CE693: Adv. Computer Networking

L-15 P2P

Acknowledgments: Lecture slides are from the graduate level Computer Networks course thought by Srinivasan Seshan at CMU. When slides are obtained from other sources, a a reference will be noted on the bottom of that slide. A full list of references is provided on the last slide.

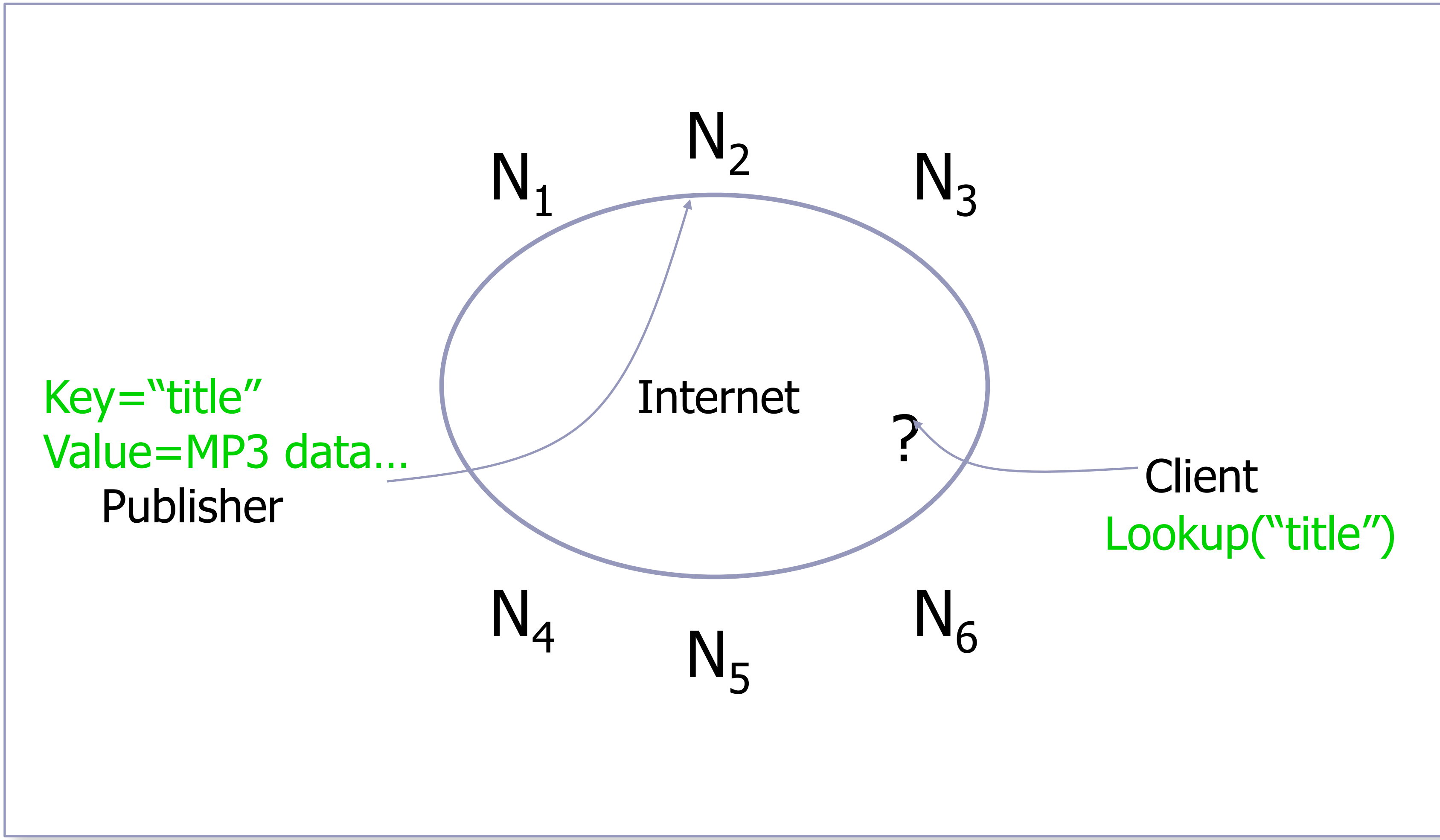
Overview

- P2P Lookup Overview
- Centralized/Flooded Lookups
- Routed Lookups – Chord
- Comparison of DHTs

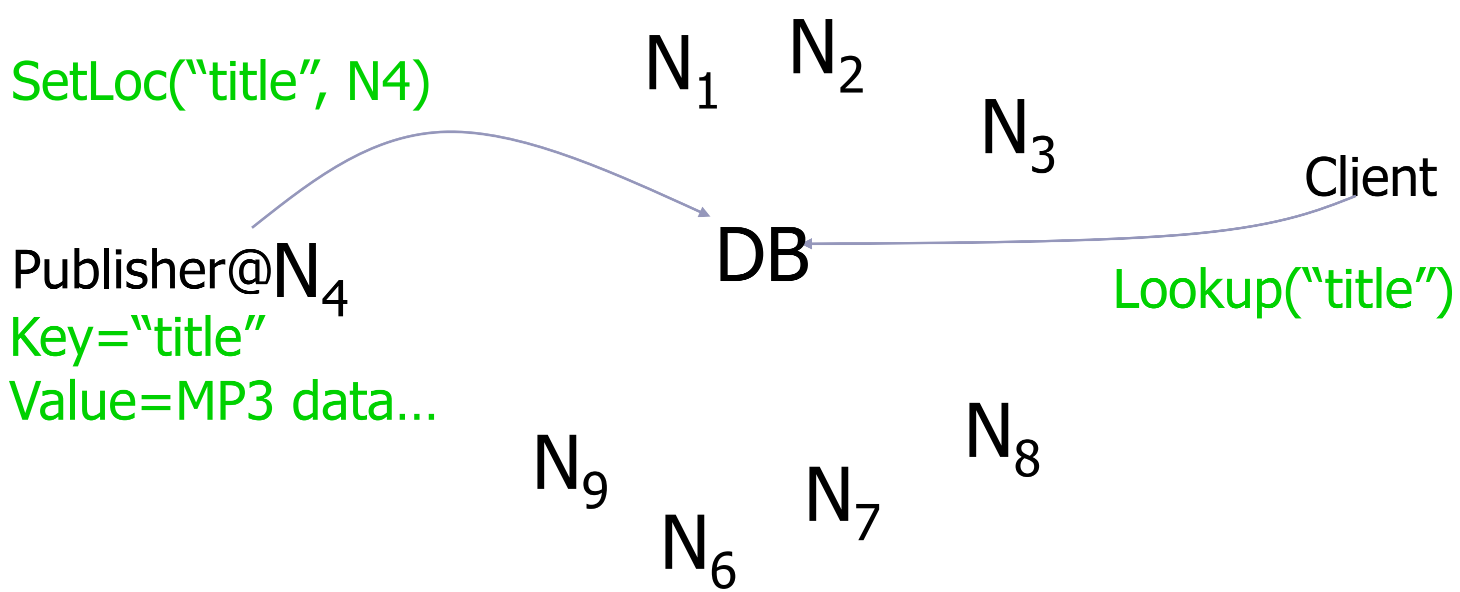
Peer-to-Peer Networks

- Typically each member stores/provides access to content
- Basically a replication system for files
  - Always a tradeoff between possible location of files and searching difficulty
  - Peer-to-peer allow files to be anywhere → searching is the challenge
  - Dynamic member list makes it more difficult
- What other systems have similar goals?
  - Routing, DNS

The Lookup Problem

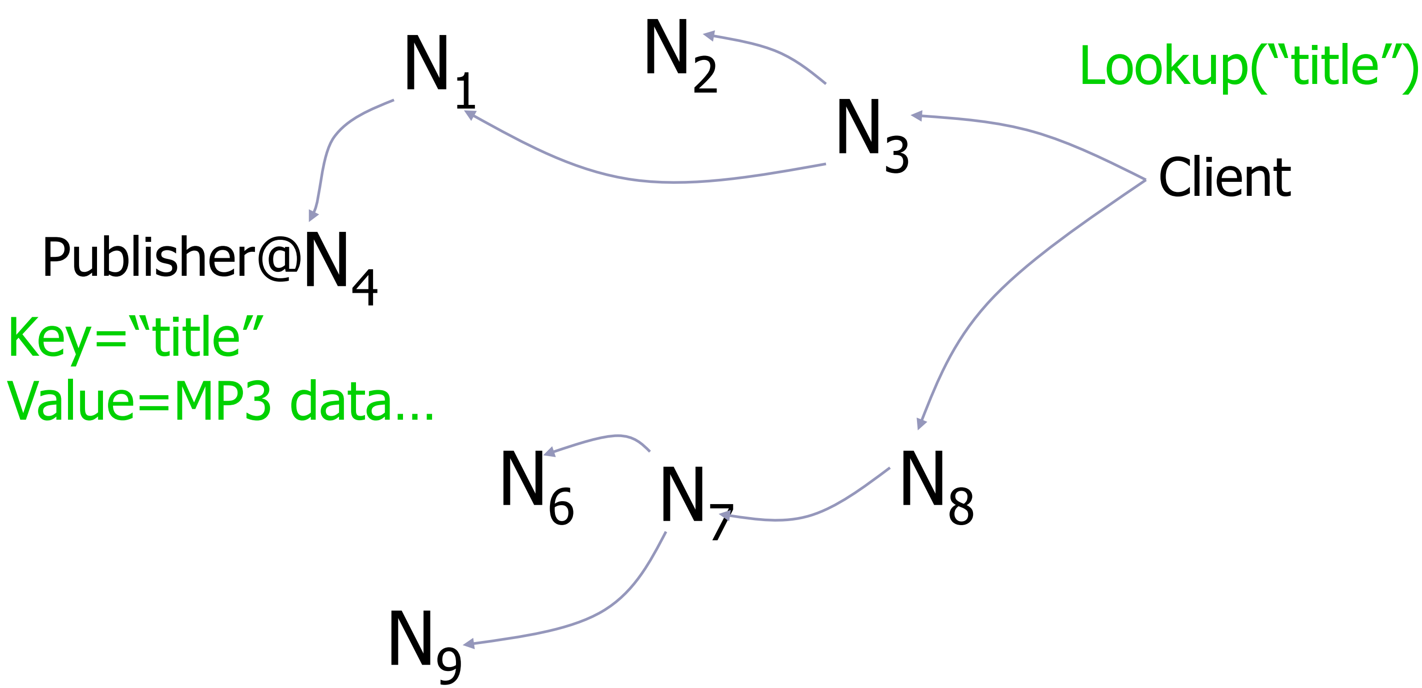


Centralized Lookup (Napster)



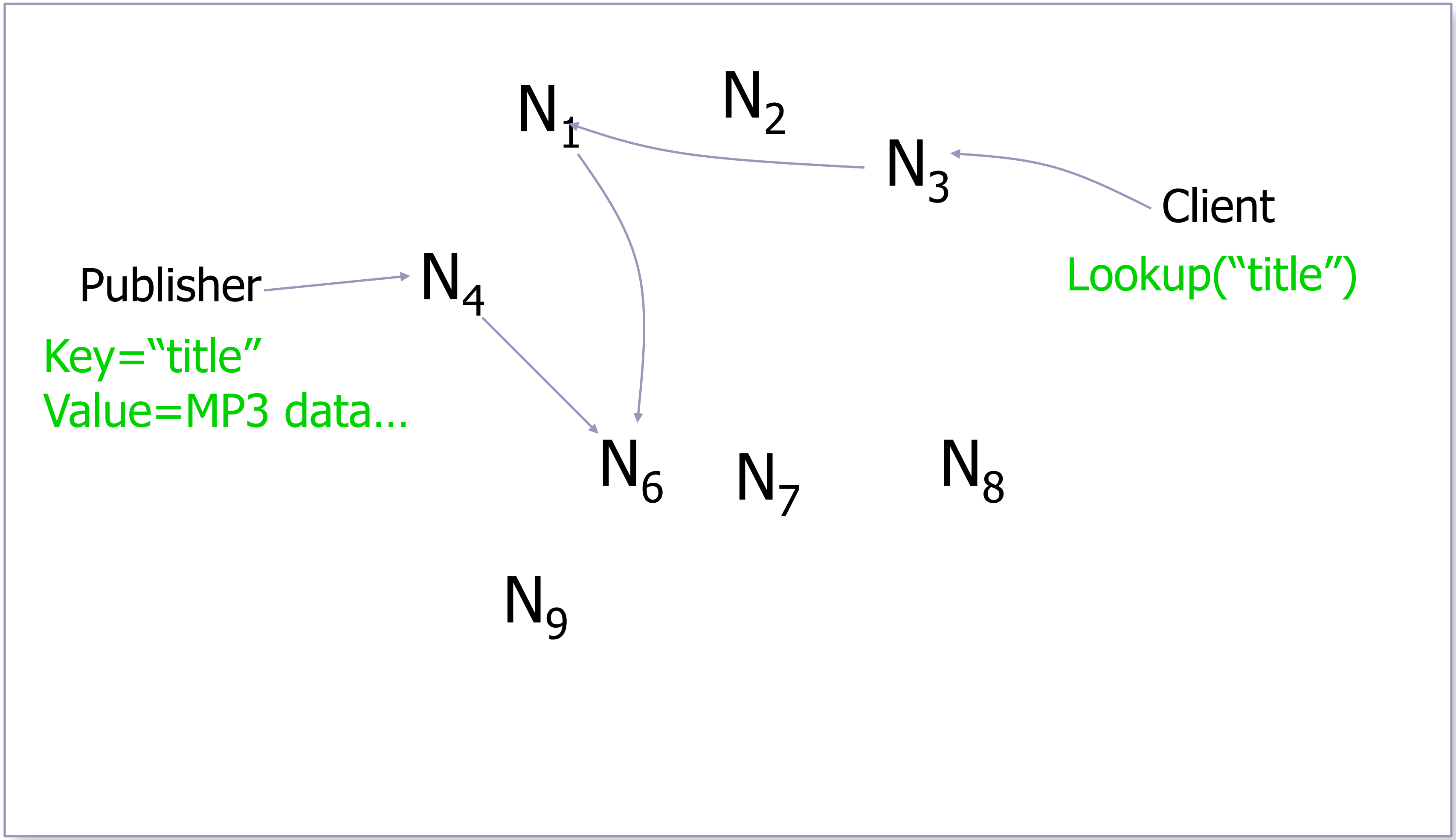
Simple, but O(N) state and a single point of failure

Flooded Queries (Gnutella)



Robust, but worst case O(N) messages per lookup

Routed Queries (Chord, etc.)



Overview

- P2P Lookup Overview
- Centralized/Flooded Lookups
- Routed Lookups – Chord
- Comparison of DHTs

Centralized: Napster

- Simple centralized scheme → motivated by ability to sell/control
- How to find a file:
  - On startup, client contacts central server and reports list of files
  - Query the index system → return a machine that stores the required file
    - Ideally this is the closest/least-loaded machine
  - Fetch the file directly from peer

Centralized: Napster

- Advantages:
  - Simple
  - Easy to implement sophisticated search engines on top of the index system
- Disadvantages:
  - Robustness, scalability
  - Easy to sue!

Flooding: Old Gnutella

- On startup, client contacts any servent (**serv**er + cli**ent**) in network
  - Servent interconnection used to forward control (queries, hits, etc)
- Idea: broadcast the request
- How to find a file:
  - Send request to all neighbors
  - Neighbors recursively forward the request
  - Eventually a machine that has the file receives the request, and it sends back the answer
  - Transfers are done with HTTP between peers

Flooding: Old Gnutella

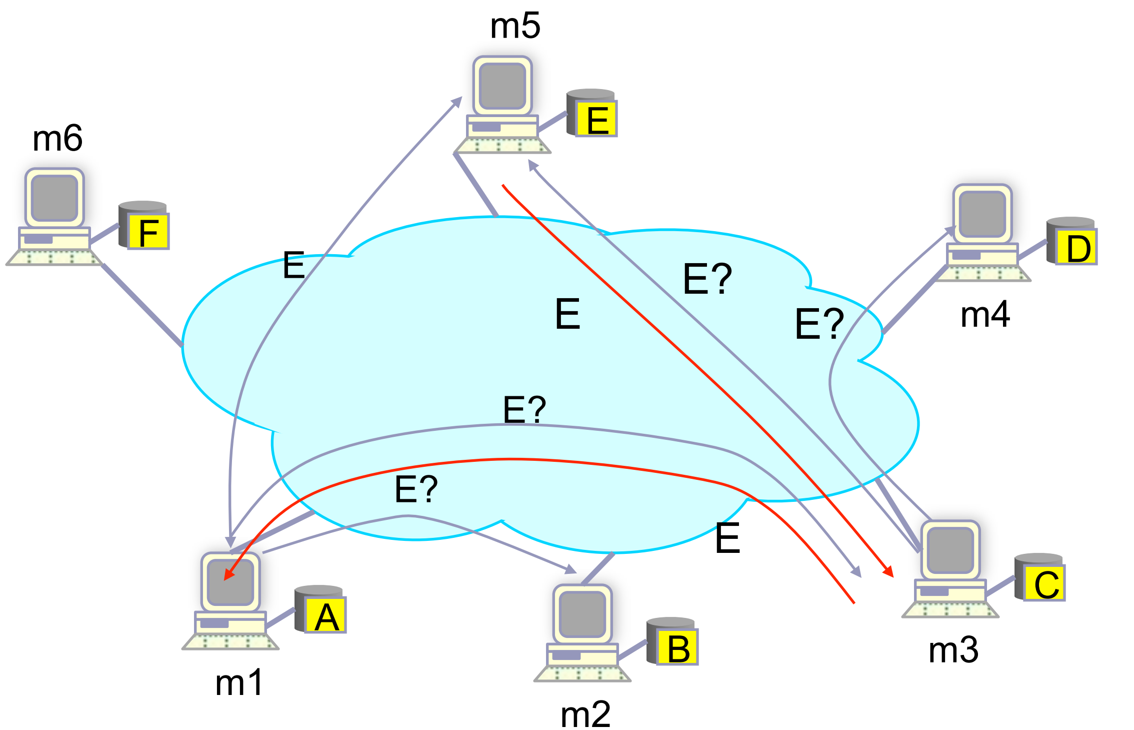
- Advantages:
  - Totally decentralized, highly robust
- Disadvantages:
  - Not scalable; the entire network can be swamped with request (to alleviate this problem, each request has a TTL)
  - Especially hard on slow clients
    - At some point broadcast traffic on Gnutella exceeded 56kbps – what happened?
    - Modem users were effectively cut off!

Flooding: Old Gnutella Details

- Basic message header
  - Unique ID, TTL, Hops
- Message types
  - Ping – probes network for other servents
  - Pong – response to ping, contains IP addr, # of files, # of Kbytes shared
  - Query – search criteria + speed requirement of servent
  - QueryHit – successful response to Query, contains addr + port to transfer from, speed of servent, number of hits, hit results, servent ID
  - Push – request to servent ID to initiate connection, used to traverse firewalls
- Ping, Queries are flooded
- QueryHit, Pong, Push reverse path of previous message

Flooding: Old Gnutella Example

Assume: m1's neighbors are m2 and m3;
m3's neighbors are m4 and m5;…



Flooding: Gnutella, Kazaa

- Modifies the Gnutella protocol into two-level hierarchy
  - Hybrid of Gnutella and Napster
- Supernodes
  - Nodes that have better connection to Internet
  - Act as temporary indexing servers for other nodes
  - Help improve the stability of the network
- Standard nodes
  - Connect to supernodes and report list of files
  - Allows slower nodes to participate
- Search
  - Broadcast (Gnutella-style) search across supernodes
- Disadvantages
  - Kept a centralized registration → allowed for law suits ☹

Overview

- P2P Lookup Overview
- Centralized/Flooded Lookups
- Routed Lookups – Chord
- Comparison of DHTs

Routing: Structured Approaches

- Goal: make sure that an item (file) identified is always found in a reasonable # of steps
- Abstraction: a distributed hash-table (DHT) data structure
  - insert(id, item);
  - item = query(id);
  - Note: item can be anything: a data object, document, file, pointer to a file…
- Proposals
  - CAN (ICIR/Berkeley)
  - Chord (MIT/Berkeley)
  - Pastry (Rice)
  - Tapestry (Berkeley)
  - …

Routing: Chord

- Associate to each node and item a unique *id* in an *uni*-dimensional space
- Properties
  - Routing table size O(log(*N*)) , where *N* is the total number of nodes
  - Guarantees that a file is found in O(log(*N*)) steps

Aside: Hashing

- Advantages
  - Let nodes be numbered 1..m
  - Client uses a ***good*** hash function to map a URL to 1..m
  - Say hash (url) = *x*, so, client fetches content from node *x*
  - No duplication – not being fault tolerant.
  - One hop access
  - Any problems?
    - What happens if a node goes down?
    - What happens if a node comes back up?
    - What if different nodes have different views?

Robust hashing

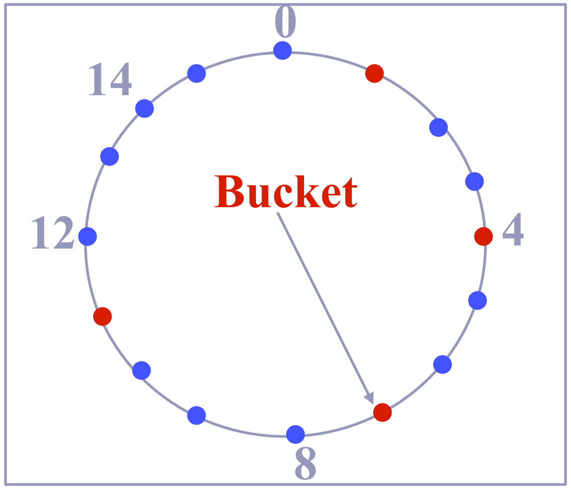
- Let 90 documents, node 1..9, node 10 which was dead is alive again
- % of documents in the wrong node?
  - 10, 19-20, 28-30, 37-40, 46-50, 55-60, 64-70, 73-80, 82-90
  - *Disruption coefficient* = ½
  - Unacceptable, use consistent hashing – idea behind Akamai!

Consistent Hash

- "view" = subset of all hash buckets that are visible
- Desired features
  - Balanced – in any one view, load is equal across buckets
  - Smoothness – little impact on hash bucket contents when buckets are added/removed
  - Spread – small set of hash buckets that may hold an object regardless of views
  - Load – across all views # of objects assigned to hash bucket is small

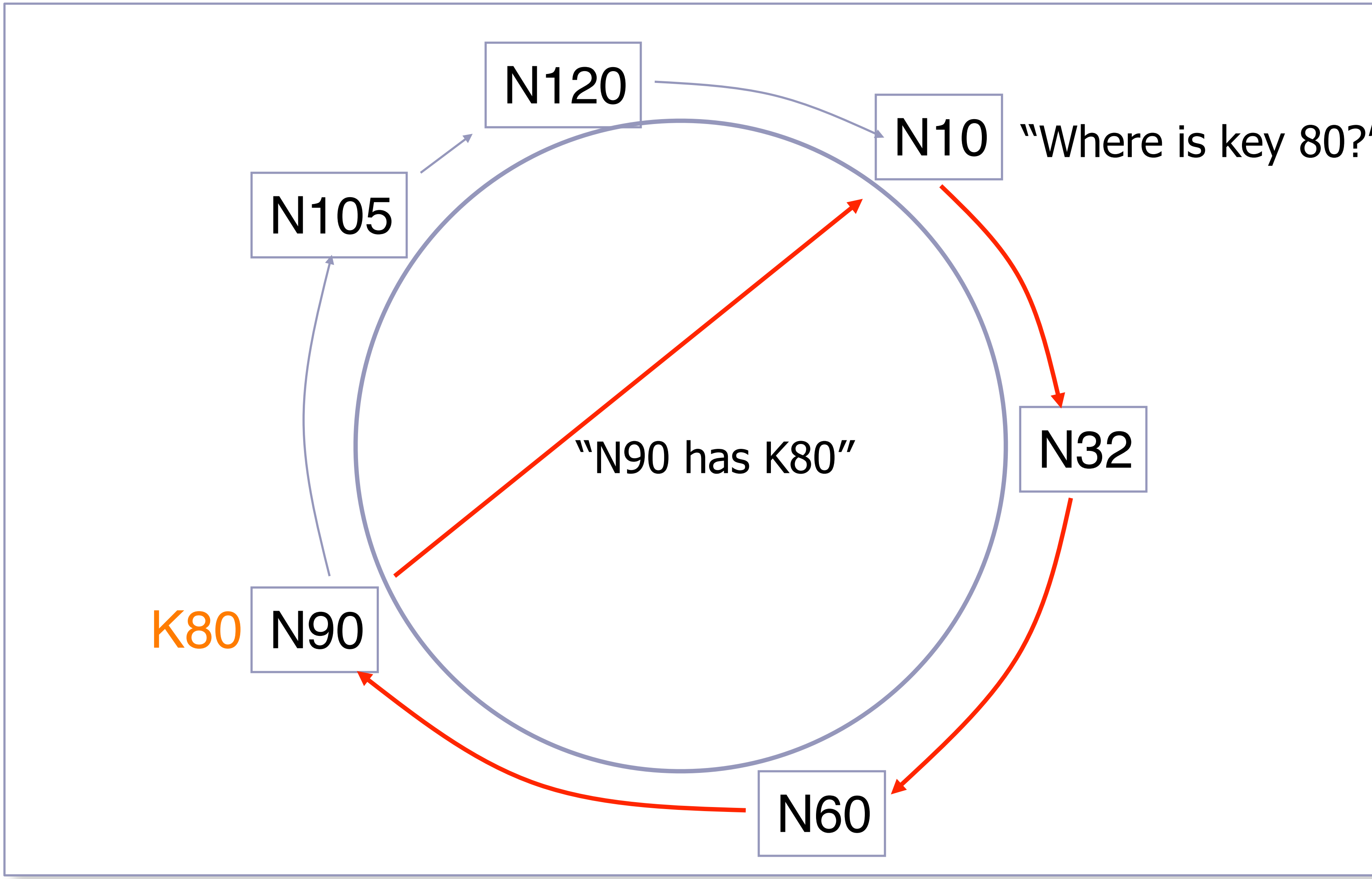
Consistent Hash – Example

- Construction
  - Assign each of C hash buckets to random points on mod 2^n circle, where, hash key size = *n.*
  - Map object to random position on circle
  - Hash of object = closest clockwise bucket

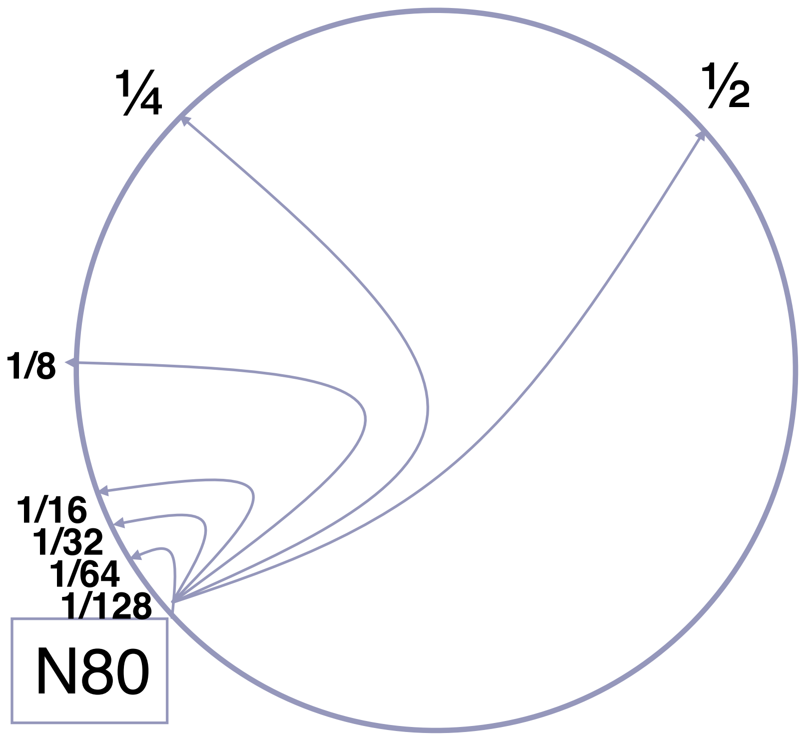


- Smoothness → addition of bucket does not cause much movement between existing buckets
- Spread & Load → small set of buckets that lie near object
- Balance → no bucket is responsible for large number of objects

Routing: Chord Basic Lookup



Routing: Finger table - Faster Lookups

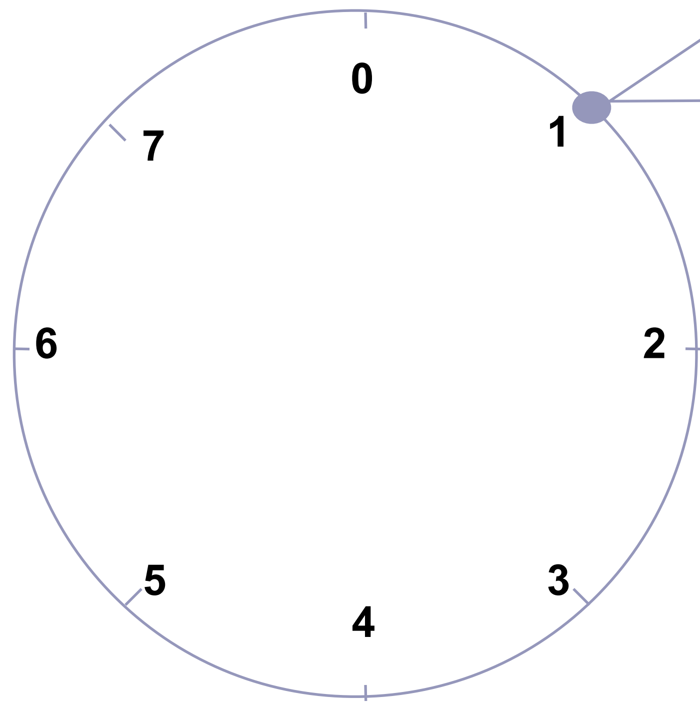


Routing: Chord Summary

- Assume identifier space is $0 \ldots 2^m$
- Each node maintains
  - Finger table
    - Entry *i* in the finger table of *n* is the first node that succeeds or equals $n + 2^i$
  - Predecessor node
- An item identified by *id* is stored on the successor node of *id*

Routing: Chord Example

- Assume an identifier space 0..8
- Node n1:(1) joins→all entries in its finger table are initialized to itself

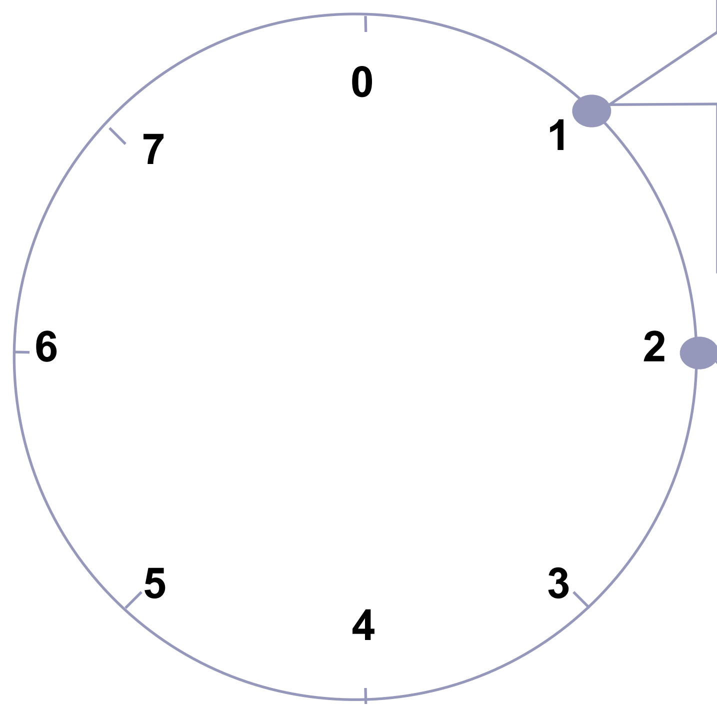


Succ. Table

i	*id*+2^i	succ
0	2	1
1	3	1
2	5	1

Routing: Chord Example

- Node n2:(2) joins



Succ. Table (node 1)

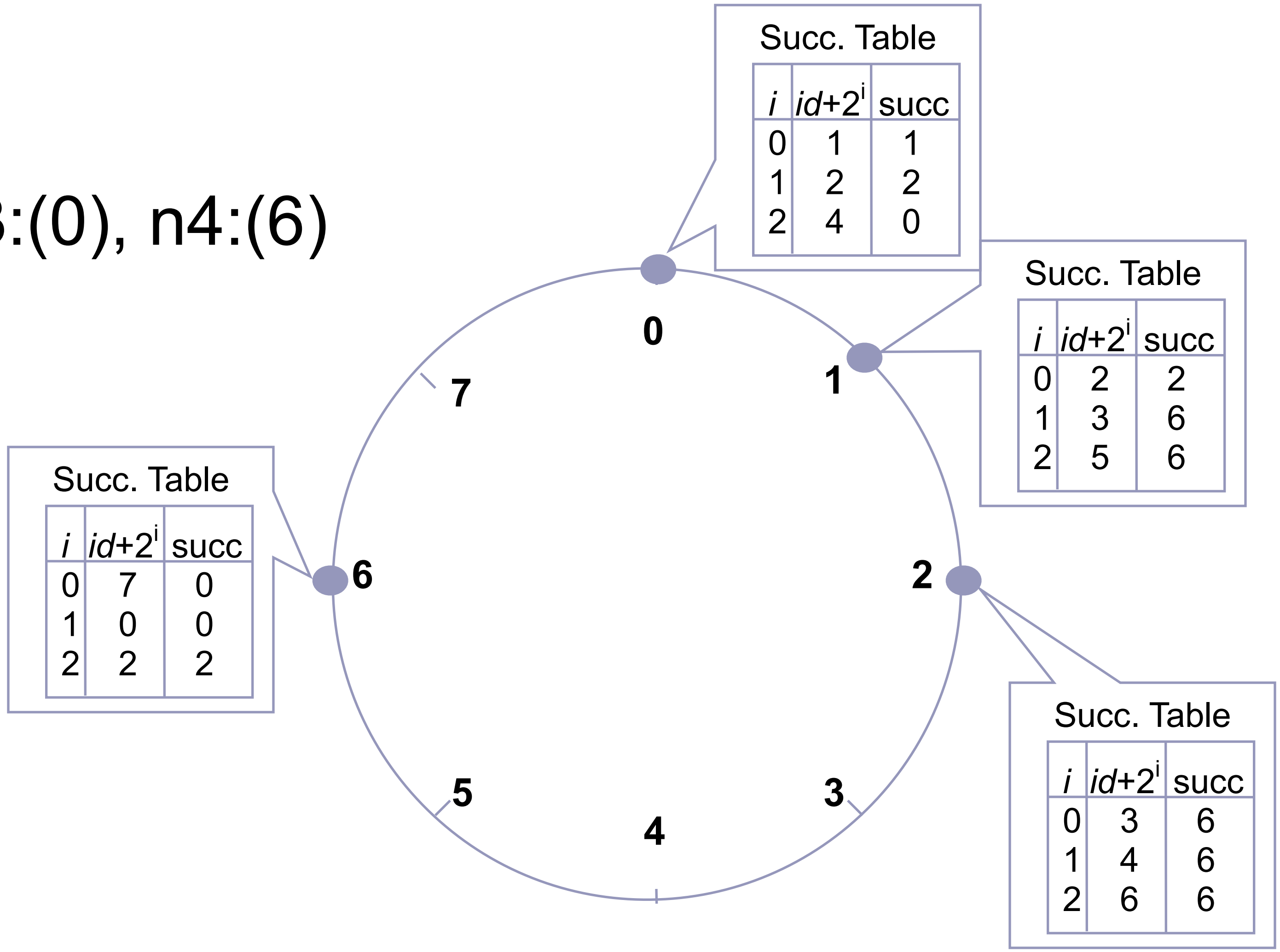
i	*id*+2^i	succ
0	2	2
1	3	1
2	5	1

Succ. Table (node 2)

i	*id*+2^i	succ
0	3	1
1	4	1
2	6	1

Routing: Chord Example

- Nodes n3:(0), n4:(6) join



Succ. Table (node 0)

i	*id*+2^i	succ
0	1	1
1	2	2
2	4	0

Succ. Table (node 1)

i	*id*+2^i	succ
0	2	2
1	3	6
2	5	6

Succ. Table (node 2)

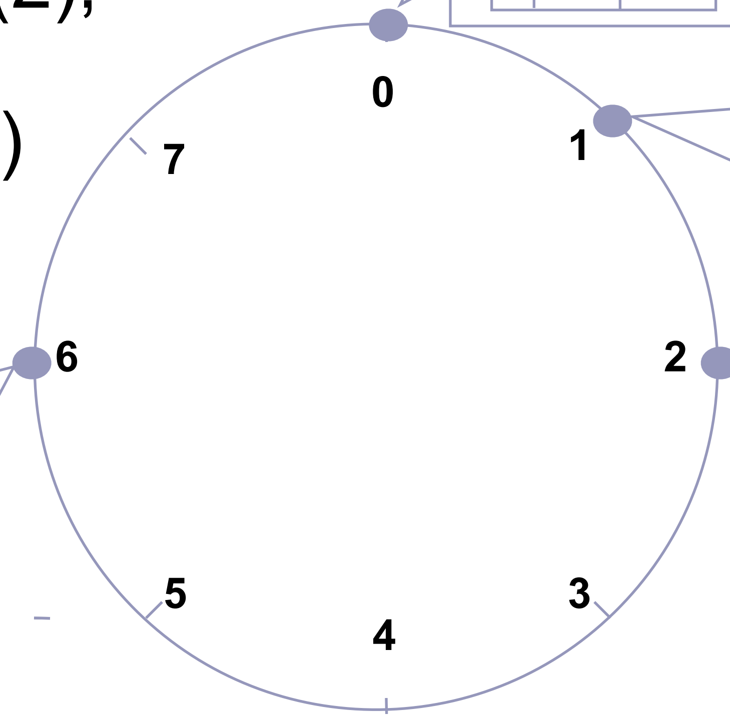
i	*id*+2^i	succ
0	3	6
1	4	6
2	6	6

Succ. Table (node 6)

i	*id*+2^i	succ
0	7	0
1	0	0
2	2	2

Routing: Chord Examples

- Nodes: n1:(1), n2(2), n3(0), n4(6)
- Items: f1:(7), f2:(1)



Node 0 — Succ. Table (Items: 7)

i	*id*+2^i	succ
0	1	1
1	2	2
2	4	0

Node 1 — Succ. Table (Items: 1)

i	*id*+2^i	succ
0	2	2
1	3	6
2	5	6

Node 2 — Succ. Table

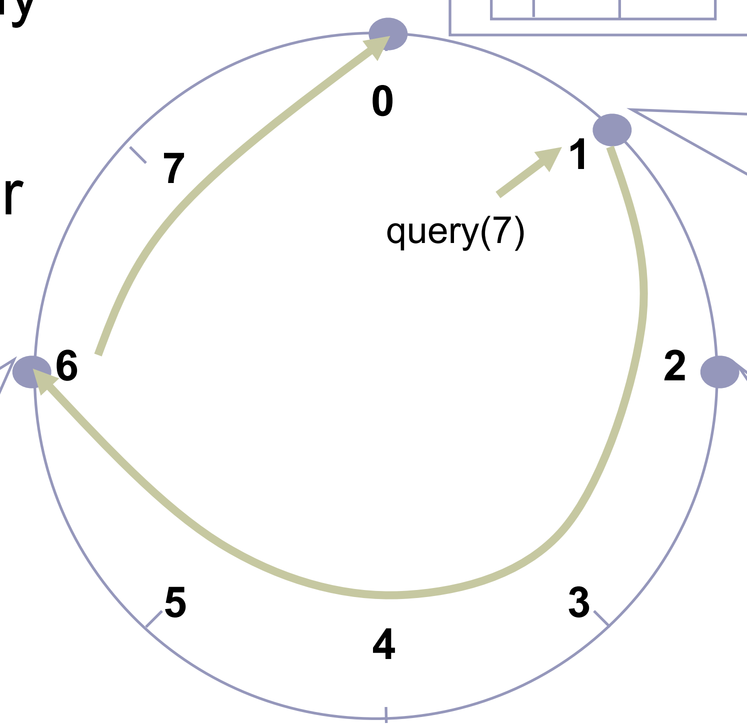
i	*id*+2^i	succ
0	3	6
1	4	6
2	6	6

Node 6 — Succ. Table

i	*id*+2^i	succ
0	7	0
1	0	0
2	2	2

Routing: Query

- Upon receiving a query for item *id*, a node
- Check whether stores the item locally
- If not, forwards the query to the largest node in its successor table that does not exceed *id*



Node 0 — Succ. Table

i	*id*+2^i	succ
0	1	1
1	2	2
2	4	0

Items: 7

Node 1 — Succ. Table

i	*id*+2^i	succ
0	2	2
1	3	6
2	5	6

Items: 1

Node 2 — Succ. Table

i	*id*+2^i	succ
0	3	6
1	4	6
2	6	6

Node 6 — Succ. Table

i	*id*+2^i	succ
0	7	0
1	0	0
2	2	2

What can DHTs do for us?

- Distributed object lookup
  - Based on object ID
- De-centralized file systems
  - CFS, PAST, Ivy
- Application Layer Multicast
  - Scribe, Bayeux, Splitstream
- Databases
  - PIER

Overview

- P2P Lookup Overview
- Centralized/Flooded Lookups
- Routed Lookups – Chord
- Comparison of DHTs

Comparison

- Many proposals for DHTs
  - Tapestry (UCB) -- Symphony (Stanford) -- 1hop (MIT)
  - Pastry (MSR, Rice) -- Tangle (UCB) -- conChord (MIT)
  - Chord (MIT, UCB) -- SkipNet (MSR,UW) -- Apocrypha (Stanford)
  - CAN (UCB, ICSI) -- Bamboo (UCB) -- LAND (Hebrew Univ.)
  - Viceroy (Technion) -- Hieras (U.Cinn) -- ODRI (TexasA&M)
  - Kademlia (NYU) -- Sprout (Stanford)
  - Kelips (Cornell) -- Calot (Rochester)
  - Koorde (MIT) -- JXTA’s (Sun)
- What are the right design choices? Effect on performance?

Deconstructing DHTs

Two observations:

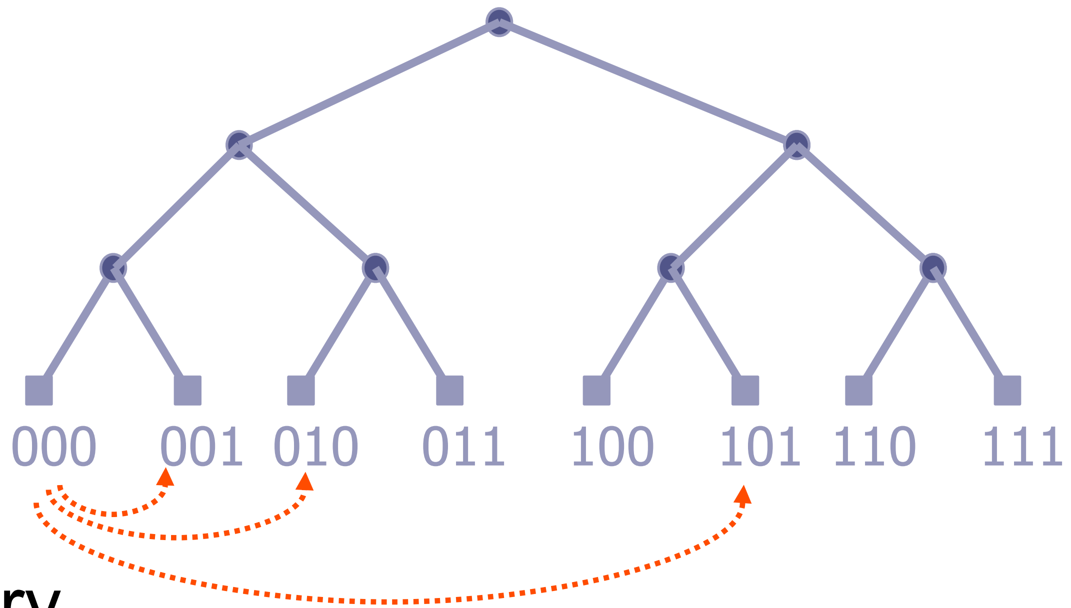
1. Common approach
   - N nodes; each labeled with a virtual identifier (128 bits)
   - define “distance” function on the identifiers
   - routing works to reduce the distance to the destination

2. DHTs differ primarily in their definition of “distance”
   - typically derived from (loose) notion of a routing geometry

DHT Routing Geometries

- Geometries:
  - Tree (Plaxton, Tapestry)
  - Ring (Chord)
  - Hypercube (CAN)
  - XOR (Kademlia)
  - Hybrid (Pastry)
- What is the impact of geometry on routing?

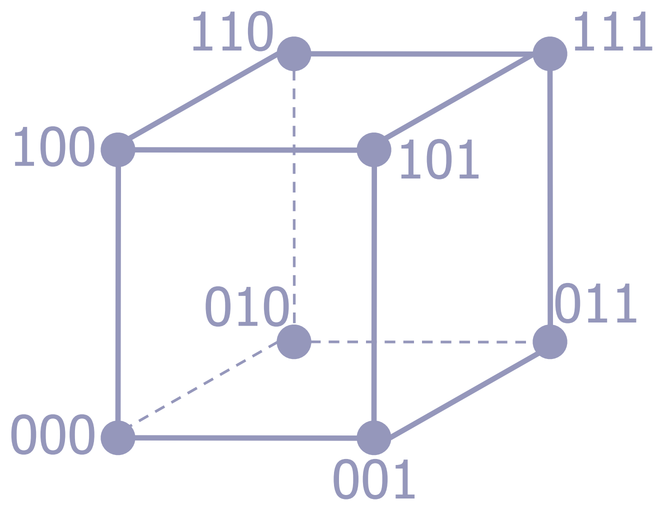
Tree (Plaxton, Tapestry)



Geometry

- nodes are leaves in a binary tree
- distance = height of the smallest common subtree
- logN neighbors in subtrees at distance 1,2,…,logN

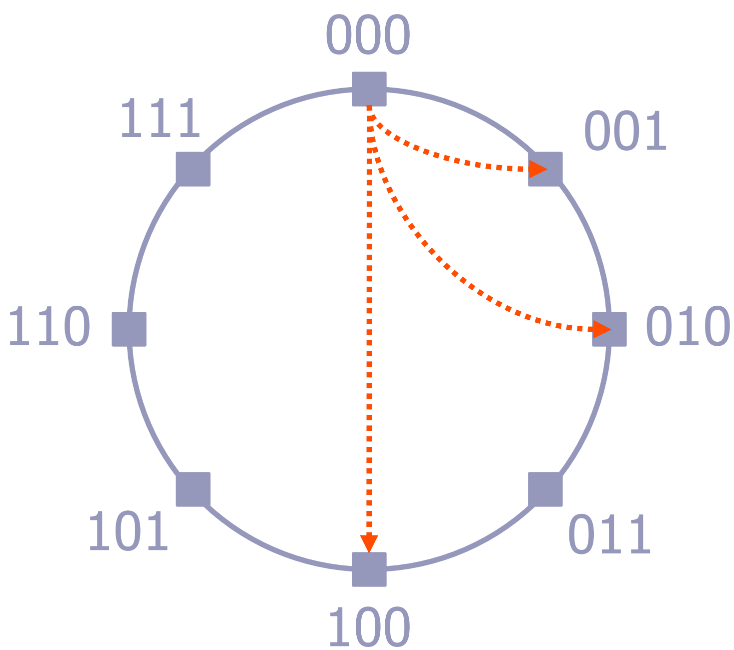
Hypercube (CAN)



Geometry

- nodes are the corners of a hypercube
- distance = #matching bits in the IDs of two nodes
- logN neighbors per node; each at distance=1 away

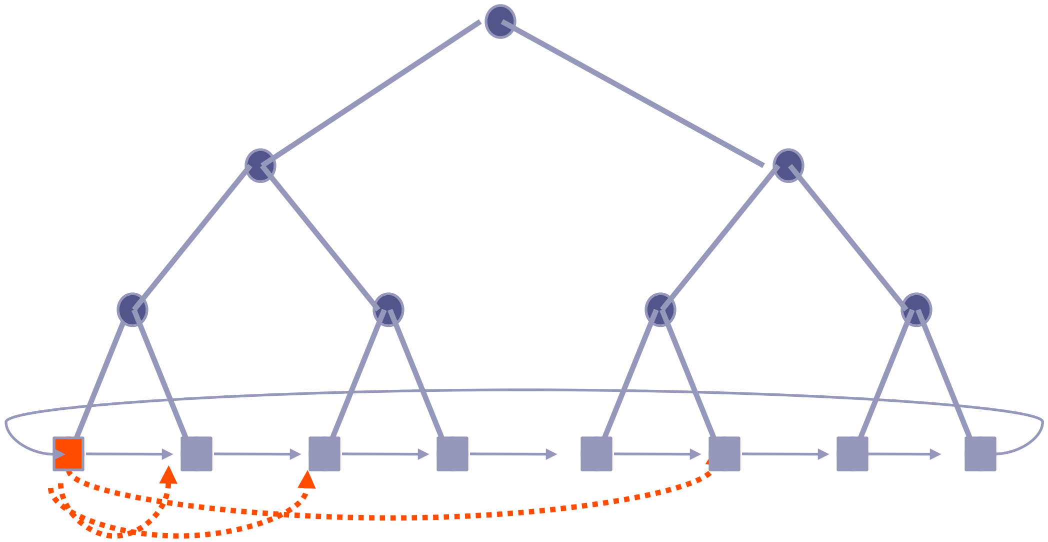
Ring (Chord)



Geometry

- nodes are points on a ring
- distance = numeric distance between two node IDs
- logN neighbors exponentially spaced over 0…N

Hybrid (Pastry)



Geometry:

- combination of a tree and ring
- two distance metrics
- default routing uses tree; fallback to ring under failures
  - neighbors picked as on the tree

XOR (Kademlia)

00 ⟷ 01 ⟷ 10 ⟷ 11

01 ⟷ 00 ⟷ 11 ⟷ 10

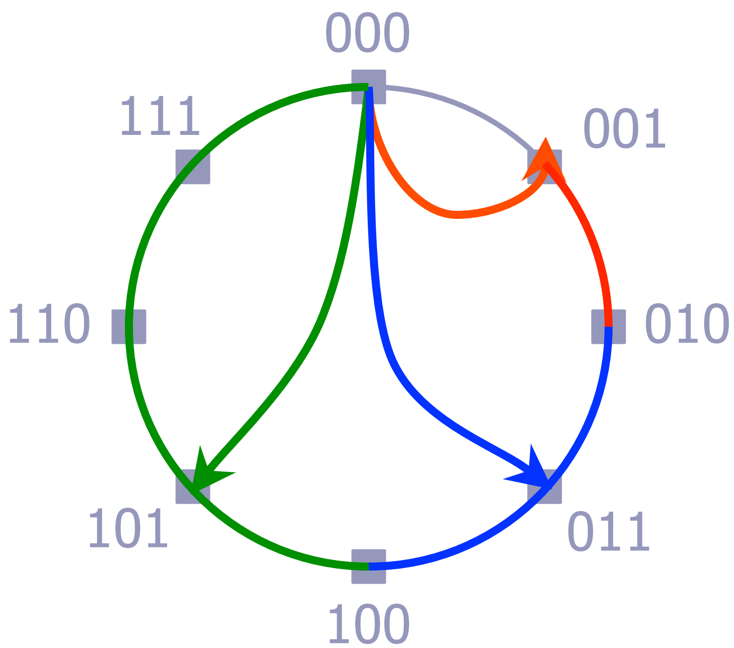
Geometry:

- distance(A,B) = A **XOR** B
- logN neighbors per node spaced exponentially
- not a ring because there is no single consistent ordering of all the nodes

Geometry's Impact on Routing

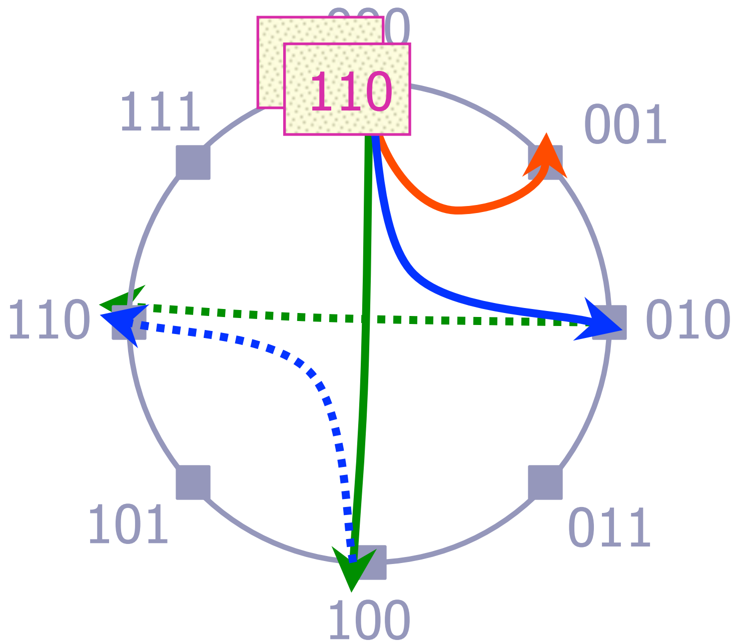
- Routing
  - Neighbor selection: how a node picks its routing entries
  - Route selection: how a node picks the next hop
- Proposed metric: flexibility
  - amount of freedom to choose neighbors and next-hop paths
    - FNS: flexibility in neighbor selection
    - FRS: flexibility in route selection
  - intuition: captures ability to "tune" DHT performance
  - single predictor metric dependent only on routing issues

FNS for Ring Geometry



- Chord algorithm picks **ith** neighbor at **2^i** distance
- A different algorithm picks **ith** neighbor from **$[2^i , 2^{i+1})$**

FRS for Ring Geometry



- Chord algorithm picks neighbor closest to destination
- A different algorithm picks the best of alternate paths

Flexibility: at a Glance

Flexibility	Ordering of Geometries
Neighbors (FNS)	**Hypercube << Tree, XOR, Ring, Hybrid** (1) (2^{i-1})
Routes (FRS)	**Tree << XOR, Hybrid < Hypercube < Ring** (1) (logN/2) (logN/2) (logN)

Geometry → Flexibility → Performance?

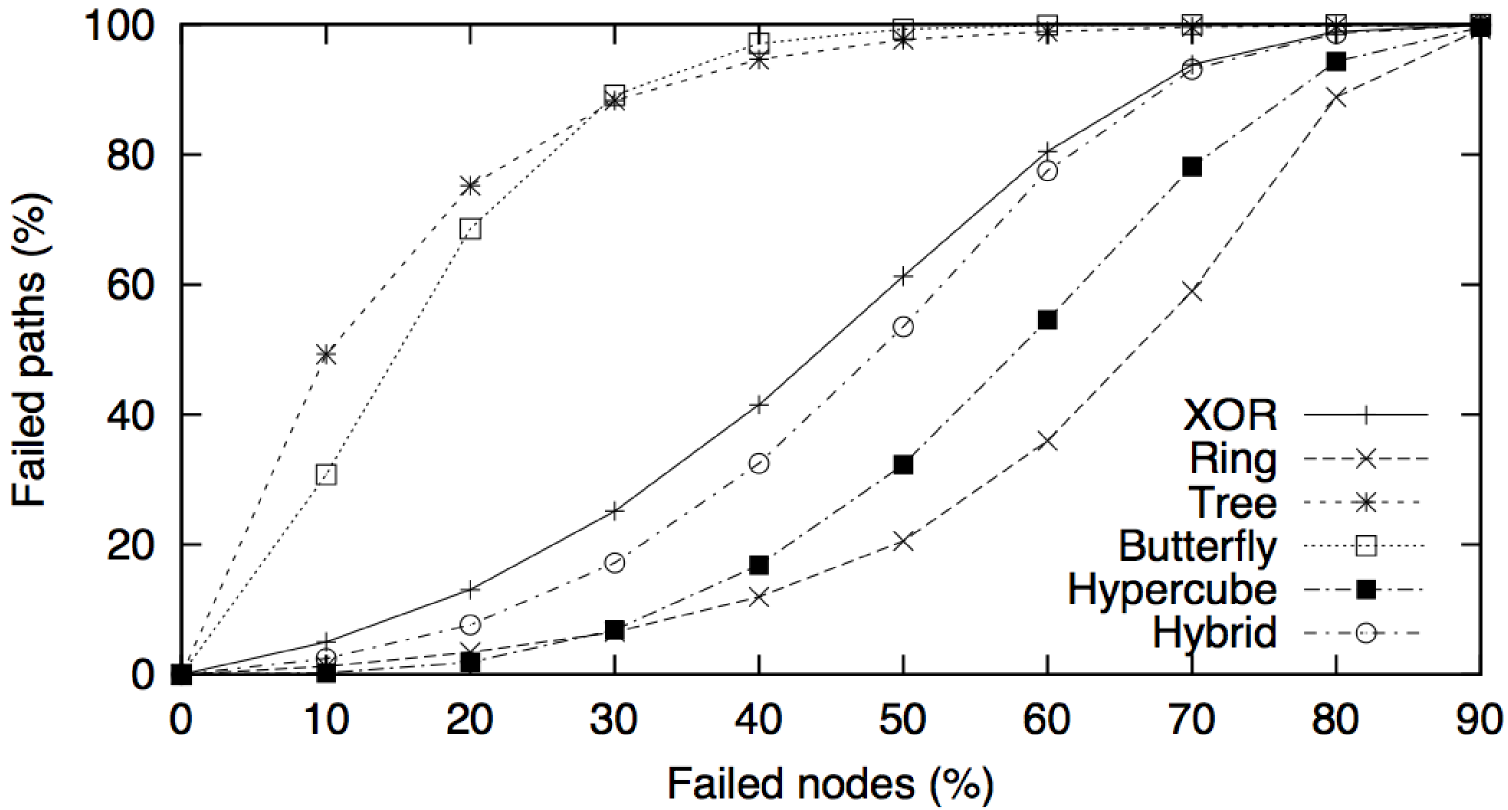
Validate over three performance metrics:

1. resilience
2. path latency
3. path convergence

Metrics address two typical concerns:

- ability to handle node failure
- ability to incorporate proximity into overlay routing

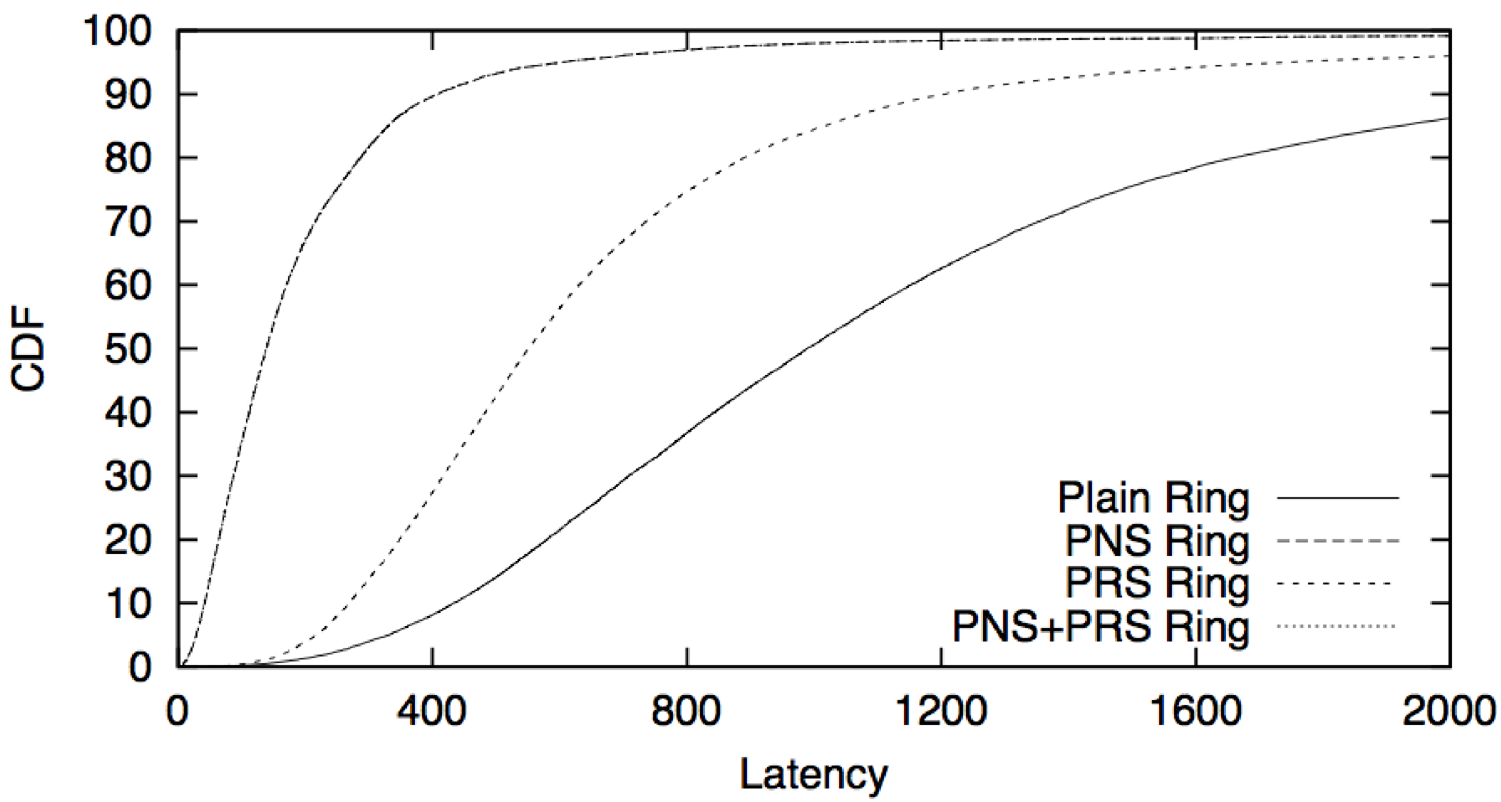
Does flexibility affect static resilience?



Tree << XOR ≈ Hybrid < Hypercube < Ring

Flexibility in Route Selection matters for Static Resilience

Which is more effective, FNS or FRS?



Plain << FRS << FNS ≈ FNS+FRS

Neighbor Selection is much better than Route Selection

Understanding DHT Routing: Conclusion

- What makes for a "good" DHT?
  - one answer: a flexible routing geometry

- Result: Ring is most flexible

Next Lecture

- DNS, Web and P2P
- Required readings
  - Peer-to-Peer Systems
  - Do incentives build robustness in BitTorrent?