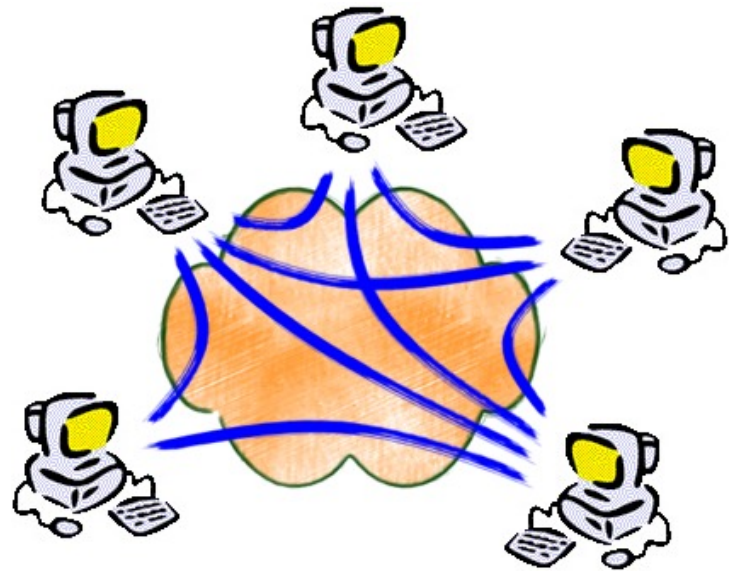


# Making peer-to-peer systems scalable

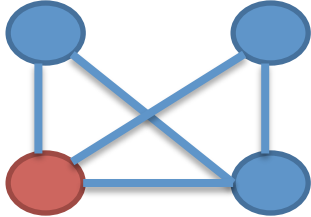
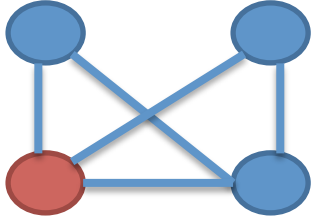
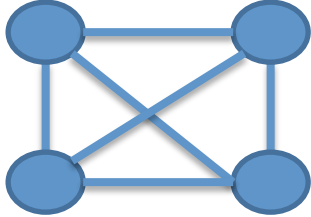
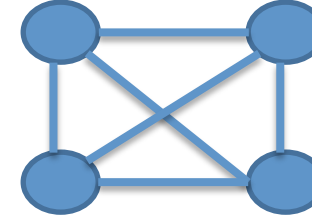
Presented by Elisavet Kozyri

# What is a peer-to-peer system?

- A distributed application architecture that partitions tasks or work loads between peers
- Main actions:
  - Find the owner of the file (indexing)
  - Get the file from the owner



# Popular P2P Systems

	indexing	get file
Napster (1999 – 2001)		
Gnutella (2000 – now)		

# What was it missing?

Scalable indexing mechanism

# Goals

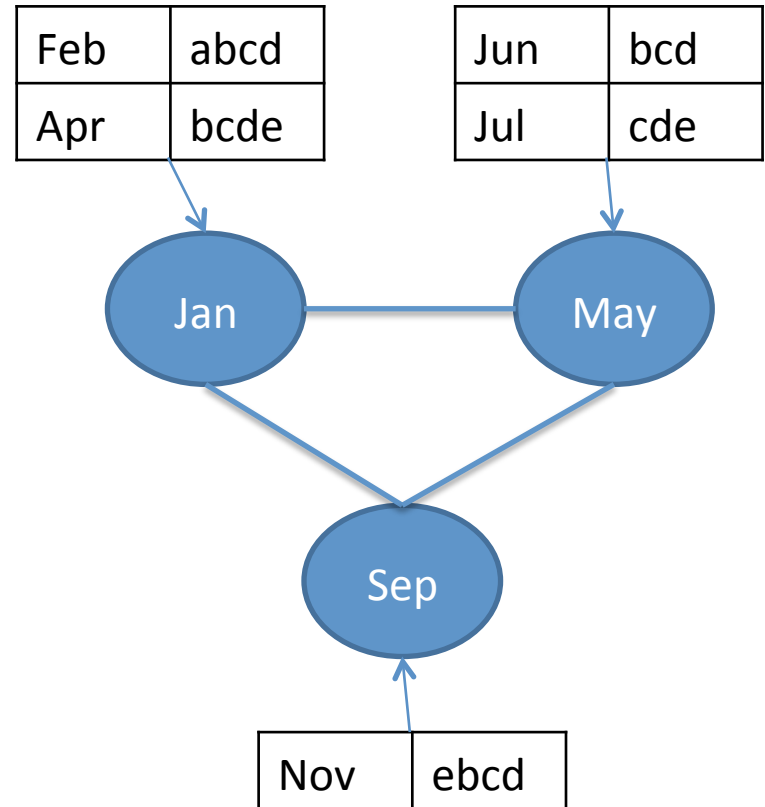
- Each node (peer) should be *responsible* for certain files
- System remain robust during and after the *arrival* and *departure* of nodes

# Observation

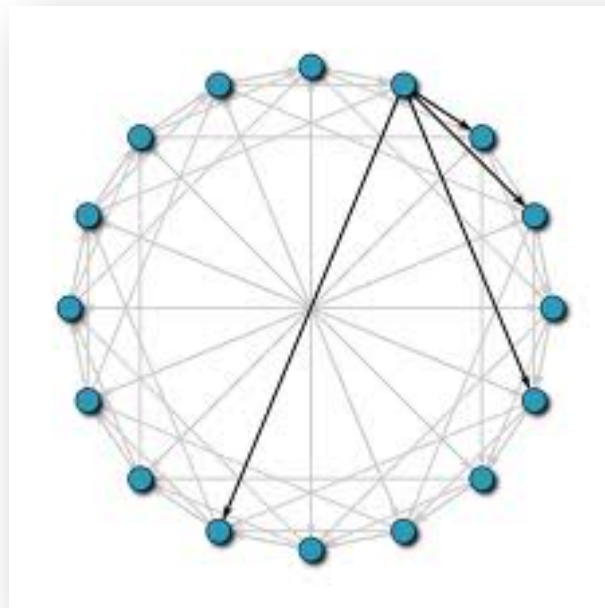
- At the new system:
  - Given an identifier of data it should find the owner.
  - If a node joins or leaves it should rearrange the data.
- Is it similar to hash table?
  - node  $\Leftrightarrow$  bucket
  - identifier  $\Leftrightarrow$  key
  - data  $\Leftrightarrow$  value
- Distributed hash table

# Distributed Hash Table

- Abstraction:
  - Simple interface
  - Similar to hash table
  - Pairs (key, value) are spread across the nodes
- Implementation:
  - Name space partitioning
  - Each node responsible for a fraction
  - Each node has ID from the same name space
  - Nodes communicate over an overlay network to route lookup requests



# Chord: A DHT Implementation





# Chord: Authors

- **Ion Stoica**

Associate Professor of CS at Berkeley

- **Robert Morris**

Professor in the EECS department at MIT

- **David Karger**

Professor in the EECS department at MIT

- **M. Frans Kaashoek**

Professor in the EECS department at MIT

- **Hari Balakrishnan**

Professor in the EECS department at MIT

# Chord: Goals

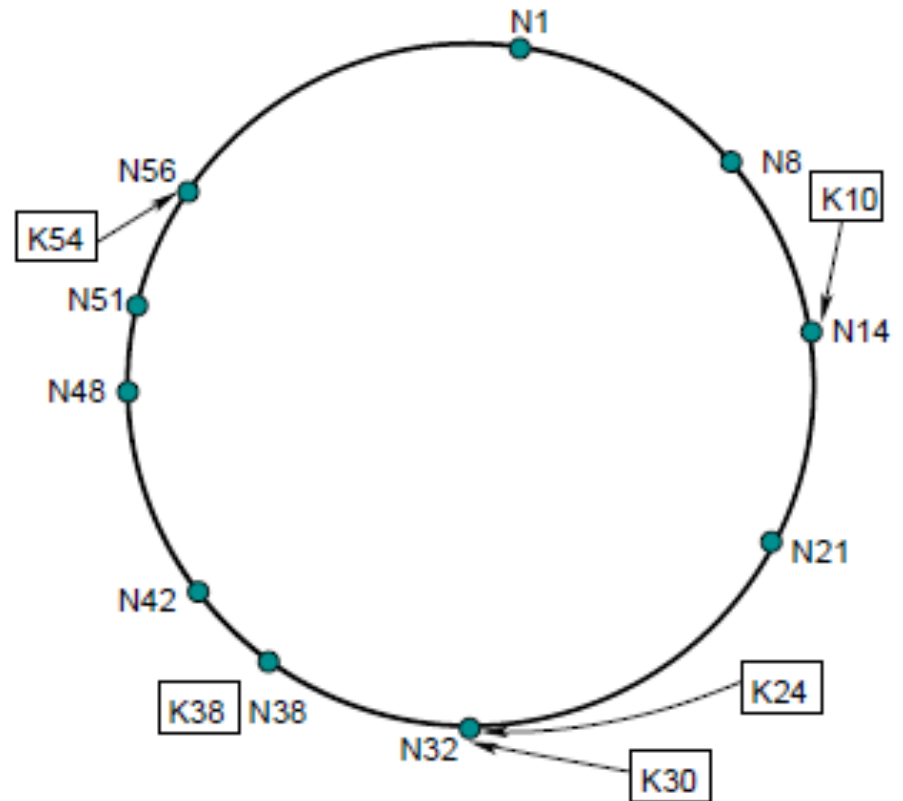
- Load balance
  - Keys are spread evenly over the nodes
- Decentralization
  - No node is more important than any other
- Scalability
  - The cost of lookups grows as the log of the number of nodes
- Availability
  - The node responsible for the key can always be found
- Flexible naming
  - No constraints on the structure of the keys

# Chord: Protocol

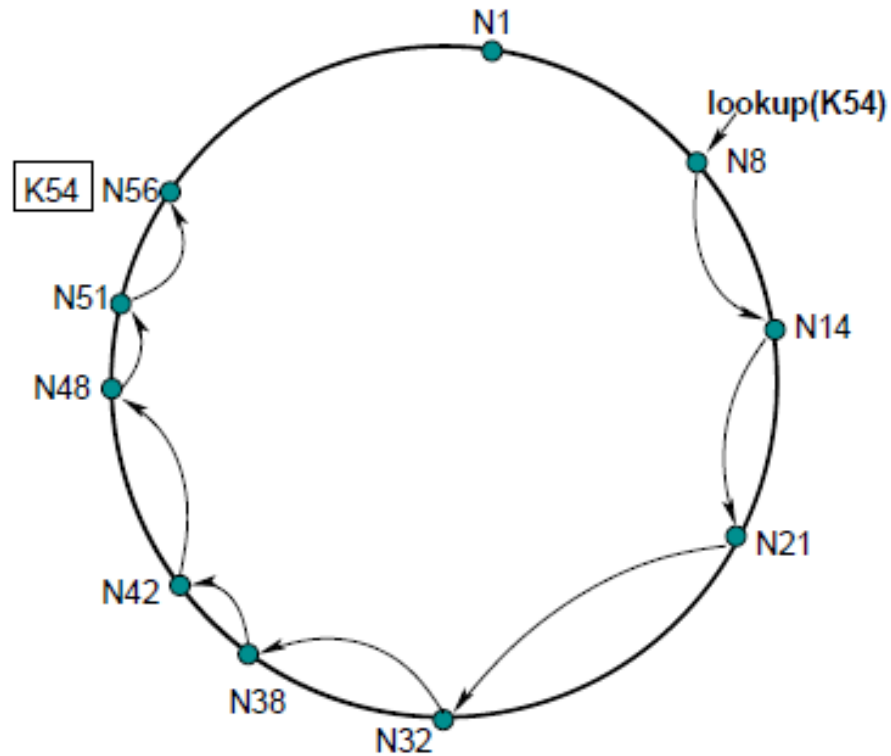
- Hashing
  - Determines the name space and the way it is partitioned among nodes.
- Routing
  - Determines the way lookup requests will be routed among nodes to reach their destination.
- Joining
  - Determines the way the system adopts itself at the arrival of a node

# Chord: Hashing

- Consistent hash function
- Each node and key has an m-bit identifier
- Identifiers ordered in an identifier circle
- Key k belongs to the node which identifier is the first clockwise from k

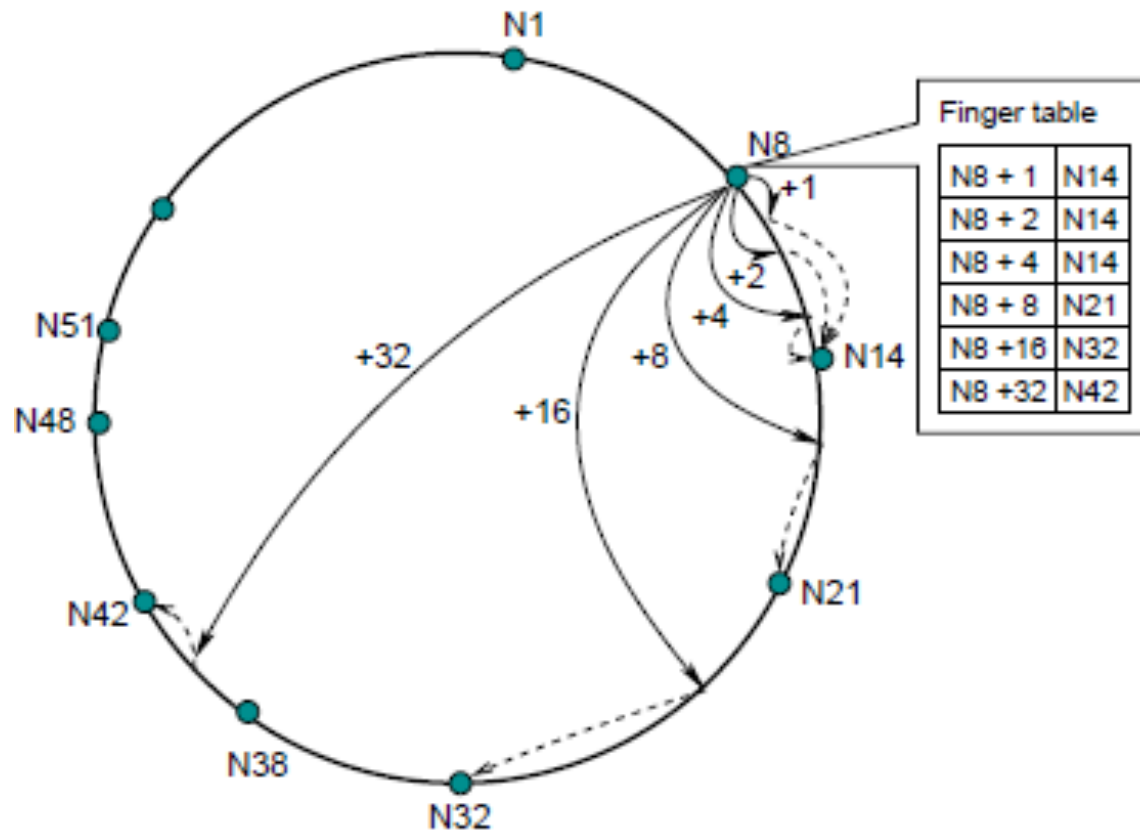


# Chord: Inefficient Routing



Complexity:  $O(N)$

# Chord: Efficient Routing

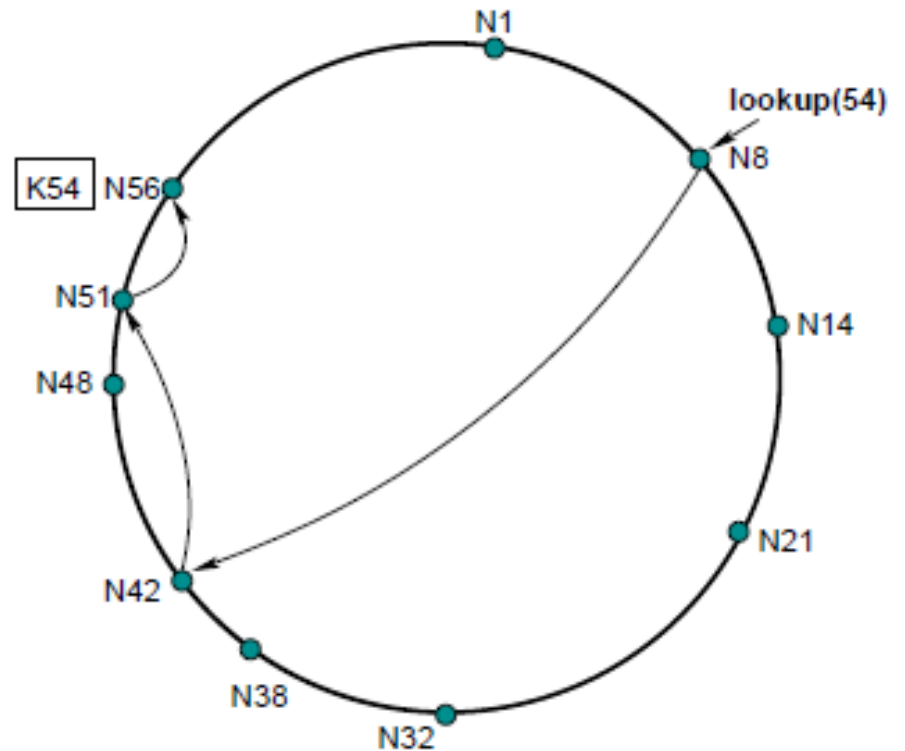


Complexity:  $O(\log N)$

# Chord: Routing

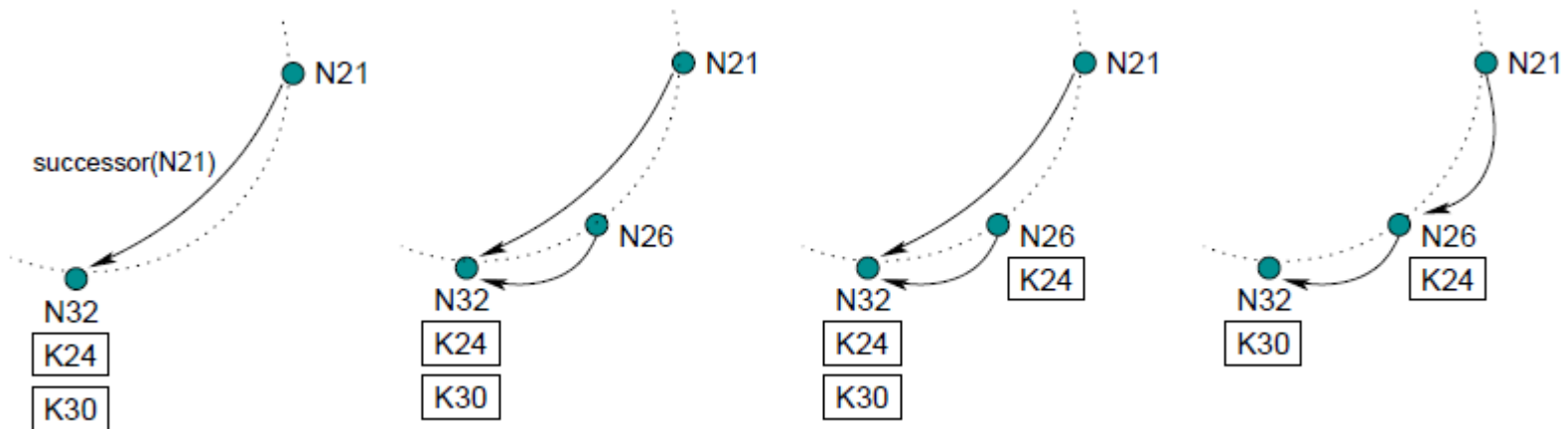
$N8 + 1$	N14
$N8 + 2$	N14
$N8 + 4$	N14
$N8 + 8$	N21
$N8 + 16$	N32
$N8 + 32$	N42

$N42 + 1$	N48
$N42 + 2$	N48
$N42 + 4$	N48
$N42 + 8$	N51
$N42 + 16$	N1
$N42 + 32$	N14



# Chord: Node joins

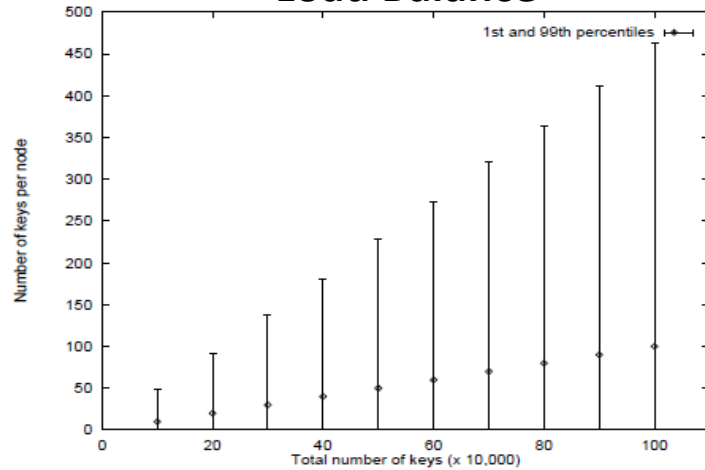
- Stabilization
- Ensure node's successor pointer is up to date
- Ex:  $N26.join(N42) \rightarrow N26.stabilize \rightarrow N26.notify(N32) \rightarrow N21.stabilize \rightarrow N26.notify(N26)$



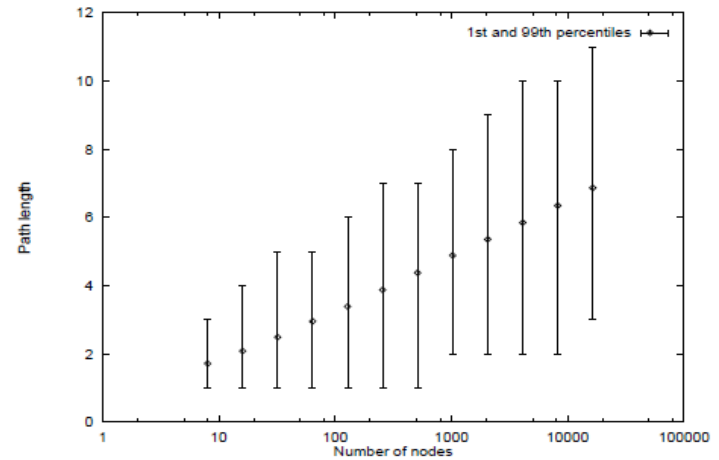


# Chord: Evaluation

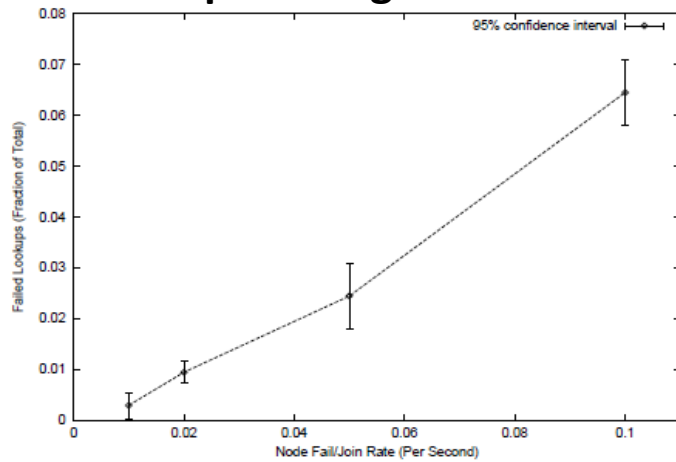
## Load Balance



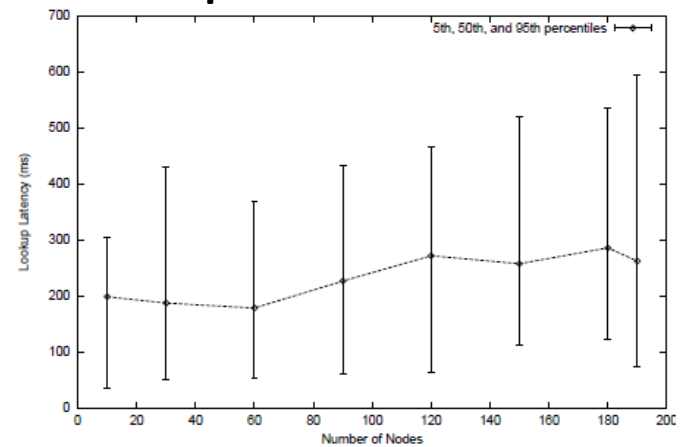
## Path Length



## Lookups During Stabilization

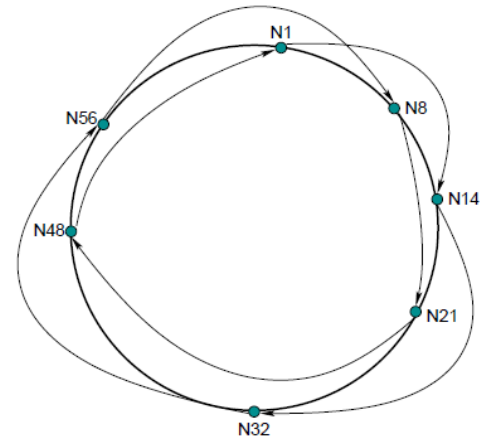


## Experimental Results



# Chord: Discussion

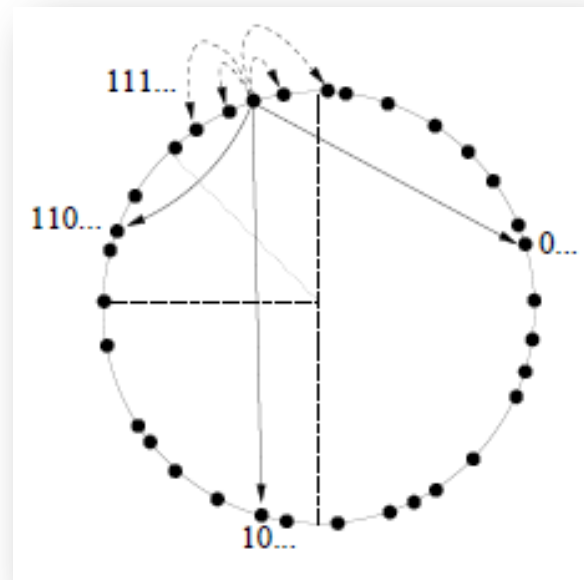
- Basic principle of routing algorithm: Longest Shooting
- Network locality?
- Stabilization: “we separate our correctness and performance goals”
- Lookups eventually succeed
- Is Chord globally consistent ?
- Anonymity?
- General comments



# Other DHT Implementations

- Pastry
- CAN
- Tapestry
- PRR
- Viceroy
- Kademlia

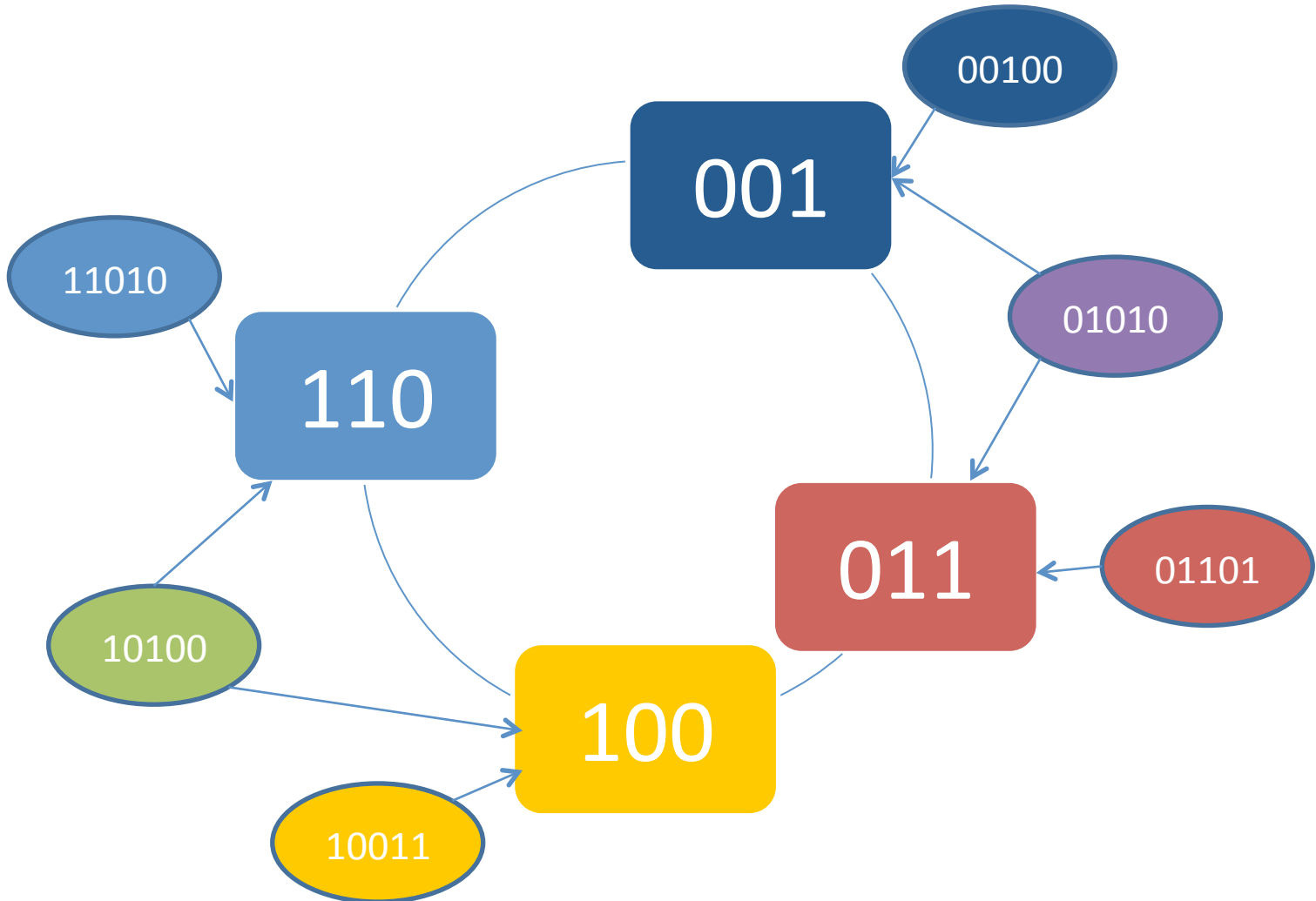
# Pastry



# Pastry: Hashing

- Circular namespace
- Each node is randomly assigned a 128-bit identifier (nodeID)
- Each object is assigned an identifier *at least* 128 bits long (objID)
- An object belongs to a node if nodeID is numerically closest to the 128 most significant bits of the objID
- An object is stored on  $k$  nodes

# Pastry: Hashing



# Pastry: Routing

- nodeID

Level 0			Level 1			Level 2			Level 3		
1	1	0	0	1	0	0	0	1	1	1	1

A message whose destID matches the local node's nodeID up to level  $l$  is forwarded to a node whose nodeID matches the destID up to least  $l+1$ .

- Routing table

For each level  $l$ , the routing table contains the IP address of  $2^{b-1}$  nodes that have the same nodeID prefix as the local node up to level  $l-1$ , but differ at level  $l$ .

From all possible nodes, the *closest* are selected.

CAN

Content-Addressable Network

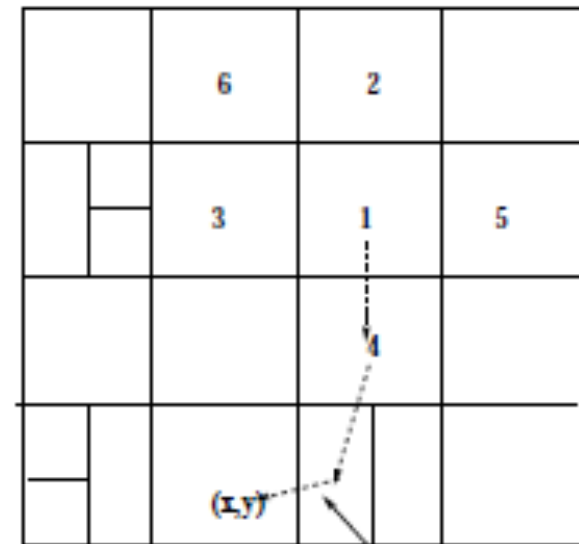


# CAN: Hashing

- d-torus
- Each node owns a zone within the overall space
- A key is mapped onto a point in the space
- If the point  $P$  belongs to the zone of node  $n$ , then the corresponding (key, value) is stored at  $n$ .

# CAN: Routing

- Routing table: IP address and virtual coordinate zone of each immediate neighbors
- A node forwards the message to the neighbor with coordinates closest to destination



sample routing path from node 1 to point (x,y)

*1's coordinate neighbor set = {2,3,4,5}*  
*7's coordinate neighbor set = {}*

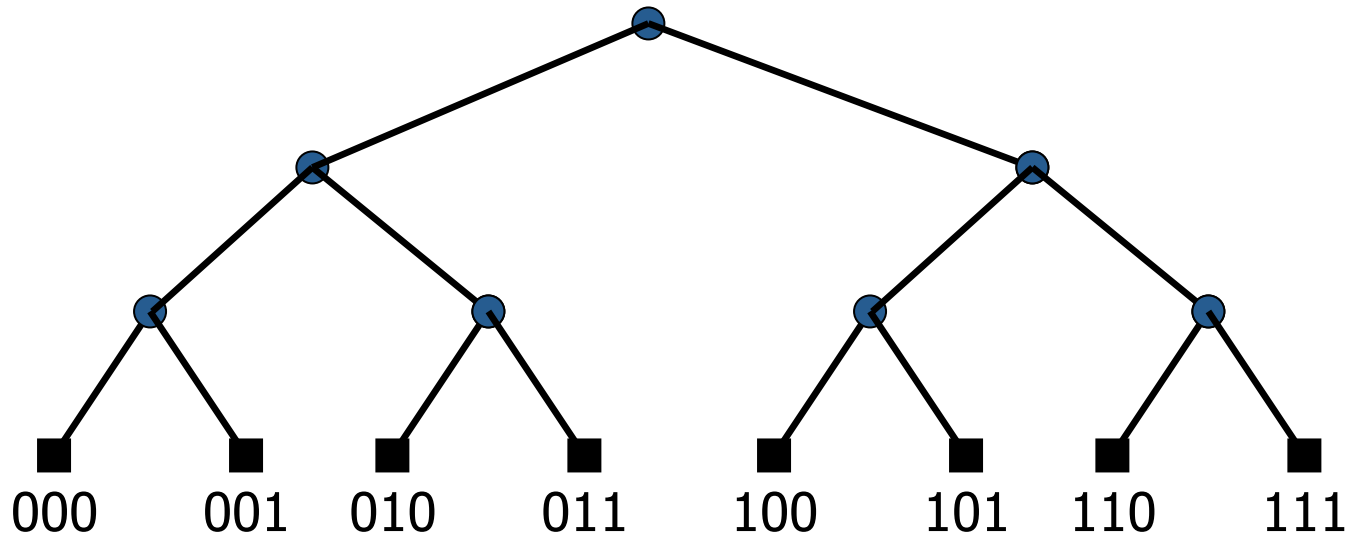
# Comparison of DHT Geometries

- “The Impact of DHT Routing Geometry of Resilience and Proximity”
- **K. Gummadi:** Head of Networked Systems Research Group at Max Planck Institute for Software Systems
- **R. Gummadi:** Assistant Professor, ECE, UMass Amherst
- **S. Gribble:** Associate Professor, CSE, University of Washington
- **S. Ratnasamy:** Researcher at Intel Research Berkeley
- **S. Shenker:** Professor, EECS, UC Berkeley
- **I. Stoica:** Associate Professor, CS, US Berkeley

# Comparison of DHT Geometries

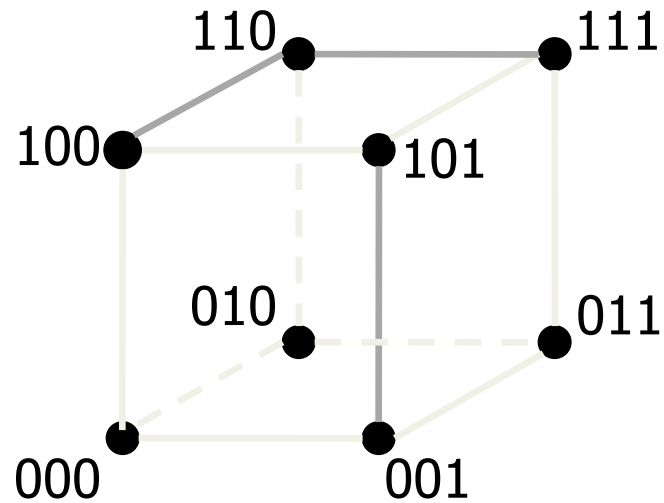
- Resilience
  - The ability of the DHT implementation to route during and after arrivals/departures of nodes.
- Proximity
  - The ability of the DHT implementation to adapt to the underlying Internet topology.
- Flexibility
  - Neighbor Selection
  - Route Selection

# Tree



- PRR, Tapestry
- Generous flexibility in choosing neighbors
- No flexibility in route selection

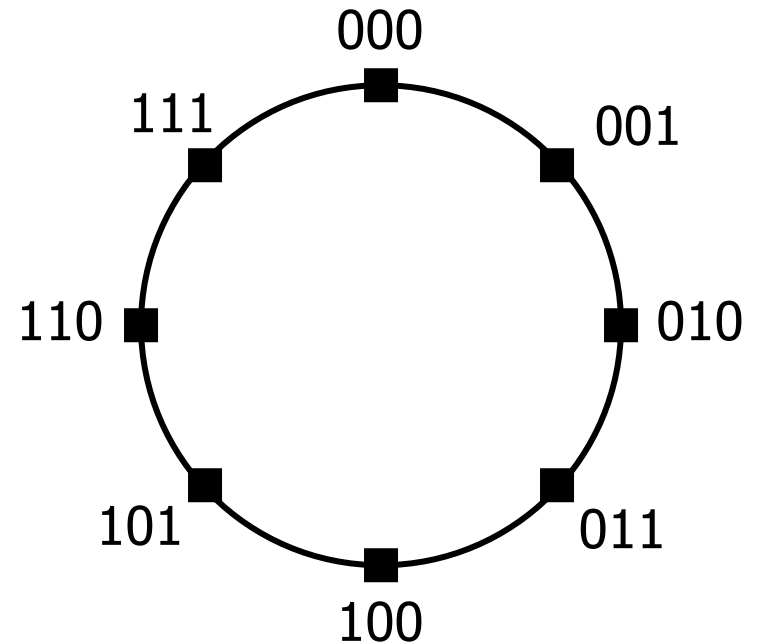
# Hypercube



- CAN
- Flexibility in route selection
- No flexibility in choosing neighbors

# Ring

- Chord
- Changes
  - $i^{\text{th}}$  neighbor of  $a$  belongs to  $[(a + 2^i), (a + 2^{i+1})]$
  - multiple paths
- Flexibility in route selection
- Flexibility in choosing neighbors



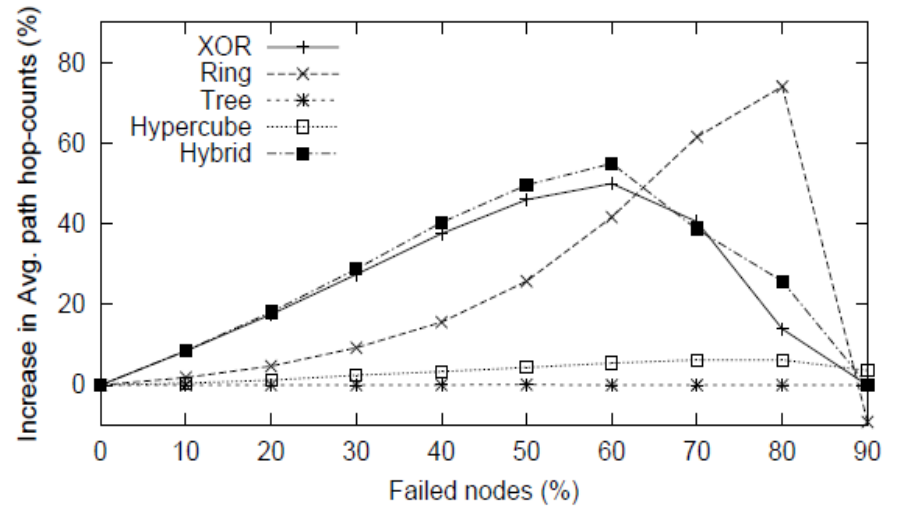
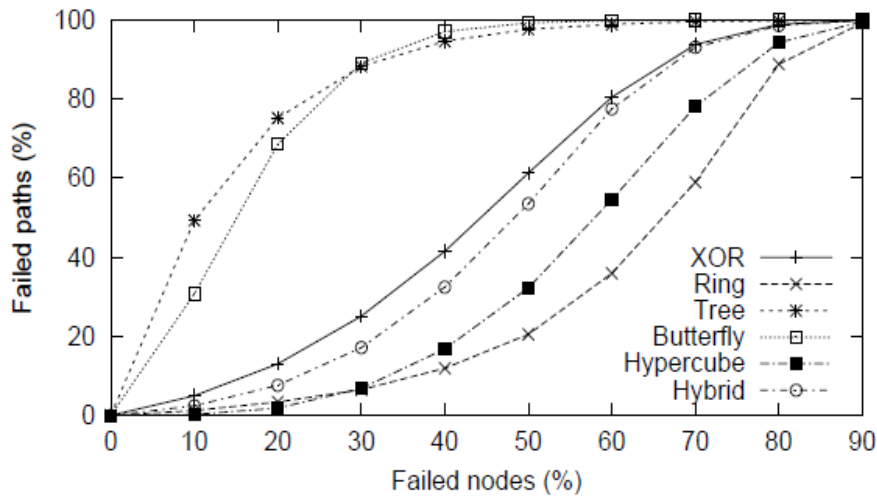
# Hybrid

- Pastry (ring + tree)
- Flexibility in route selection
- Flexibility in choosing neighbors



# Static Resilience

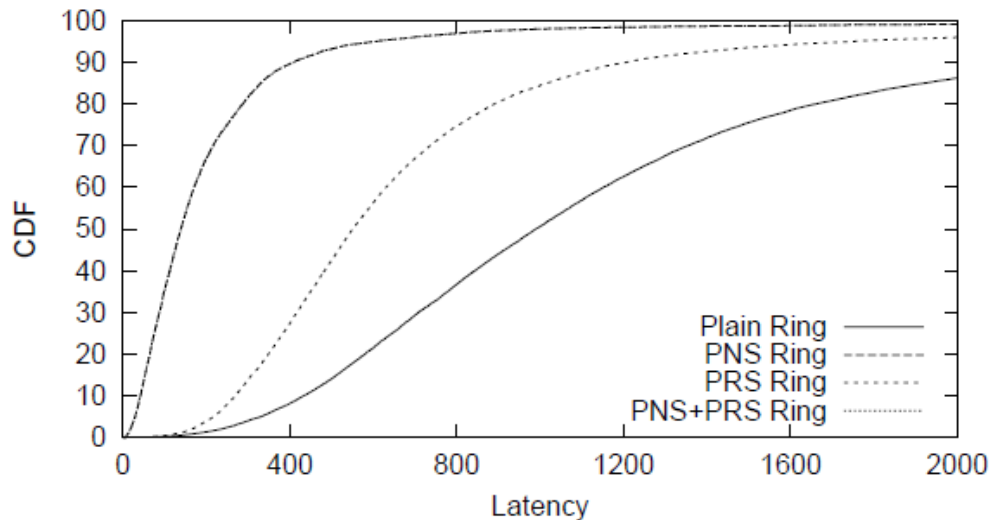
Static resilience  $\Leftrightarrow$  Routing flexibility



# Proximity

- Proximity Neighbor Selection (PNS)
- Proximity Route Selection (PRS)

What is the best?



# Discussion

- Are ring geometries the best?
- What is the importance of sequential neighbors?
- How does neighbors flexibility influence resilience/proximity?
- Chord, CAN, Pastry: Are they used today?
- Which is the best?
- General comments

# References

- Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems, Antony Rowston, Peter Druschel
- A Scalable Content-Addressable Network, Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard karp, Scott Shenker
- Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications, Ion Stoica, Robert Morris, David Liben-Nowell, David Karger, M. Frans Kaashoek, Frank Dabek, Hari Balakrishnan
- Geometry shapes from Krishna's SIGCOMM talk

Thank you!