Peer to Peer

Presented by

Bo Peng
(bpeng@cs.cornell.edu)

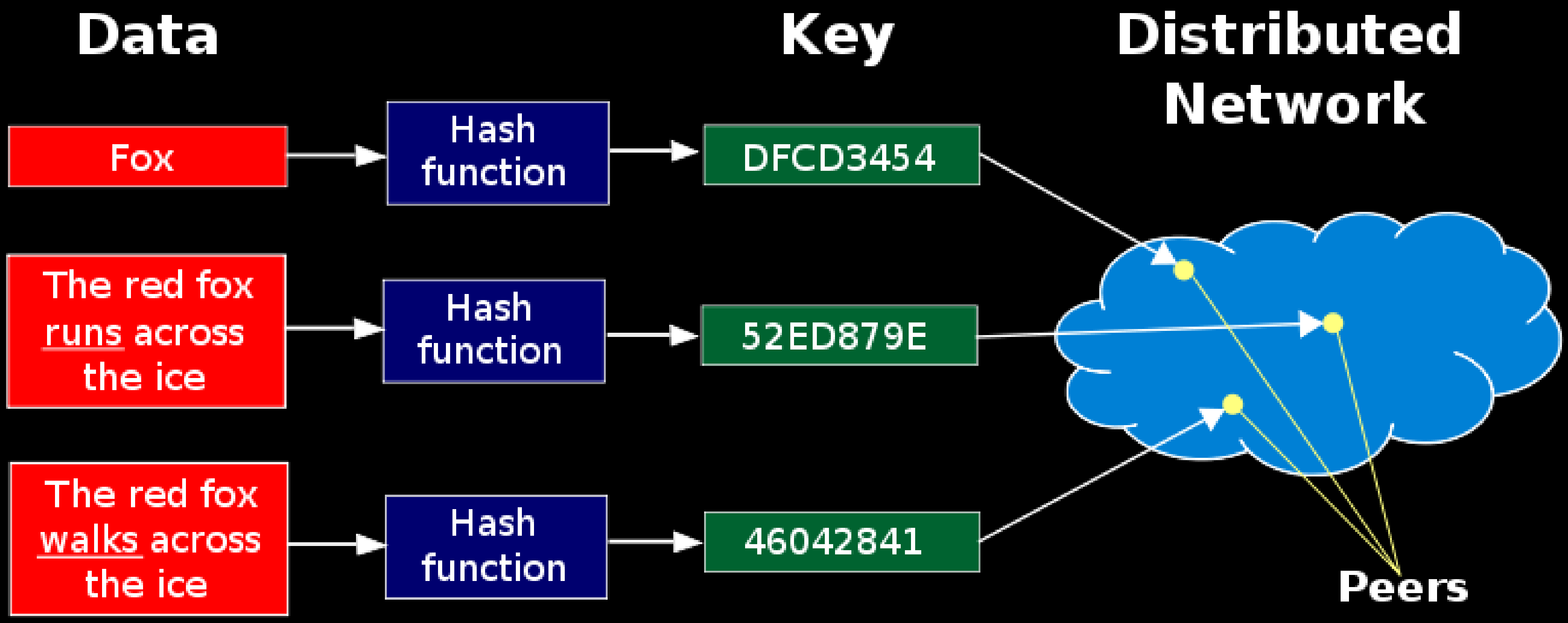
OCT 22 2009

Distributed Hash Tables

- DHTs are decentralized distributed systems providing hash-table-like lookup service
- Ideal substrate for distributed applications

  (distributed file systems, peer-to-peer file sharing , cooperative web caching, etc.)

  – Efficient lookup
  – Minimal cost of fault tolerance
  – Extreme scalability


Data
Key
Distributed Network
Fox
Hash function
DFCD3454
The red fox runs across the ice
Hash function
52ED879E
The red fox walks across the ice
Hash function
46042841
Peers

DHT History

- Motivated by peer-to-peer systems research (Napster, Gnutella, Freenet)
  - Napster: central index server
  - Gnutella: flooding query model
  - Freenet: fully distributed, but employed a heuristic key based routing
- Uses a more structured key based routing
  - The decentralization of Gnutella and Freenet
  - The efficiency and guaranteed results of Napster
  - One drawback : only directly support exact-match search, rather than keyword search
- Chord, CAN, Pastry, and Tapestry (2001)

Agenda

- Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications *(SIGCOMM’01)*

- The Impact of DHT Routing Geometry on Resilience and Proximity *(SIGCOMM’03)*

Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications

Ion Stoica (UC Berkeley)

Robert Morris (MIT)

David Karger (MIT)

M. Frans Kaashoek (MIT)

Hari Balakrishnan (MIT)

Takeaway Points

Chord:

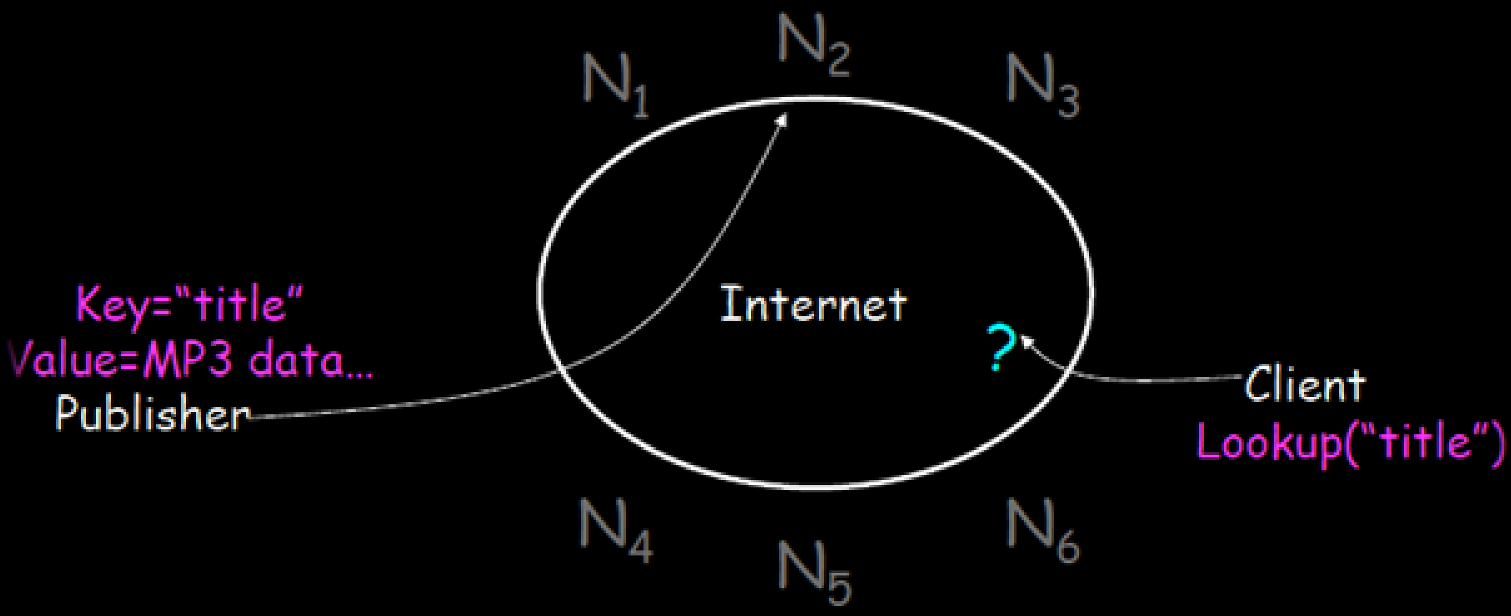
- Provides peer-to-peer hash lookup service
- Simple Geometry (Ring)
- Efficient: *O(log N) messages per lookup*
- Robust: as nodes fail and join
- Good substrate for peer-to-peer systems

Outline

- What is Chord?
- Chord hash lookup
- Maintain routing table
- Simulation

Problem

- Core operation in peer-to-peer systems
  - The lookup problem: to efficiently locate the node that stores a particular data item



What is chord

- Definition:
  - A scalable distributed protocol for peer-to-peer lookup
- Operation:
  - Supports only one operation: given a key, it maps the key onto a node
- Functionality:
  - Solves problem of locating a data item in a collection of distributed nodes, considering frequent node's joins and leaves

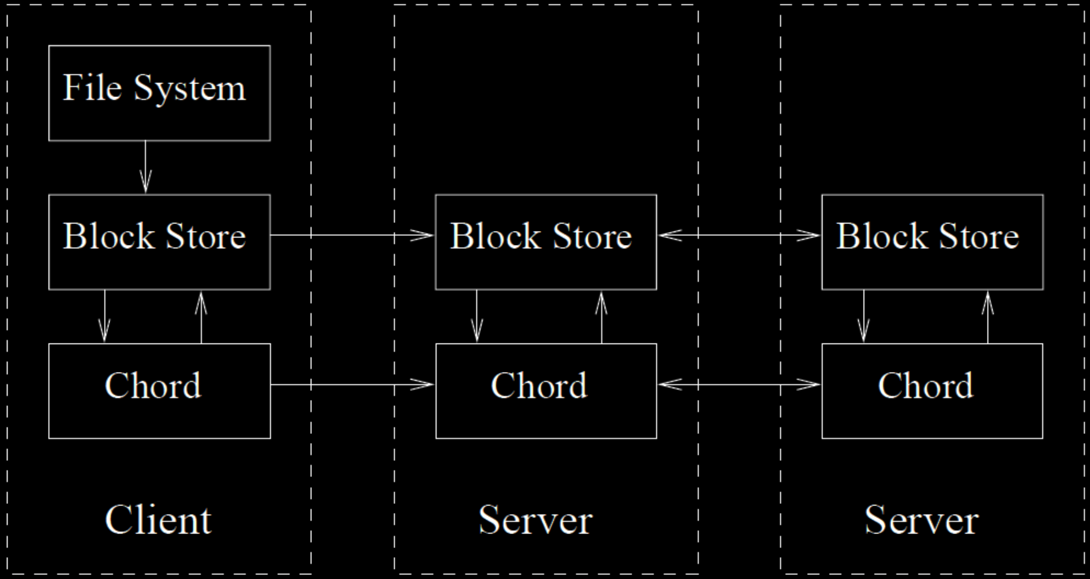
http://pdos.csail.mit.edu/chord/

Design Objectives

- Load Balance
- Decentralization
- Scalability
- Availability
- Flexible Naming

Application Support

- IP Address = Lookup(key)
- Notification
- Example : Cooperative Mirroring



Outline

- What is Chord?
- Chord hash lookup
- Maintain routing table
- Simulation

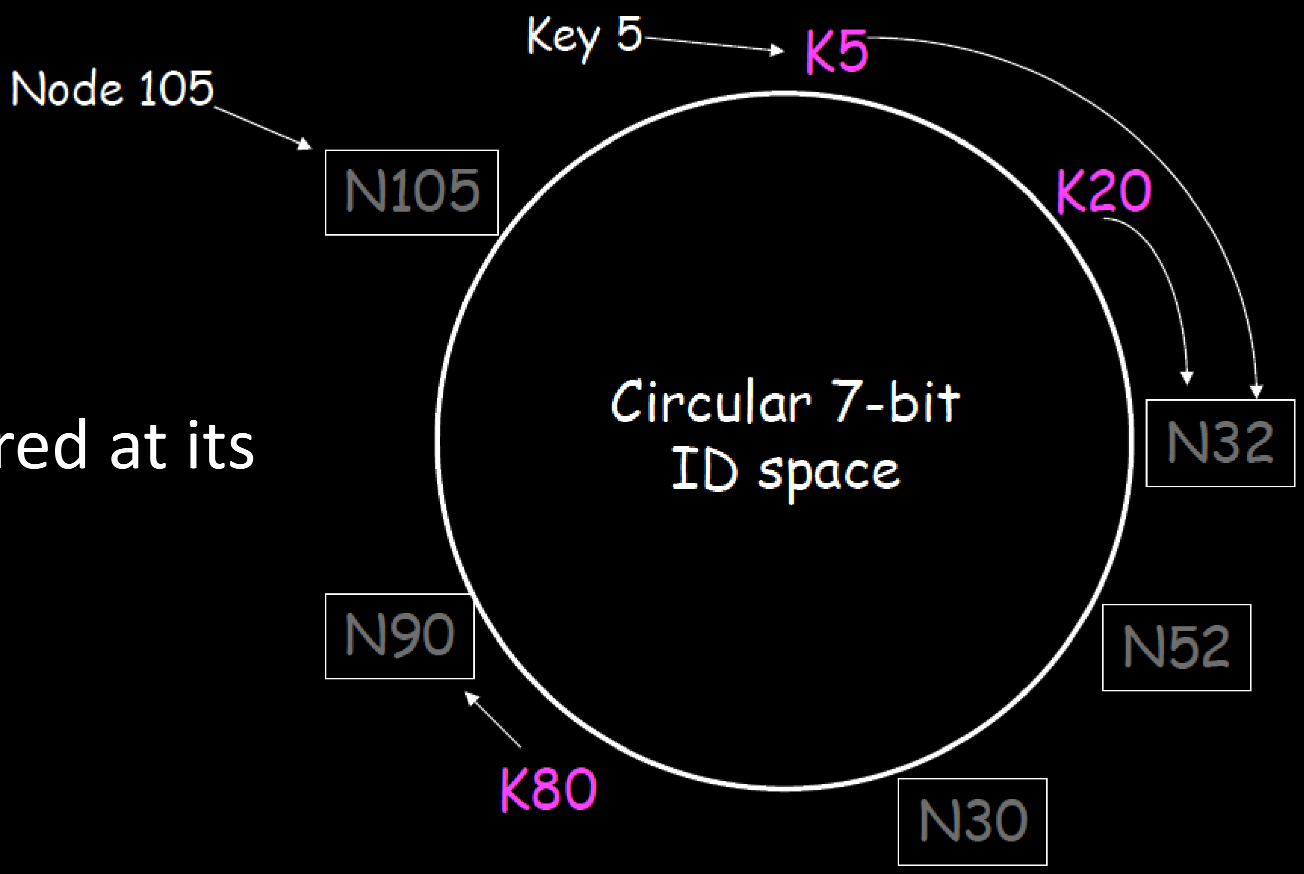
Identifier Space

- *m*-bit identifier space
  - Key identifier = SHA-1(key)
  - Node identifier = SHA-1(IP address)
- Successor
  - The node with next higher ID of the current key or node

How to map key IDs onto node IDs?

- Consistent Hashing:

A *key* is stored at its successor



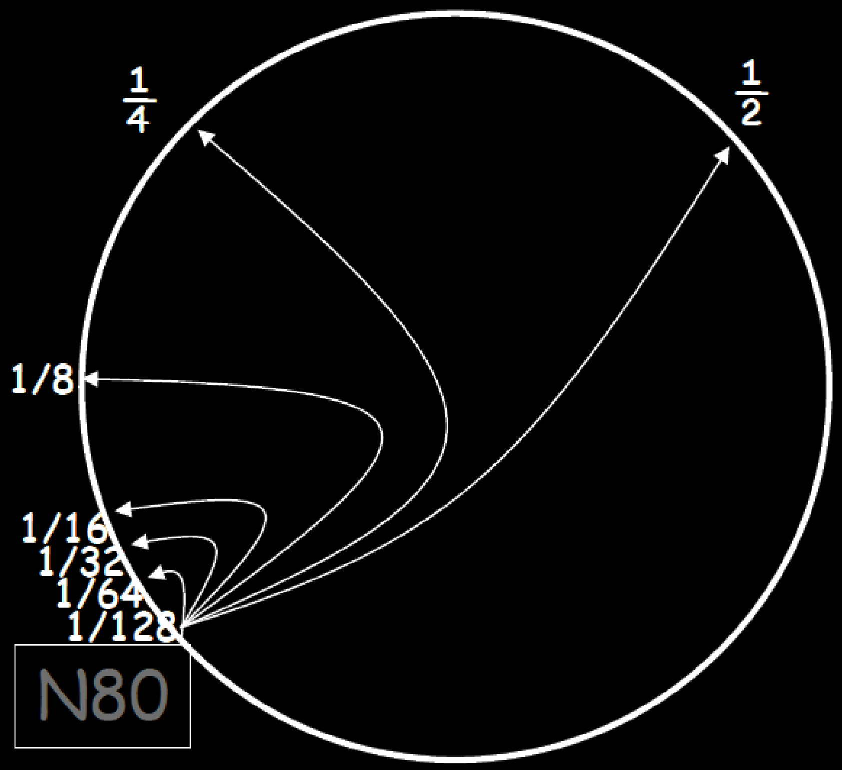
Scalable Key Location

- Finger Table

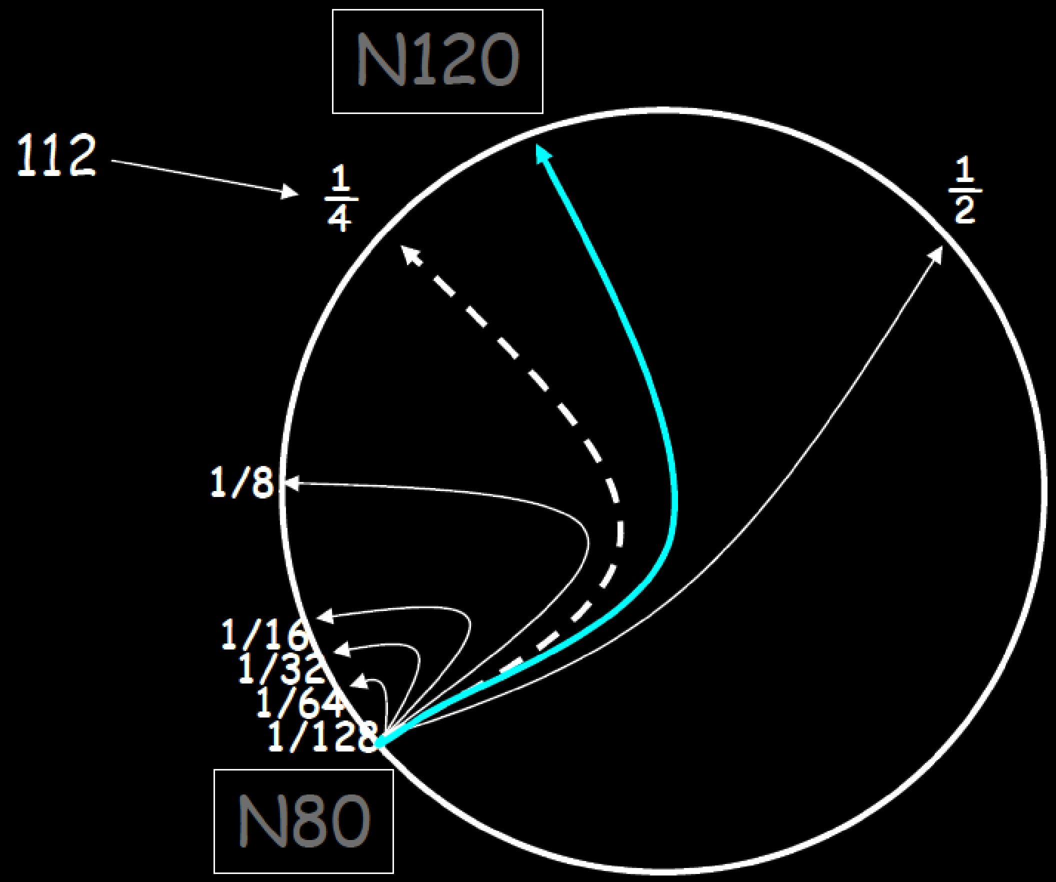
Notation	Definition
finger$[k]$.*start*	$(n + 2^{k-1}) \bmod 2^m, 1 \le k \le m$
.*interval*	$[$*finger*$[k]$.*start*, *finger*$[k+1]$.*start*$)$
.*node*	first node $\ge n$.*finger*$[k]$.*start*
successor	the next node on the identifier circle; *finger*$[1]$.*node*
predecessor	the previous node on the identifier circle

Scalable Key Location (con.)

- Each node knows m other nodes in the ring

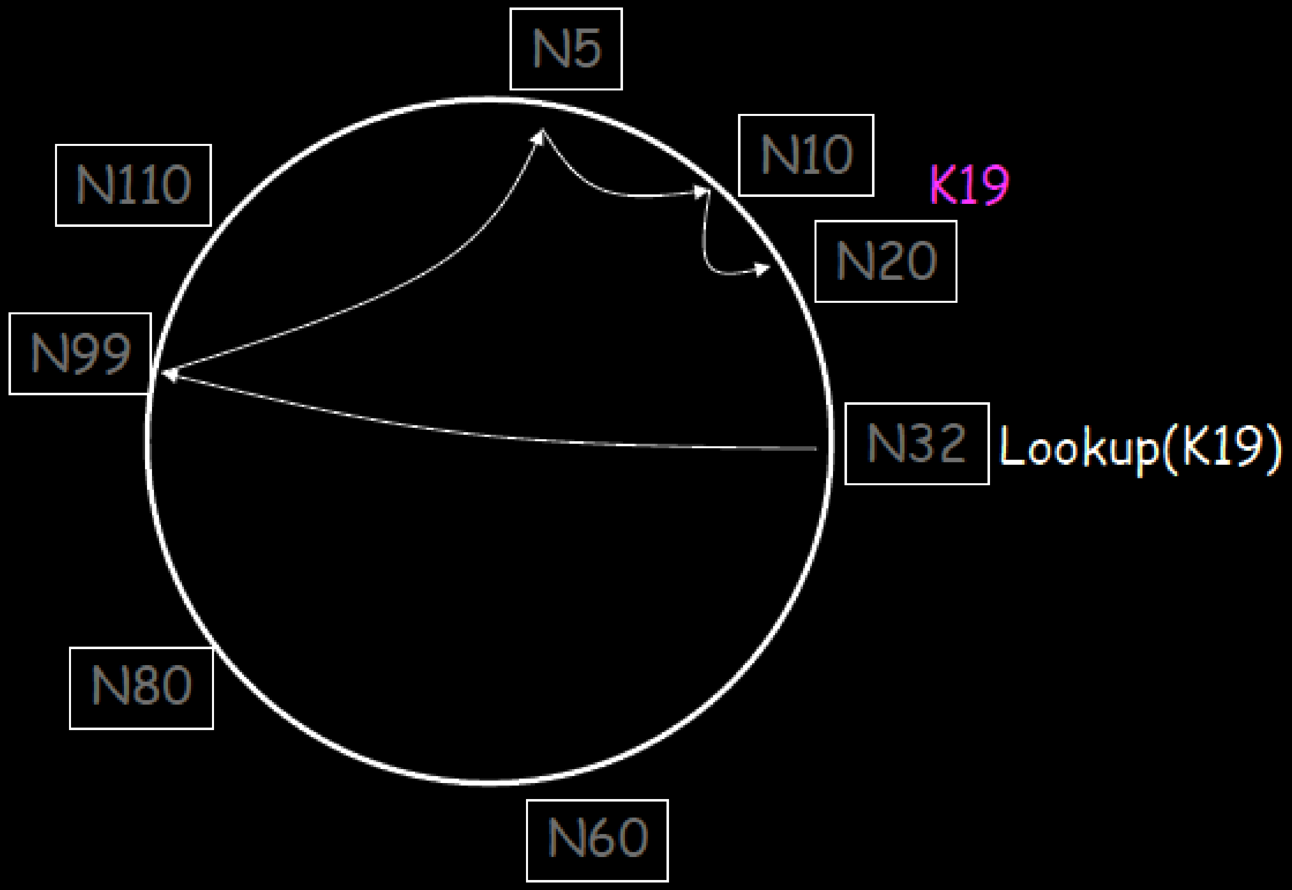


Scalable Key Location (con.)



Scalable Key Location (con.)

- Lookups take *O(log N)* hops



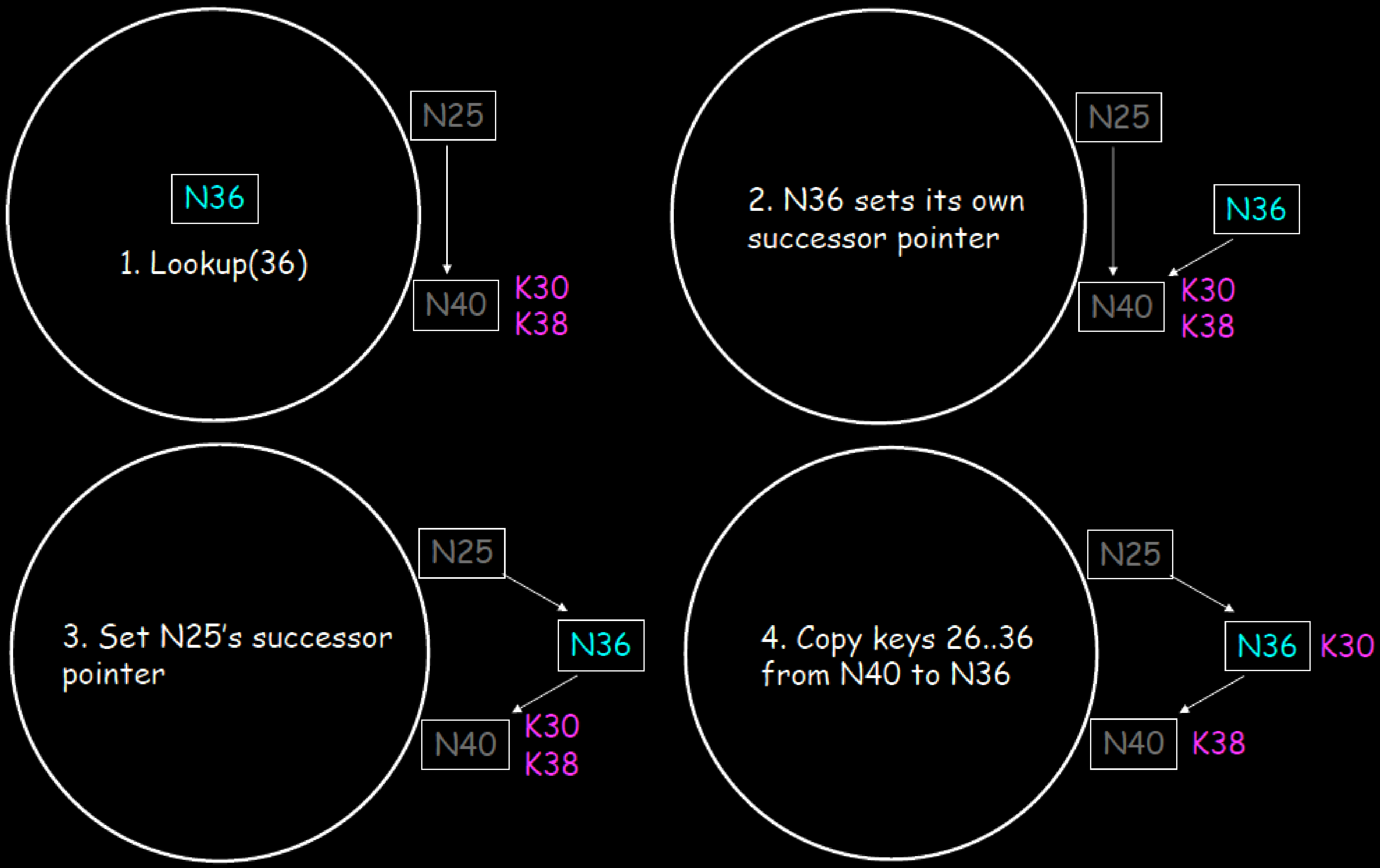
Outline

- What is Chord?
- Chord hash lookup
- Maintain routing table
- Simulation

Stabilizing

- Functionality:
  – To handle concurrent node joins/fails/leaves
- Operation:
  – Keep successor pointers up to date, then verify and correct finger table entries
  – Nodes periodically run stabilization protocol

Node Joins


N25
N36
1. Lookup(36)
N40
K30
K38
N25
2. N36 sets its own successor pointer
N36
N40
K30
K38
N25
3. Set N25's successor pointer
N36
N40
K30
K38
N25
4. Copy keys 26..36 from N40 to N36
N36
K30
N40
K38

Failure Recovery

Successor Lists:

- Each node knows *r* immediate successors
- After failure, will know first live successor
- Correct successors guarantee correct lookups
- Guarantee is with some probability
  - Can choose *r* to make probability of lookup failure arbitrarily small

Outline

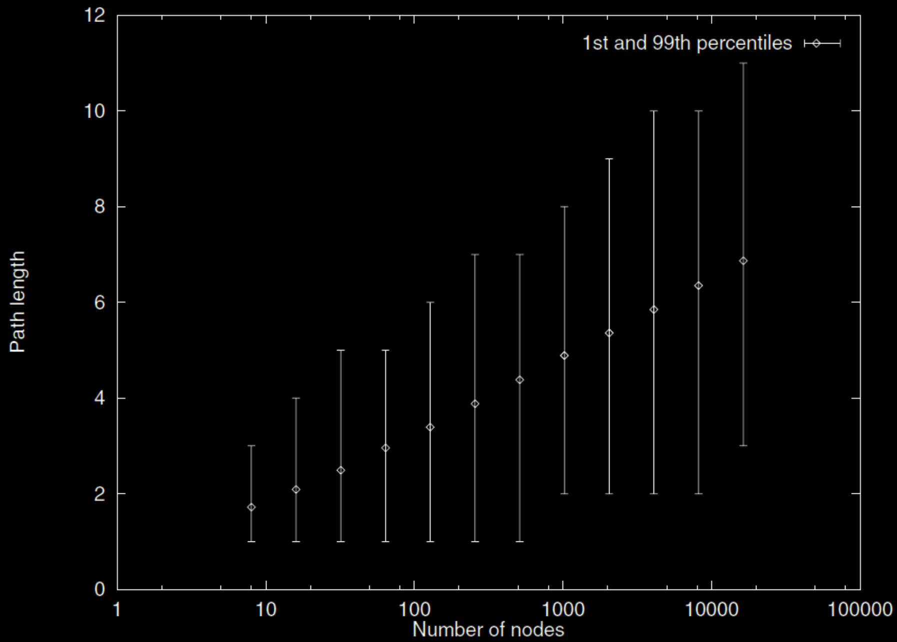
- What is Chord?
- Chord hash lookup
- Maintain routing table
- Simulation

Simulation

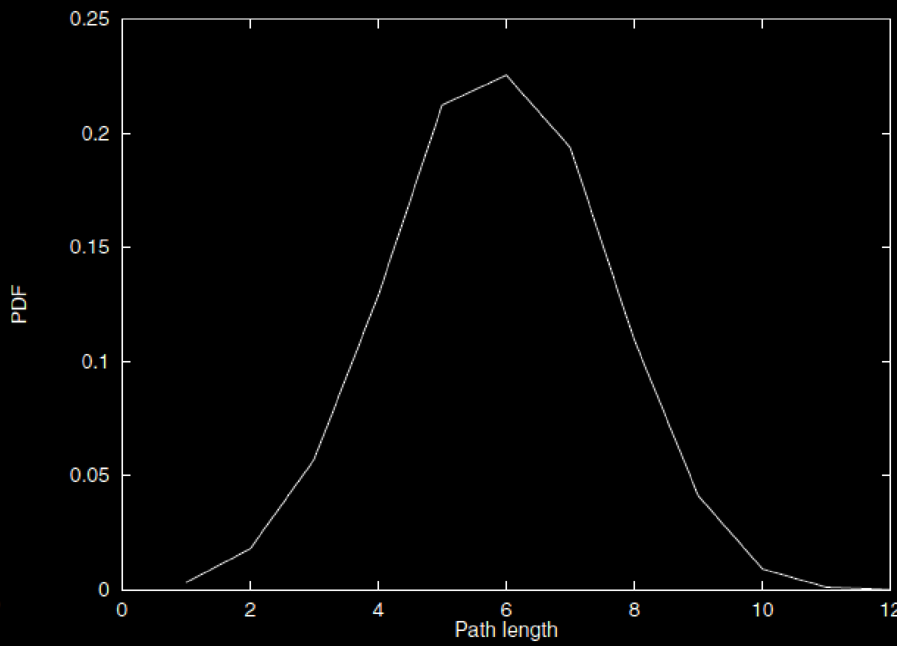
- Network Scale:
  - 10^4 nodes & 10^5 to 10^6 keys
- Chord Implementation:
  - Iterative (Recursive)
- Results confirm theoretical analysis:
  - Efficiency
  - Scalability
  - Robustness

Path Length

PL as a function of Network Size

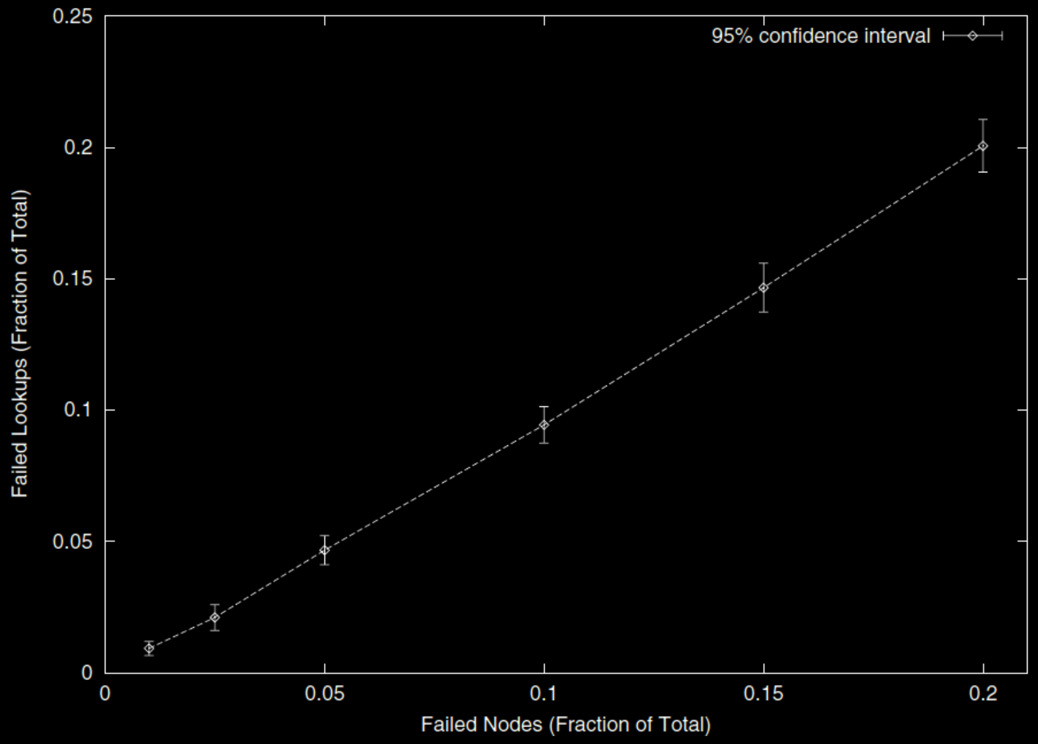


PDF of the PL in the case of a 2^{12} node network

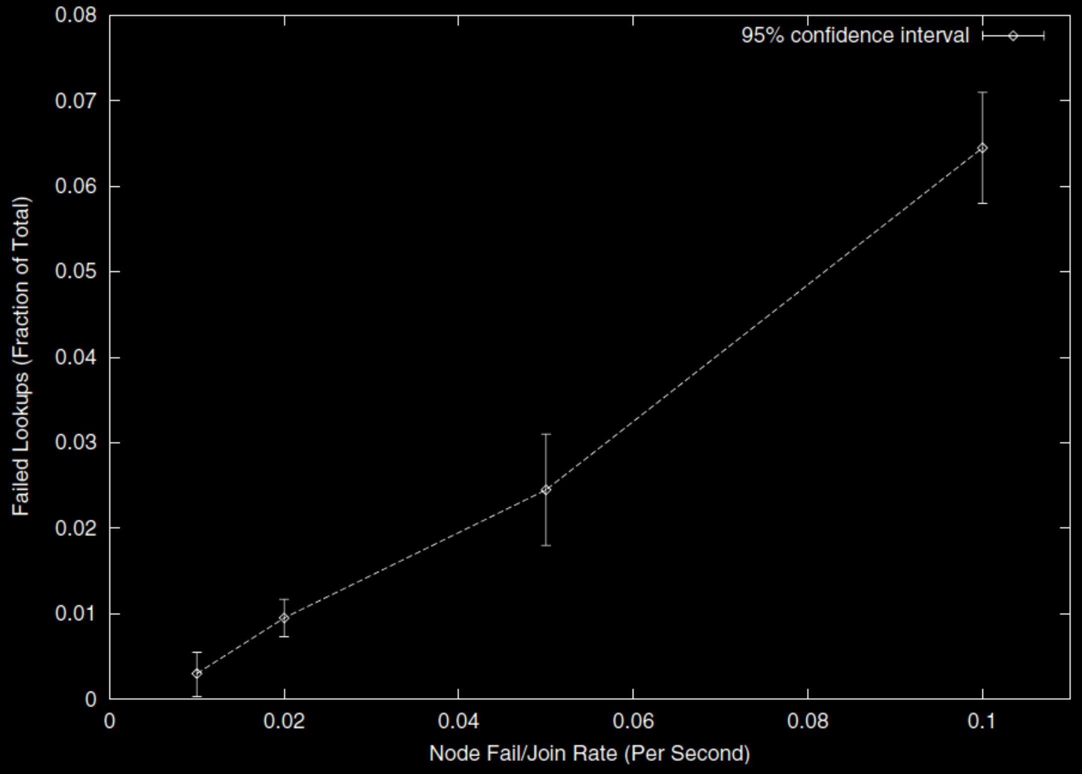


Lookup Cost is *O(log N)*

Failed Lookups -- Failed Nodes

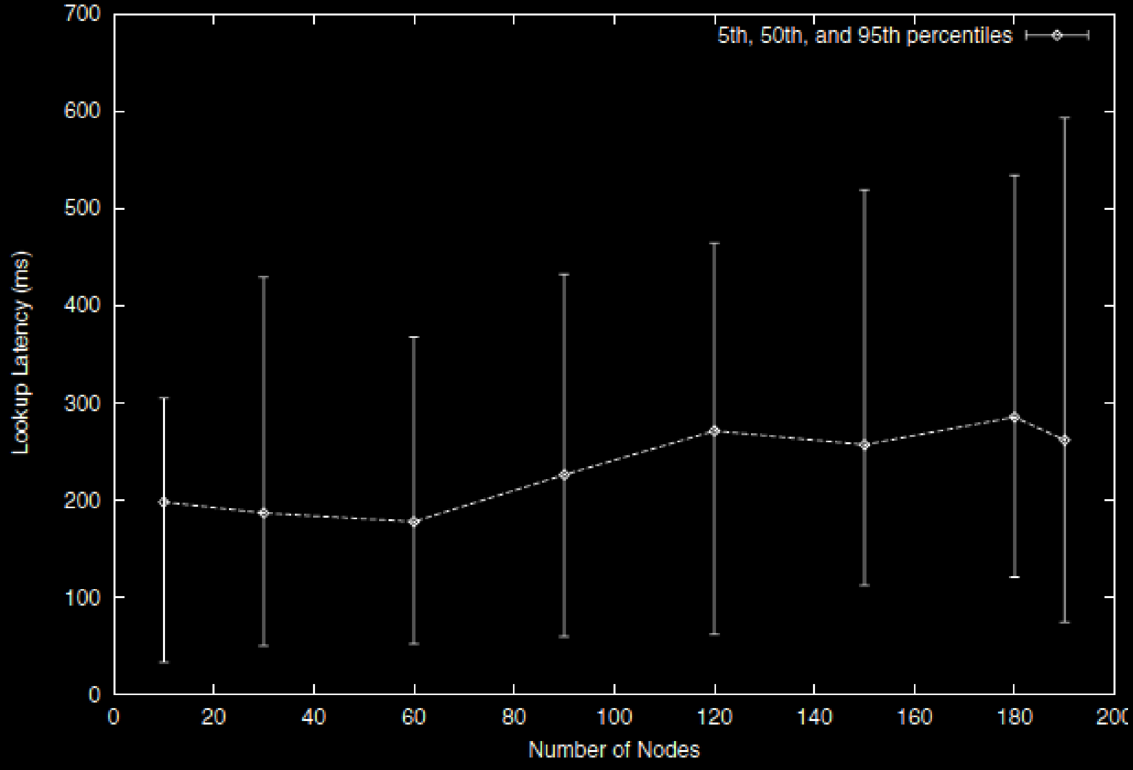


Failed Lookups – Node Fail/Join Rate



Experimental Results

- Chord Prototype



Chord Summary

- Chord provides peer-to-peer hash lookup service
- Efficient:
  - *O(log N) messages per lookup*
- Scalable:
  - *O(log N) states per node*
- Robust:
  - Survives massive failures, joins or leaves
- Good primitive for peer-to-peer systems

Limitations:

- No anonymity (Chord designates nodes for data items)
- Network locality is not well exploited

The Impact of DHT Routing Geometry on Resilience and Proximity

Krishna Gummadi (MPI-SWS)

Ramakrishna Gummadi (USC)

Steven Gribble (U Washington)

Sylvia Ratnasamy (Intel)

Scott Shenker (UC Berkeley)

Ion Stoica (UC Berkeley)

Takeaway Points

- Comparison of Different Geometries
  - Ring, Tree, Hypercube, Butterfly, XOR
- Flexibility
  - Flexibility Neighbor Selection (FNS)
  - Flexibility Routing Selection (FRS)
- Static Resilience
- Path Latency
  - Proximity methods (PRS & PNS)

Outline

- DHTs Design
- Static Resilience
- Proximity

Motivation

- New DHTs constantly proposed
- Isolated analysis

Goals:

- Separate fundamental design choices from algorithmic details
- Understand the impact of different DHT routing geometries on reliability and efficiency

Component-based Analysis

- Break DHT design into independent components
  - *Routing-level:* neighbor & route selection
  - *System-level:* caching, replication, querying policy etc.
- Analyze impact of each component choice separately compare with black-box analysis

Geometry & Algorithm

- *Algorithm* : exact rules for selecting neighbors, routes
  - Chord, CAN, Tapestry, Pastry, etc.
- *Geometry* : an algorithm's underlying structure that inspires a DHT design
  - Distance function is the formal representation of Geometry
  - Many algorithms can have same geometry:
    - Chord, Symphony => Ring

Comparison

Geometry	Algorithm
Ring	Chord, Symphony
Hypercube	CAN
Tree	Plaxton
Butterfly	Viceroy
Hybrid = Tree + Ring	Tapestry, Pastry
XOR d(id1, id2) = id1 XOR id2	Kademlia

Flexibility

- The algorithmic freedom left after the geometry is chosen
  - Neighbor selection
    - *FNS:* number of node choices for a neighbor
  - Route selection
    - *FRS:* average number of route choices for a destination

property	tree	hypercube	ring	butterfly	xor	hybrid
Neighbor Selection	$n^{\log n/2}$	1	$n^{\log n/2}$	1	$n^{\log n/2}$	$n^{\log n/2}$
Route Selection	1	$c_1(\log n)$	$c_1(\log n)$	1	1	1

Geometry => Flexibility => Performance

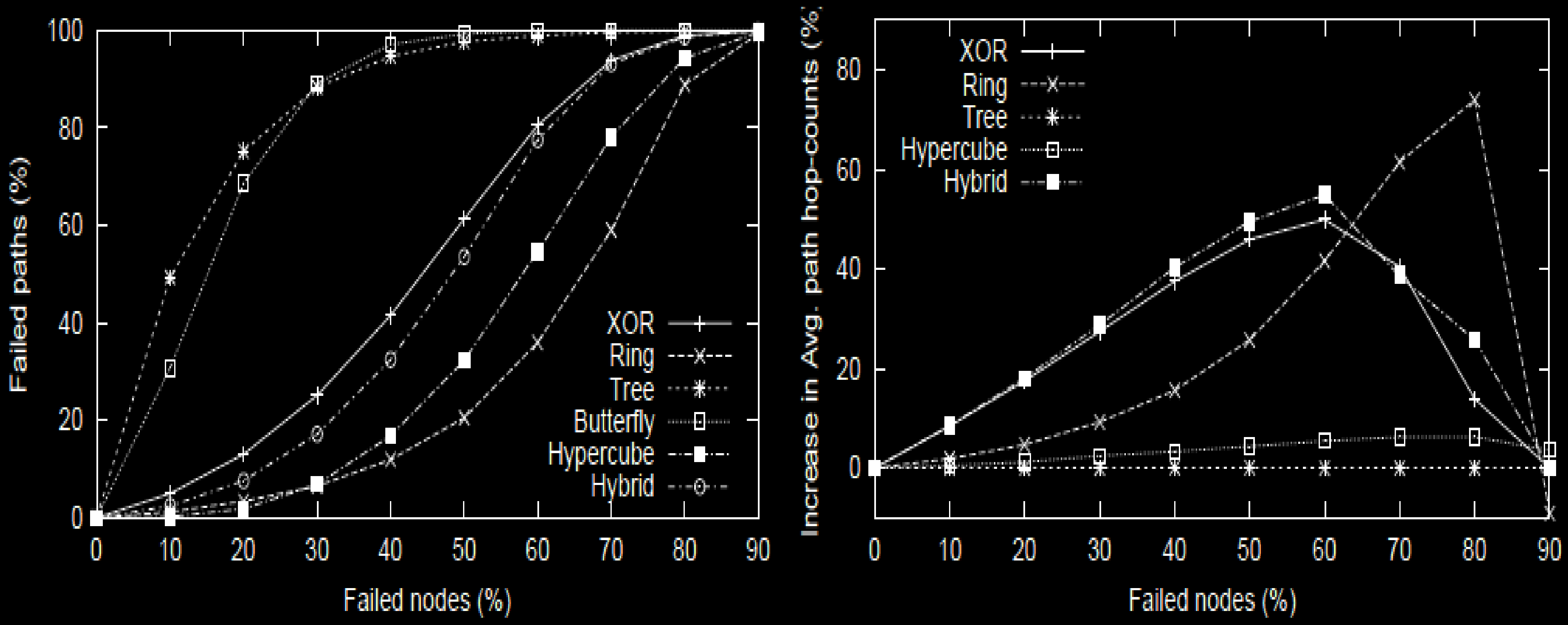
Outline

- DHTs Design
- Static Resilience
- Proximity

Static Resilience

- Resilience:
  - Robust Routing
- Static Resilience:
  - One of the three aspects of resilience
  - We keep the routing table static (except for deleting failed nodes)
  - Measures the extent to which DHTs can route around trouble
- Evaluation metrics:
  - % paths failed
  - % increase in path length

Static Resilience & Geometries



Flexibility

=> Static Resilience

Outline

- DHTs Design
- Static Resilience
- Proximity

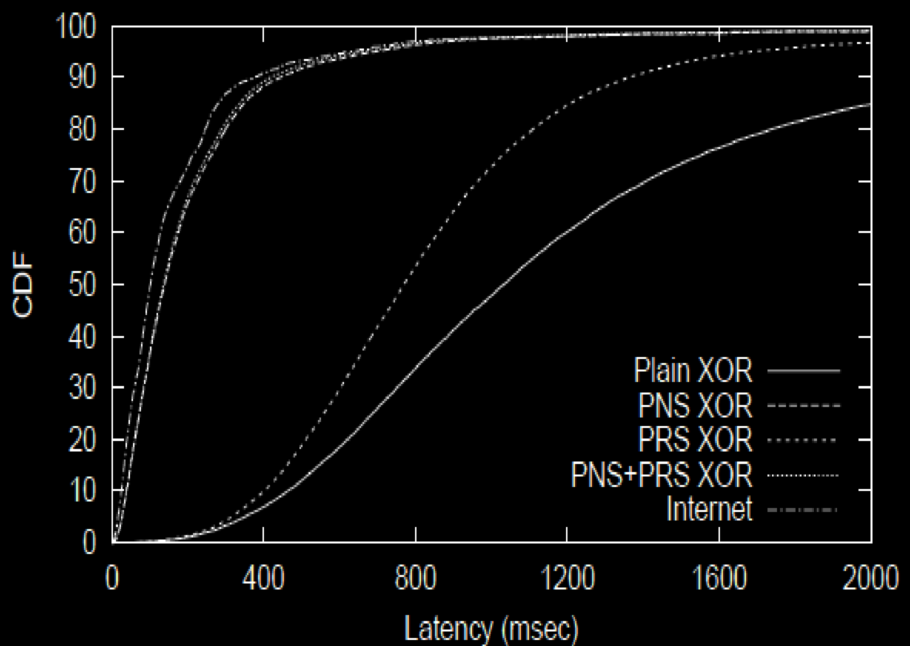
Path Latency

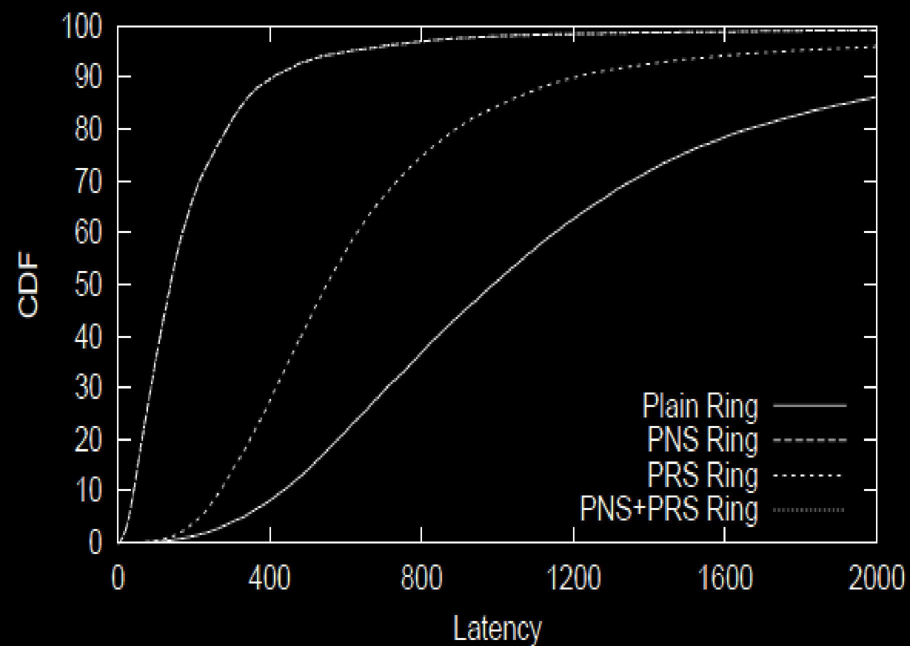
- DHTs are designed to route effectively in terms of *hopcount*
- End-to-end latency issues approached through *proximity methods*
  - Proximity Neighbor Selection (PNS): neighbors are chosen based on proximity
  - Proximity Route Selection (PRS): the choice of next-hop when routing to a destination depend on the proximity of neighbors
  - *Proximity Identifier Selection (PIS)

Proximity

- Goal: Minimize end-to-end overlay path latency
- Both PNS and PRS can reduce latency
  - Tree supports PNS, Hypercube supports PRS, Ring & XOR have both

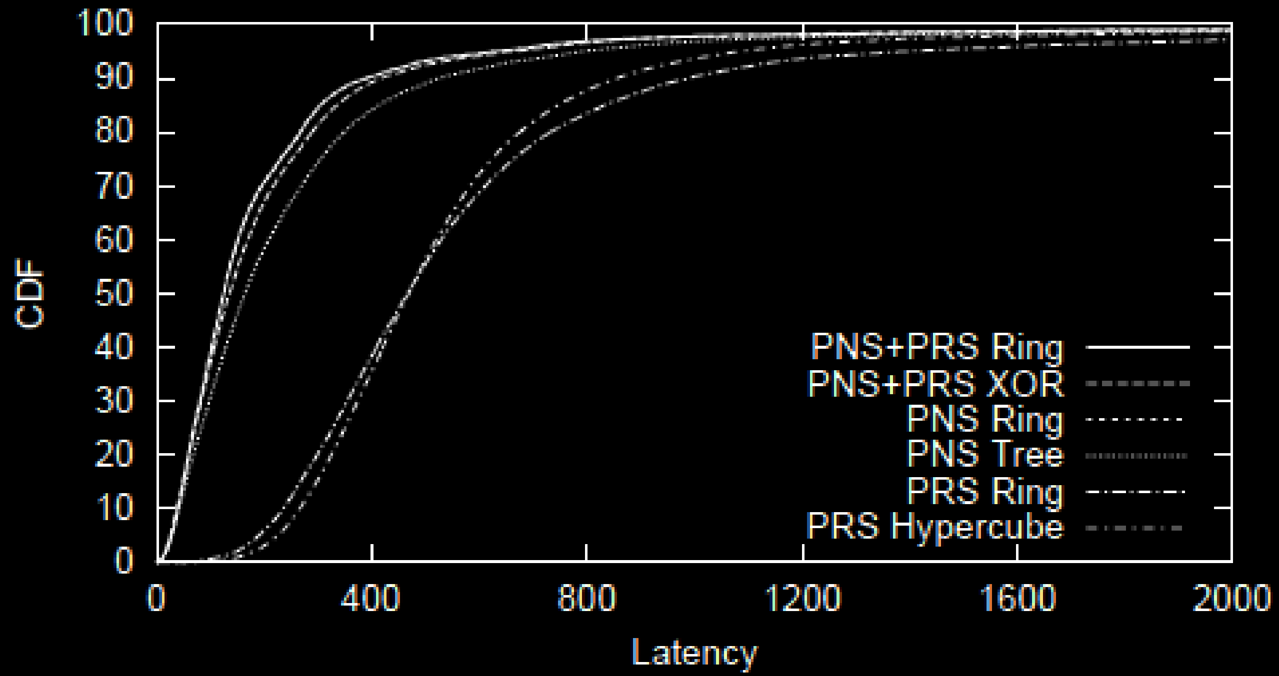
PNS or PRS?





- Plain << PRS << PNS ≈ PNS+PRS

Does Geometry Matter?



Proximity Summary

- Both addressed path latency issues
- Performance
- Independency

Flexibility

=> Path Latency

Limitations

- Geometry?
  - a distance function on an identifier space
- Other factors of DHTs?
  - Algorithmic details, symmetry in routing table entries
- Completeness?
  - Other DHT algorithms

Conclusion

- Routing Geometry is a fundamental design choice
  - Geometry => Flexibility
  - Flexibility => Performance (Resilience & Proximity)
- Ring has the greatest flexibility
  - Great routing performance

Why not the Ring?

Thank You!