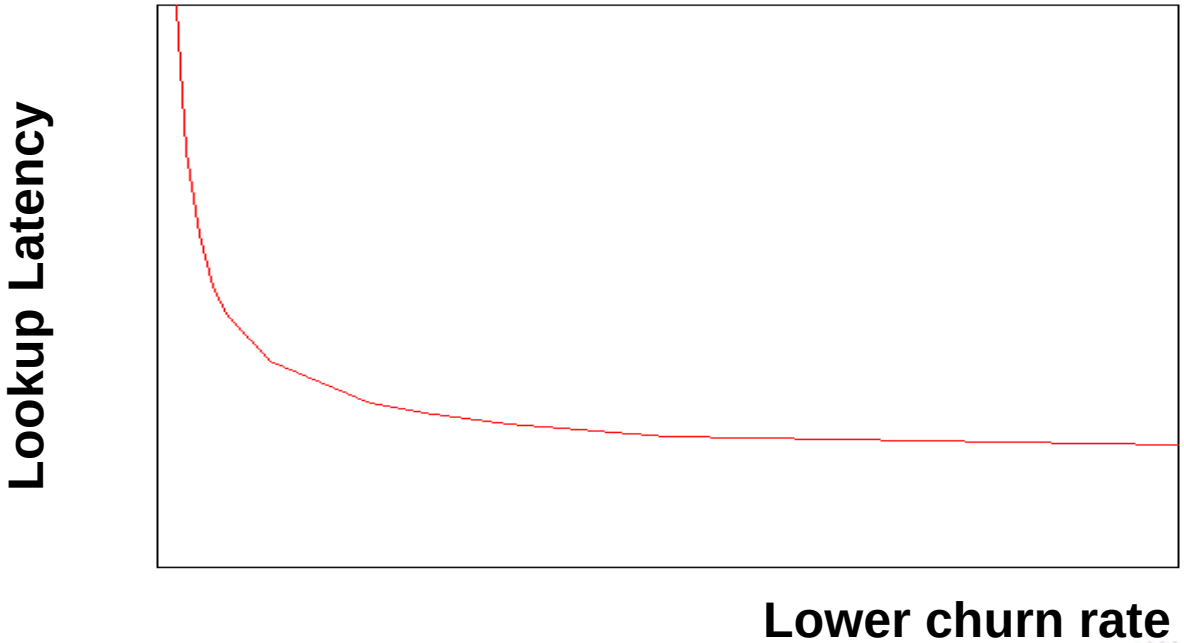
Bandwidth Efficient Management of DHT Routing Tables

Jinyang Li, Jeremy Stribling,
Robert Morris, Frans Kaashoek

DHT designs reflect churn assumptions

- Chord (Kademlia, Tapestry, Pastry)
  - ✓ small routing table, low maintenance traffic
  - ✗ O(log N) lookup hops
  - – Good choice for high churn
- OneHop
  - ✗ large routing table, high maintenance traffic
  - ✓ O(1) lookup hop
  - – Good choice for low churn

An ideal DHT would be adaptive



- Node access link capacity is the main limit
- DHT must use link efficiently for given churn

Accordion Design

- Goal: routing table that minimizes latency
- Use b/w budget to search for new nodes
  - To reduce average lookup hops
- Evict nodes likely to be dead
  - To reduce lookup timeouts
- Table size is determined by the equilibrium of acquisition and eviction process

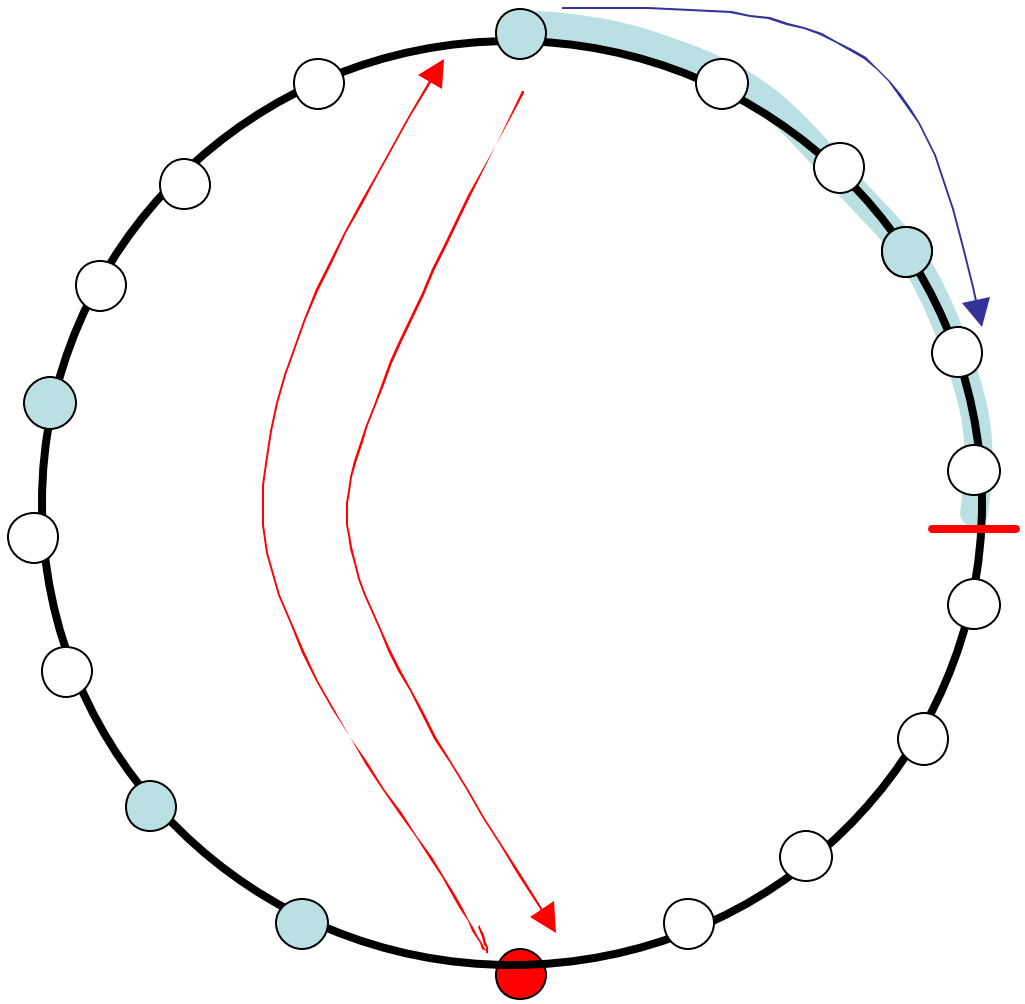
How to use the bandwidth budget

- Use as much b/w as possible in lookups
  - Piggyback “learning” info on every lookup
  - Lookup in parallel to learn and avoid timeouts
- Use remaining b/w to actively search for nodes.

Routing state distribution

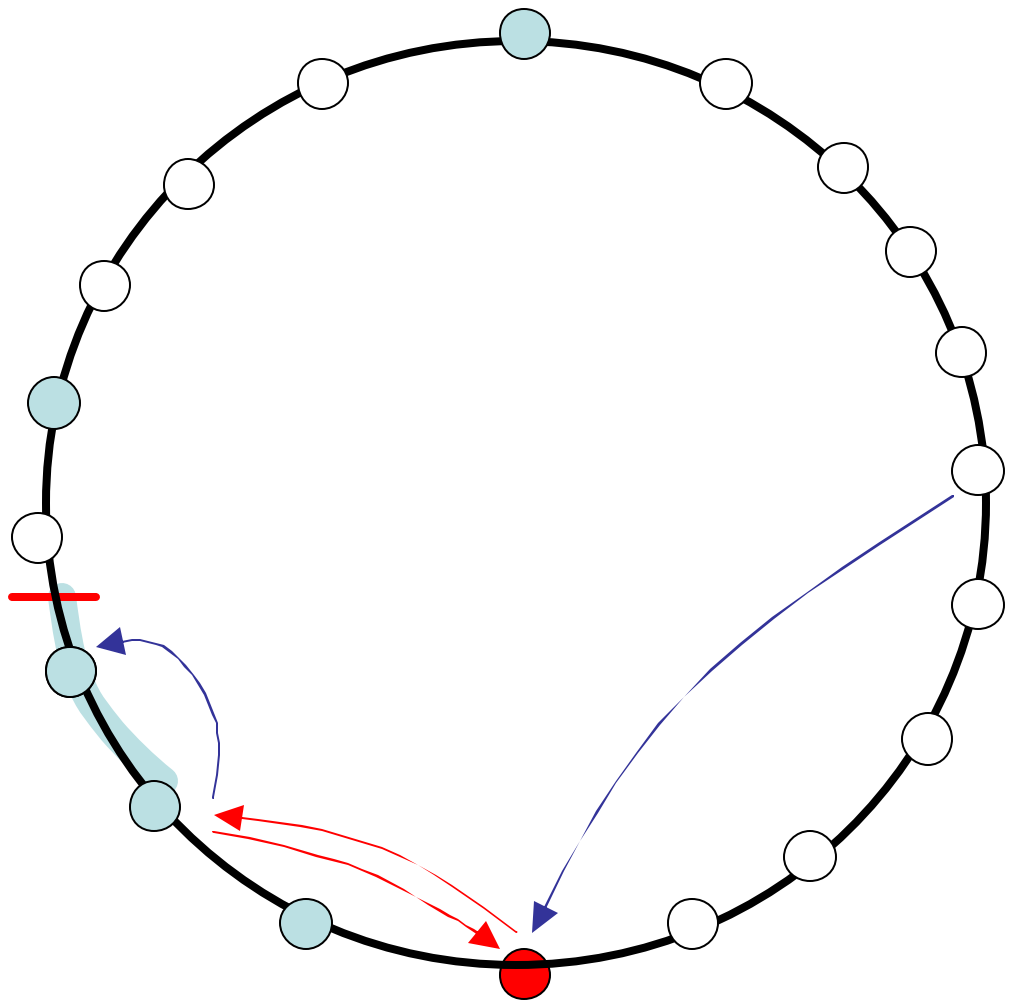
- Keep all nodes you learn about
- Guide active search to get 1/x distribution
  - i.e. you know more nodes closer in ID space
  - Yields log(n) lookup hops
  - Any node can be in the routing table
  - Allows non-quantized table sizes

Learning with 1/x distribution



- Most recursive lookup hops are short, at the end
- Nodes learn from next hops in the 1/x distribution

Learning with 1/x distribution



- Most recursive lookup hops are short, at the end
- Nodes learn from next hops in the 1/x distribution

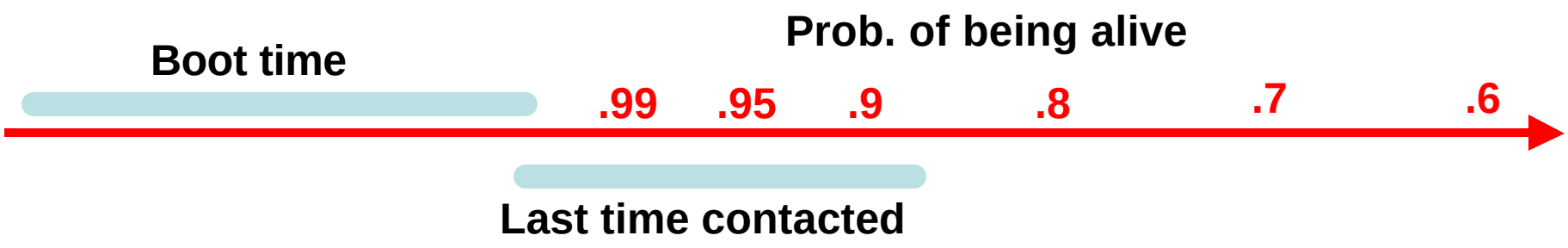
Avoiding lookup timeouts

- Evict nodes that are likely dead
- Issue parallel recursive lookups

How to decide if a node might be dead?

- Too expensive to ping frequently.
- Assume nodes that are up for a long time are more likely to remain up.
- Node learn *boot time* and *last time contacted* in addition to ID.

How to decide if a node might be dead?

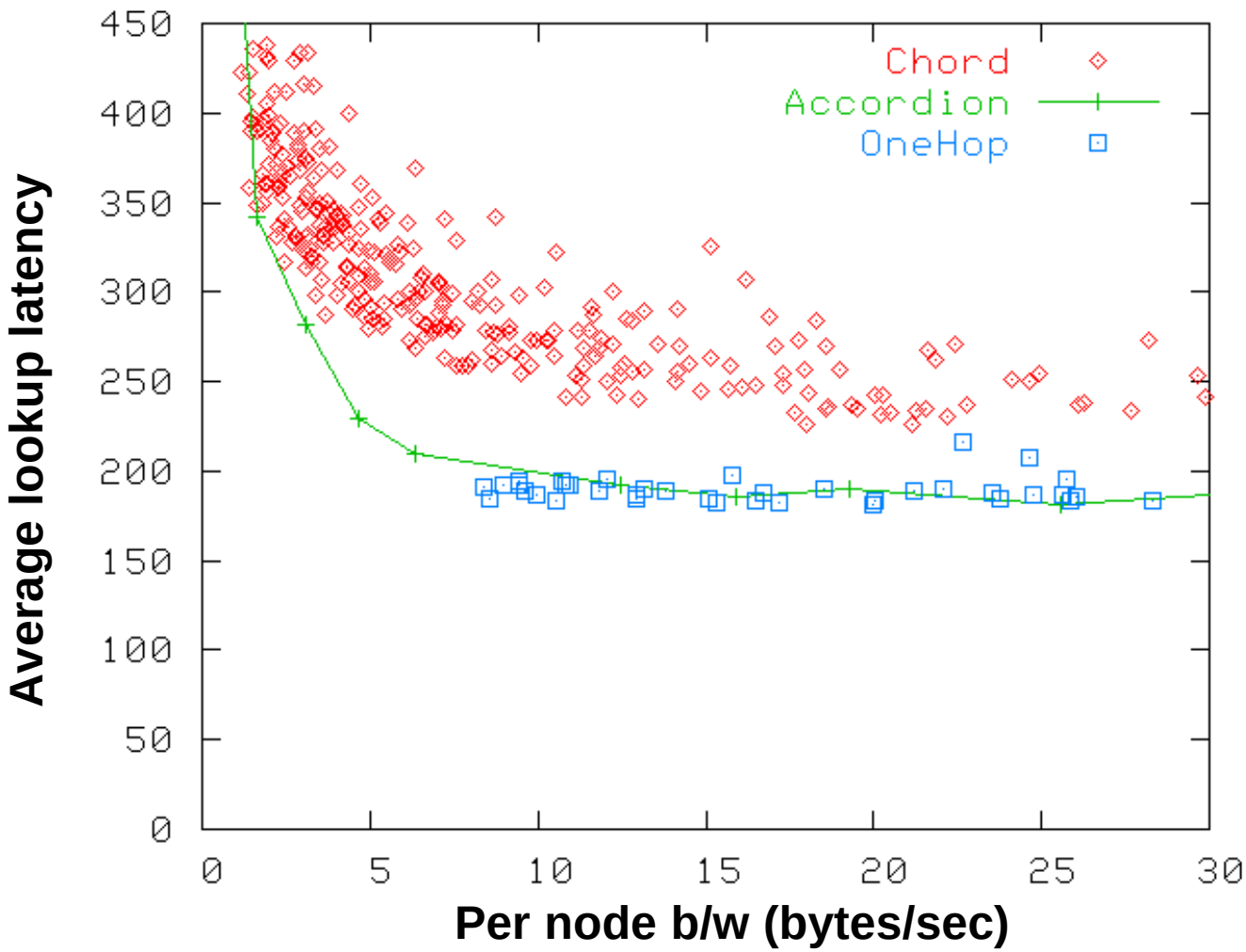


- Estimate a neighbor's prob. of being alive based on *boot time* and *last time contacted*
- Choose a eviction threshold based on lookup parallelism such that the prob. of timeout < .1

Evaluations

- Simulation scenario:
  - 1024 nodes
  - latency matrix of 1024 DNS server measurements.
  - Pareto up/down time distribution with median 1 hour.
  - Each node looks up a random key every 10 minutes.

Accordion is b/w efficient



Conclusions

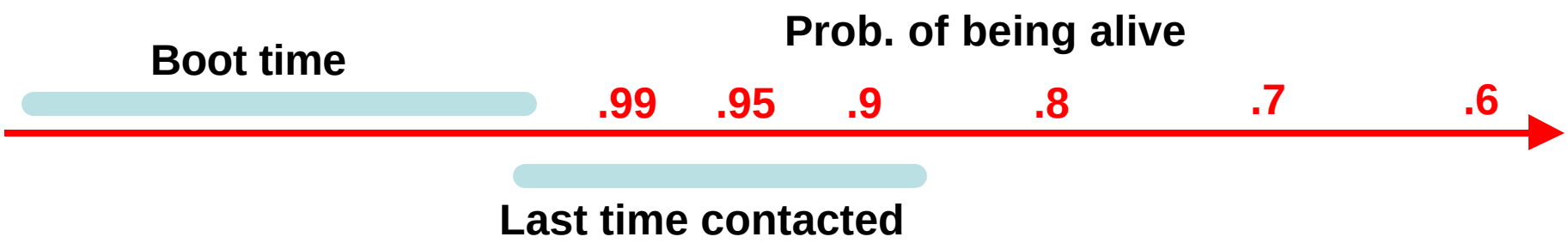
- Accordion adjusts routing table size according to churn
- Accordion can efficiently allocate b/w budget to maintain its routing table

How to use the bandwidth budget

- Use as much b/w as possible in lookups
  - Piggyback “learning” info on every lookup
  - Look in parallel to learn and avoid timeouts
- Use remaining b/w to actively search for nodes according to 1/x distribution.
  - i.e. you know more nodes closer in ID space
  - Yields log(n) lookup hops
  - Any node can be in the routing table
  - Allows non-quantized table sizes
  - Learning from lookup approximates 1/x distribution

How to decide if a node might be dead?

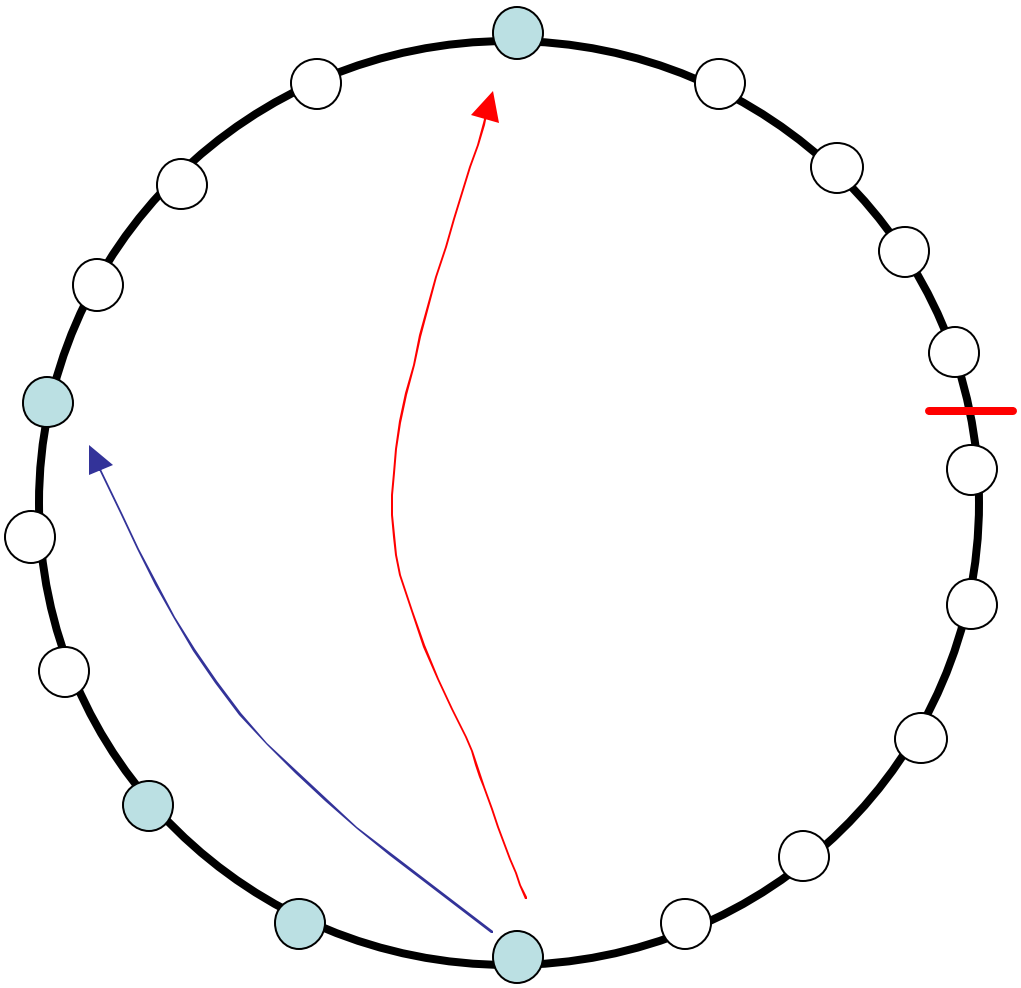
- Too expensive to ping frequently
- Assume nodes that are up for a long time are more likely to remain up
- Node learn *boot time* and *last time contacted* in addition to ID



- Choose a eviction threshold based on lookup parallelism such that the prob. of timeout < .1

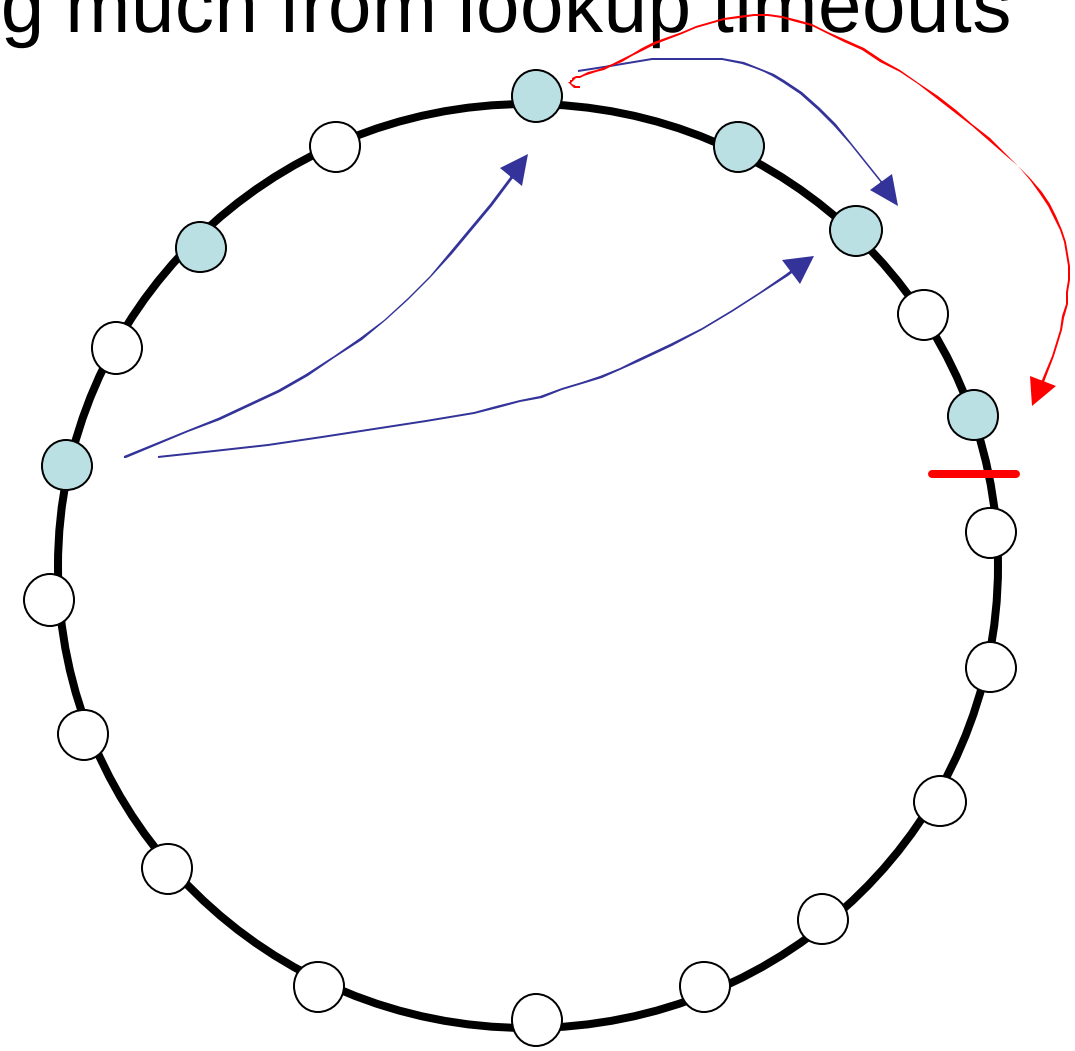
Parallelizing recursive lookups

- Allows a lower timeout threshold of eviction.



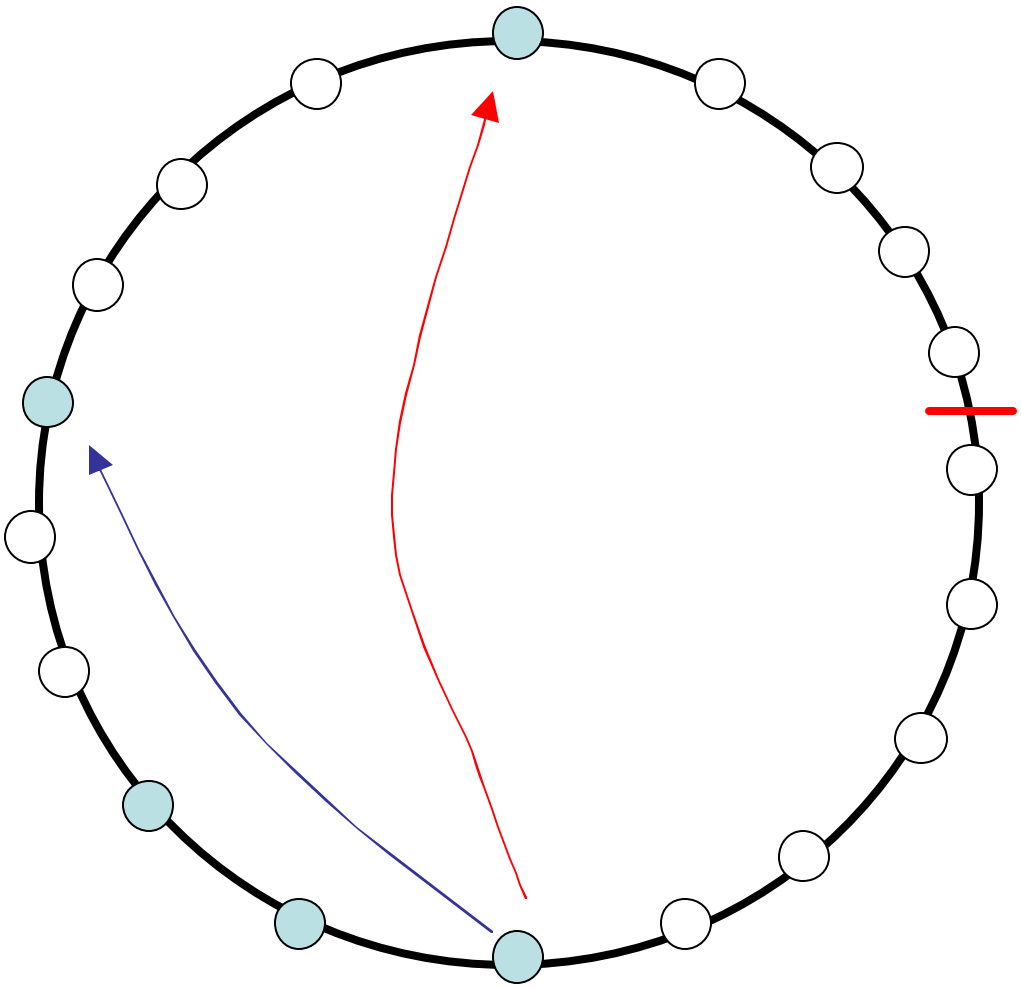
Parallelizing recursive lookups

- Increase effective routing table size without suffering much from lookup timeouts



Parallelizing recursive lookups

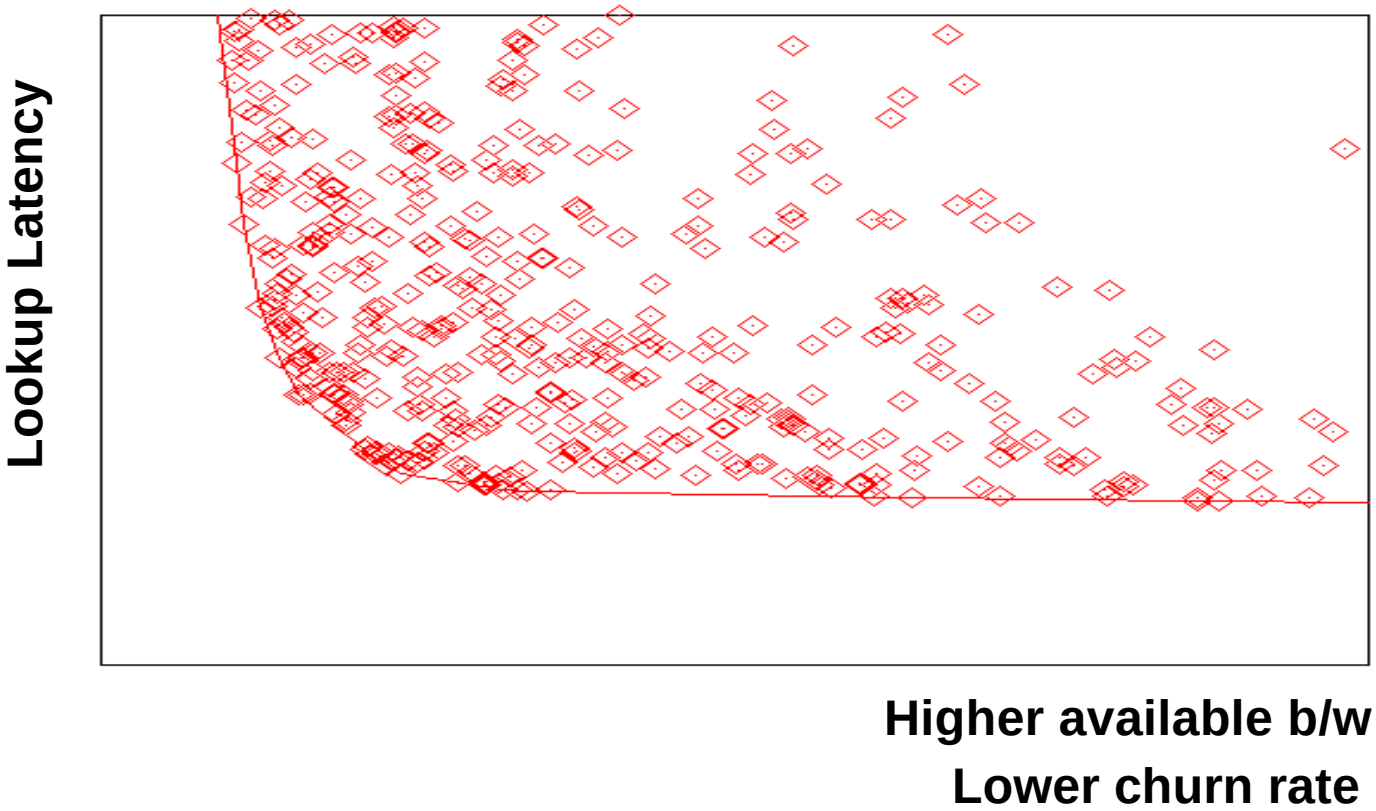
- Reduce lookup timeouts.



Exploring routing tables

- When there’s no user lookup, a node fills its b/w budget by exploring new neighbors.
- No multi-hop lookups. A node asks existing neighbors for new nodes that fill the biggest gap scaled by 1/x.

An ideal DHT is b/w efficient



- Existing protocols' b/w efficiency depends on how well parameters are tuned

Accordion is b/w efficient