- Implementation errors
- Performance – latency
  - DNS lookups contribute more than one second to 20% object retrievals
- Performance – misconfigurations
  - Inconsistent responses from 14% of domains.
- Performance – load imbalance. Large load at root servers.
- Update propagation – large scale caching cause stale records

Beehive: overview

Proactive caching layer with O(1) average lookup performance built on top of structured DHTs such as Chord.

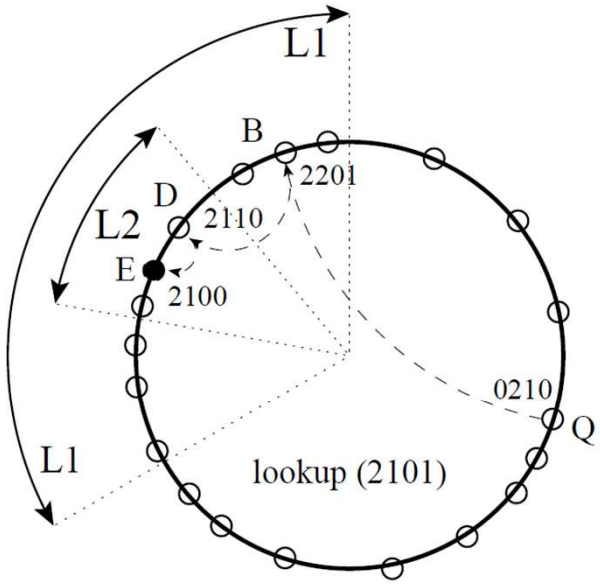


Figure 3: Proactive Caching in Beehive: Caching an object at all nodes with i matching prefix-digits ensures that it can be located in i hops. Beehive achieves $O(1)$ average lookup time with minimal replication of objects.

Beehive: analytical model

Minimizes bandwidth and space consumption by posing the following optimization problem:

> minimize total number of replicas subject to the constraint that the aggregate lookup latency is less than a desired constant *C*.

They derived closed-form solution that:

- Captures the space-time tradeoff
- Minimizes bandwidth and storage overhead by picking optimal number of replicas required to achieve target performance
- Besides decreased latency, replicating objects achieves load balancing
- Use level of replication to quickly locate and update other replicas

Replication parameters are derived from the aggregate access frequencies.

CoDoNS: Evaluation

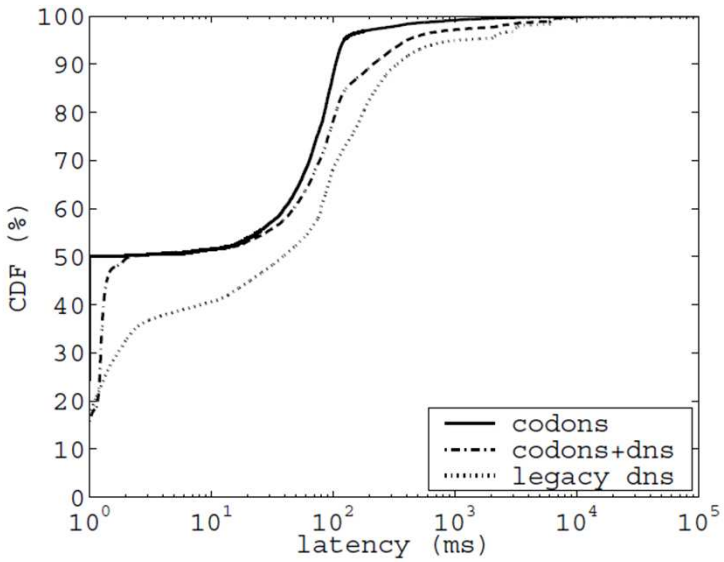


Figure 5: Cumulative Distribution of Latency: CoDoNS achieves low latencies for name resolution. More than 50% of queries incur no network delay as they are answered from the local CoDoNS cache.

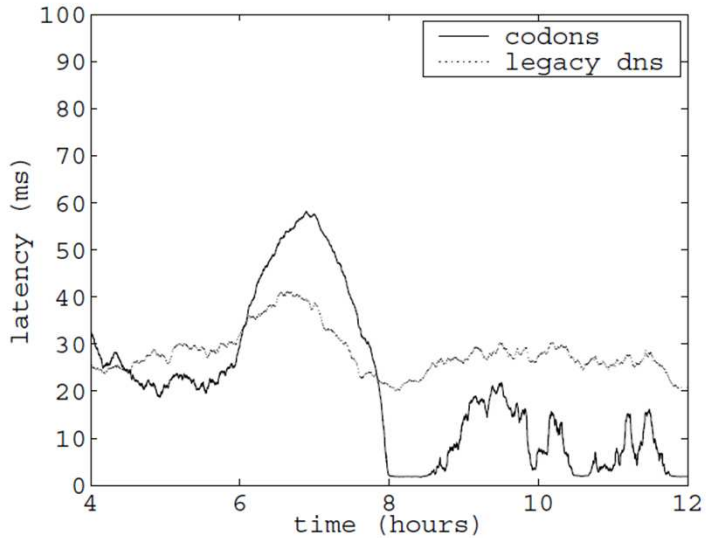
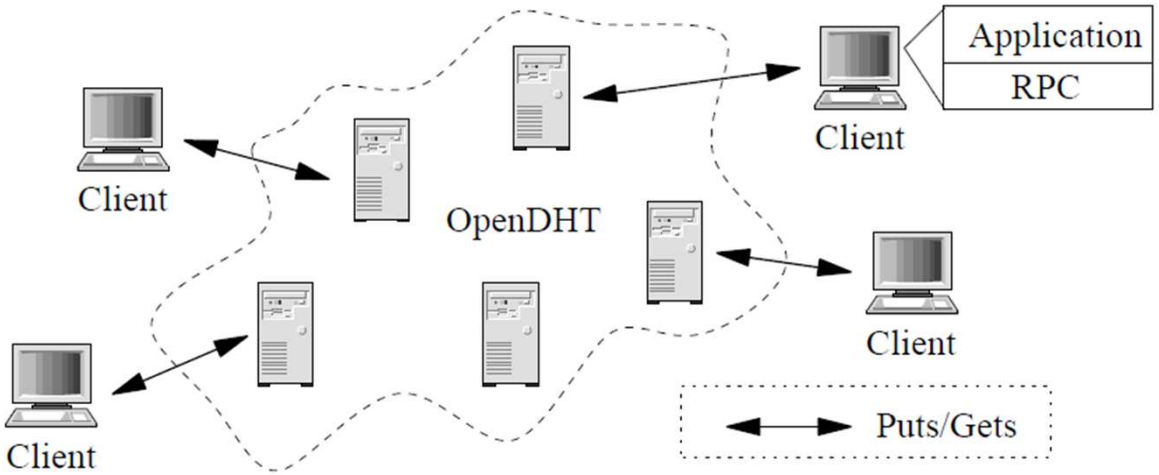


Figure 7: Median Latency vs Time as a flash-crowd is introduced at 6 hours: CoDoNS detects the flash-crowd quickly and adapts the amount of caching to counter it, while continuing to provide high performance.

OpenDHT

Design goals:

- Public storage facility, mutually untrusting clients
- Fair storage allocation
- Simple interface
- Sharing both among applications and among clients
  (compare to Dynamo where there is an instance per application)



Storage allocation: Design

As discussed before, DHT service may expose different levels of interface (routing, lookup, storage)

OpenDHT chose **storage** for ease-of-use with additional workarounds to provide **lookup** functionality (discussed later)

Persistent public storage eventually fills up with **orphaned data**

Solution: **revolving door** approach

- Old data is overwritten with new data; clients must periodically refresh
- Previous work relies on charging to enforce fairness
- Keeping clients informed when data is deleted – **time to live (TTL)**
- Charging is not acceptable in OpenDHT; need other mechanisms

Interface

(i) simple, (ii) no restriction on keys, (iii) authentication

Procedure	Functionality
$put(k, v, H(s), t)$	Write (k, v) for TTL t can be removed with secret s
$get(k)$ returns $\{(v, H(s), t)\}$	Read all v stored under k returned value(s) unauthenticated
$remove(k, H(v), s, t)$	Remove (k, v) put with secret s $t >$ than TTL remaining for put
$put\text{-}immut(k, v, t)$	Write (k, v) for TTL t immutable $(k = H(v))$
$get\text{-}immut(k)$ returns (v, t)	Read v stored under k returned value immutable
$put\text{-}auth(k, v, n, t, K_P, \sigma)$	Write (k, v), expires at t public key K_P; private key K_S can be removed using nonce n $\sigma = \{H(k, v, n, t)\}_{K_S}$
$get\text{-}auth(k, H(K_P))$ returns $\{(v, n, t, \sigma)\}$	Read v stored under $(k, H(K_P))$ returned value authenticated
$remove\text{-}auth(k, H(v), n, t, K_P, \sigma)$	Remove (k, v) with nonce n parameters as for *put-auth*

k – 160-bit key (output of SHA-1)

all values are stored

removes are treated like puts

robust against *squatting*, but vulnerable to *drowning*

immutable puts – solve drowning, but restrict key choice

signed puts – gets return only previously signed puts

ReDiR (recursive distributed rendezvous)

- Adds **lookup** interface

Procedure	Functionality
join(*host*, *id*, *namespace*)	adds (*host*, *id*) to the list of hosts providing functionality of *namespace*
lookup(*key*, *namespace*)	returns (*host*, *id*) in *namespace* whose *id* most immediately follows *key*

- Nodes willing to provide a service **join** corresponding DHT
- Each node is associated with **id** from a 160-bit **key space**
- Each application is identified by a **namespace**
- Recall *i*3 – service nodes provide forwarding functionality
- Trigger (*k, d*) is stored on node returned by *lookup(k)*, service node forwards packets it receives for key *k* to *d*.
- Naïve approach – store ip/port of all nodes that joined *ns* under key *ns* → not scalable; linear growth

ReDiR

- Store two dimensional quad-tree in OpenDHT
- Each tree node is a list of (IP, port) pairs of *some* clients
- Nodes identified by level i and position from left j stored inside OpenDHT at key $H(ns, i, j)$
- Given a branching factor b, there are at most b^i nodes
- Associate with each node (i, j) b intervals in DHT keyspace

ReDiR

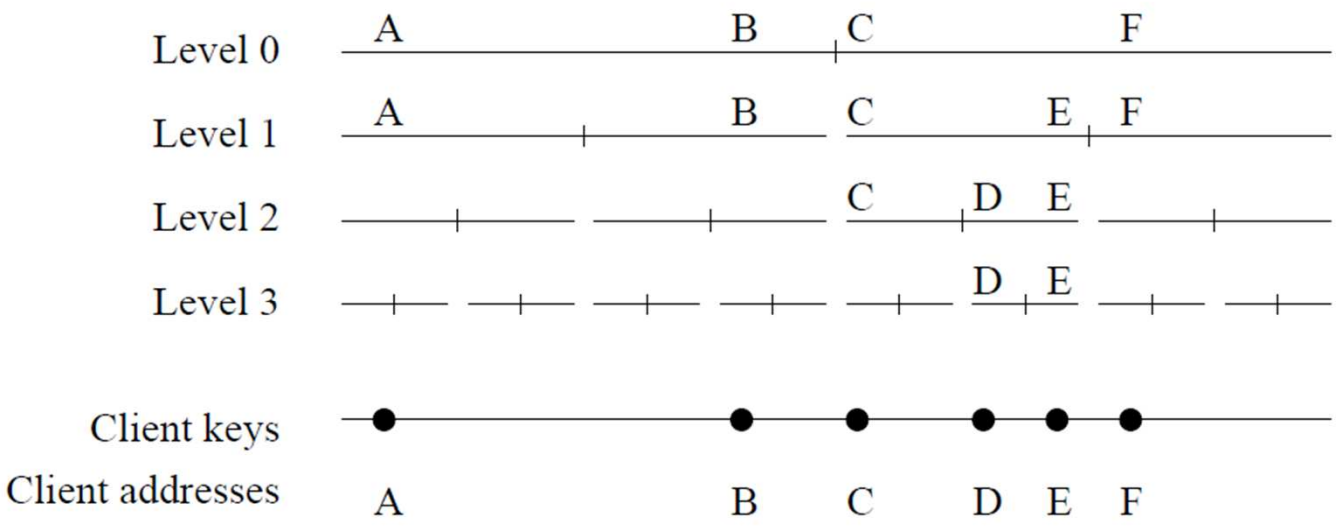
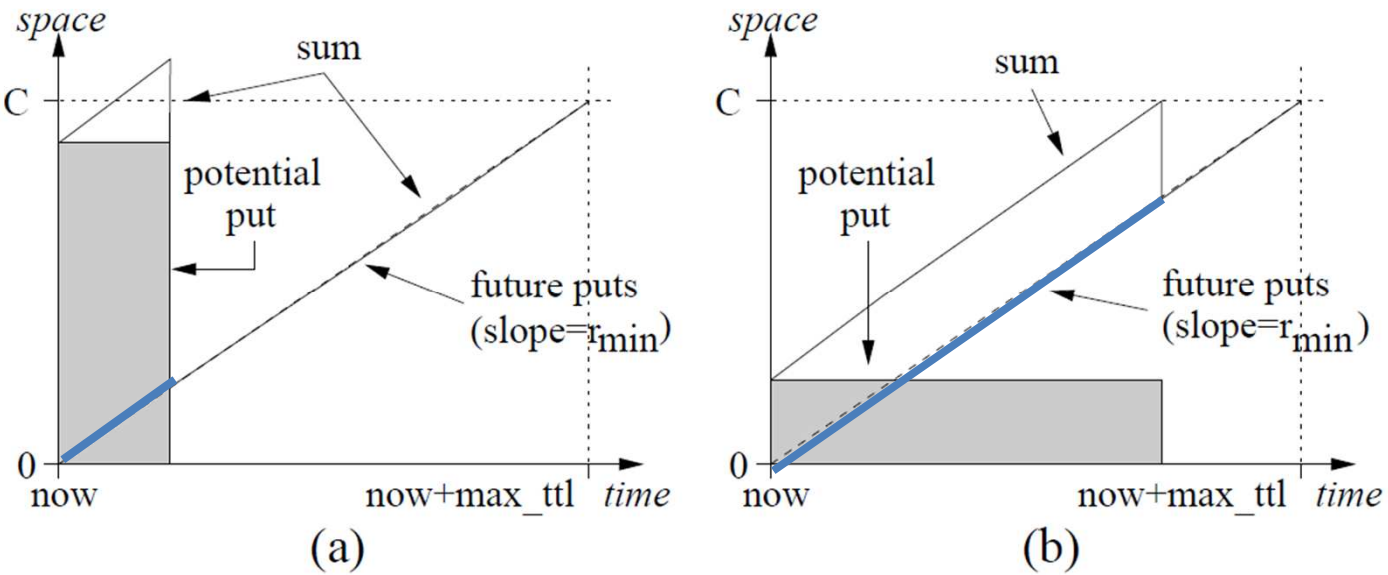


Figure 2: ***An example ReDiR tree with branching factor*** $b = 2$**. Each tree node is shown as a contiguous line representing the node's interval of the keyspace, and the two intervals associated with each node are separated by a tick. The names of registered application hosts (**A **through** F**) are shown above the tree nodes at which they would be stored.**

Fair Space-Time Algorithm

- Global fairness is hard to achieve, involves many tradeoffs
- Hence, per-disk model of fairness
- **Preventing starvation**: require each node to be able to accept at a rate $r_{min} = C / T$, C = disk capacity, T = maximum TTL
- Puts **must not endanger** reserved rate r_{min}



Fair Space-Time Algorithm

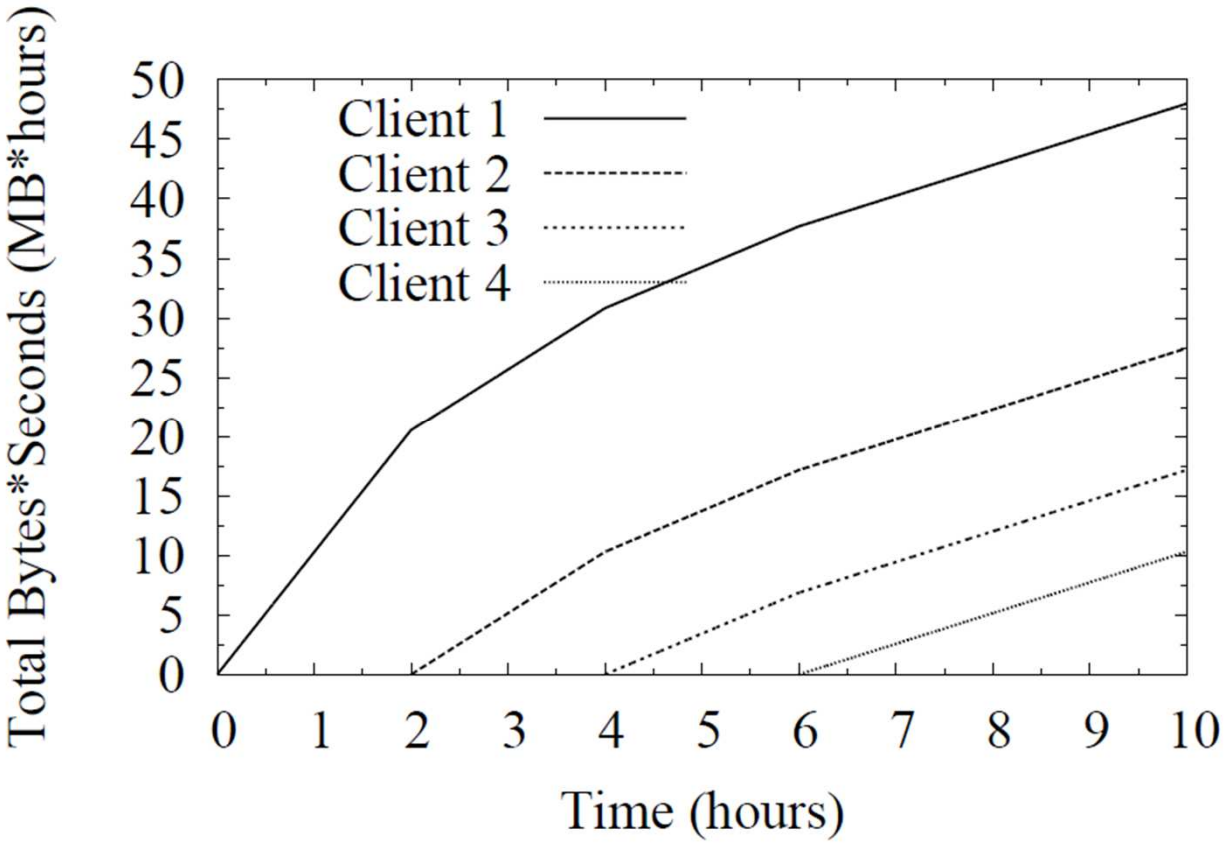
- Define *commitment* := size * TTL of a put
- Aim to equalize **rate** of commitments independently of each client's history (similar to Fair Queuing)
- Maintain a system virtual time $v(t)$ = maximum start time of all puts accepted before t

Algorithm

- Each nodes maintains a bounded queue for each client with puts pending
- Drop new puts if client's queue is full; else, compute start time & enqueue
- Select the put with the lowest start time
- If "admission control test" (starvation check) is passed, accept. Otherwise, sleep until the node can accept pending put

Evaluation – non-starvation

All clients put above their fair rates, but begin putting at different times



Evaluation – fair allocation

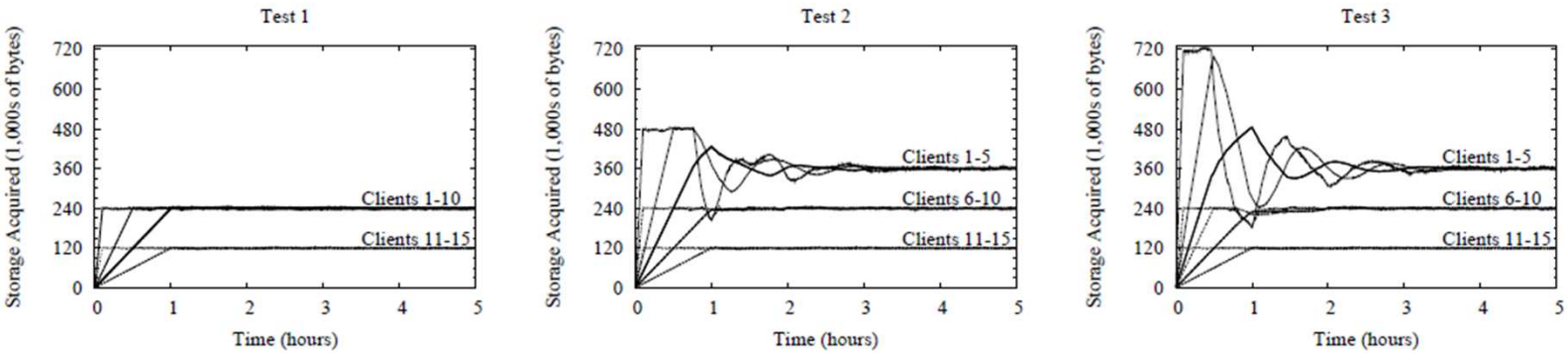


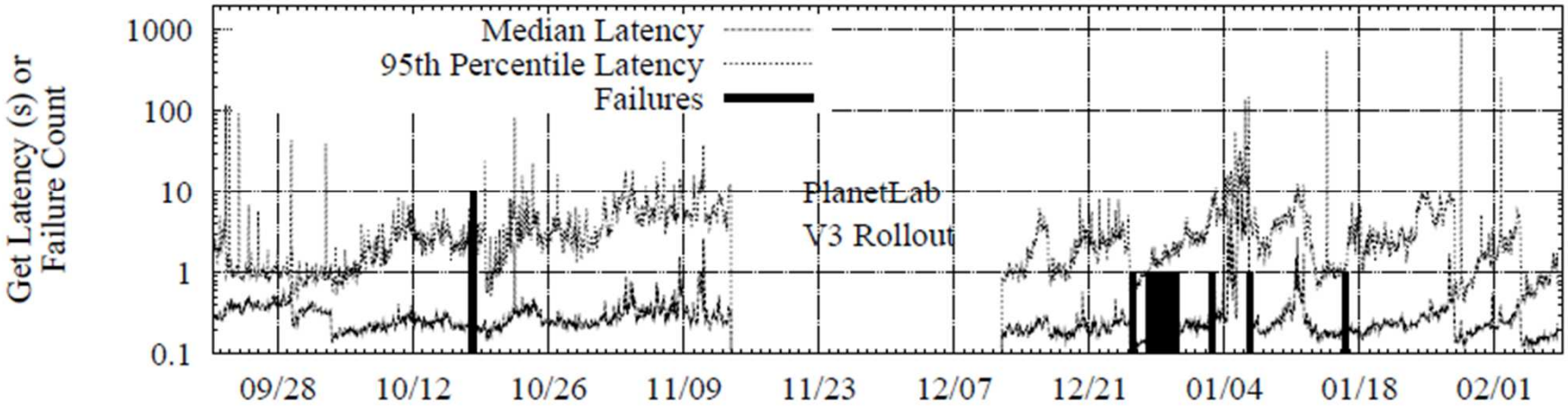
Figure 5: ***Fair allocation despite varying put sizes and TTLs.*** **See text for description.**

			Test 1				Test 2				Test 3			
Client	Size	TTL	Bid	50th	90th	Avg	Bid	50th	90th	Avg	Bid	50th	90th	Avg
1	1000	60	1.0	0	974	176	2.0	5222	10851	5126	3.0	6605	12949	6482
2	1000	30	1.0	0	0	9	2.0	7248	11554	6467	3.0	7840	13561	7364
3	1000	12	1.0	0	0	9	2.0	8404	12061	7363	3.0	8612	14173	8070
4	500	60	1.0	0	409	56	2.0	7267	11551	6490	3.0	7750	13413	7368
5	200	60	1.0	0	0	13	2.0	8371	12081	7349	3.0	8566	14125	8035
6	1000	60	1.0	0	861	163	1.0	396	1494	628	1.0	446	2088	933
7	1000	30	1.0	0	0	12	1.0	237	1097	561	1.0	281	1641	872
8	1000	12	1.0	0	0	9	1.0	221	1259	604	1.0	249	1557	940
9	500	60	1.0	0	409	63	1.0	123	926	467	1.0	187	1162	770
10	200	60	1.0	0	0	14	1.0	0	828	394	1.0	6	1822	804
11	1000	60	0.5	0	768	160	0.5	398	1182	475	0.5	444	1285	531
12	1000	30	0.5	0	0	6	0.5	234	931	320	0.5	261	899	328
13	1000	12	0.5	0	0	5	0.5	214	938	306	0.5	235	891	311
14	500	60	0.5	0	288	37	0.5	137	771	226	0.5	171	825	249
15	200	60	0.5	0	0	7	0.5	0	554	103	0.5	0	715	131

Table 3: ***Queuing times in milliseconds for each of the clients in the multiple size and TTL tests.*** **Sizes are in bytes; TTLs are in minutes. A "bid" of 1.0 indicates that a client is putting often enough to fill 1/15th of the disk in an otherwise idle system.**

Deployment

- Deployed on PlanetLab (200 hosts) for 3.5 months
- Over 9 million puts and gets each; only 28 lost values



References

OpenDHT: A Public DHT Service and Its Uses. Sean Rhea, Brighten Godfrey, Brad Karp, John Kubiatowicz, Sylvia Ratnasamy, Scott Shenker, Ion Stoica, and Harlan Yu. *Proceedings of ACM SIGCOMM 2005*, August 2005.

Ion Stoica, Daniel Adkins, Shelley Zhuang, Scott Shenker, Sonesh Surana, "Internet Indirection Infrastructure," *Proceedings of ACM SIGCOMM*, August, 2002.

G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels. Dynamo: amazon's highly available keyvalue store. *Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles*, pages 205–220, 2007.

The Design and Implementation of a Next Generation Name Service for the Internet. Venugopalan Ramasubramanian and Emin Gun Sirer. *Proceedings of SIGCOMM*, Portland, Oregon, August 2004.