Fixing the Embarrassing Slowness of OpenDHT on PlanetLab

Sean Rhea, Byung-Gon Chun,
John Kubiatowicz, and Scott Shenker
UC Berkeley (and now MIT)

December 13, 2005

Distributed Hash Tables (DHTs)

- Same interface as a traditional hash table
  - put(*key*, *value*) — stores *value* under *key*
  - get(*key*) — returns all the values stored under *key*
- Built over a distributed overlay network
  - Partition key space over available nodes
  - Route each put/get request to appropriate node

DHTs: The Hype

- High availability
  - Each key-value pair replicated on multiple nodes
- Incremental scalability
  - Need more storage/tput? Just add more nodes.
- Low latency
  - Recursive routing, proximity neighbor selection, server selection, etc.

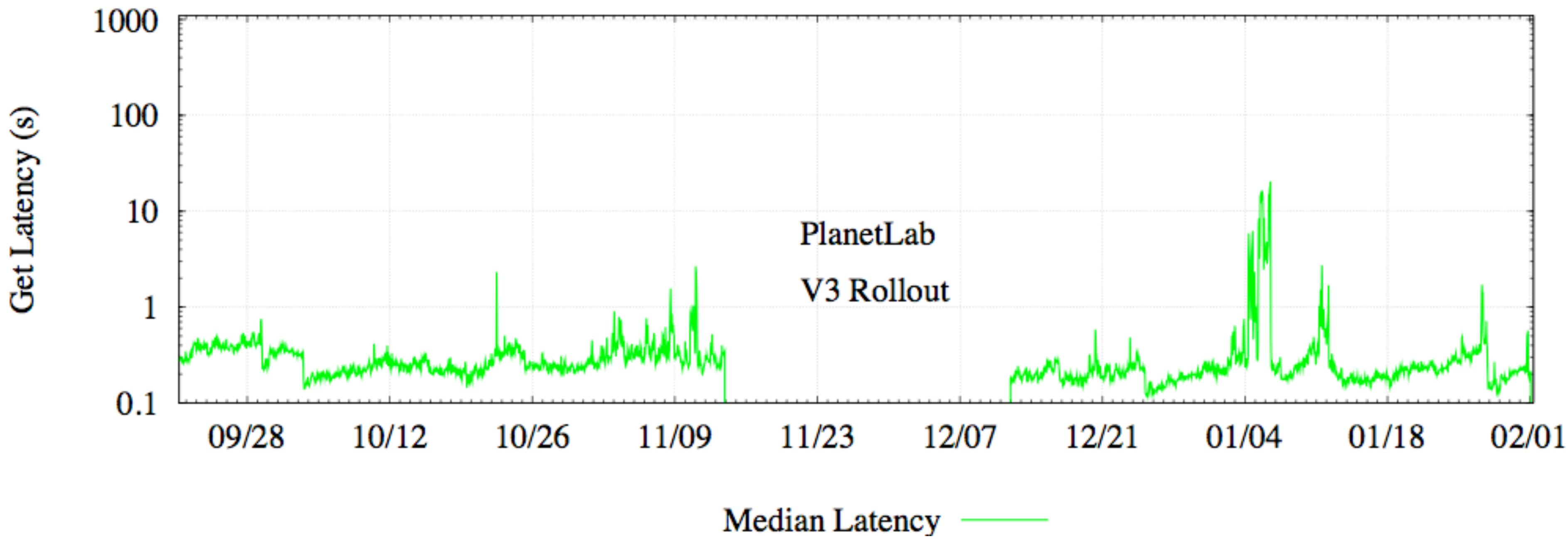
DHTs: The Hype

- Promises of DHTs realized only “in the lab”
  - Use isolated network (Emulab, ModelNet)
  - Measure while PlanetLab load is low
  - Look only at median performance
- Our goal: make DHTs perform “in the wild”
  - Network not isolated, machines shared
  - Look at long term 99th percentile performance
  - (Caveat: no outright malicious behavior)

Why We Care

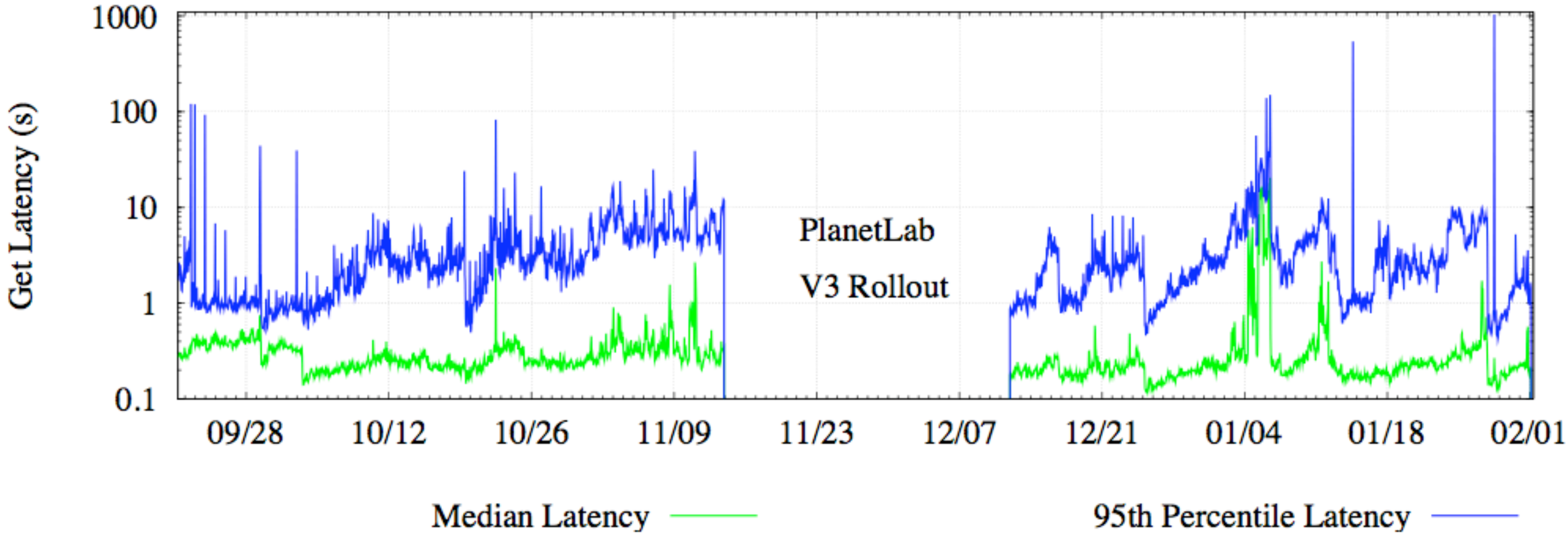
- Promise of P2P was to harness idle capacity
  - Not supposed to need dedicated machines
- Running OpenDHT service on PlanetLab
  - No control over what else is running
  - Load can be really bad at times
  - Up 24/7: have to weather good times and bad
  - Good median performance isn't good enough

Original OpenDHT Performance



- Long-term median get latency < 200 ms
  - Matches performance of DHASH on PlanetLab
  - Median RTT between hosts ~ 140 ms

Original OpenDHT Performance

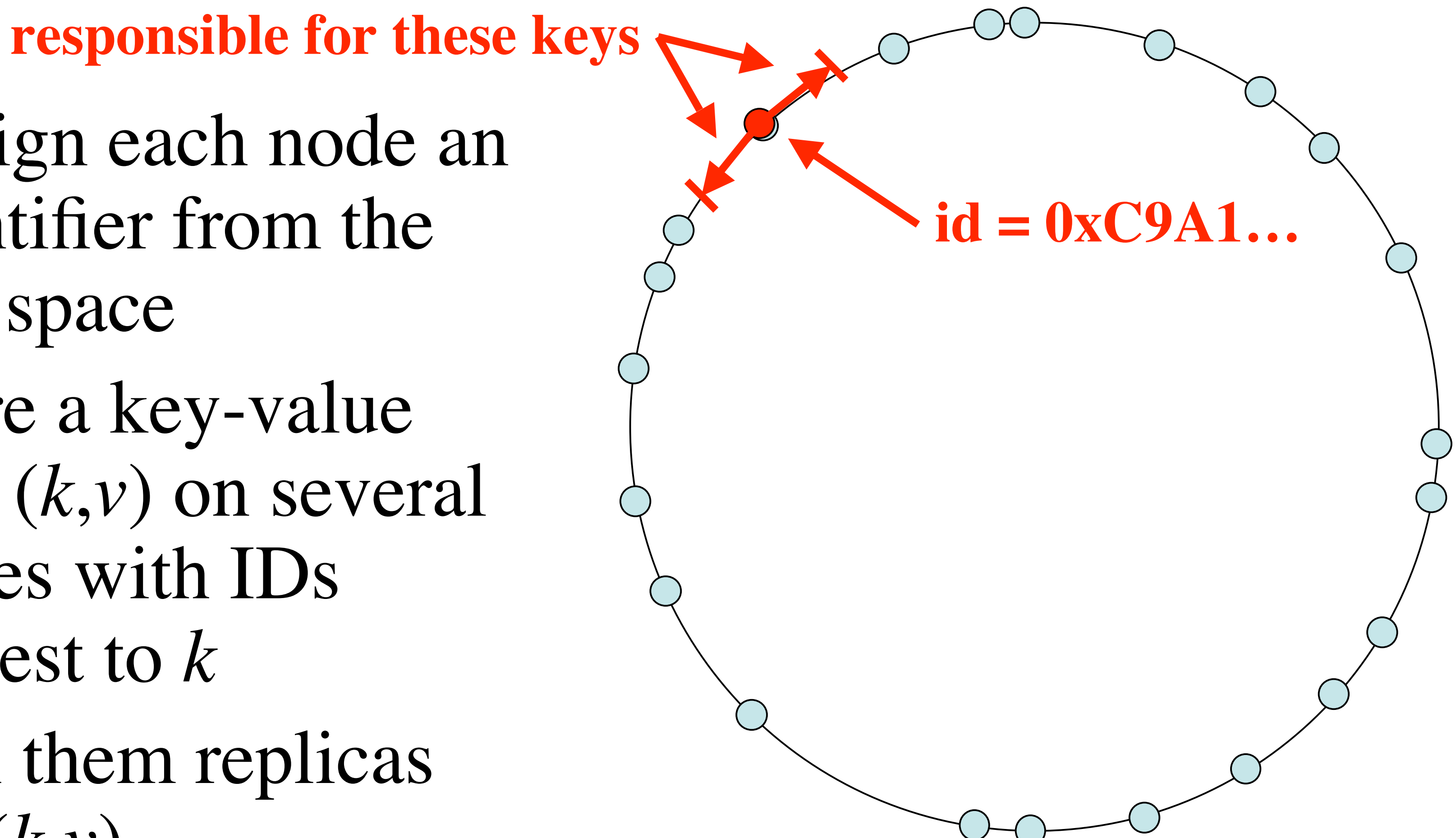


- But 95th percentile get latency is atrocious!
  - Generally measured in *seconds*
  - And even median spikes up from time to time

Talk Overview

- Introduction and Motivation
- How OpenDHT Works
- The Problem of Slow Nodes
- Algorithmic Solutions
- Experimental Results
- Related Work and Conclusions

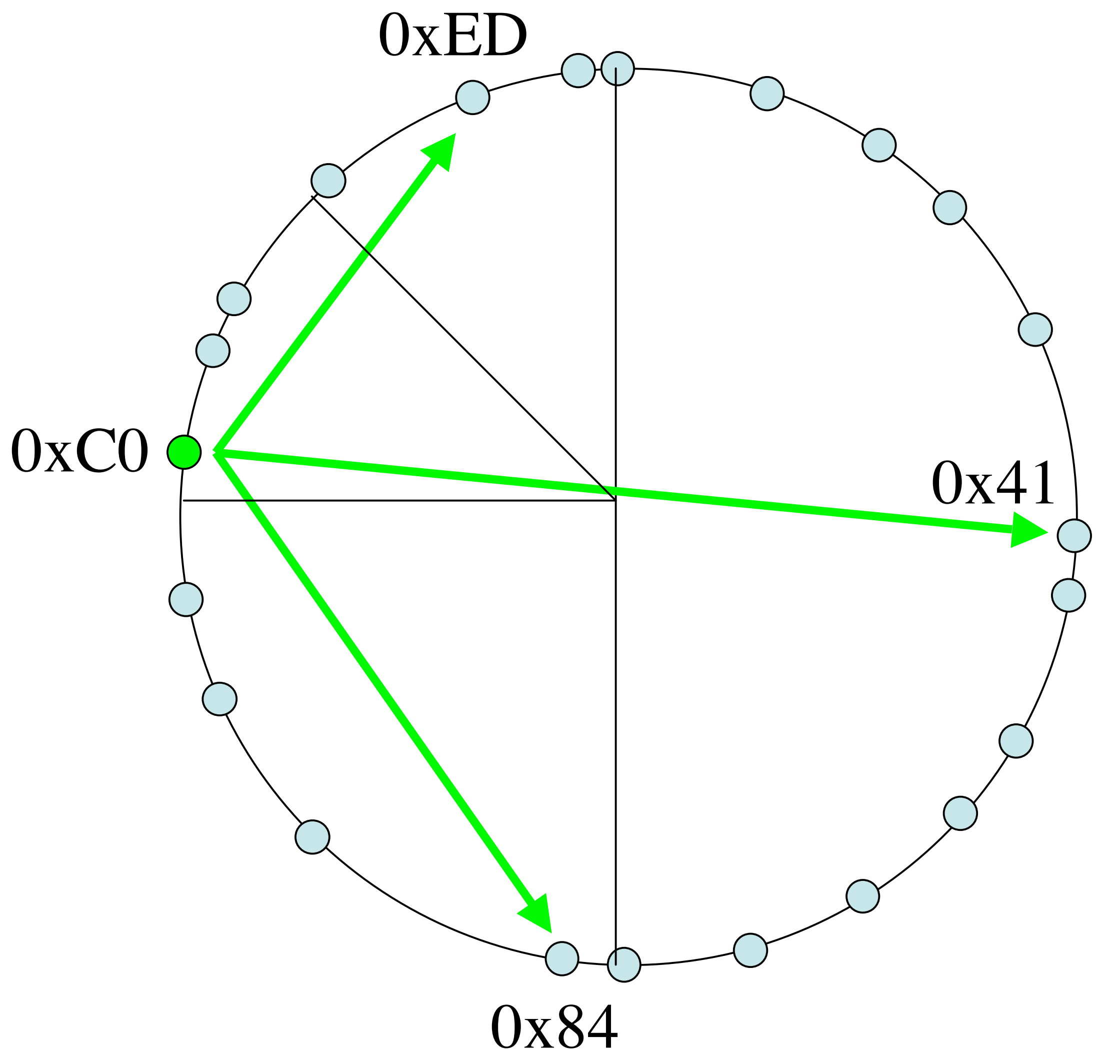
OpenDHT Partitioning



- Assign each node an identifier from the key space
- Store a key-value pair (*k*,*v*) on several nodes with IDs closest to *k*
- Call them replicas for (*k*,*v*)

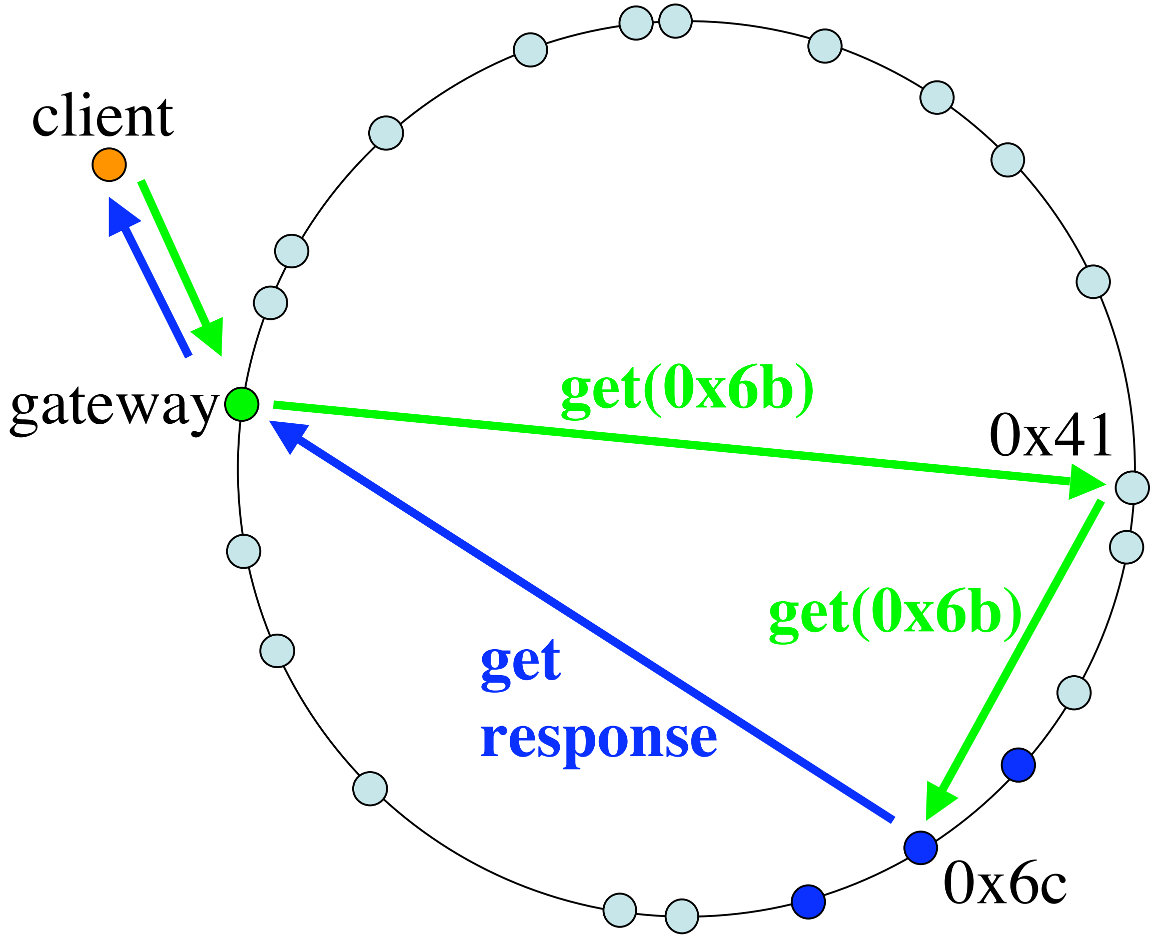
OpenDHT Graph Structure

- Overlay neighbors match prefixes of local identifier
- Choose among nodes with same matching prefix length by network latency



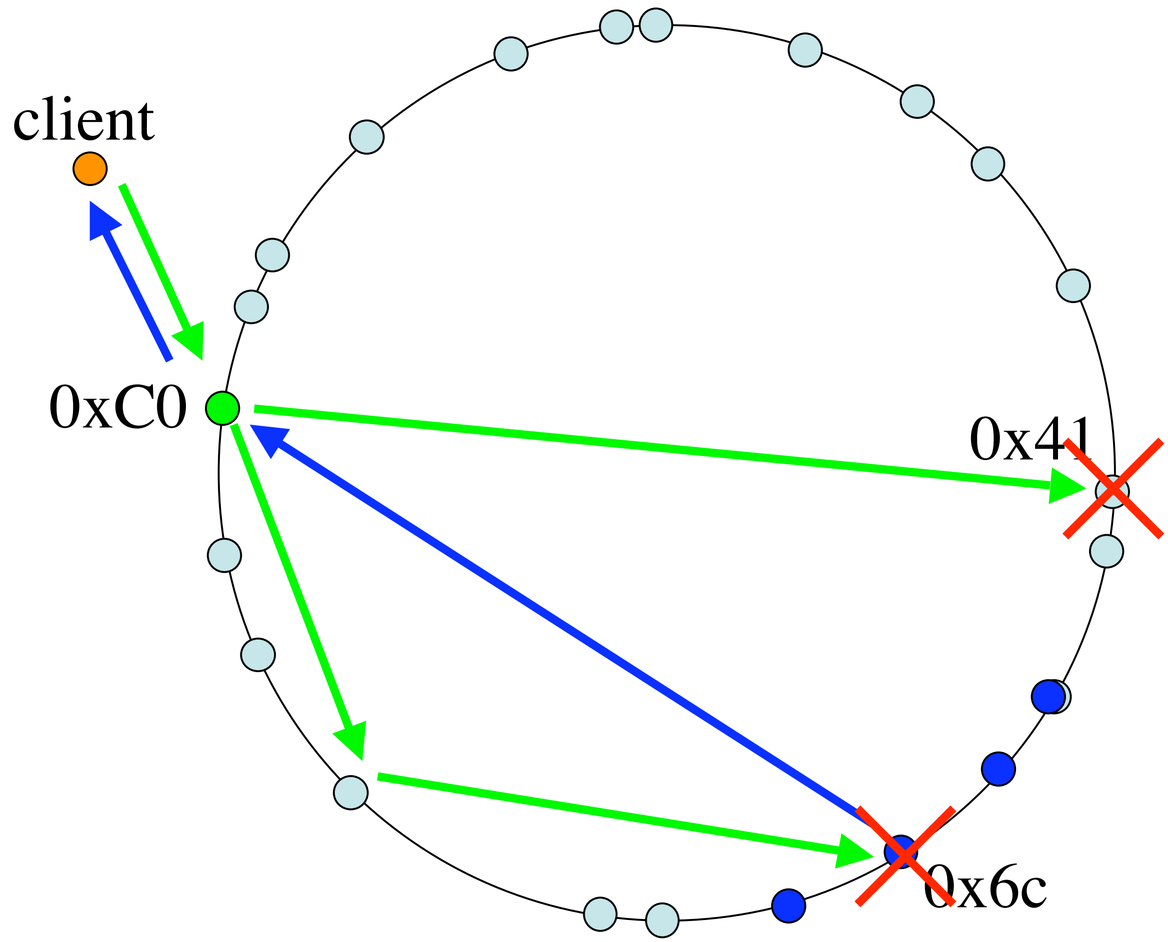
Performing Gets in OpenDHT

- Client sends a get request to gateway
- Gateway routes it along neighbor links to first replica encountered
- Replica sends response back directly over IP



Robustness Against Failure

- If a neighbor dies, a node routes through its next best one
- If replica dies, remaining replicas create a new one to replace it



The Problem of Slow Nodes

- What if a neighbor doesn't fail, but just slows down temporarily?
  - If it stays slow, node will replace it
  - But must adapt slowly for stability
- Many sources of slowness are short-lived
  - Burst of network congestion causes packet loss
  - User loads huge Photoshop image, flushing buffer cache
- In either case, gets will be delayed

Flavors of Slowness

- At first, slowness may be unexpected
  – May not notice until try to route through a node
  – First few get requests delayed
- Can keep history of nodes' performance
  – Stop subsequent gets from suffering same fate
  – Continue probing slow node for recovery

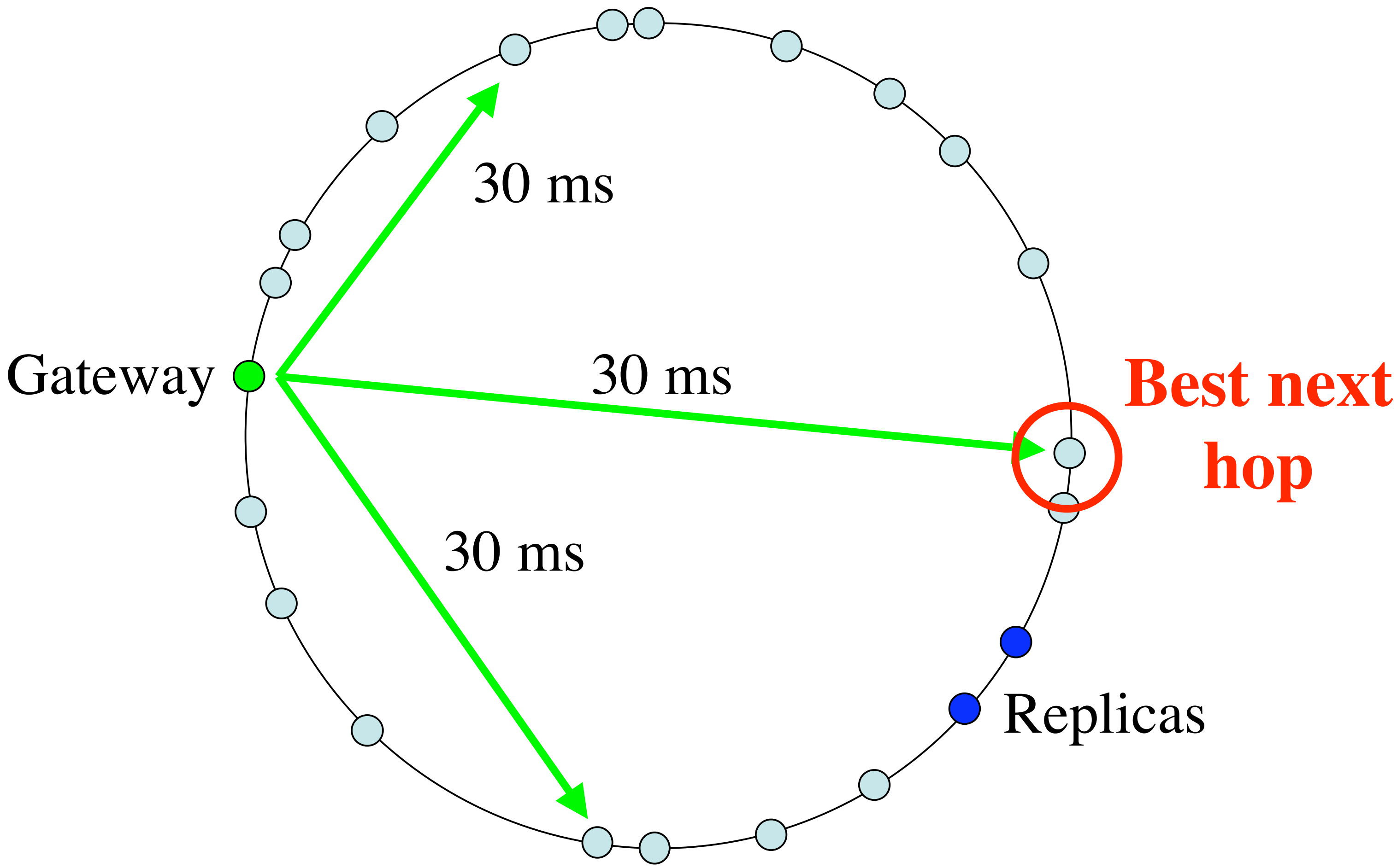
Talk Overview

- Introduction and Motivation
- How OpenDHT Works
- The Problem of Slow Nodes
- Algorithmic Solutions
- Experimental Results
- Related Work and Conclusions

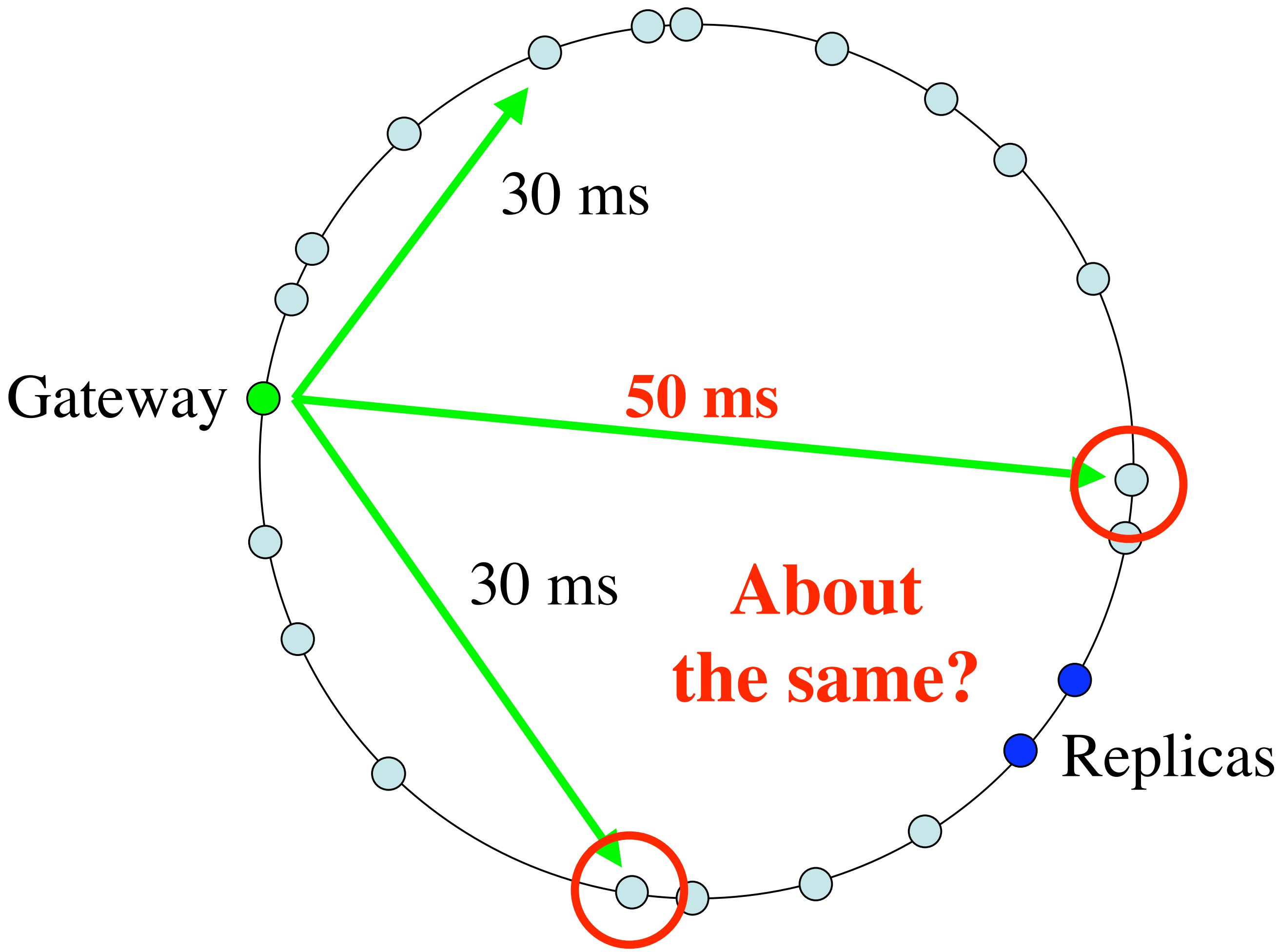
Two Main Techniques

- Delay-aware routing
  - Guide routing not just by progress through key space, but also by past responsiveness

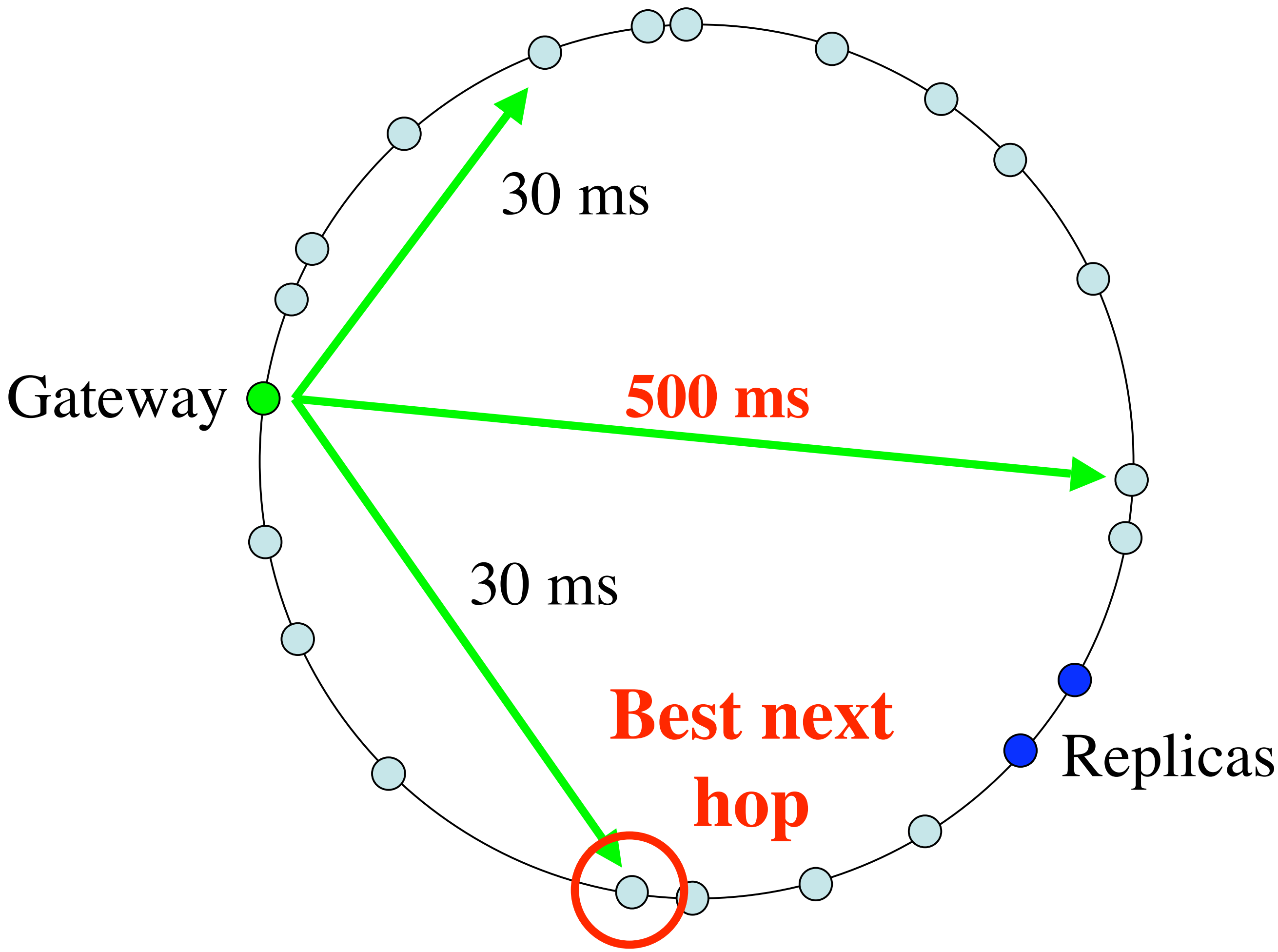
Delay-Aware Routing



Delay-Aware Routing



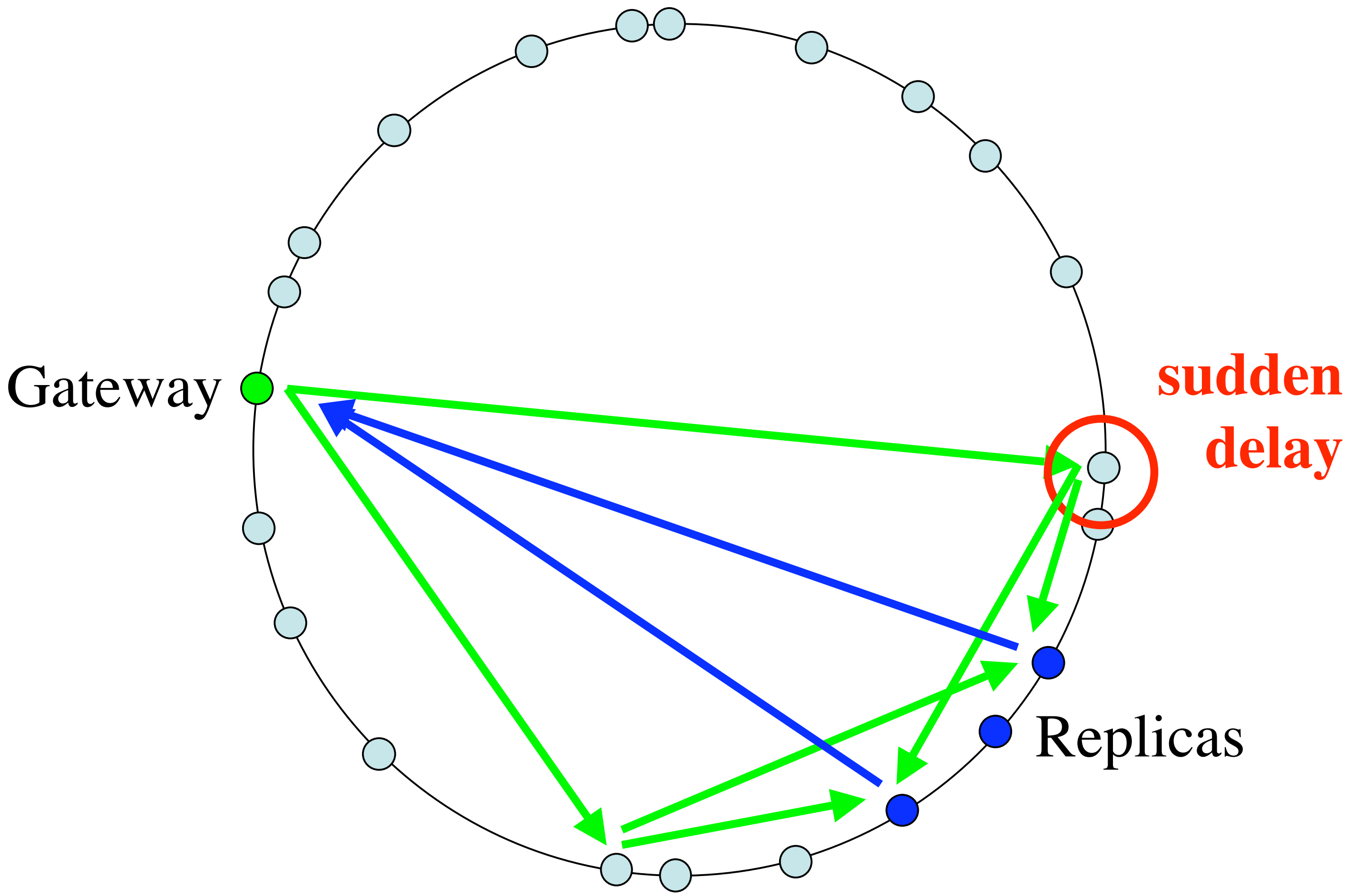
Delay-Aware Routing



Two Main Techniques

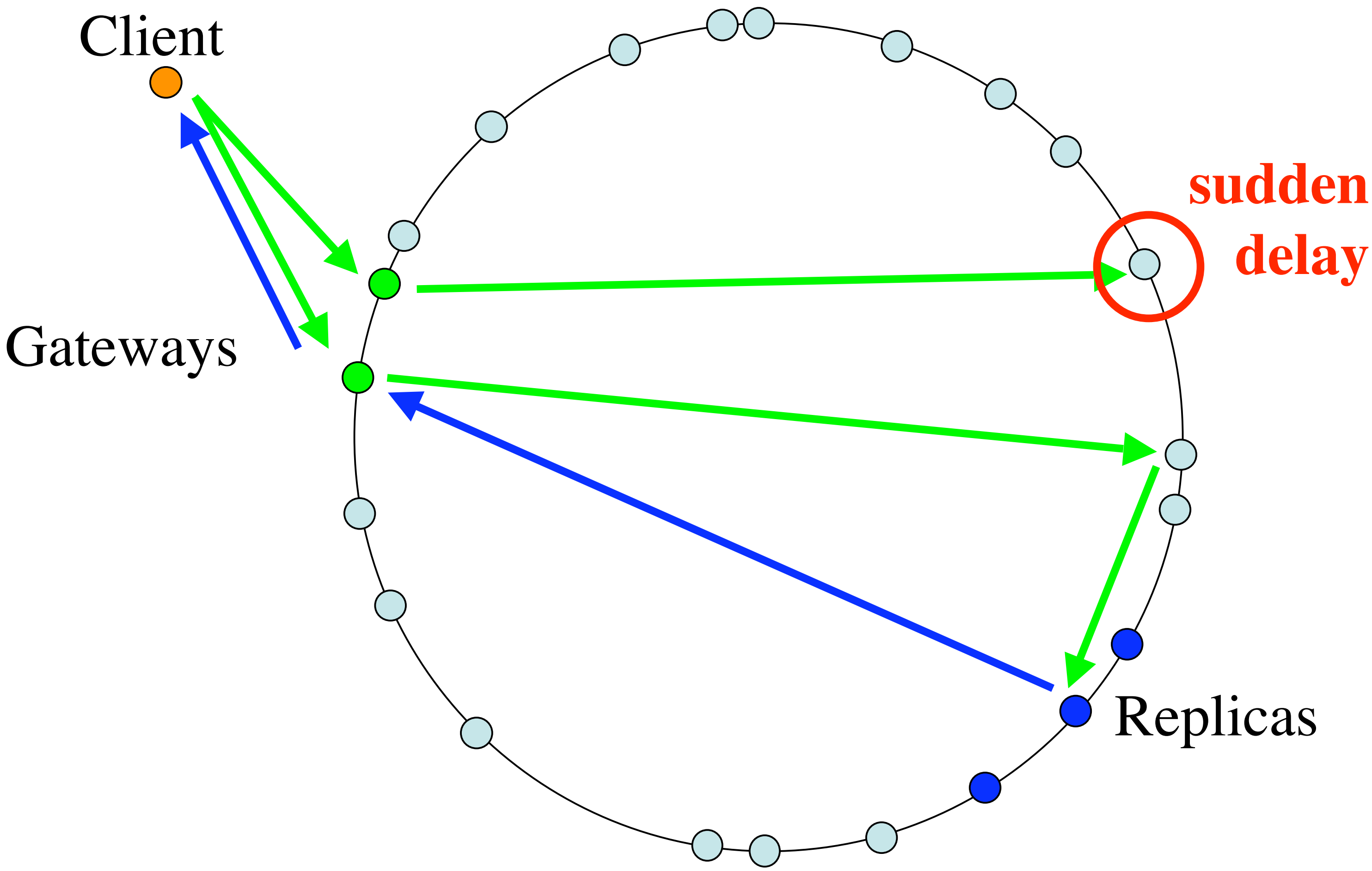
- Delay-aware routing
  - Guide routing not just by progress through key space, but also by past responsiveness
  - Cheap, but must first observe slowness
- Added parallelism
  - Send each request along multiple paths

Naïve Parallelism



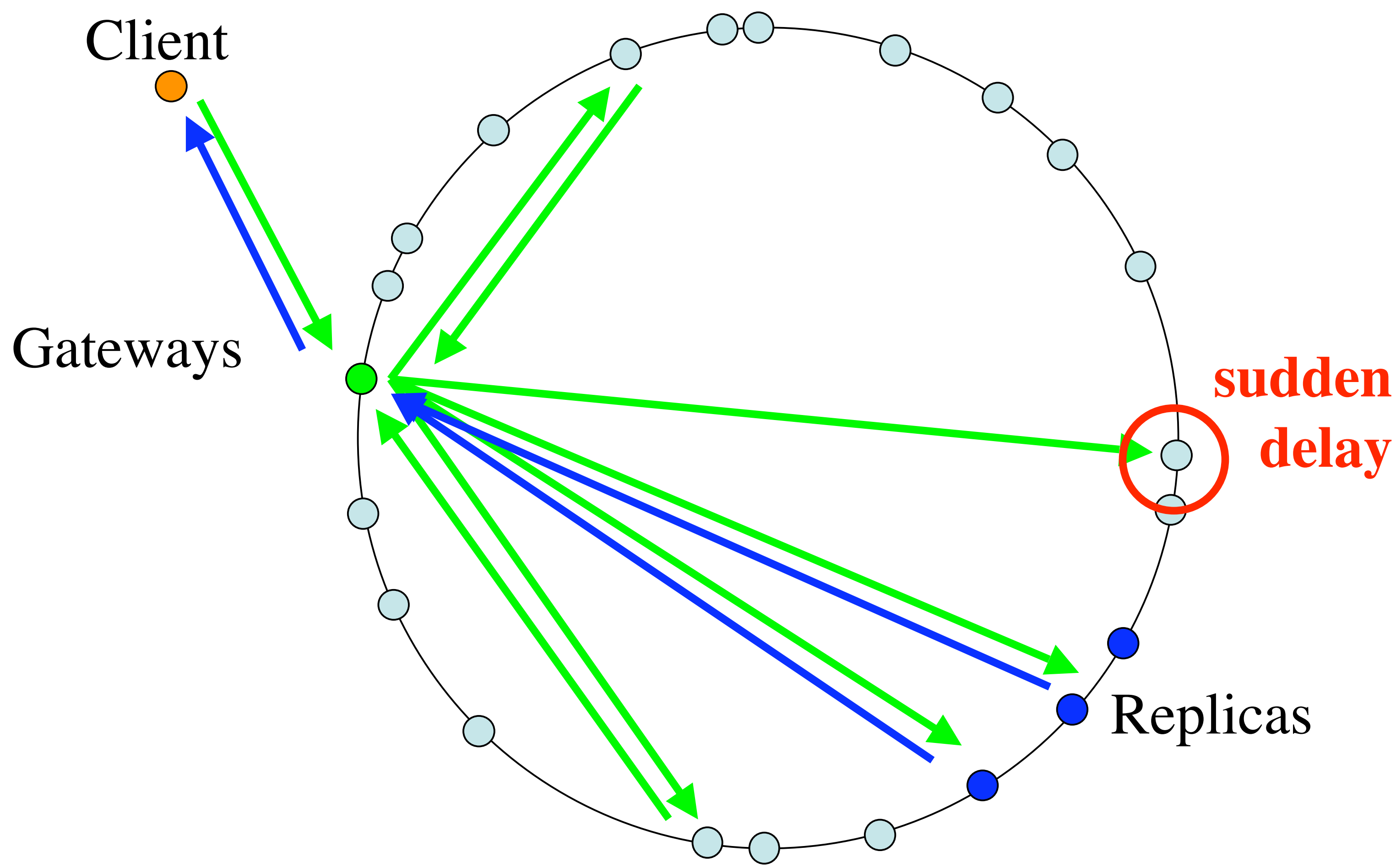
Multiple Gateways

(Only client replicates requests.)



Iterative Routing

(Gateway maintains *p* concurrent RPCs.)



Two Main Techniques

- Delay-aware routing
  - Guide routing not just by progress through key space, but also by past responsiveness
  - Cheap, but must first observe slowness
- Added parallelism
  - Send each request along multiple paths
  - Expensive, but handles unexpected slowness

Talk Overview

- Introduction and Motivation
- How OpenDHT Works
- The Problem of Slow Nodes
- Algorithmic Solutions
- Experimental Results
- Related Work and Conclusions

Experimental Setup

- Can’t get reproducible numbers from PlanetLab
  - Both available nodes and load change hourly
  - But PlanetLab is the environment we care about
- Solution: run all experiments concurrently
  - Perform each get using every mode (random order)
  - Look at results over long time scales:
    6 days; over 27,000 samples per mode

Delay-Aware Routing

Mode	Latency (ms) 50th	Latency (ms) 99th	Cost Msgs	Cost Bytes
Greedy	150	4400	5.5	1800
Delay-Aware	100	1800	6.0	2000

- Latency drops by 30-60%
- Cost goes up by only ~10%

Multiple Gateways

# of Gateways	Latency (ms)		Cost	
	50th	99th	Msgs	Bytes
1	100	1800	6.0	2000
2	70	610	12	4000
3	57	440	17	5300

- Latency drops by a further 30-73%
- But cost doubles or worse

Iterative Routing

# of Gateways	Mode	50th	99th	Cost Msgs	Cost Bytes
1	Recursive	100	1800	6.0	2000
3	Recursive	57	440	17	5300
1	3-way Iterative	120	790	15	3800
2	3-way Iterative	76	360	27	6700

- Parallel iterative not as cost effective as just using multiple gateways

Talk Overview

- Introduction and Motivation
- How OpenDHT Works
- The Problem of Slow Nodes
- Algorithmic Solutions
- Experimental Results
- Related Work and Conclusions

Related Work

- Google MapReduce
  - Cluster owned by single company
  - Could presumably make all nodes equal
  - Turns out it's cheaper to just work around the slow nodes instead
- Accordion
  - Another take on recursive parallel lookup
- Other related work in paper

Conclusions

- Techniques for reducing get latency
  - Delay-aware routing is a clear win
  - Parallelism very fast, but costly
  - Iterative routing not cost effective

- OpenDHT get latency is now quite low
  - Was 150 ms on median, 4+ seconds on 99th
  - Now under 100 ms on median, 500 ms on 99th
  - Faster than DNS [Jung et al. 2001]

Thanks!

For more information:
http://opendht.org/