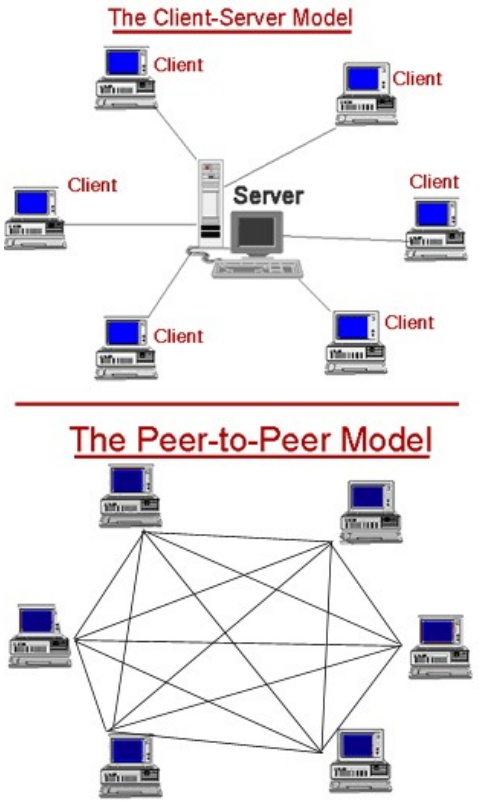
Scalability In Peer-to-Peer Systems

Presented by Stavros Nikolaou

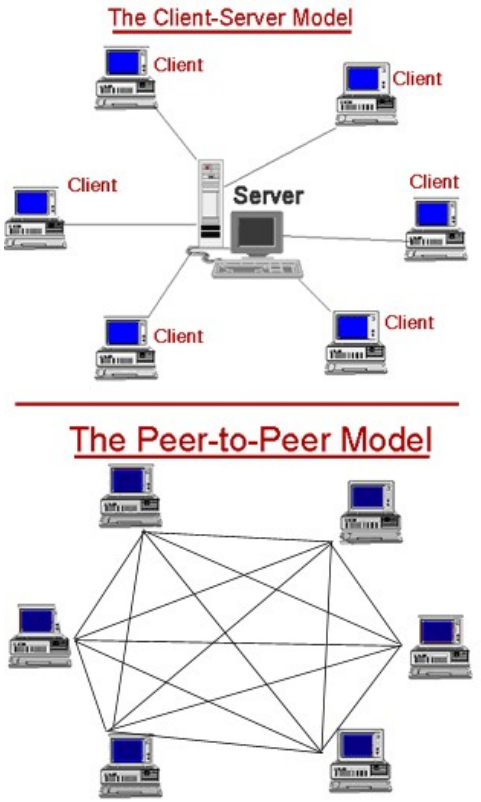
Background on Peer-to-Peer Systems

- Definition:
  - Distributed systems/applications featuring:
    - No centralized control, no hierarchical organization
    - Peers (nodes) of equal functionality



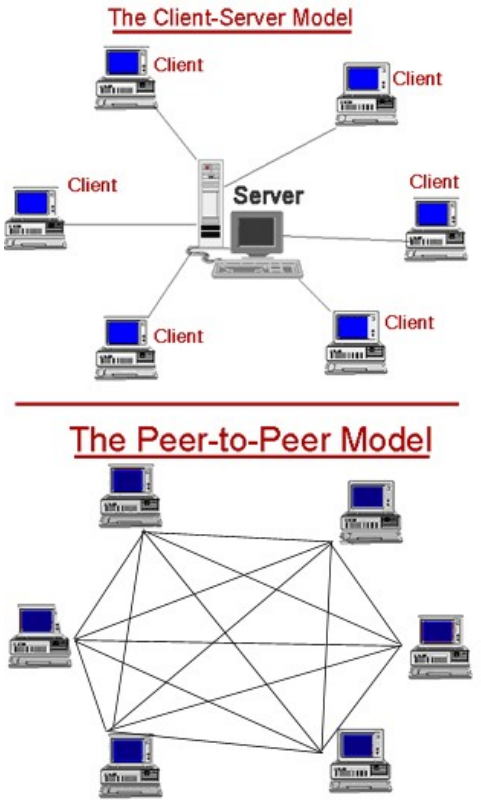
Background on Peer-to-Peer Systems

- Featured Applications:
  - File sharing (Napster, Bit Torrent etc.)
  - Content Delivery (PPLive, FreeCast, CoralCDN etc.)
  - Domain Name Systems
  - Communication networks (Skype)



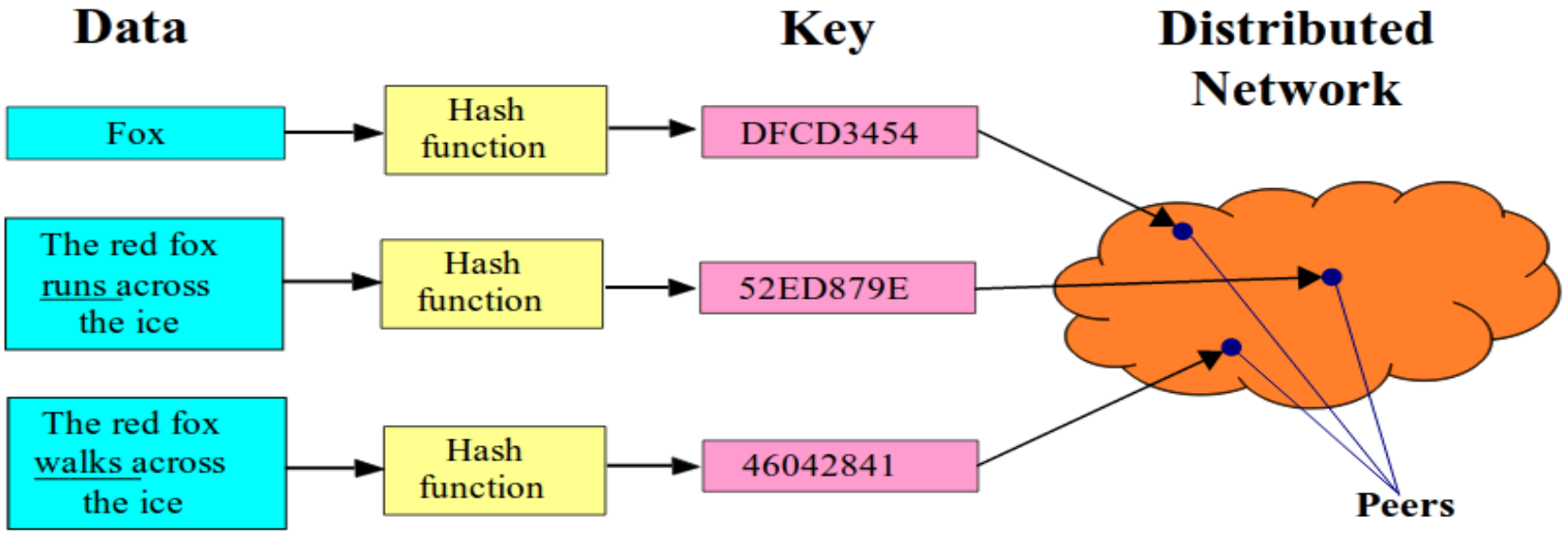
Background on Peer-to-Peer Systems

- Architecture:
  - Structured: organized following specific criteria and algorithms
    => overlays with specific topologies and properties
    - Common Case: Distributed Hash Tables (DHTs)
  - Unstructured: pure, hybrid, centralized



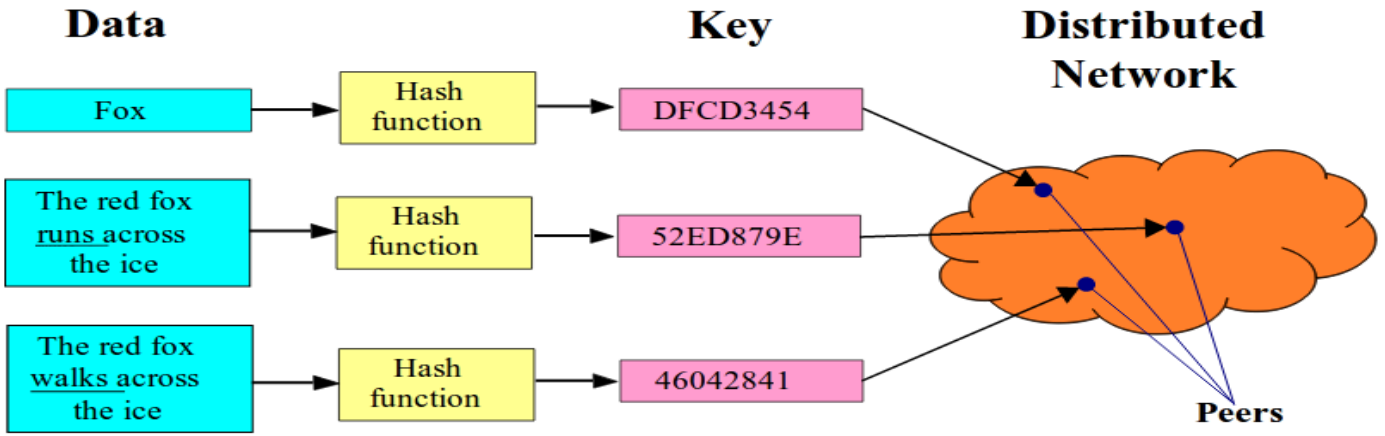
Background on DHTs

- DHT: decentralized, distributed system providing a hash-table like lookup service
  - Responsibility for maintaining (key, value) pairs: distributed among nodes
    - Minimize maintained state and adaptation costs
    - Tolerant to network changes (joins, leaves, failures of nodes)
    - Highly scalable to the size of the network



Background on DHTs

- DHT: decentralized, distributed system providing a hash-table like lookup service
  - Key-based routing:
    - Nodes and objects → IDs
    - Similar IDs → similar set of nodes
    - Suitable for “exact matches” but not for keyword matches (no closeness notion)



Background on DHTs

- DHT: decentralized, distributed system providing a hash-table like lookup service
  - Used for: Constructing distributed services/applications
  - Motivation: P2P systems such as *Chord, CAN, Pastry,* and *Tapestry*
  - Applications: BTDigg (Bit Torrent search engine), CoralCDN, Freenet (distributed data storage)

Summary

- Problem:
  - Myriad DHT designs in literature
  - Which one is the best (if there is one globally accepted)?
  - How do we choose?

Papers

- *Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications*
  - Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, Hari Balakrishnan

- *The Impact of DHT Routing Geometry on Resilience and Proximity*
  - K. Gummadi, R. Gummadi, S. Gribble, S. Ratnasamy, S. Shenker, I. Stoica

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

- Background of authors:
  - Ion Stoica: Professor of Computer Science Division at University of California, Berkeley
  - Robert Morris: Professor of EECS at MIT, Cambridge, MA
  - David Liben-Nowell: Associate Professor in the Department of Computer Science at Carleton College
  - David R. Karger: Professor of EECS at MIT, Cambridge, MA
  - M. Frans Kaashoek: Professor of EECS at MIT, Cambridge, MA
  - Frank Dabek: Engineer at Google
  - Hari Balakrishnan: Professor of EECS at MIT, Cambridge, MA

Summary

- Problem:
  - *Efficiently* locate the node that stores a particular data item in a peer-to-peer (P2P) network
- Motivation:
  - Location of data items is a core operation of many P2P systems
  - How to map a hash-table to a dynamic distributed set of nodes
- Contribution:
  - Chord – a distributed, scalable protocol for lookup in dynamic P2P systems
  - Features:
    - Efficiently Adaptable: churn (nodes joining and leaving)
    - Scalable: communication and maintained state costs scale logarithmically with the number of nodes

Take Away Messages

- Chord: Peer-to-Peer hash lookup protocol
- Efficiency: O(logN) messages for locating a key
- Scalability: O(logN) state size
- Robustness: surviving massive node failures/joins
  - Eventual succession of lookups
- *Promising infrastructure for scalable Peer-to-Peer applications*

System Model Basics

- Chord:
  - Scalable, distributed protocol for lookup operations in P2P networks
    - Only operation: key → node

- Driving Principles:
  - Load Balancing: evenly spread keys over all nodes
  - Decentralization: no node is more important than any other
  - Scalability: lookup cost grows logarithmically to the number of nodes
  - Availability: node responsible for a key can always be found
  - Flexible Naming: no constraints on the structure of the keys it lookups

- Example applications:
  - cooperative mirroring (replication),
  - timed shared storage (continuous availability),
  - distributed indexes (keyword searching)

Outline: Protocol Overview

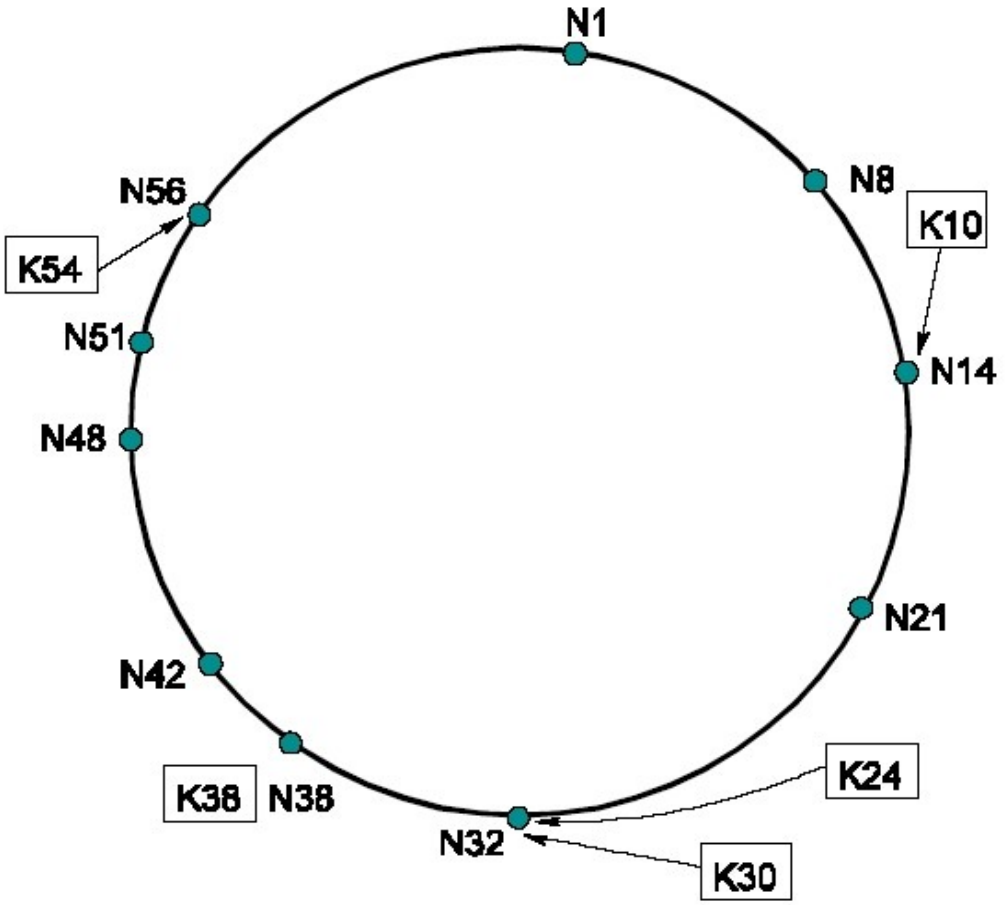
- Locating Keys
  - Consistent Hashing: Determines key space and mapping.
  - Routing: Determines how lookup requests for keys reache their corresponding destination nodes.
- Joining Nodes
  - Determines how the system reacts to dynamically joining nodes.
- Failure Recovery
  - Determines how node failures are tolerated.

Outline: Protocol Overview

- Locating Keys
  - Consistent Hashing: Determines key space and mapping.
  - Routing: Determines how lookup requests for keys reache their corresponding destination nodes.
- Joining Nodes
  - Determines how the system reacts to dynamically joining nodes.
- Failure Recovery
  - Determines how node failures are tolerated.

Consistent Hashing

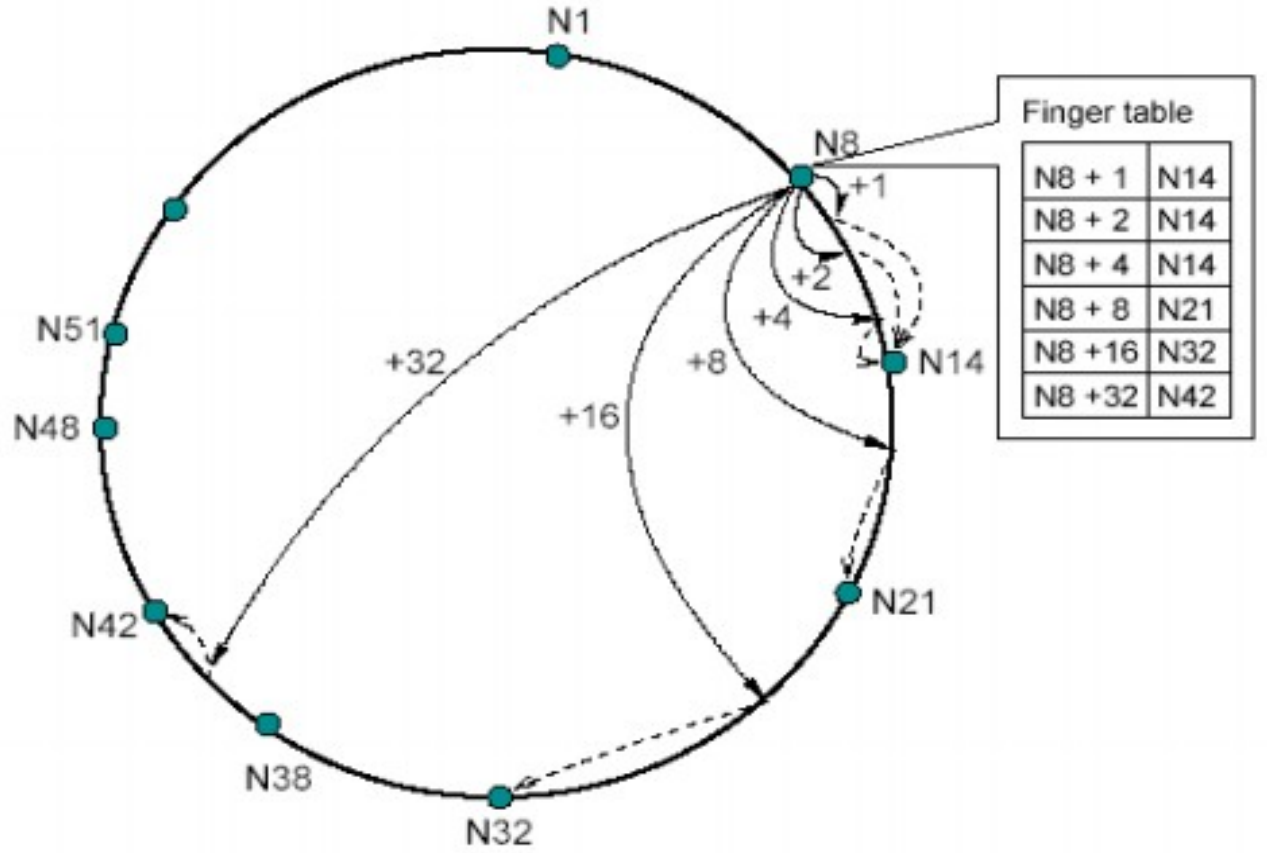
- Identifier Space:
  - m-bit ids for both nodes and keys
  - Key id = SHA-1(key)
  - Node id = SHA-1(node IP)
- IDs ordered in identifier circle mod 2^m
- A key with id *k* is mapped to the node *n* whose id is equal or follows *k* in the identifier space. *n* is called the successor of *k.*



Outline: Protocol Overview

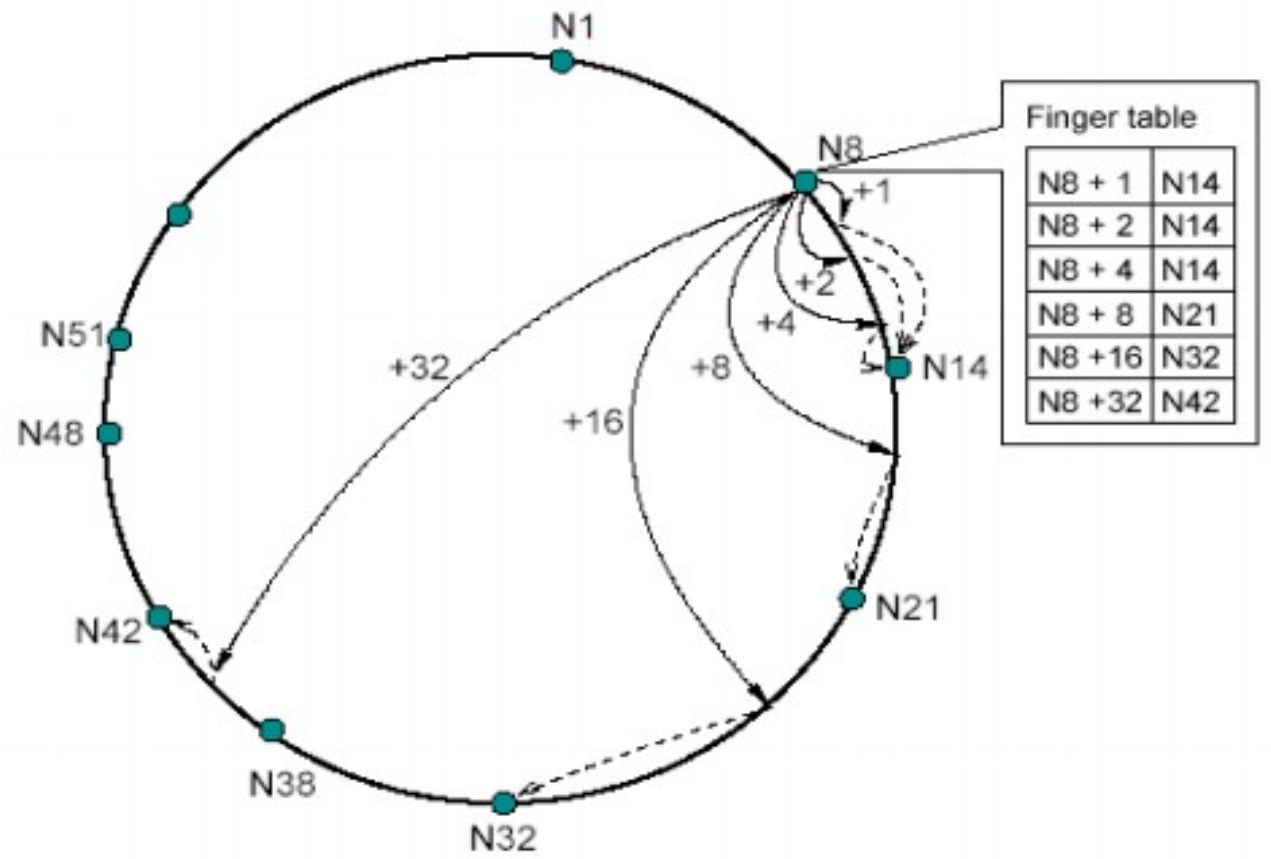
- Locating Keys
  - Consistent Hashing: Determines key space and mapping.
  - Routing: Determines how lookup requests for keys reache their corresponding destination nodes.
- Joining Nodes
  - Determines how the system reacts to dynamically joining nodes.
- Failure Recovery
  - Determines how node failures are tolerated.

Routing



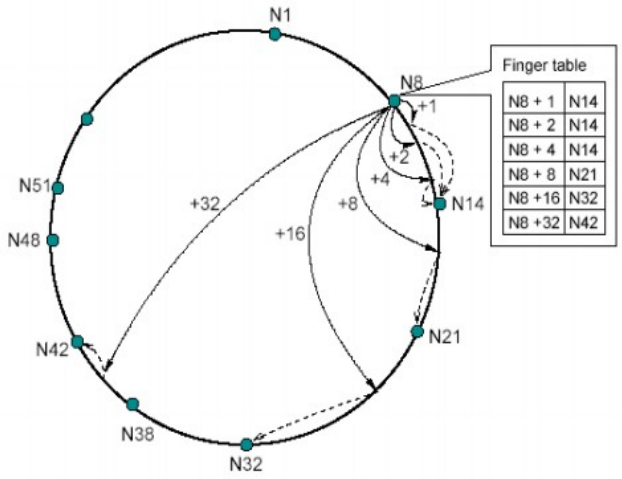
- Each node maintains a table of m entries (finger table)
- i^{th} entry:id (and IP) of the first node whose clockwise id distance from the current node is at least 2^{i-1}.
- Each node:
  - Stores information about a small amount of nodes (mostly the closer ones)
  - The successor of any arbitrary key cannot always reside in the finger table

Routing

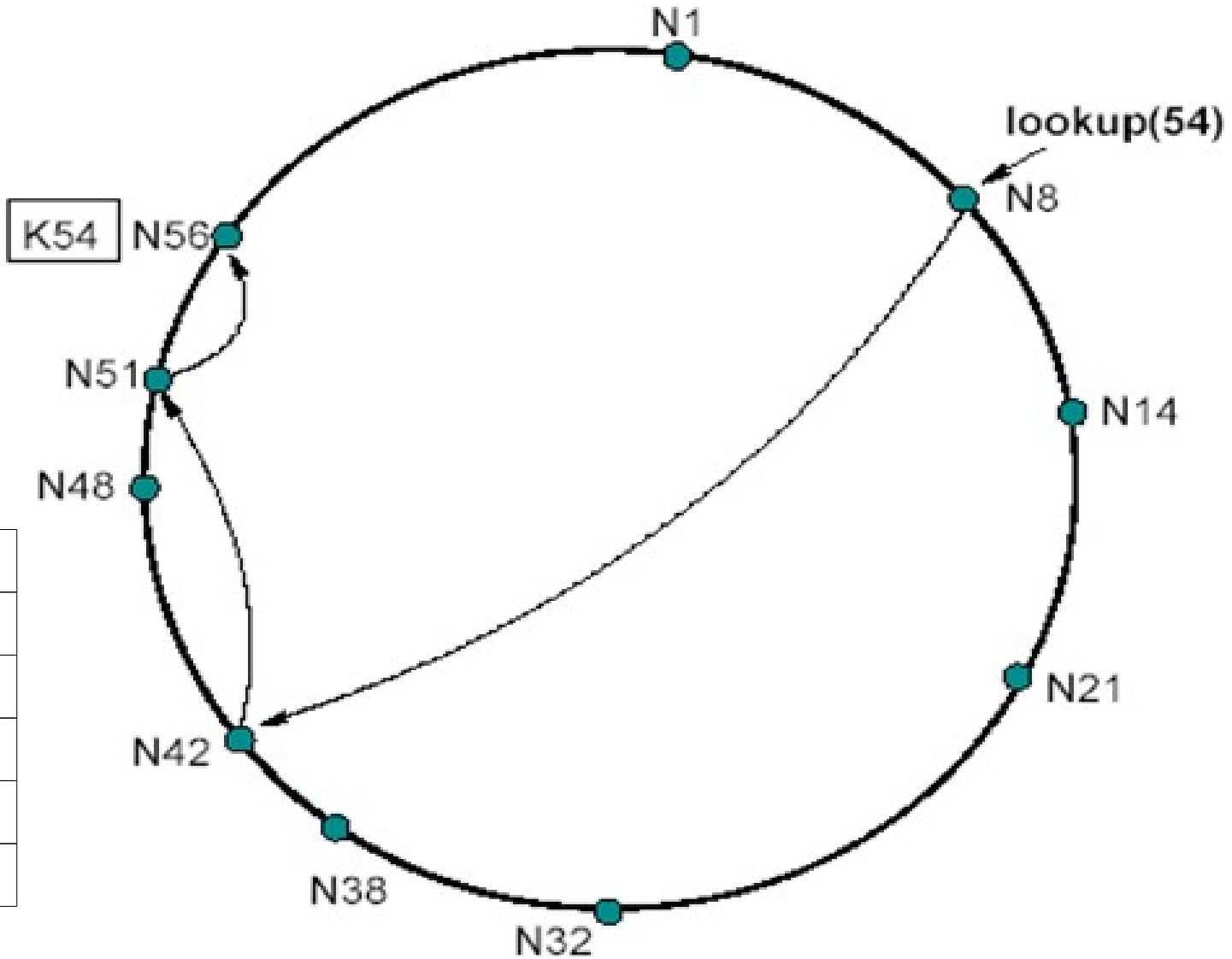


- Each time the request is forwarded towards the closest known predecessor of the key.
- Each such forward results in halving the distance to the target identifier.
- Due to randomness (uniform node id distribution on the identifier circle) → Complexity: *O(logN) with high probability*, N = size of the network

Routing



N42 + 1	N48
N42 + 2	N48
N42 + 4	N48
N42 + 8	N51
N42 + 16	N1
N42 + 36	N14

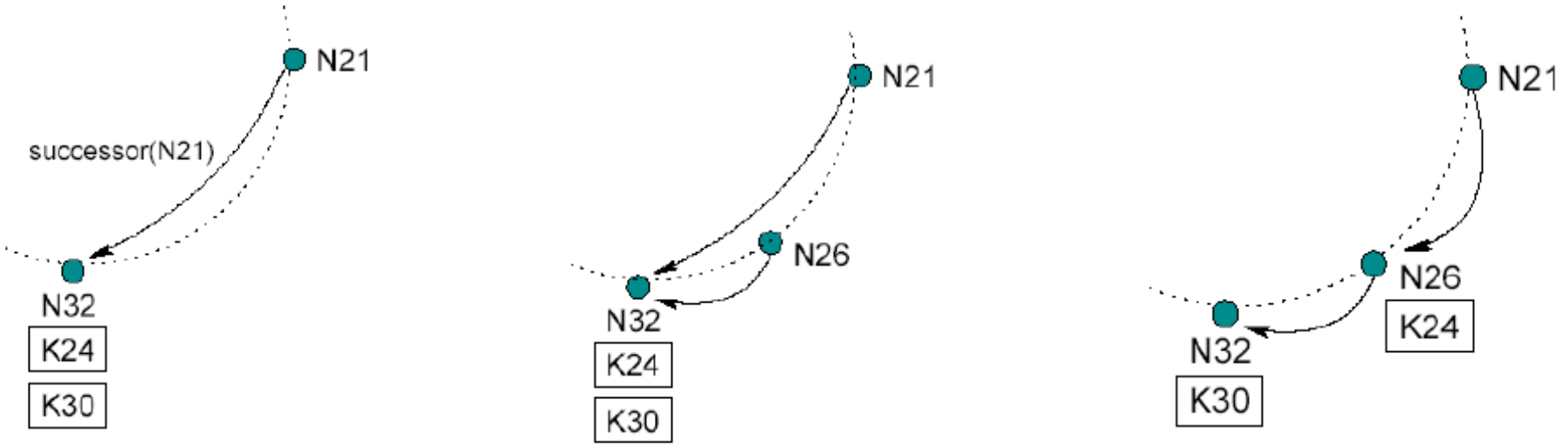


Outline: Protocol Overview

- Locating Keys
  - Consistent Hashing: Determines key space and mapping.
  - Routing: Determines how lookup requests for keys reache their corresponding destination nodes.
- Joining Nodes
  - Determines how the system reacts to dynamically joining nodes.
- Failure Recovery
  - Determines how node failures are tolerated.

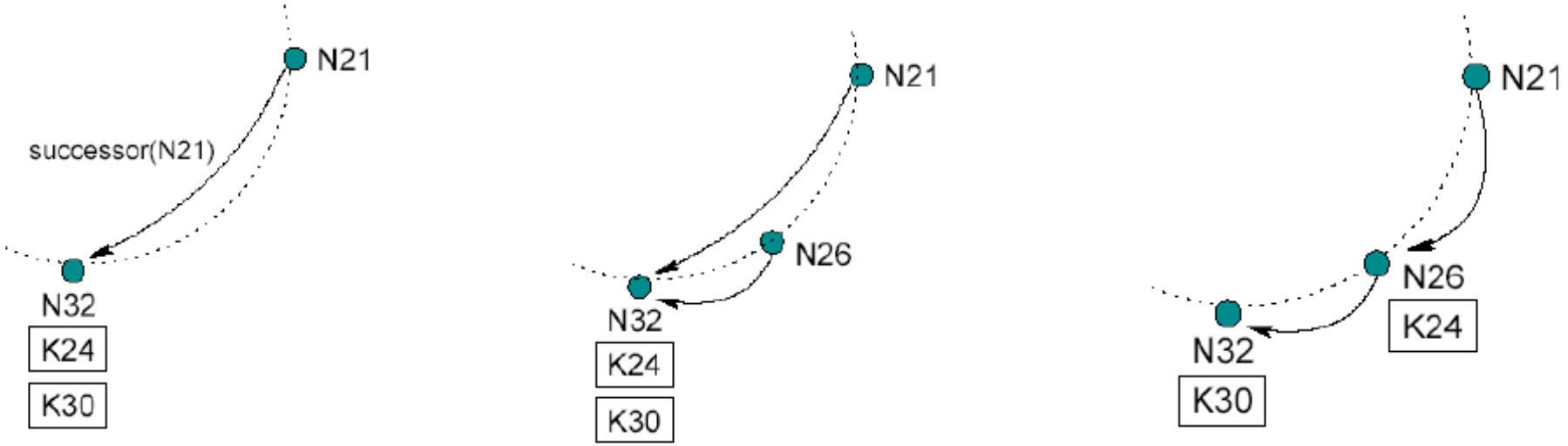
Node Join and Stabilization

- Invariants:
  - Nodes' successors must always be correctly maintained
  - successor(*k*) must always be responsible for *k*.
  - Complexity: $O(log^2 N)$ (including the messages required for fixing the finder tables)



- Each node additionally stores its predecessor

Node Join and Stabilization



Stabilization:

- For concurrent joins of multiple nodes
- Keeps successor pointers up to date; verifies and corrects finger tables.
- Runs periodically

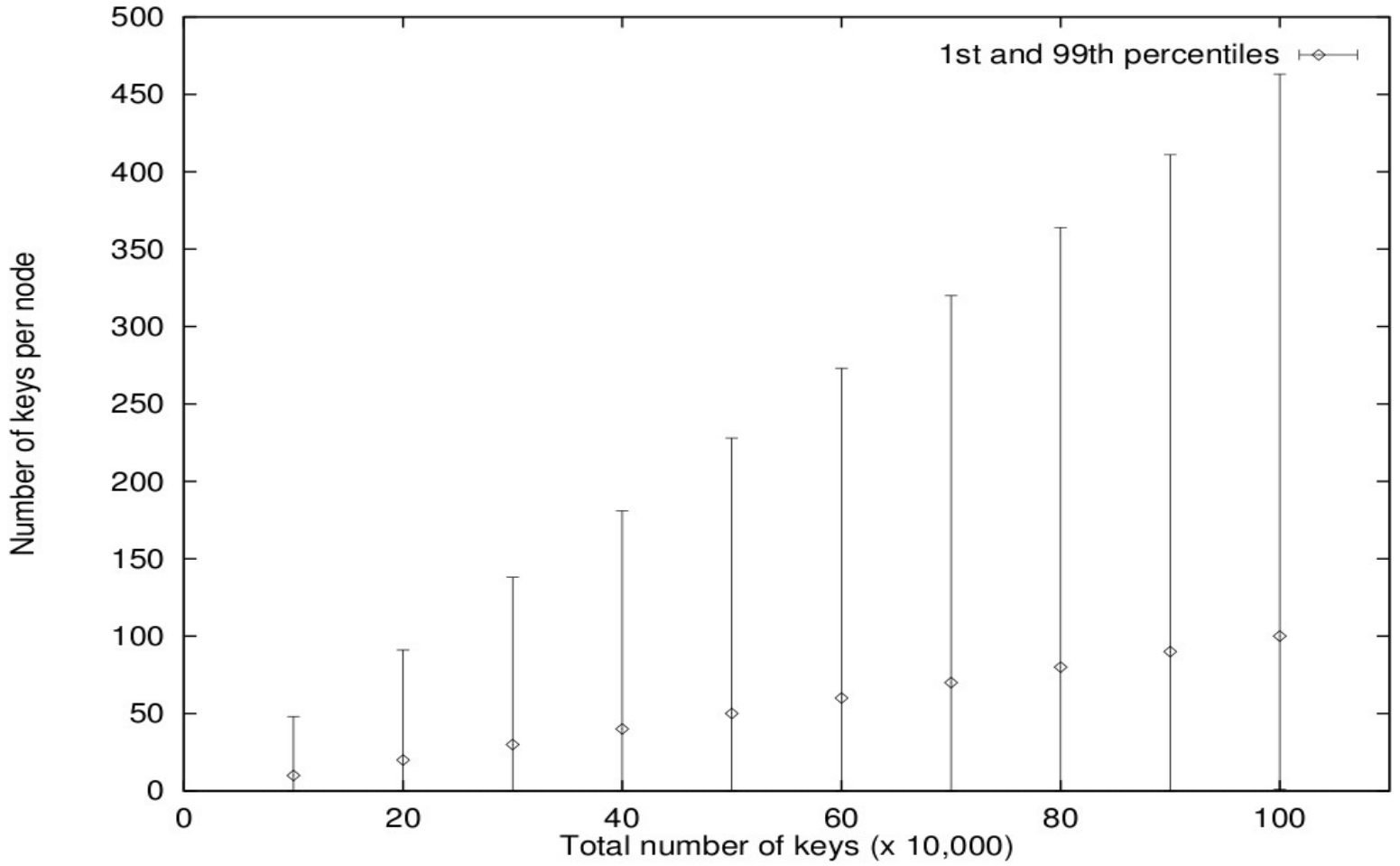
Outline: Protocol Overview

- Locating Keys
  - Consistent Hashing: Determines key space and mapping.
  - Routing: Determines how lookup requests for keys reache their corresponding destination nodes.
- Joining Nodes
  - Determines how the system reacts to dynamically joining nodes.
- Failure Recovery
  - Determines how node failures are tolerated.

Failure Recovery

- Successor lists:
  - Each node maintains a list with its *r* immediate successors
  - When a node fails, its predecessor will now next alive successor
  - Successors maintained: Correctness is guaranteed
  - Choosing *r* for making lookup failure probability arbitrarily small.

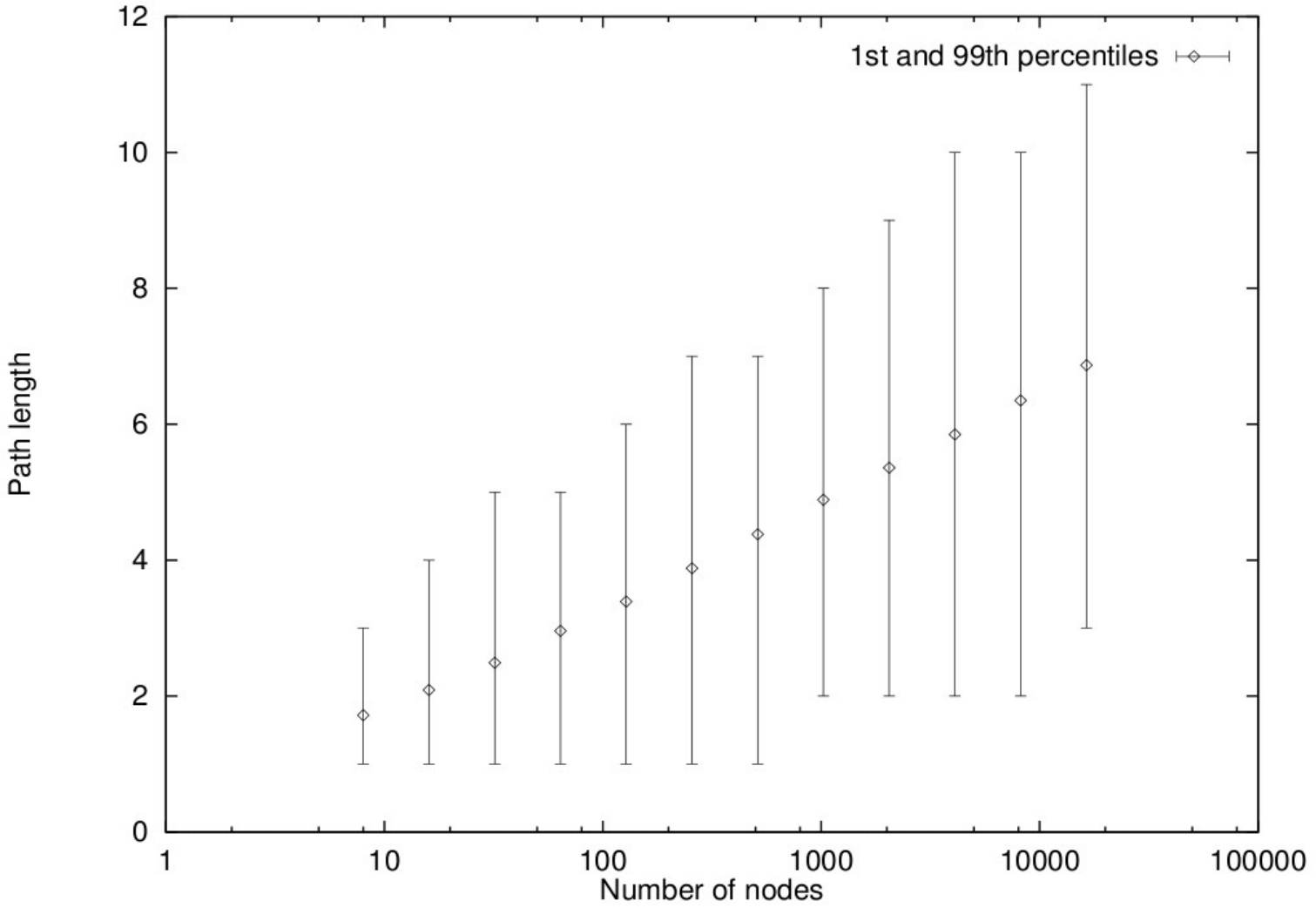
Evaluation (Simulation): Load Balance



The mean and 1st and 99th percentiles of the number of keys stored per node in a 10^4 node network.

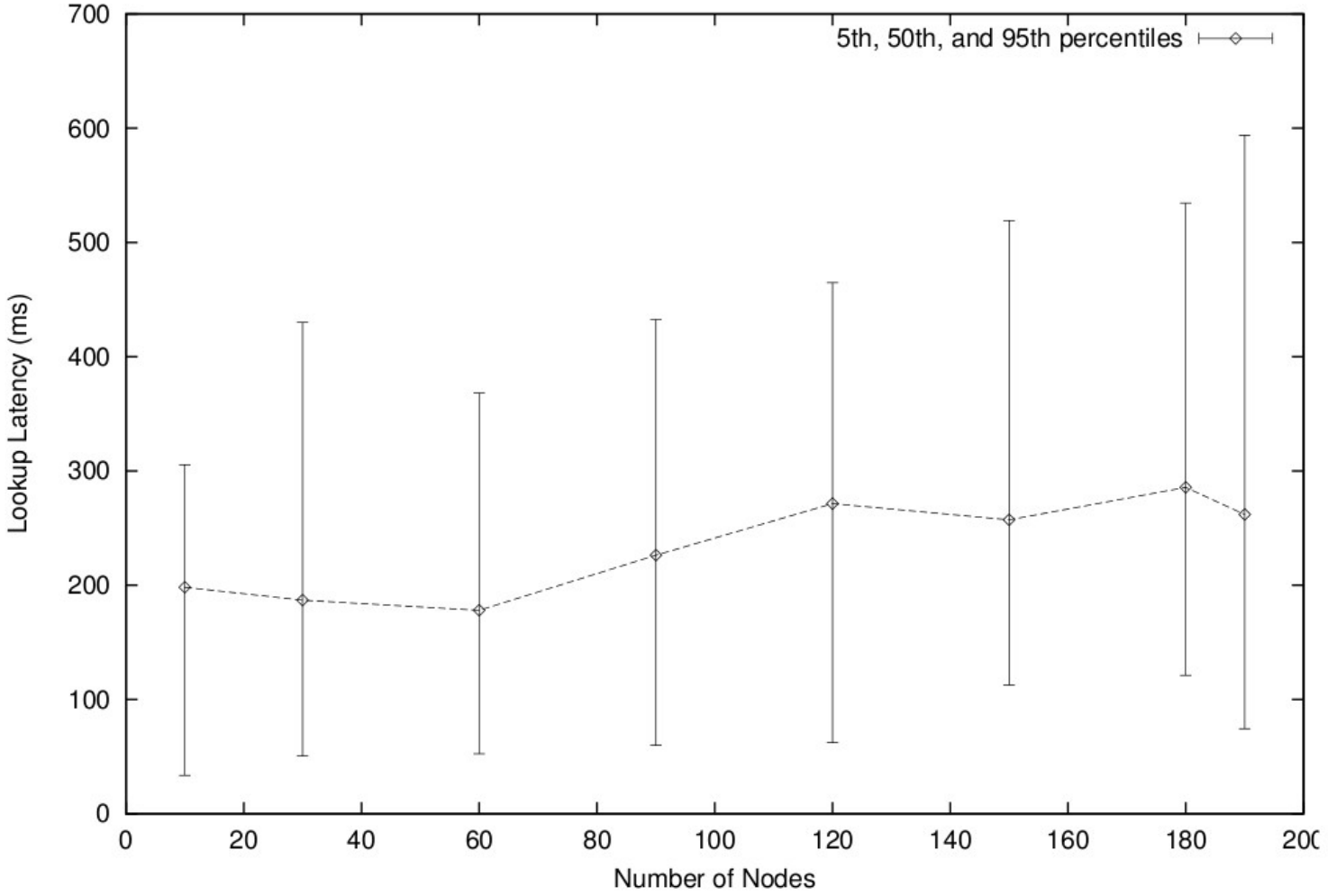
Great variations – practically non uniform distribution of nodes on the ring

Evaluation (Simulation): Path Length



The path length as a function of network size. Indeed the lookup cost is O(logN)

Evaluation (Experimental Results): Lookup Latency



Lookup latency grows slowly with the total number of nodes!

Conclusions & Discussion

- Chord: Peer-to-Peer hash lookup protocol
- Efficiency: O(logN) messages for locating a key
- Scalability: O(logN) state size
- Robustness: surviving multiple node failures/joins
  - Eventual lookups success
- *Promising infrastructure for scalable Peer-to-Peer applications*

- What's Wrong!

Conclusions & Discussion

- Issues, Open Questions:
  - Pathological Cases: Partitioned network (multiple cycles, single cycle loop)
  - Network Locality
  - Unsuitable for keyword matches
  - Anonymity (Explicit node assignment for data values)

The Impact of DHT Routing Geometry on Resilience and Proximity

- Background of authors:
  - Krishna Gummadi: University of Washington
  - Ramakrishna Gummadi: USC, Los Angeles
  - Steven Gribble:University of Washington
  - Sylvia Ratnasamy: Intel Research, Berkeley
  - Scott Shenker: ICSI, Berkeley
  - Ion Stoica: UC Berkeley

Summary – Overview (1)

- Problem:
  - Myriad DHT designs in literature
  - Which one is the best (if there is one globally accepted)?
  - How do we choose?

Summary – Overview (2)

- Motivation:
  - A lot of DHTs have been proposed.
  - They have been studied and evaluated in isolation.
  - No comparisons have been made between the main infrastructures.
    - No base of comparison (i.e. the distinguishing characteristics/criteria that we use) has been set yet.

Summary – Overview (3)

- Contribution:
  - Separation of design choices (routing geometries) from algorithmic details.
  - Proposal of a comparison base for evaluating design choices.
  - Compare different geometries (Rings, Hypercubes, Butterflies, Trees)
  - Explore geometry's effect on:
    - Flexibility: Selection of neighbors (FNS) or routes (FRS).
      - Static Resilience: How well routing is done while routing tables still in recovery.
      - Path Latency: minimize it by incorporating proximity in DHTs

DHT Categorizations

- Functionality Level:
  - Routing-Level: routing behavior of a DHT – neighbor and routing selection choice.
  - System-Level: anything else – caching, replication etc.

- Algorithm vs Geometry:
  - Algorithm: exact rules on the routing and neighbor selection scheme.
  - Geometry [abstract definition]: algorithms' underlying structure that drives the DHT design; constraints the way route/neighbors selections take place without necessarily changing the algorithm.
    - Different algorithms may have the same geometry.

Geometries Comparison

- Geometry (Example DHT Algorithm):
  - Tree (PRR, Tapestry): node ids on the leaves, distance = height of the smallest common subtree
  - Hypercube (CAN): distance = number of different bits
  - Butterfly (Viceroy): nodes in log(n) stages, correct i^{th} bit at stage i
  - Ring (Chord, Symphony)
  - XOR (Kademlia): distance = numeric value of XOR of ids
  - Hybrid (Pastry): default tree distance, failure ring distance

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 (optimal paths) Route Selection (non-optimal paths)	1 -	$c_1(\log n)$ -	$c_1(\log n)$ $2c_2(\log n)$	1 -	1 $c_2(\log n)$	1 $c_2(\log n)$
Natural support for sequential neighbors?	no	no	yes	no	no	Default routing: no Fallback routing: yes

Table 1: The neighbor and route selection flexibility at any node in various routing geometries. c_1 and c_2 are small constants.

Geometries Comparison

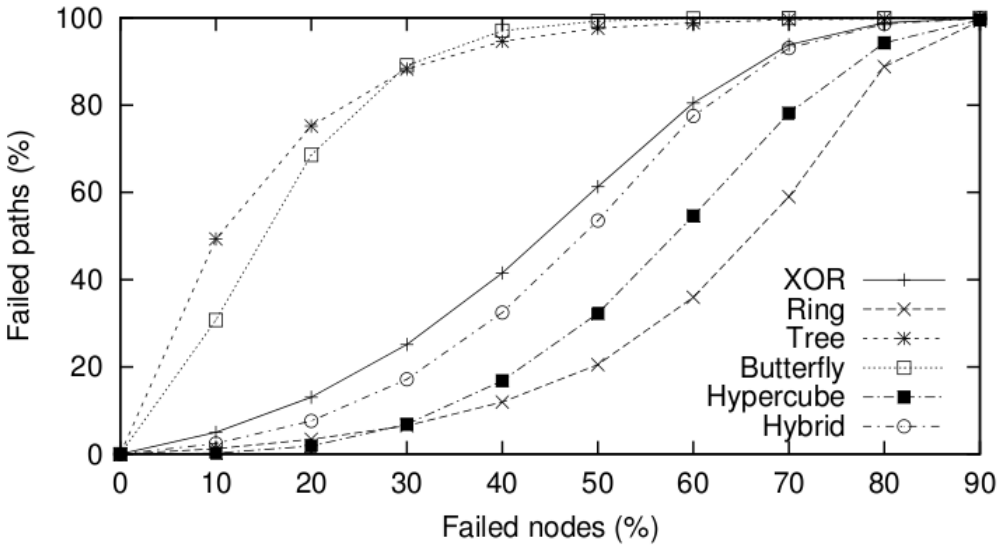
- Geometry (Example DHT Algorithm):
  - Tree (PRR, Tapestry)
  - Hypercube (CAN)
  - Butterfly (Viceroy)
  - Ring (Chord, Symphony)
  - XOR (Kademlia)
  - Hybrid (Pastry)

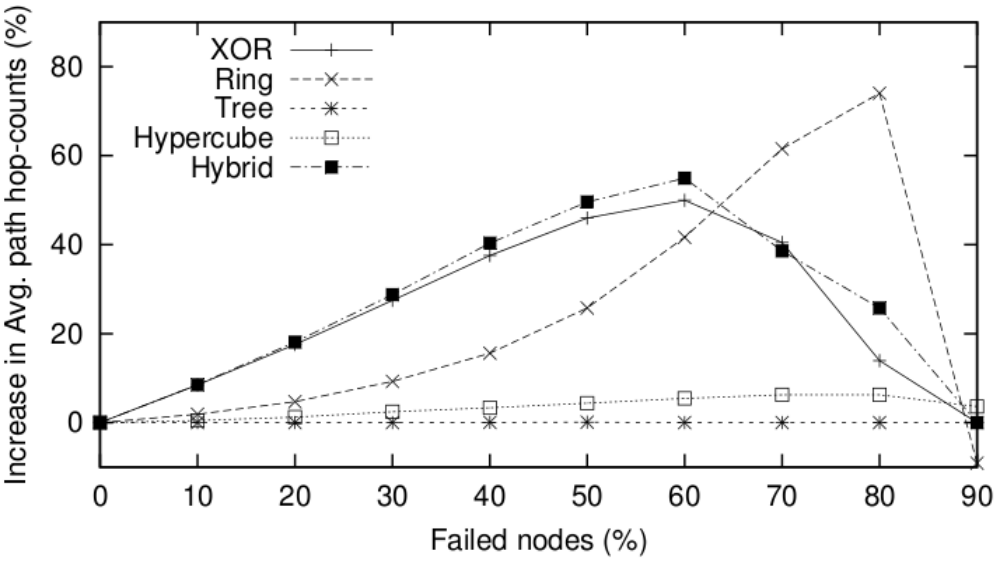
property	tree	hypercube	ring	butterfly	xor	hybrid
Neighbor Selection	$n^{\log n/2}$	1	$n^{\log n/2}$	1	$n^{\log n/2}$	$n^{\log n/2}$
Route Selection (optimal paths)	1	$c_1(\log n)$	$c_1(\log n)$	1	1	1
Route Selection (non-optimal paths)	-	-	$2c_2(\log n)$	-	$c_2(\log n)$	$c_2(\log n)$
Natural support for sequential neighbors?	no	no	yes	no	no	Default routing: no Fallback routing: yes

Resilience

Table 1: The neighbor and route selection flexibility at any node in various routing geometries. c_1 and c_2 are small constants.

Static Resilience





- Resilience:
  - How well routing is done while routing tables are still in recovery.
  - Routing tables are kept static except for the failed nodes' entries that are deleted
- Metrics:
  - % paths failed: how often routing between two nodes failed
  - % increase in path length: how much is the length of the route increased compared to the path length without failures

Routing Flexibility => Static Resilience

Geometries Comparison

- Geometry (Example DHT Algorithm):
  - Tree (PRR, Tapestry)
  - Hypercube (CAN)
  - Butterfly (Viceroy)
  - Ring (Chord, Symphony)
  - XOR (Kademlia)
  - Hybrid (Pastry)

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 (optimal paths) Route Selection (non-optimal paths)	1 -	$c_1(\log n)$ -	$c_1(\log n)$ $2c_2(\log n)$	1 -	1 $c_2(\log n)$	1 $c_2(\log n)$
Natural support for sequential neighbors?	no	no	yes	no	no	Default routing: no Fallback routing: yes

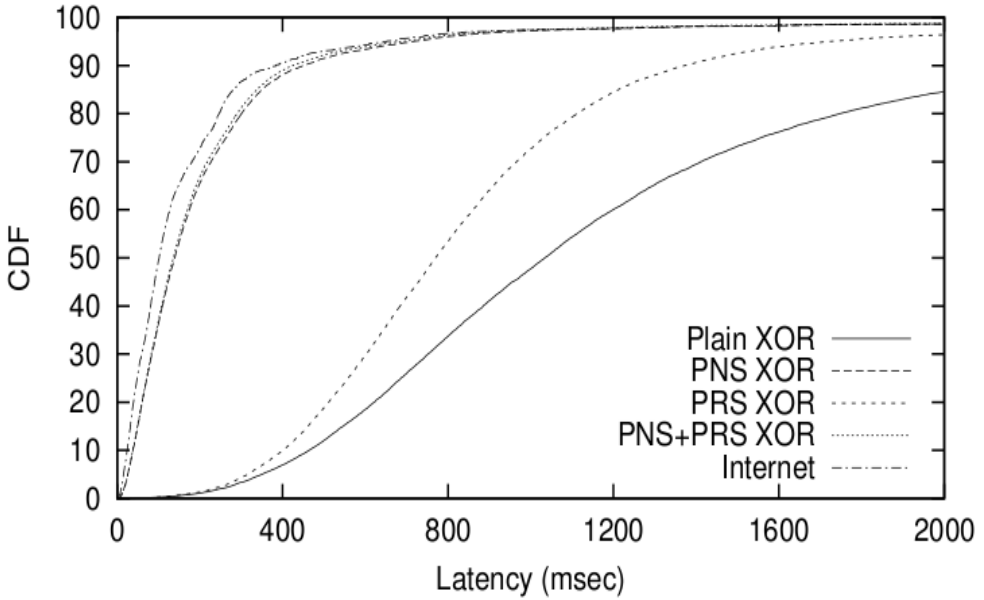
Path Latency

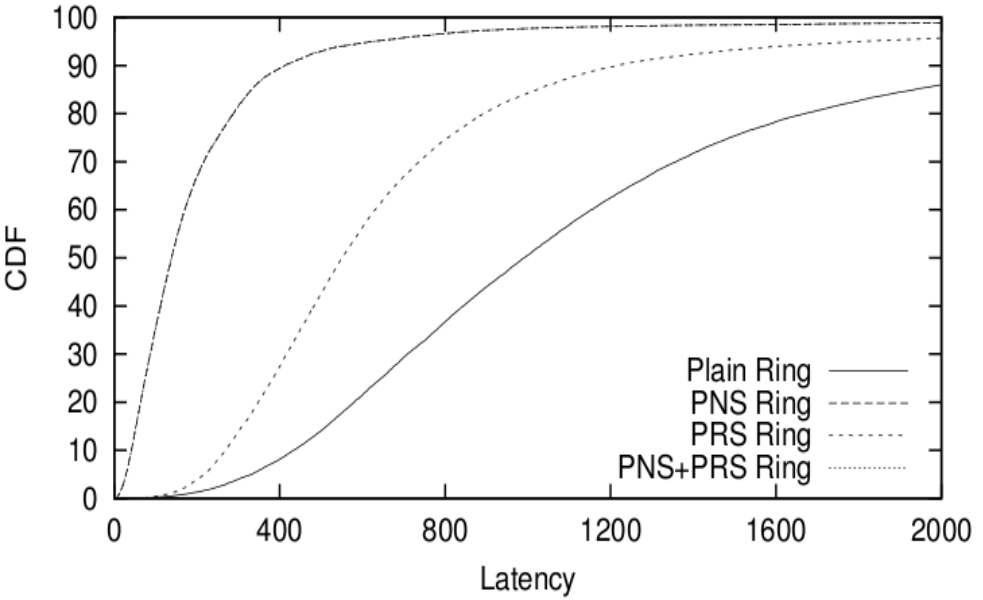
Table 1: The neighbor and route selection flexibility at any node in various routing geometries. c_1 and c_2 are small constants.

Path Latency

- DHTs are efficient in terms of hopcount
- Hopcount fails to capture end-to-end delays of links/overlay hops (e.g. a satellite link)
- Therefore proximity approaches are used to minimize end-to-end latency:
  - Proximity Neighbor Selection (PNS): neighbors on routing tables are chosen according to their proximity; (Tree, Ring, XOR)
  - Proximity Route Selection (PRS): next hops on routing are chosen according to the neighbors' proximity; (Hypercube, Ring, XOR)
  - Proximity Identifier Selection (PIS): Not used.

PNS vs PRS

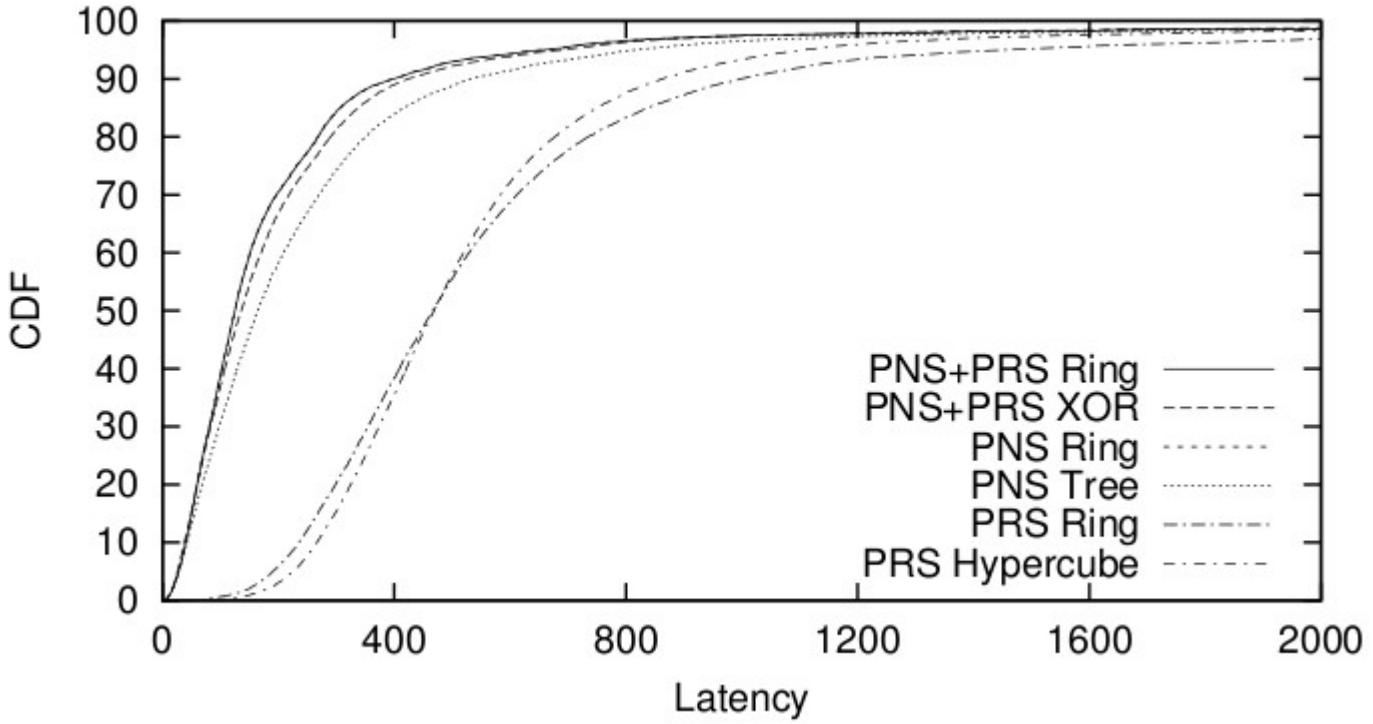




PNS+PRS ≈ PNS > PRS > Plain

- Both PNS and PRS facilitate finding shorter paths

Role of Topology



Does not depend on routing geometry.

Flexibility (in implementing PRS, PNS) => Path latency

DHTs: Conclusions & Discussion

- Rooting Geometry is a design choice of major importance:
  - Geometry affects Flexibility
  - Flexibility affects Proximity & Static Resilience thus affects performance
- Sequential neighbors and their importance in the recovery process and static resilience (though I did not discuss it in the lecture).
- Ring: topology seems by far the most promising topology (flexibility, sequential neighbors support). Why not use them?

DHTs: Conclusions & Discussion

- Issues, Open Questions:
  - Completeness: more DHT algorithms
  - Factor not considered: symmetry in the routing tables => state management overhead
  - Narrow focus on two performance aspects of DHTs: Static resilience, proximity. What about communication overhead? Storage requirements? Replication techniques for faster recovery and fault-tolerance could be studied.
  - Graphs are somewhat unclear (how have they been generated – simulation? Experimental results?)

Current State

- What happened:
  - Different file storing and lookup systems have risen
  - Cassandra (Facebook): structured key-value store
  - Dynamo (Amazon Web Services): highly available, proprietary key-value structured storage system
  - Memcached (Google, Facebook, AWS, …, everywhere really): general-purpose distributed memory caching system
  - Bit Torrent (everyone else:): peer-to-peer file sharing protocol

References

- Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. 2001. Chord: A scalable peer-to-peer lookup service for internet applications. In Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications (SIGCOMM '01). ACM, New York, NY, USA, 149-160.
- K. Gummadi, R. Gummadi, S. Gribble, S. Ratnasamy, S. Shenker, and I. Stoica. 2003. The impact of DHT routing geometry on resilience and proximity. In Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications (SIGCOMM '03). ACM, New York, NY, USA, 381-394.

Thank You!!!