EpiChord: Parallelizing the Chord Lookup Algorithm with Reactive Routing State Management

Ben Leong, Barbara Liskov, and Eric D. Demaine

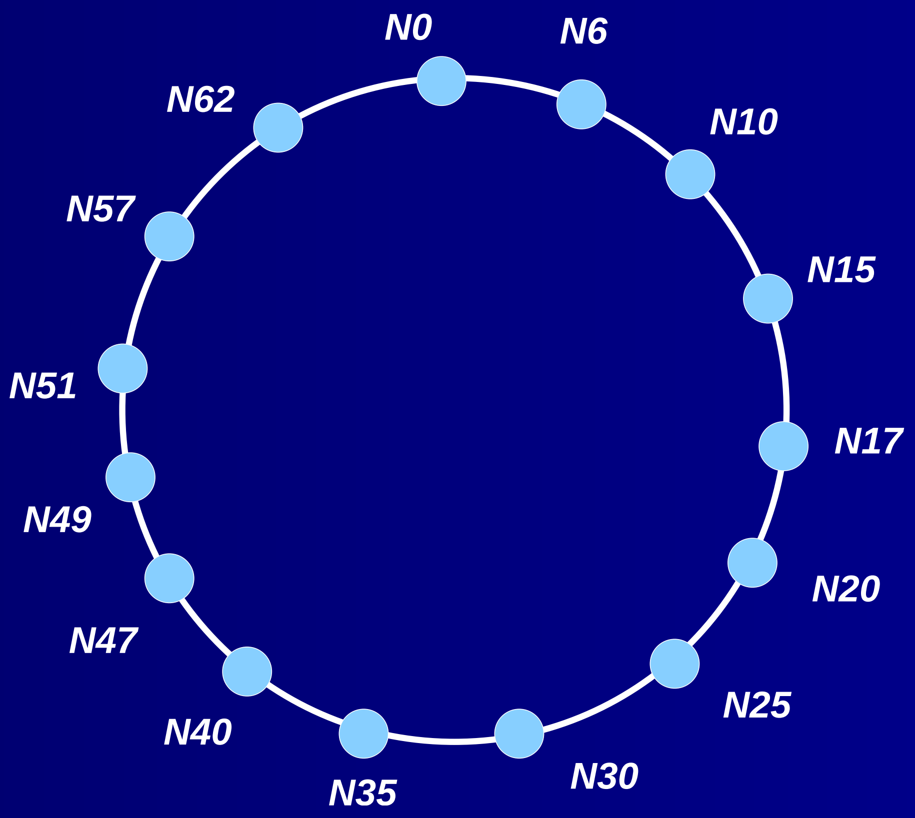
MIT Computer Science and Artificial Intelligence Laboratory

`{benleong, liskov, edemaine}@mit.edu`

Structured Peer-to-Peer Systems

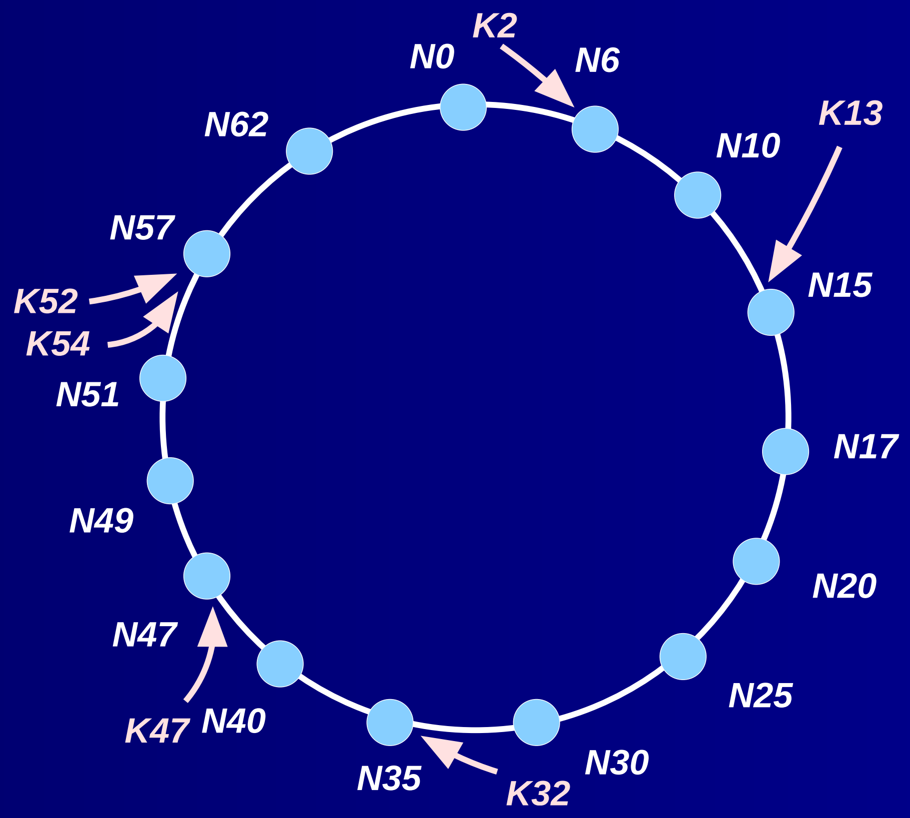
- Large scale dynamic network
- Overlay infrastructure :
  - Scalable
  - Self configuring
  - Fault tolerant
- Every node responsible for some objects
- Find node having desired object
- Challenge: Efficient Routing at Low Cost

Address Space



- Most common — one-dimensional circular address space

Mapping Keys to Nodes



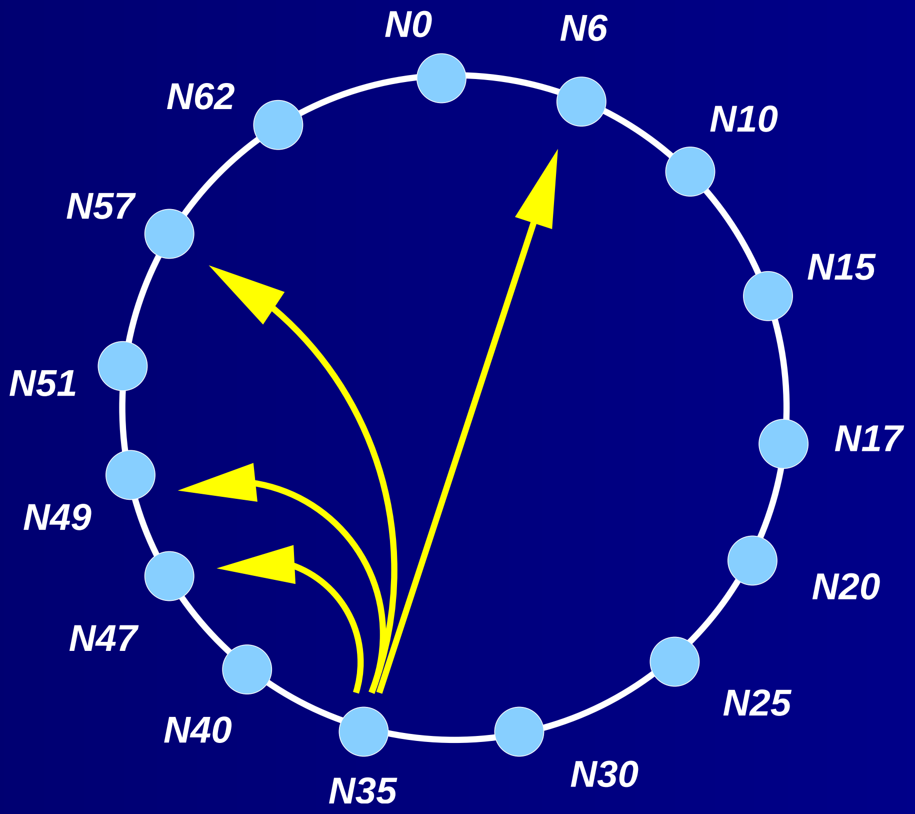
- successor of key is its owner

Distributed Hash Tables (DHTs)

- A Distributed Hash Table (DHT) is a distributed data structure that supports a *put/get* interface.
- Store and retrieve {key, value} pairs efficiently over a network of (generally unreliable) nodes
- Keep state stored per node small because of network churn $\Rightarrow$ minimize book-keeping & maintenance traffic

$\Rightarrow$ EpiChord explores the trade-offs in moving from sequential lookup to parallel lookup and from $O(\log n)$ to $O(\log n)++$ state

Chord



- Each node periodically probes $O(\log n)$ fingers
- Achieves $O(\log n)$-hop performance

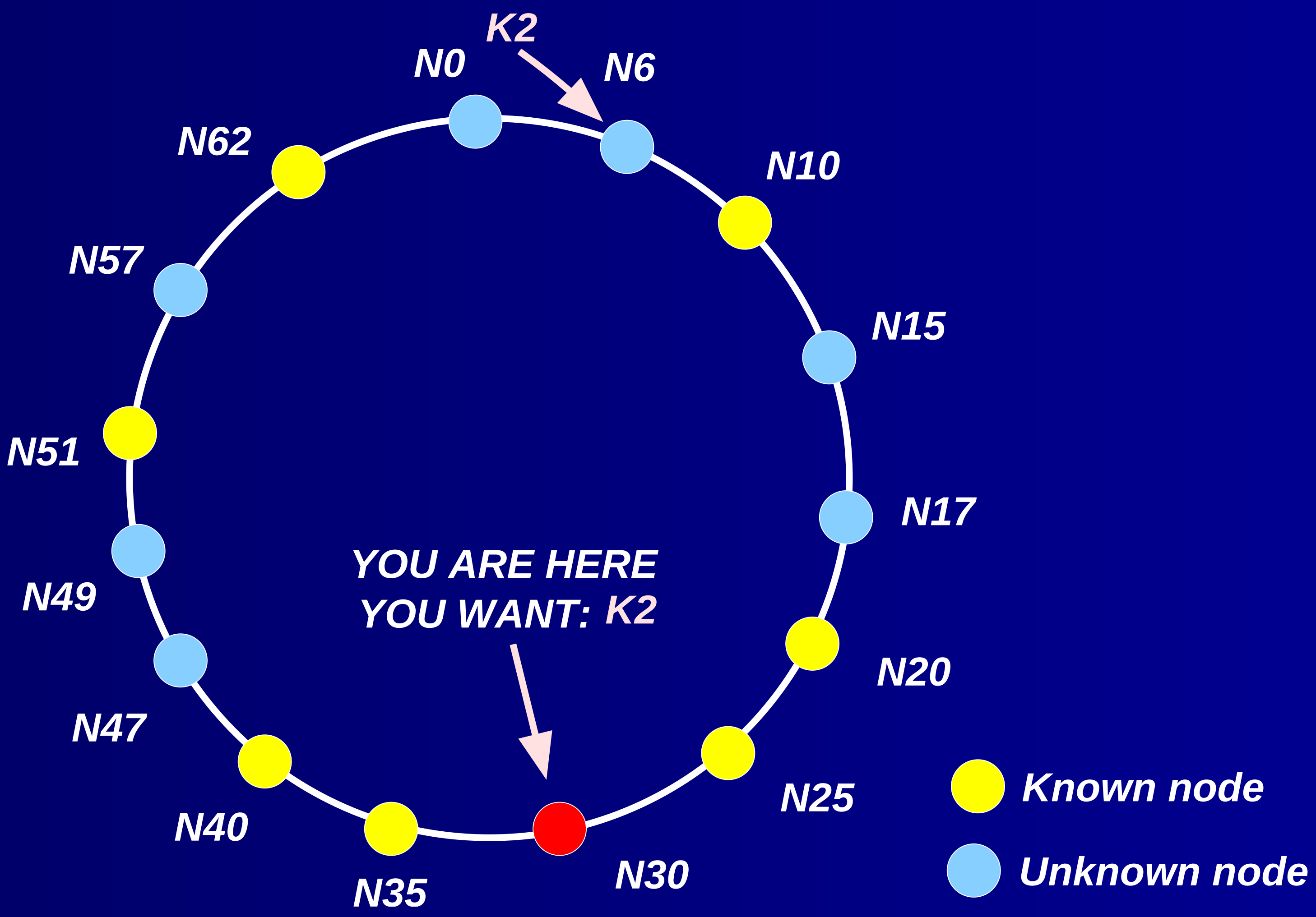
Our Goal

- We want to do better than $O(\log n)$-hop lookup without adding extra overhead.
- Use a combination of techiques:
  - Piggyback information on lookup messages
  - Allow cache to store more than $O(\log n)$ routing state
  - Issue parallel queries during lookup

Outline

- Parallel Lookup Algorithm
- Reactive Cache Management
- Simulation Results
- Related Work
- Conclusion

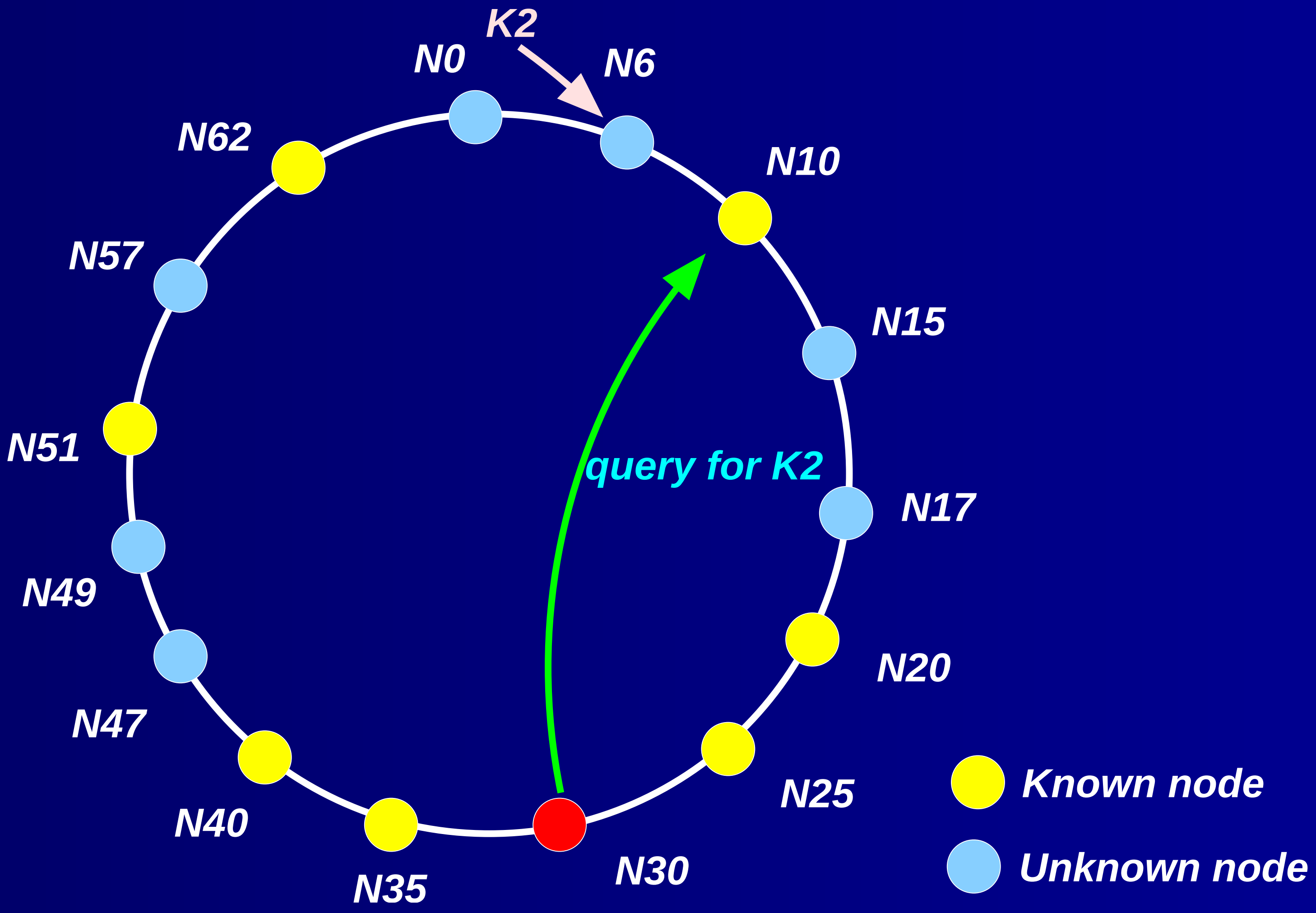
EpiChord Lookup Algorithm



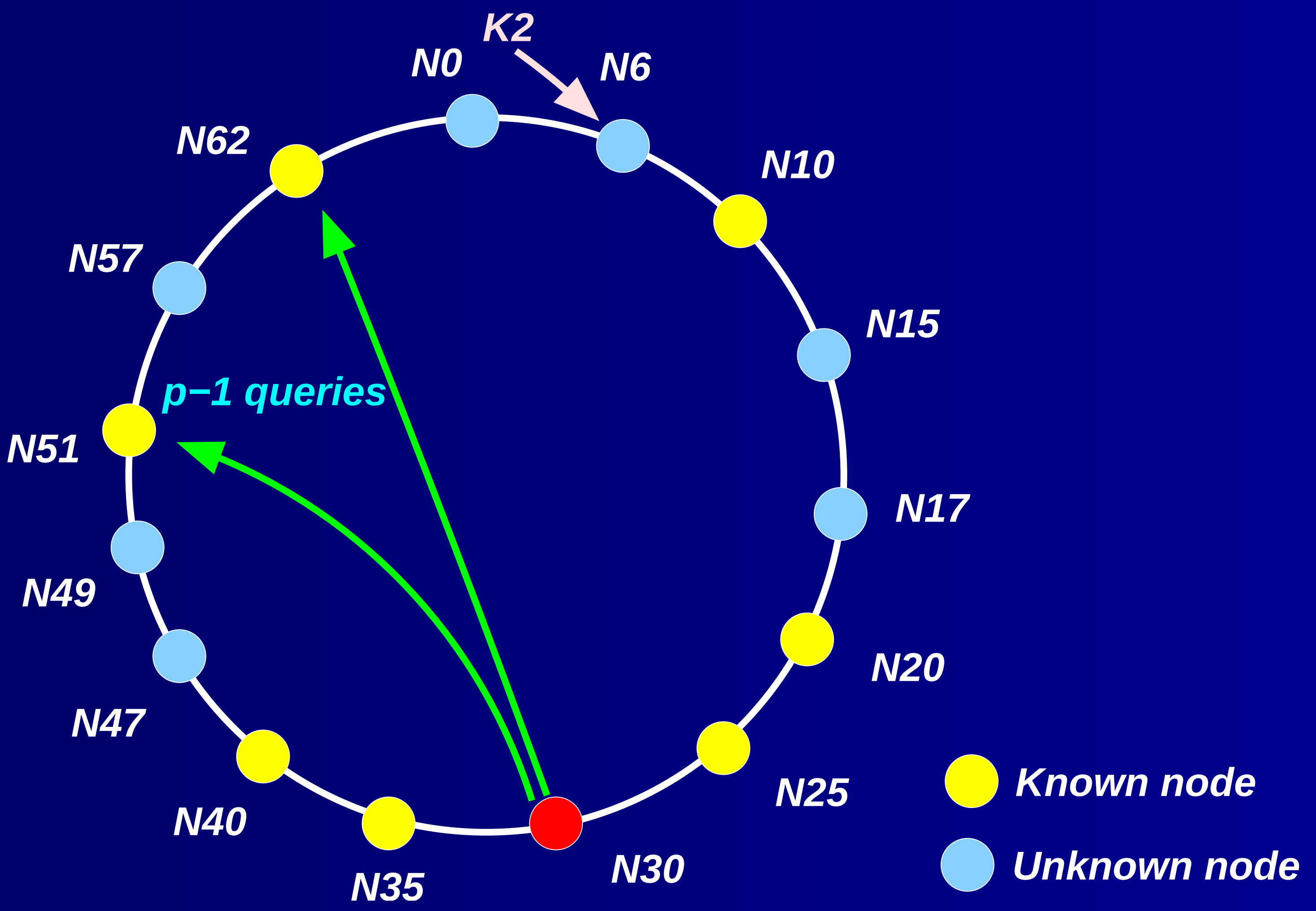
YOU ARE HERE
YOU WANT: K2

● Known node
● Unknown node

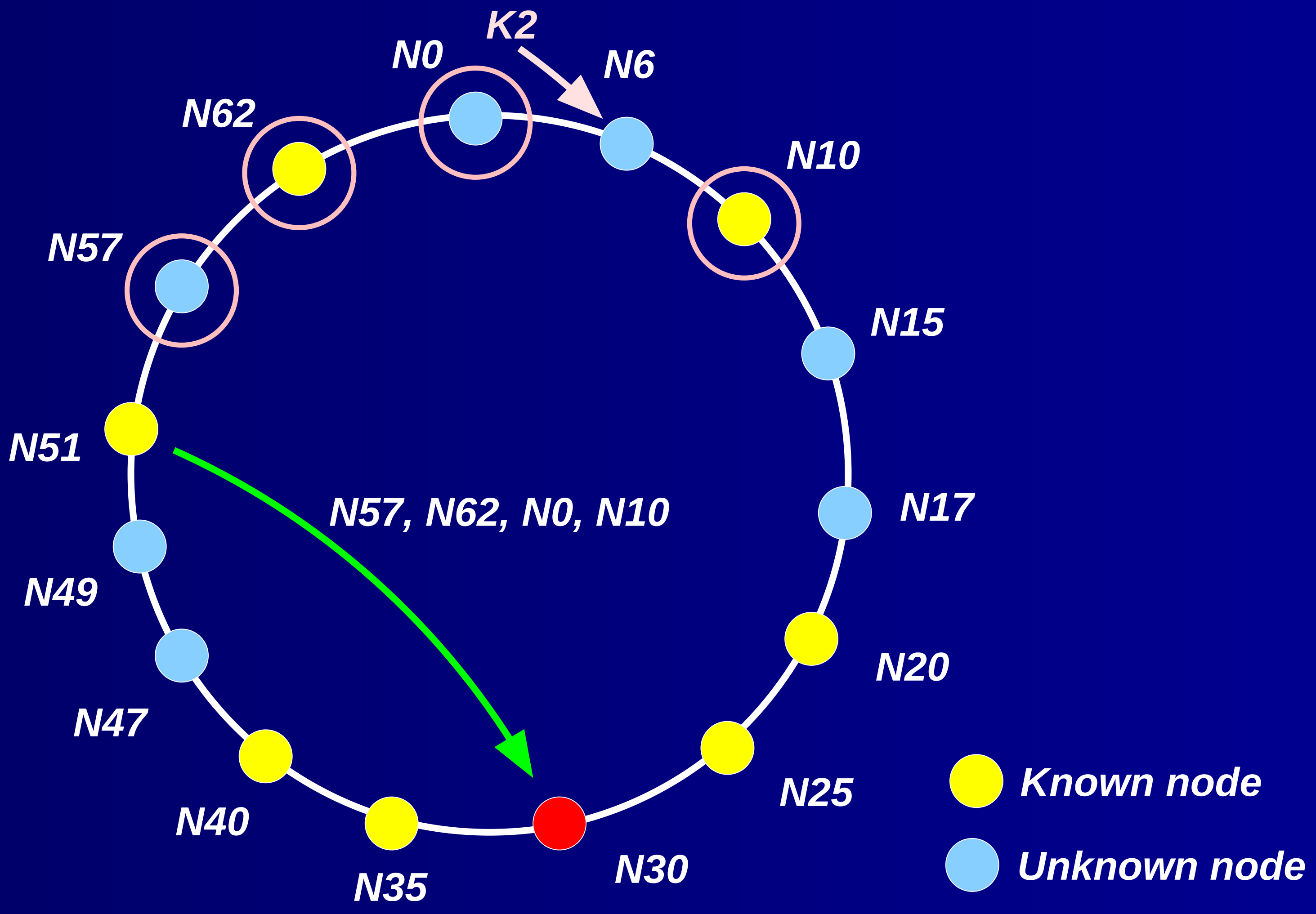
EpiChord Lookup Algorithm



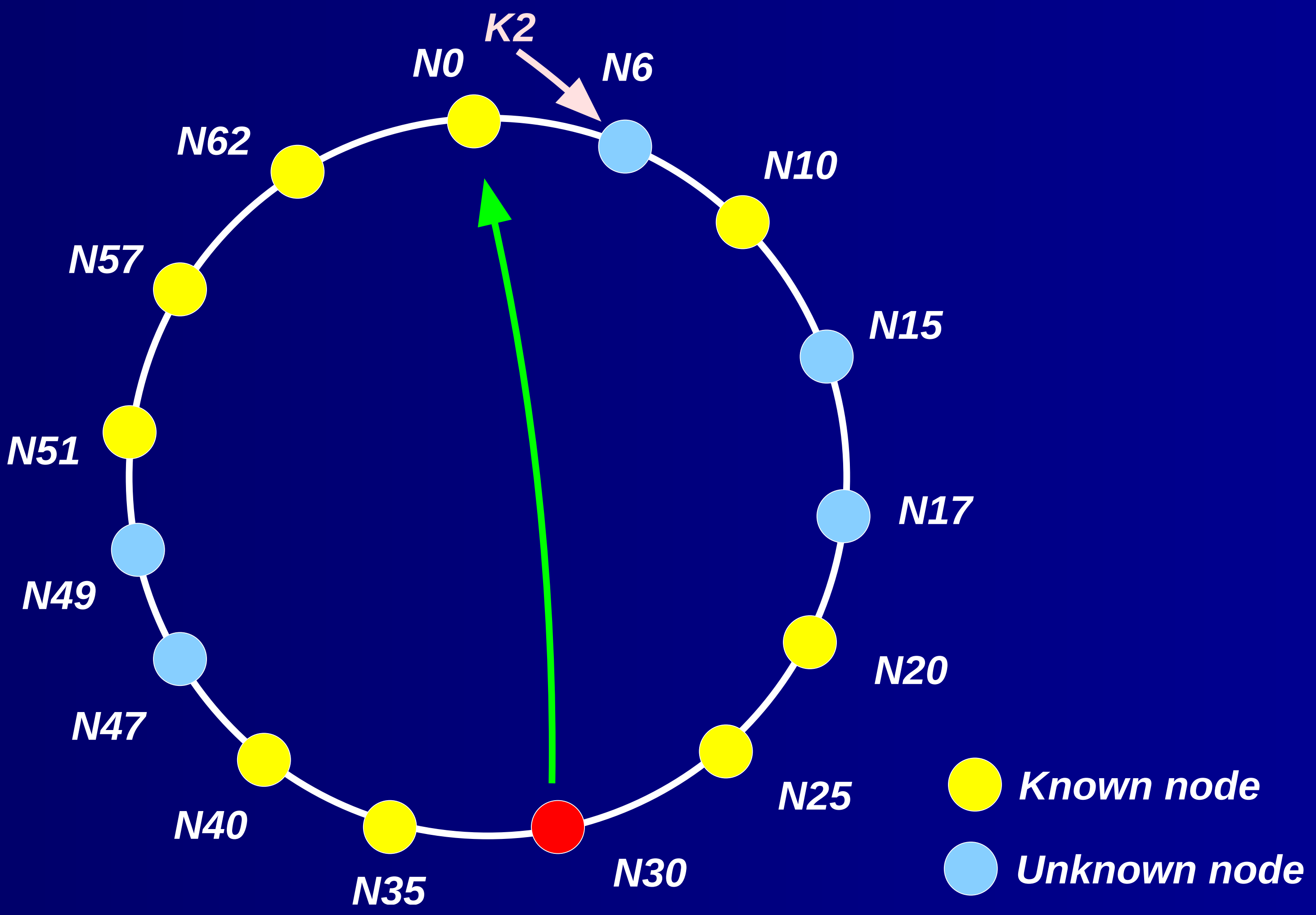
EpiChord Lookup Algorithm



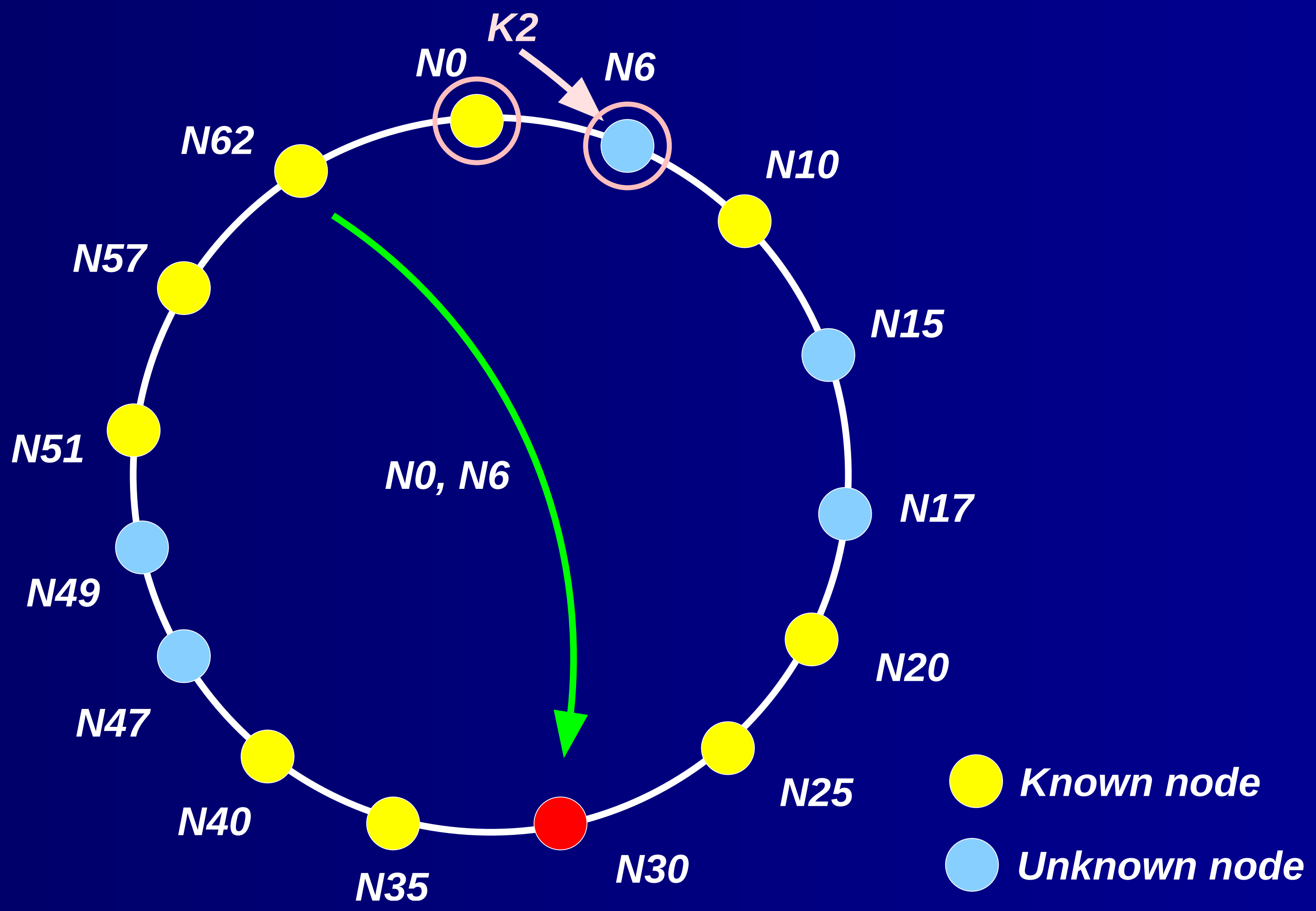
EpiChord Lookup Algorithm



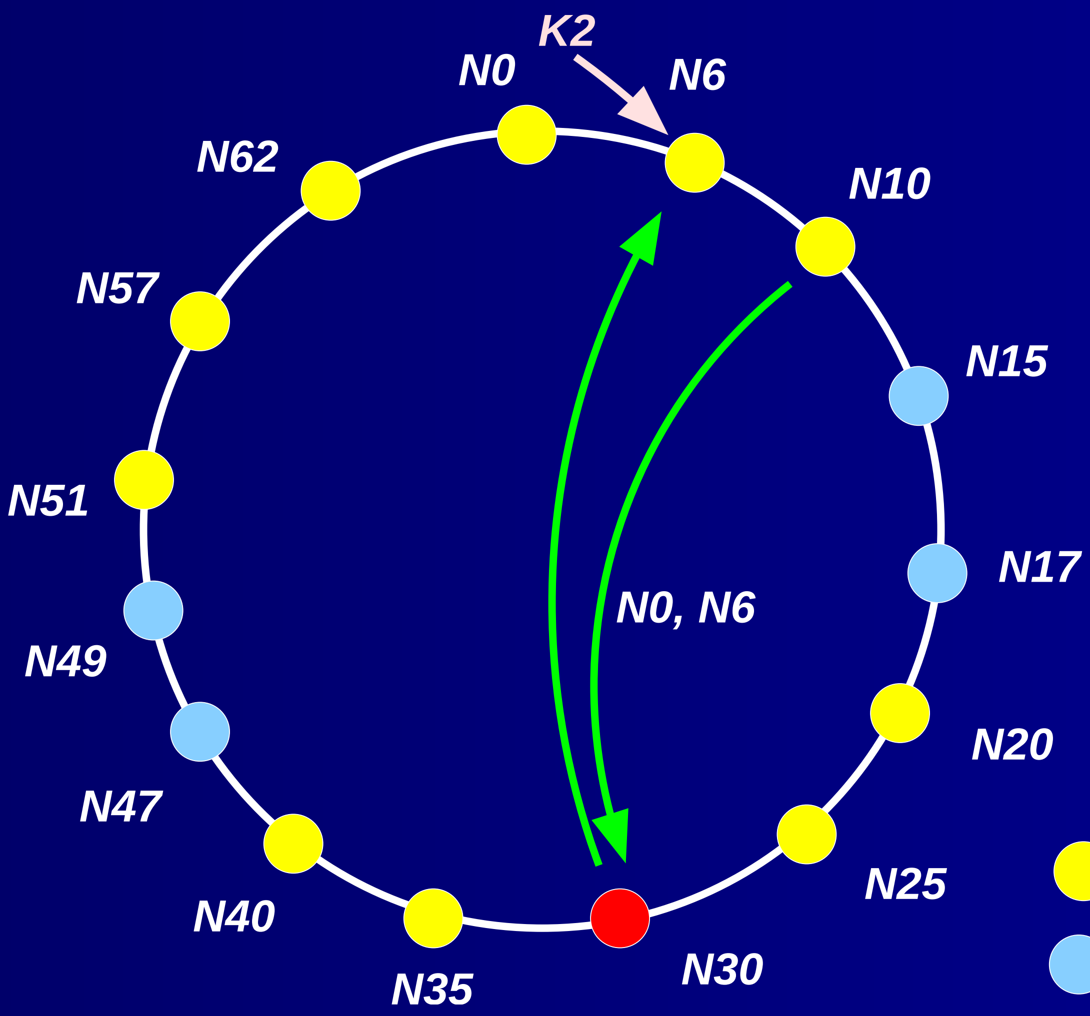
EpiChord Lookup Algorithm



EpiChord Lookup Algorithm



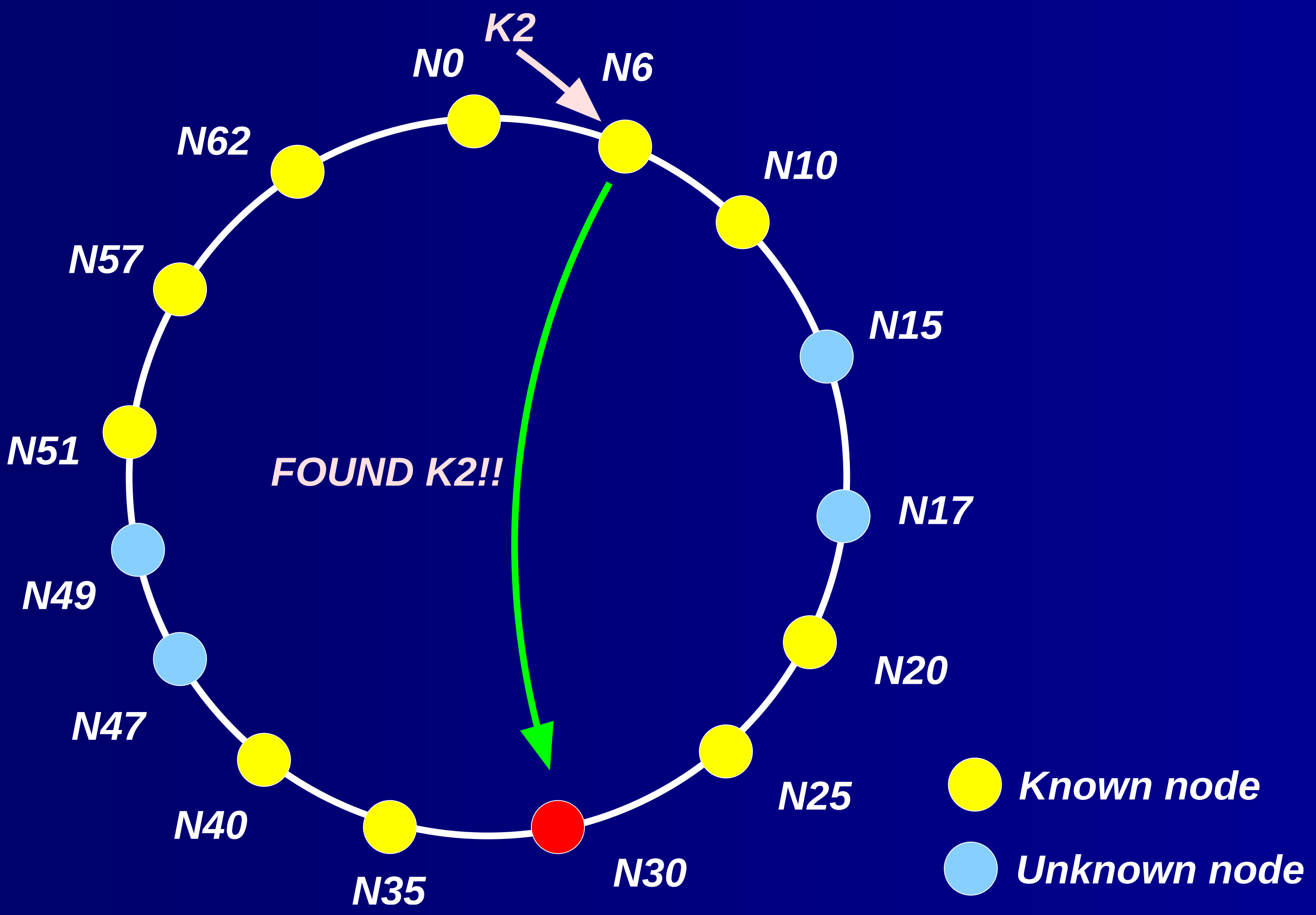
EpiChord Lookup Algorithm



● Known node

● Unknown node

EpiChord Lookup Algorithm



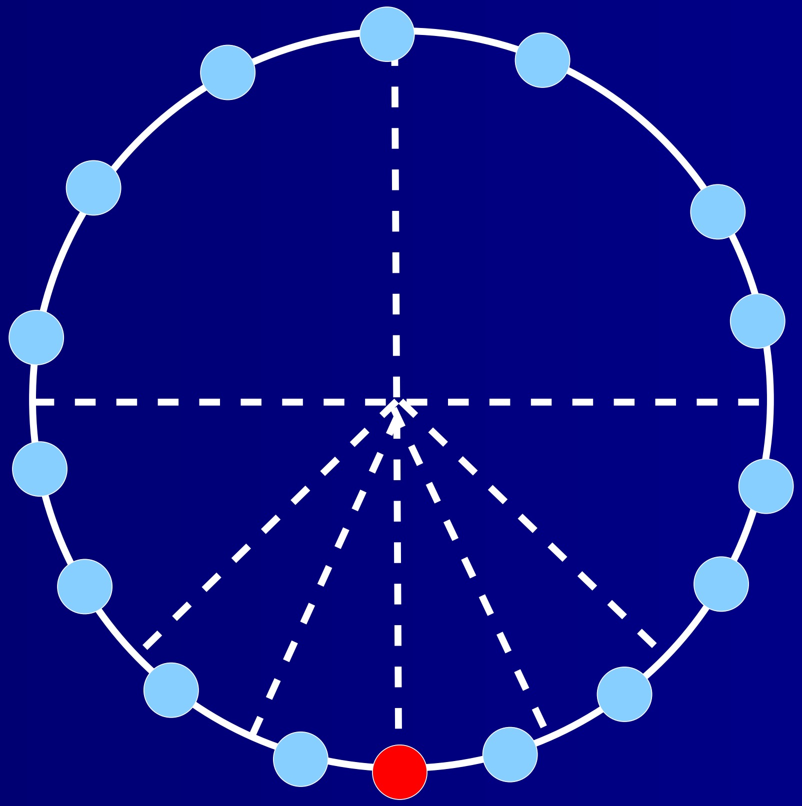
EpiChord Lookup Algorithm

- Intrinsically iterative
  - Learn about more nodes
  - Avoid redundant queries – typically $2(p+h)$ messages
- Additional policies to learn new routing entries:
  - When a node first joins network, obtains a cache transfer from successor
  - Nodes gather information by observing lookup traffic

Reactive Cache Management

- Traditional (active) approach
  $\Rightarrow$ Ping fingers periodically
- Our (reactive) approach:
  - Cache entries have a fixed expiration period
  - Divide address space into exponentially smaller slices
  - Periodically check if each slice has sufficient (j) un-expired entries
  - If not, make a lookup to the midpoint of the offending slice

Division of Address Space



- Estimate number of slices from k successors and k predecessors
- j and k are system parameters $\Rightarrow$ choose $k \geq 2j$

Summary

- Piggyback extra information on lookups
- Allow cache to contain more than $O(\log n)$ state
- Flush out old state with TTLs
- Use cache entries in parallel to avoid timeouts
- Check that cache entries are well-distributed. Fix if necessary.
- Now, let’s evaluate performance : (i) latency and (ii) cost

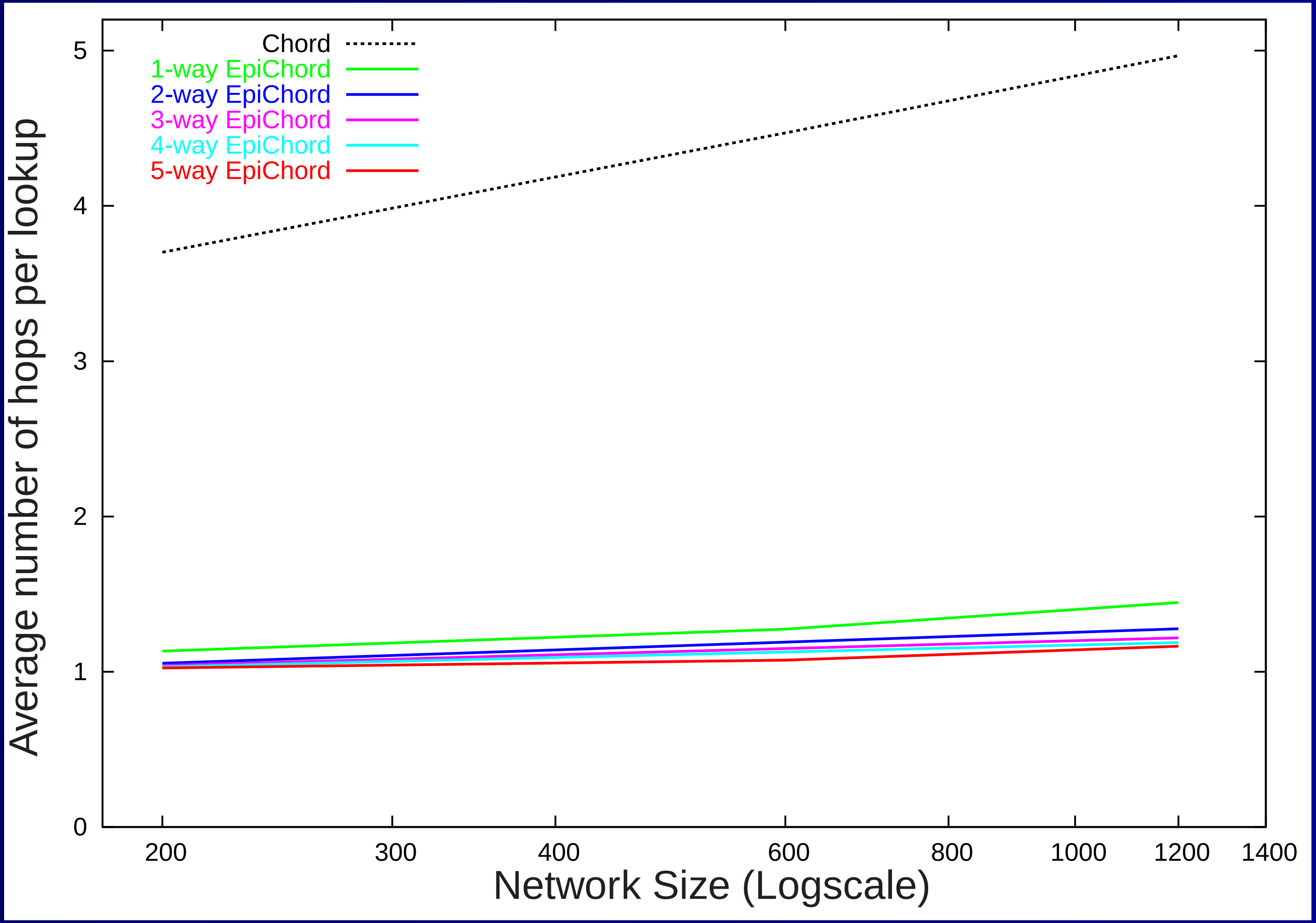
Simulation Setup

- Compare EpiChord to the *optimal* sequential Chord lookup algorithm (base 2)
- What’s optimal? We ignore Chord maintenance costs and assume that the finger tables of nodes are perfectly accurate regardless of node failures
- The competing sequential lookup algorithm is thus a reasonably strong adversary and not just a straw man

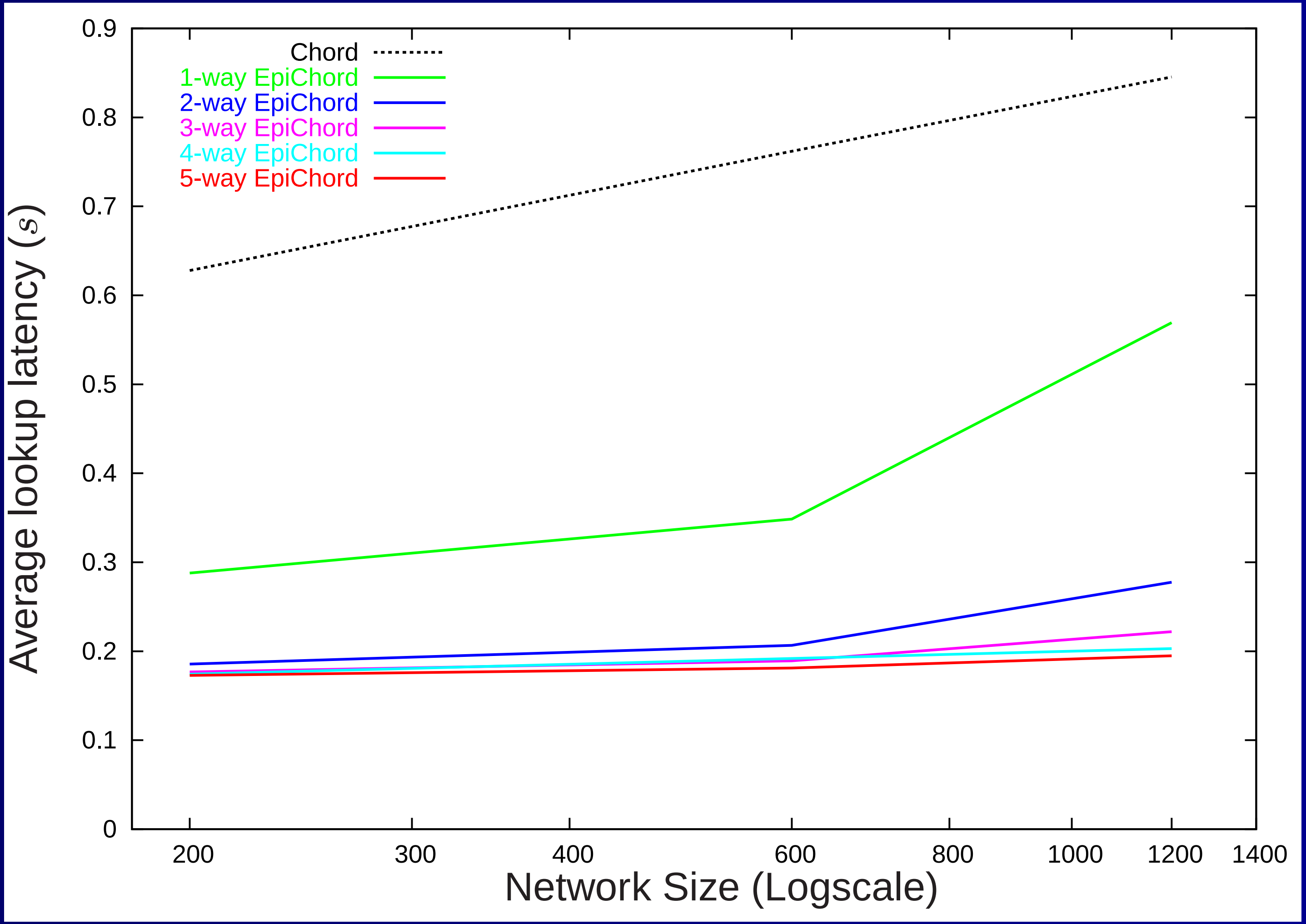
Simulation Setup

- The assumed workloads will affect comparisons (Li et al., 2004)
- Consider 2 types of workloads:
  - Lookup-Intensive
    200 to 1,200 nodes, $r \approx \frac{1}{600} \Rightarrow rn \approx 0.3$ to 2
    query rate, $Q \approx 2$ per sec
  - Churn-Intensive
    600 to 9,000 nodes, $r \approx \frac{1}{600} \Rightarrow rn \approx 1.0$ to 15
    query rate, $Q \approx 0.05$ to 0.07 per sec

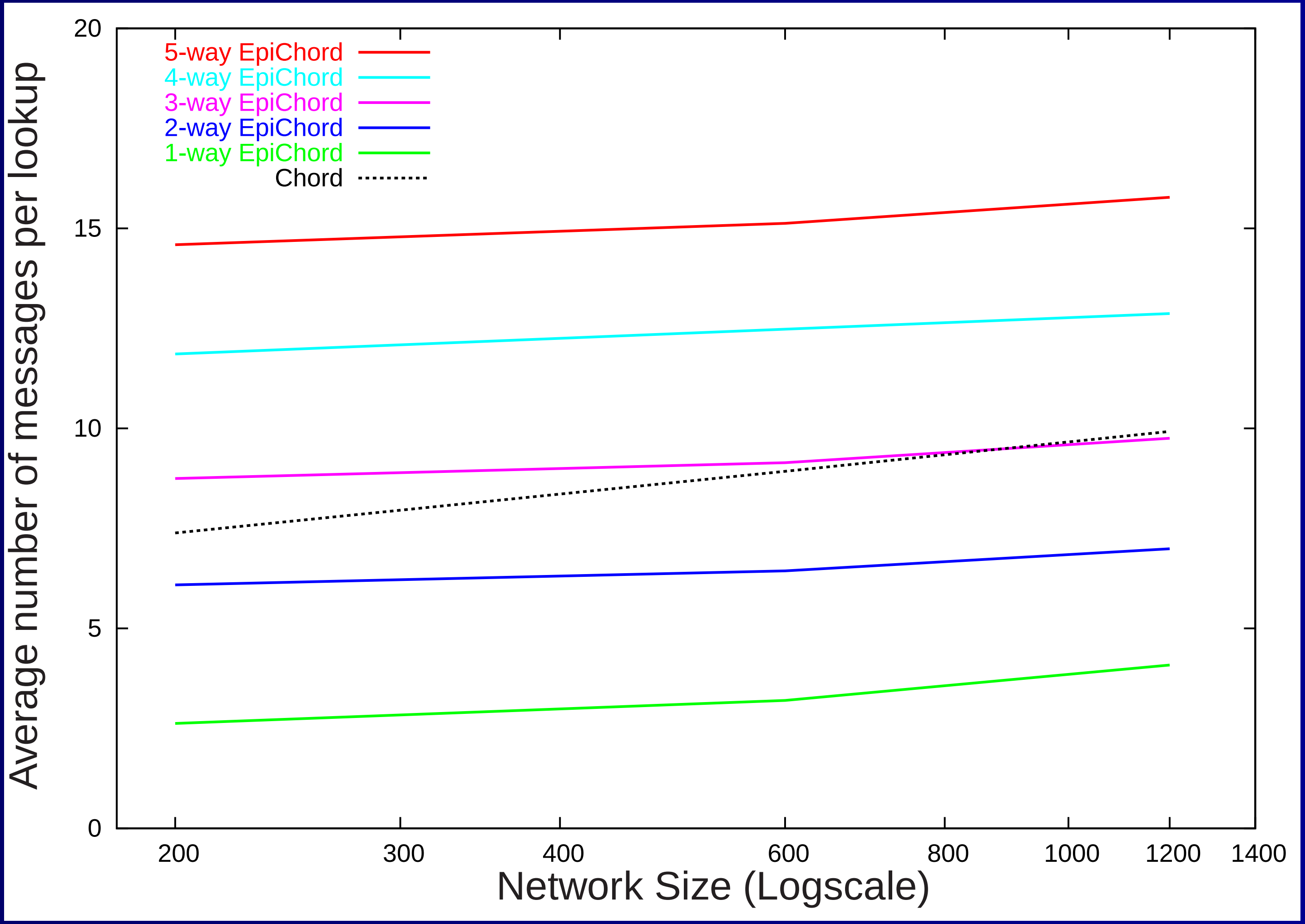
Hop Count – Lookup-Intensive



Latency – Lookup-Intensive



Messages Sent Per Lookup



Summary of Results

- Increasing p improves hop count and latency and reduces lookup failure rate
- Since our approach is iterative $\Rightarrow$ about $2(p+h)$ messages per lookup
- Higher lookup rates yield better overall performance due to caching
- Number of entries returned per query $l > 3$ does not affect performance much, so we set $l = 3$

Related Work

- Chord (Stoica et al., 2001)
- DHash++ (Dabek et al., 2004)
- Kademlia (Maymounkov and Mazieres, 2002)
- Kelips (Gupta et al., 2003)
- One-Hop (Gupta et al., 2004)

Conclusion

- Parallel lookup and reactive routing state maintenance algorithm trades off storage with better lookup performance w/o increasing bandwidth consumption
- Reduce both lookup latencies and pathlengths over Chord by a factor of 3 by issuing only 3 queries asynchronously in parallel per lookup w/o using more messages
- A parallel lookup strategy is inherently more resilient to timeouts than a sequential one

EpiChord: Parallelizing the Chord Lookup Algorithm with Reactive Routing State Management

Ben Leong, Barbara Liskov, and Eric D. Demaine

MIT Computer Science and Artificial Intelligence Laboratory

`{benleong, liskov, edemaine}@mit.edu`

Proximity

- We do not track latency information or explicitly use proximity information
- But parallel asynchronous lookup exploits proximity indirectly
- Key observation — Final sequence of lookups that returns the correct answer first is approximately equivalent to a proximity-optimized lookup sequence

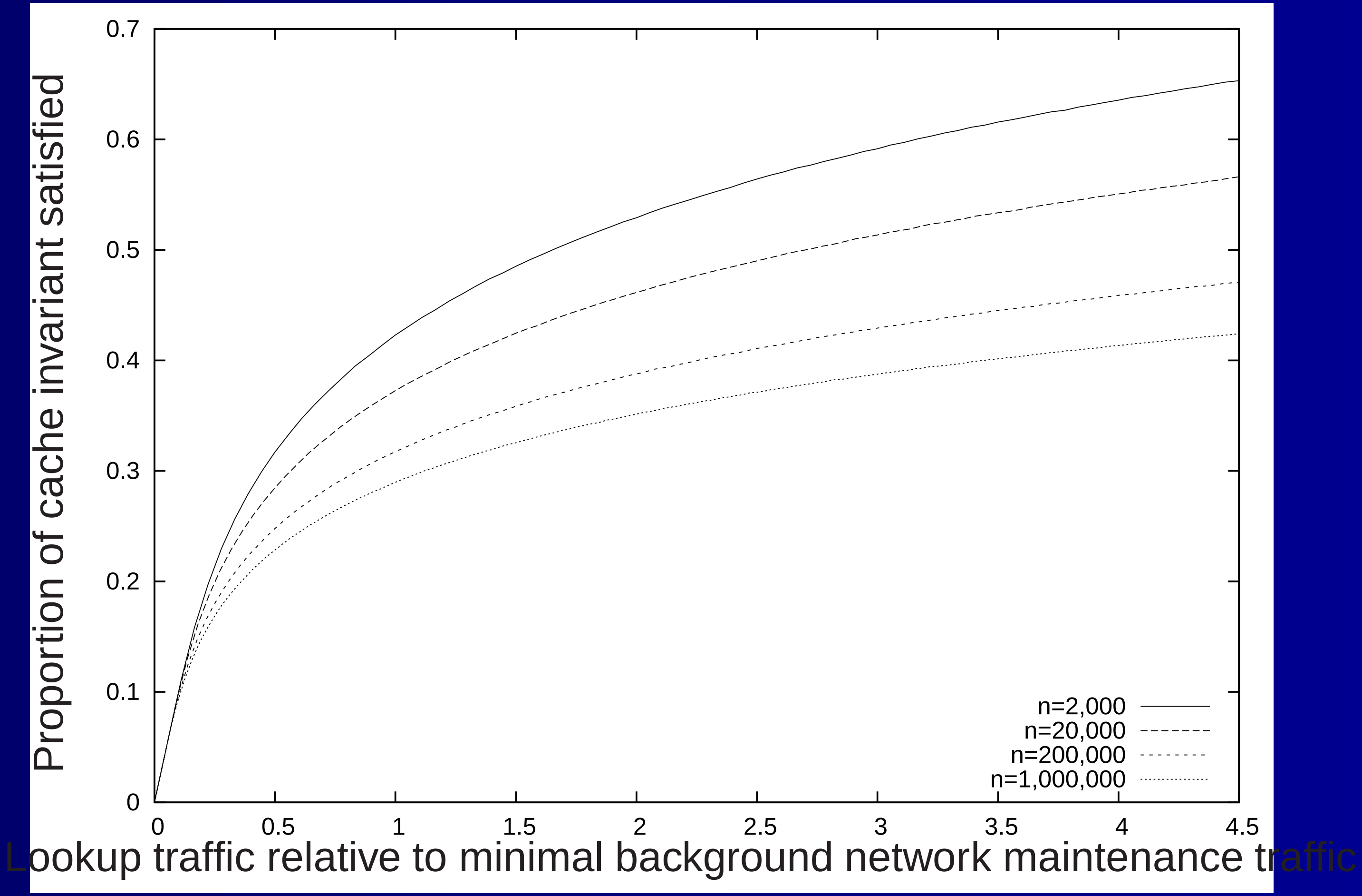
Worst-Case Performance

- If j (entries/slice) $= 1$, equivalent to Chord
- Assume a uniformly distributed workload, worst-case lookup pathlength is at most

$$\frac{1}{2}\log_\alpha n, \quad \alpha = 3j + \frac{6}{j+3} \; (j > 1)$$

- If $j = 2$, $\alpha = 7.2$ and expected worst-case lookup pathlengths are at most only $\frac{\frac{1}{2}\log_2 n}{\frac{1}{2}\log_\alpha n} = \log_\alpha 2 \approx \frac{1}{3}$ of that for Chord

Reduction in Background Probes



- Probably at least 20 to 25% savings

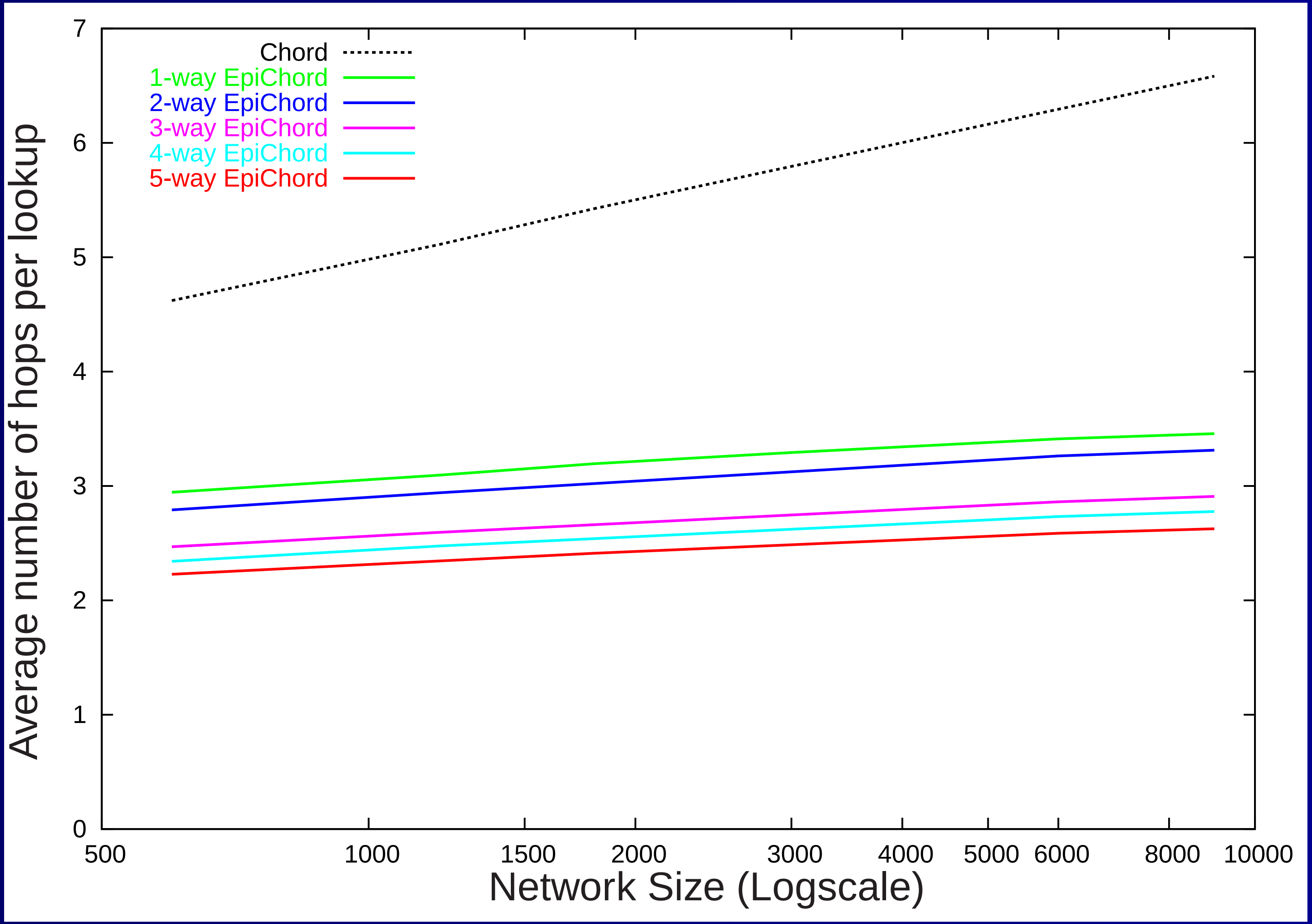
System Parameters

- Timeout = 0.5 s
- Retransmits = 3 times
- Node lifespan – exponentially distributed with mean 600 s (10 mins)
- Cache Expiration Interval = 120 s (2 mins)

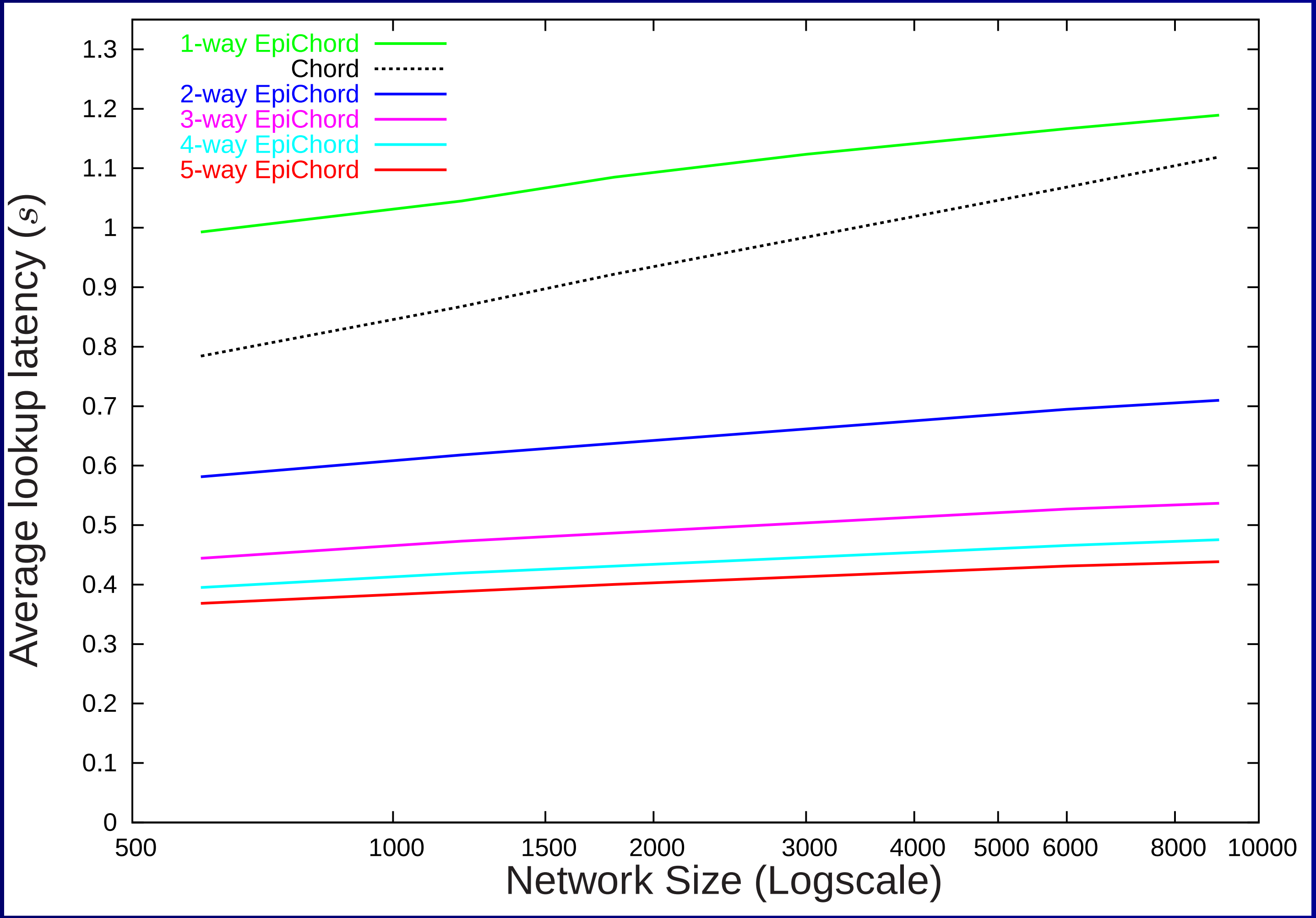
Background Maintenance Traffic

- Need to ping every 60 s for 90% validity
- $j = 2 \Rightarrow$ min routing set $4\times$ Chord
- Need only half probes because of symmetry
- Since 120 s $= 2 \times 60$ s $\Rightarrow$ background maintenance bandwidth $\leq$ Chord

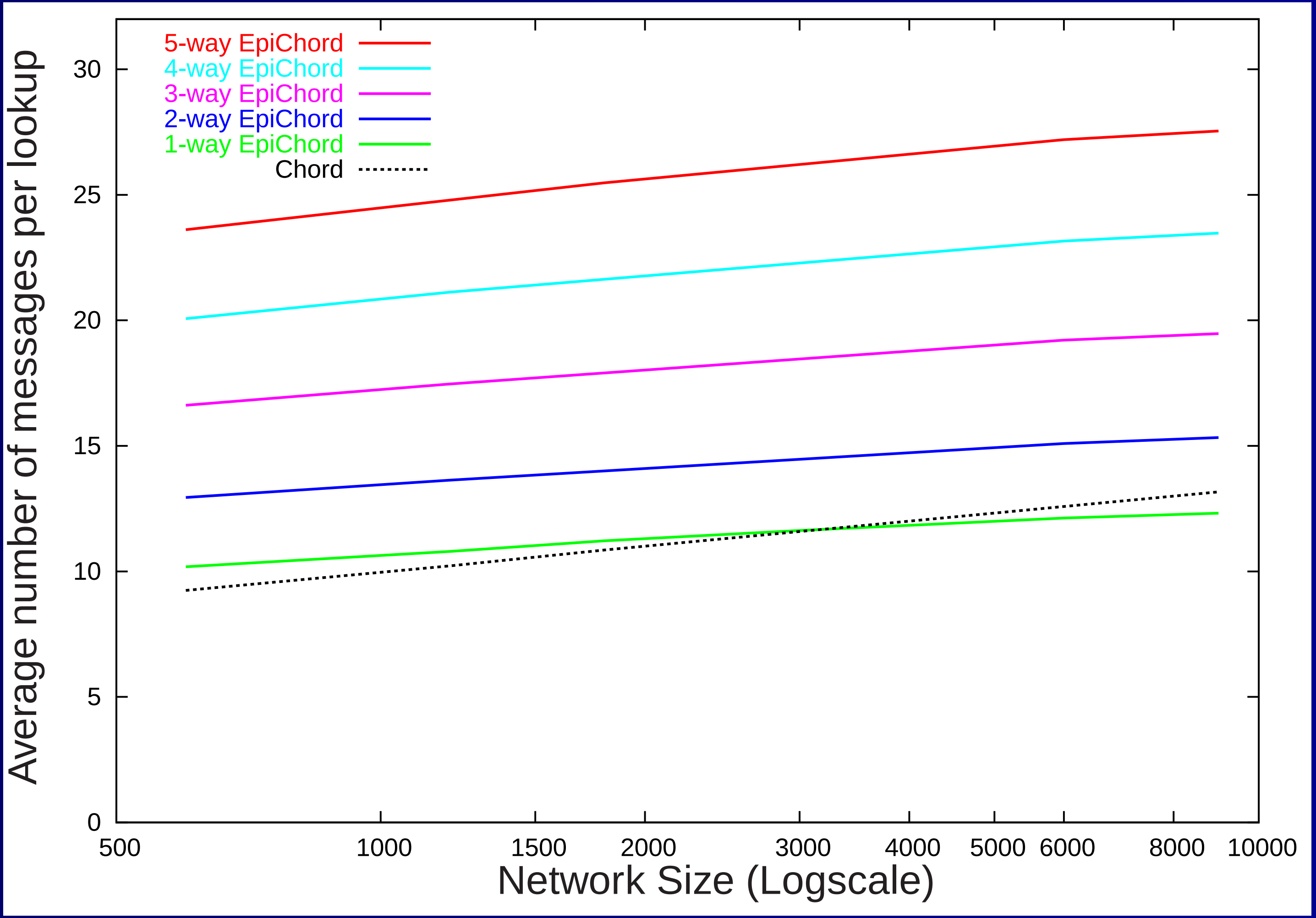
Hop Count – Churn-Intensive



Latency – Churn-Intensive



Messages Sent Per Lookup



Modelling Cache Composition

- Consider a network of steady state size n, where per unit time
    - a fraction r of the nodes leave
    - a fraction f of the cache entries are flushed
    - Each node makes Q lookups uniformly over the address space
    - p queries are sent in parallel for each lookup

Modelling Cache Composition

- Where x is the number of live nodes that is known to a node at time t, we obtain the following relation:

$$\frac{d}{dt}x(t) = \overbrace{pQ(1-\frac{x}{n})}^{\text{incoming queries}} - \overbrace{fx}^{\text{entries flushed}} - \overbrace{(1-f)rx}^{\text{nodes departed but not flushed}}$$

- This assumes that new knowledge comes only from incoming queries

Modelling Cache Composition

- Where y is the number of outdated cache entries at time t, we have the following relation:

$$\frac{d}{dt}y(t) = \overbrace{(1-f)rx}^{\text{dead nodes not flushed}} - \overbrace{fy}^{\text{dead nodes flushed}} - \overbrace{pQ\left(\frac{y}{x+y}\right)}^{\text{outdated nodes discovered by timeouts of outgoing queries}}$$

- If churn is low relative to lookup rate, cache maintenance protocol is unimportant

Modelling Cache Composition

- If churn is high, the proportion of outdated entries in the cache, γ, is

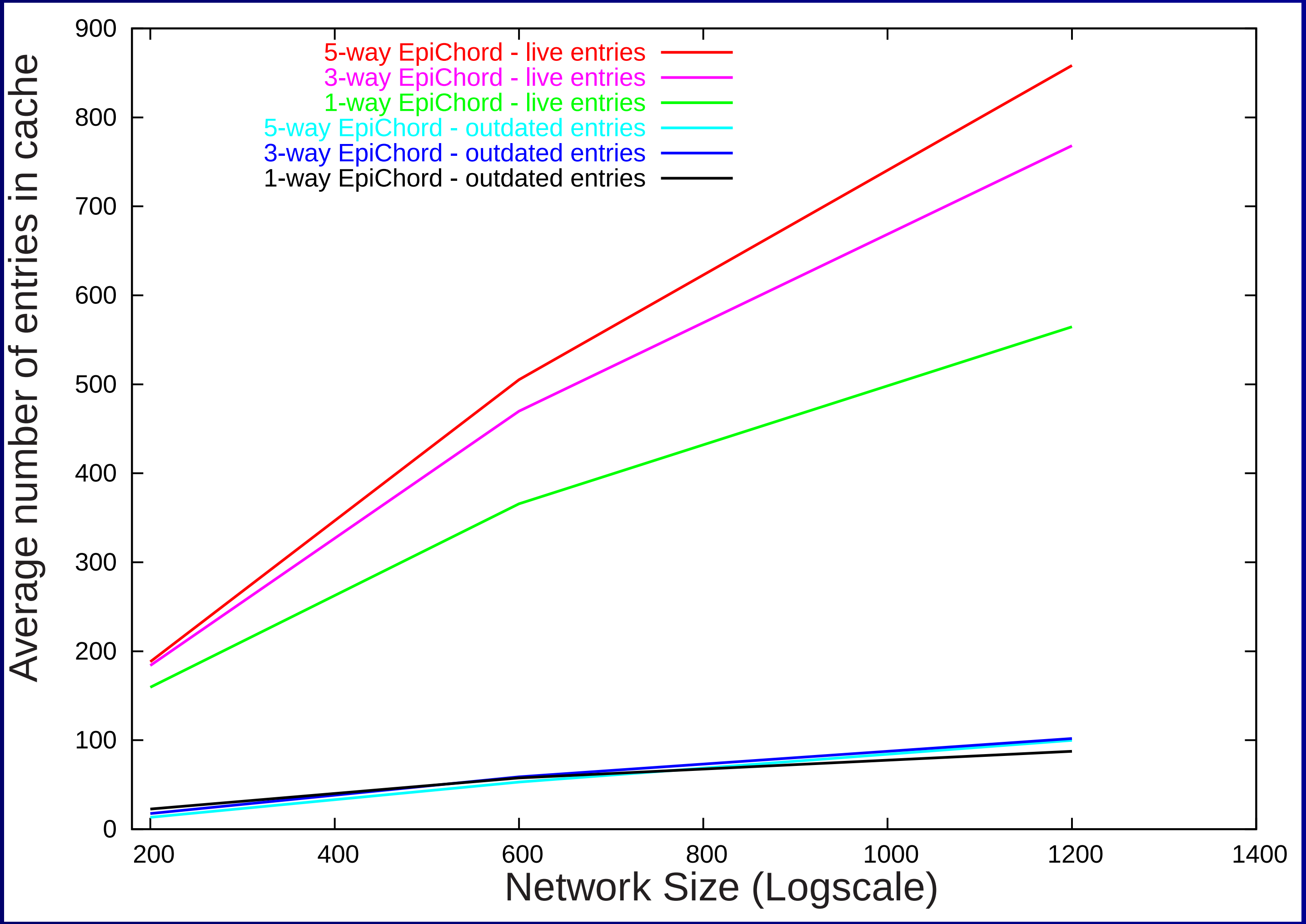
$$\gamma = \lim_{t\to\infty} \frac{y}{x+y} \approx \frac{\sqrt{1+\frac{(1-f)r}{f}}-1}{\sqrt{1+\frac{(1-f)r}{f}}}$$

- If cache entries are flushed at node failure rate,

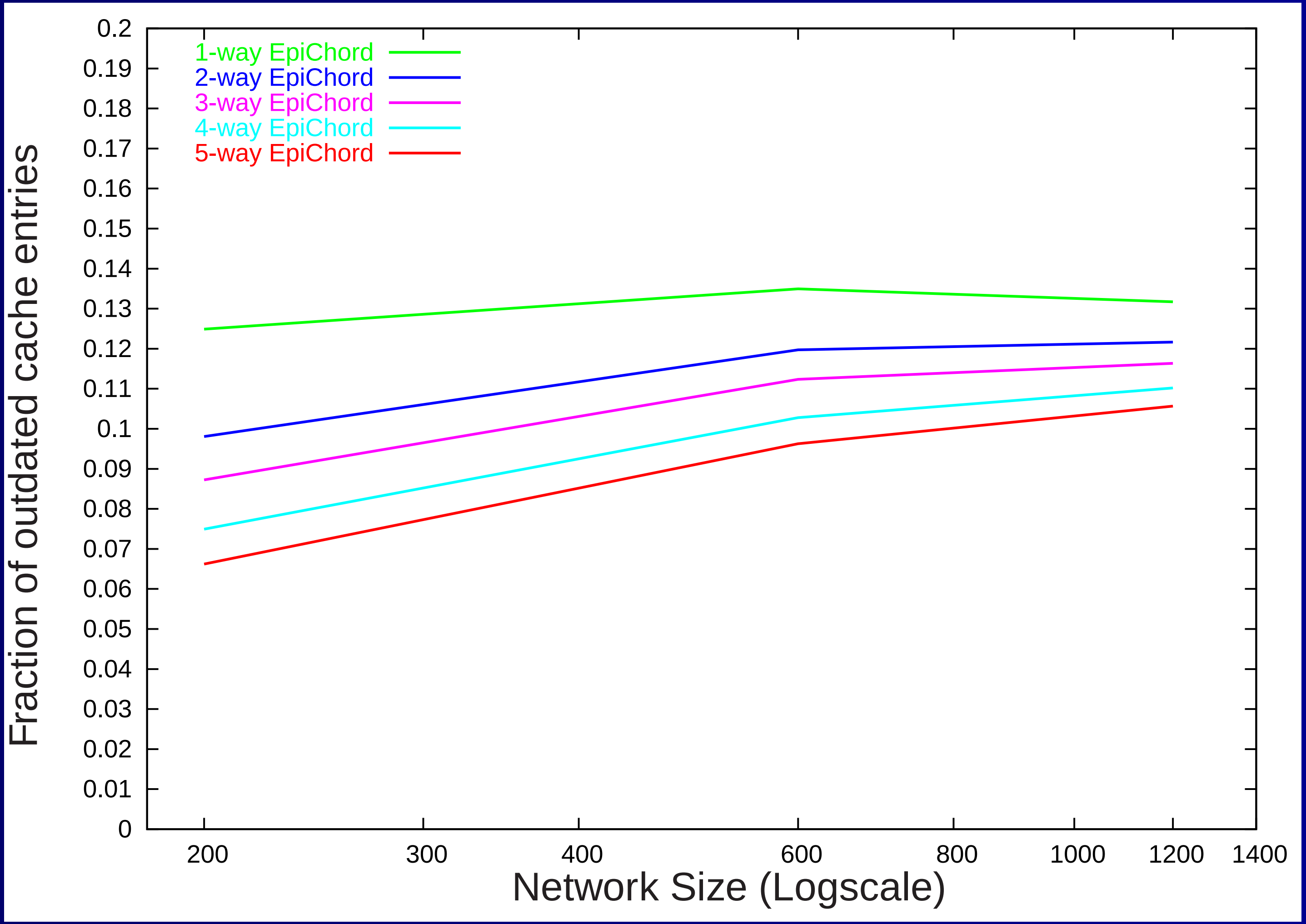
$$\gamma \approx \frac{\sqrt{2-f}-1}{\sqrt{2-f}} \leq 1 - \frac{1}{\sqrt{2}} = 0.292$$

$\Rightarrow$ most 30% of cache entries will be outdated

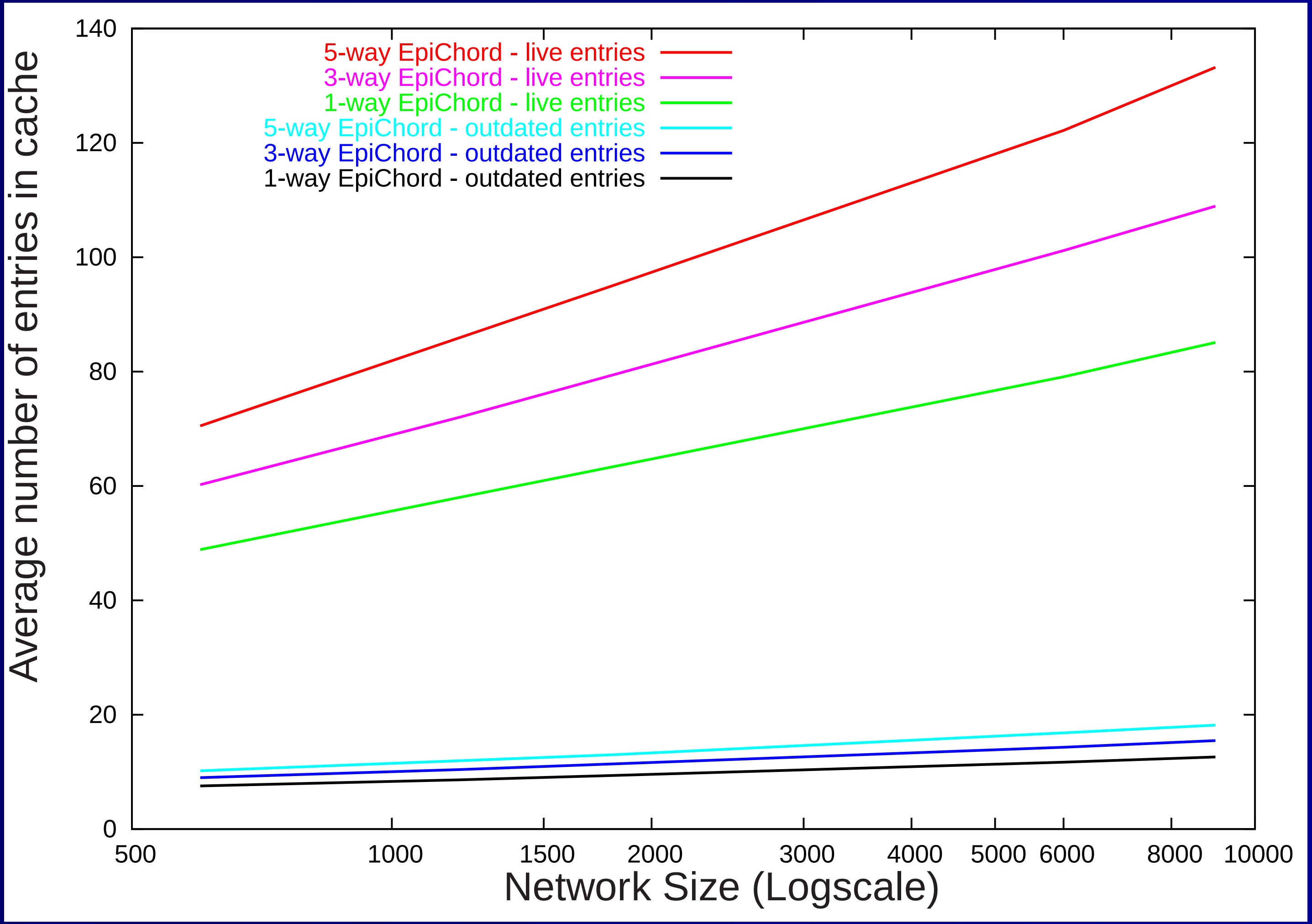
Cache – Lookup-Intensive



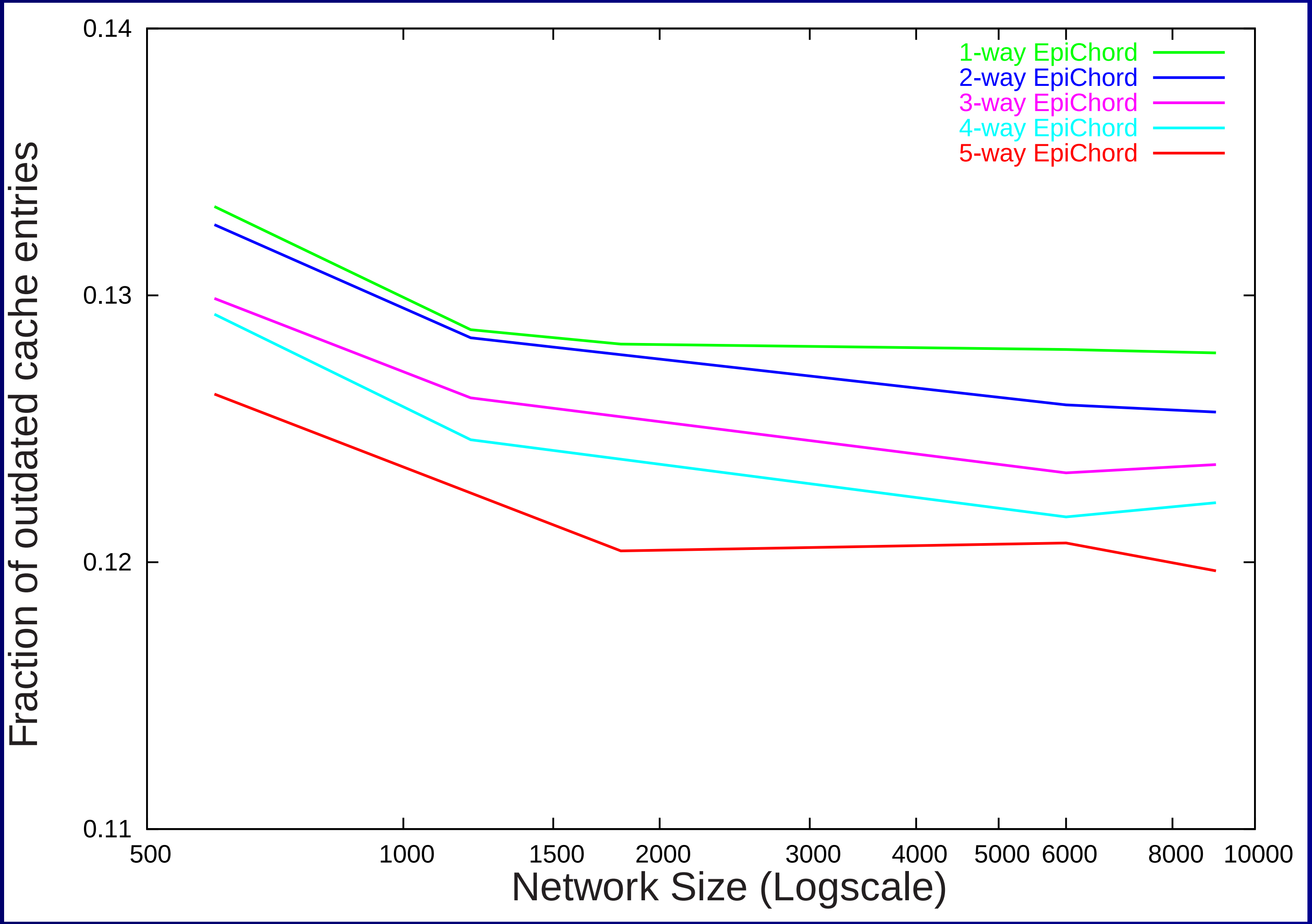
Cache – Lookup-Intensive



Cache – Churn-Intensive



Cache – Churn-Intensive



References

Dabek, F., Li, J., Sit, E., Robertson, J., Kaashoek, M. F., and Morris, R. (2004). Designing a DHT for low latency and high throughput. In *Proceedings of the 1st Symposium on Networked Systems Design and Implementation (NSDI 2004)*, pages 85–98.

Gupta, A., Liskov, B., and Rodrigues, R. (2004). Efficient routing for peer-to-peer overlays. In *Proceedings of the 1st Symposium on Networked Systems Design and Implementation (NSDI 2004)*, pages 113–126.

Gupta, I., Birman, K., Linga, P., Demers, A., and van Renesse, R. (2003). Kelips: Building an efficient and stable P2P DHT through increased memory and background overhead. In *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS ’03)*.

Li, J., Stribling, J., Morris, R., Kaashoek, M. F., and Gil, T. M. (2004). DHT routing tradeoffs in network with churn. In *Proceedings of the 3rd International Workshop on Peer-to-Peer Systems (IPTPS ’04)*.

Maymounkov, P. and Mazieres, D. (2002). Kademlia: A peer-to-peer information system based on the xor metric. In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS ’02)*.

Stoica, I., Morris, R., Karger, D., Kaashoek, F., and Balakrishnan, H. (2001). Chord: A scalable Peer-To-Peer lookup service for internet applications. In *Proceedings of the 2001 ACM SIGCOMM Conference*, pages 149–160.