Kademlia: A Peer-to-peer Information System Based on the XOR Metric

Petar Maymounkov and David Mazières
New York University

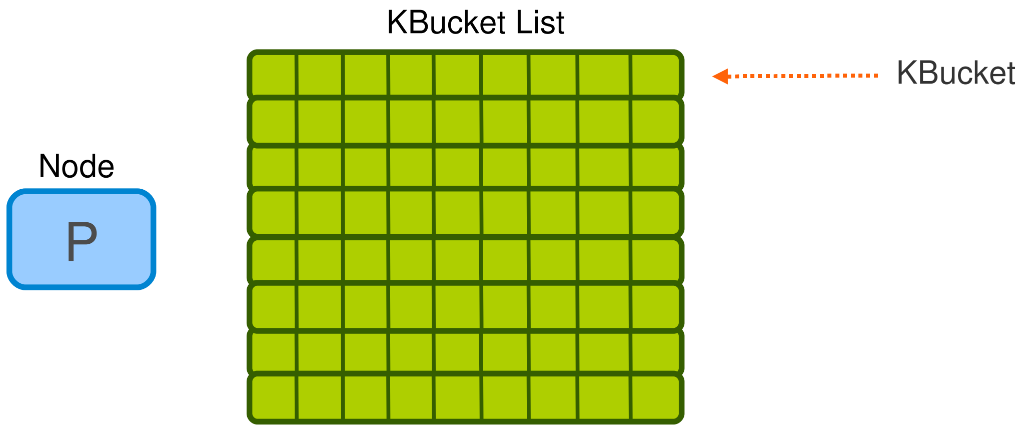
(Presented In 1th International Workshop on P2P Systems 2002)

Presented by: Amir H. Payberah
amir@sics.se

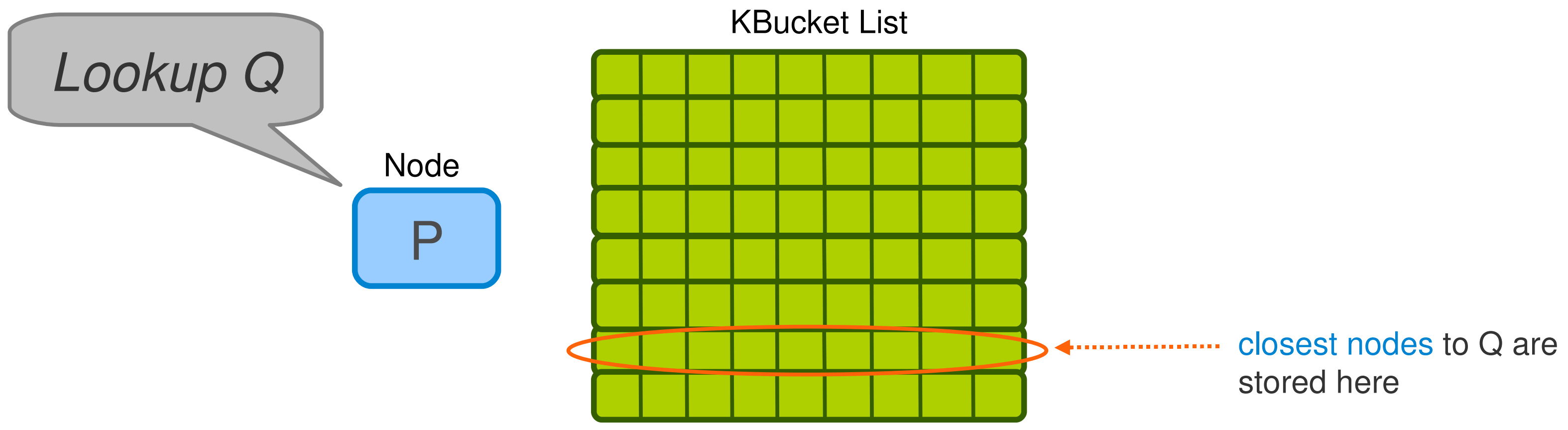
BitTorrent™

Core Idea

Core Idea - 1

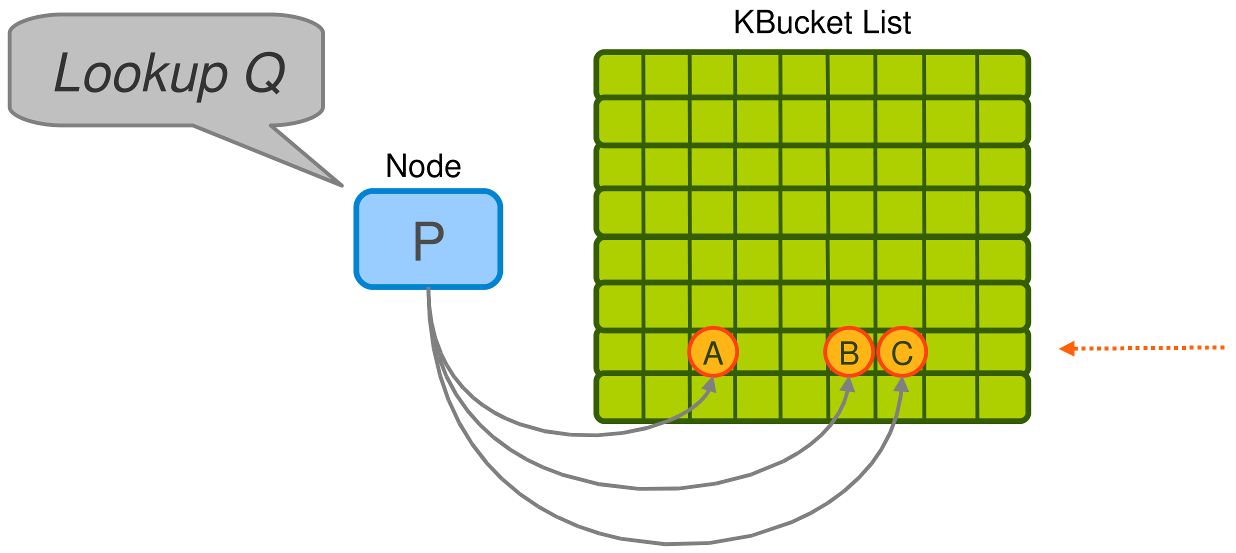


Core Idea - 2



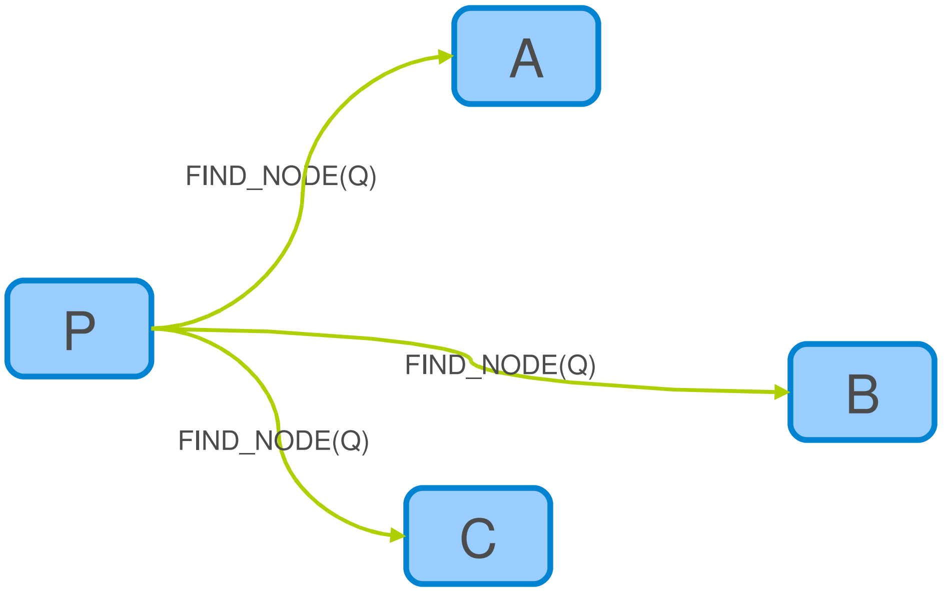
- Closest nodes in ID space
- Having more common bits in prefix (will be back to it)

Core Idea - 3

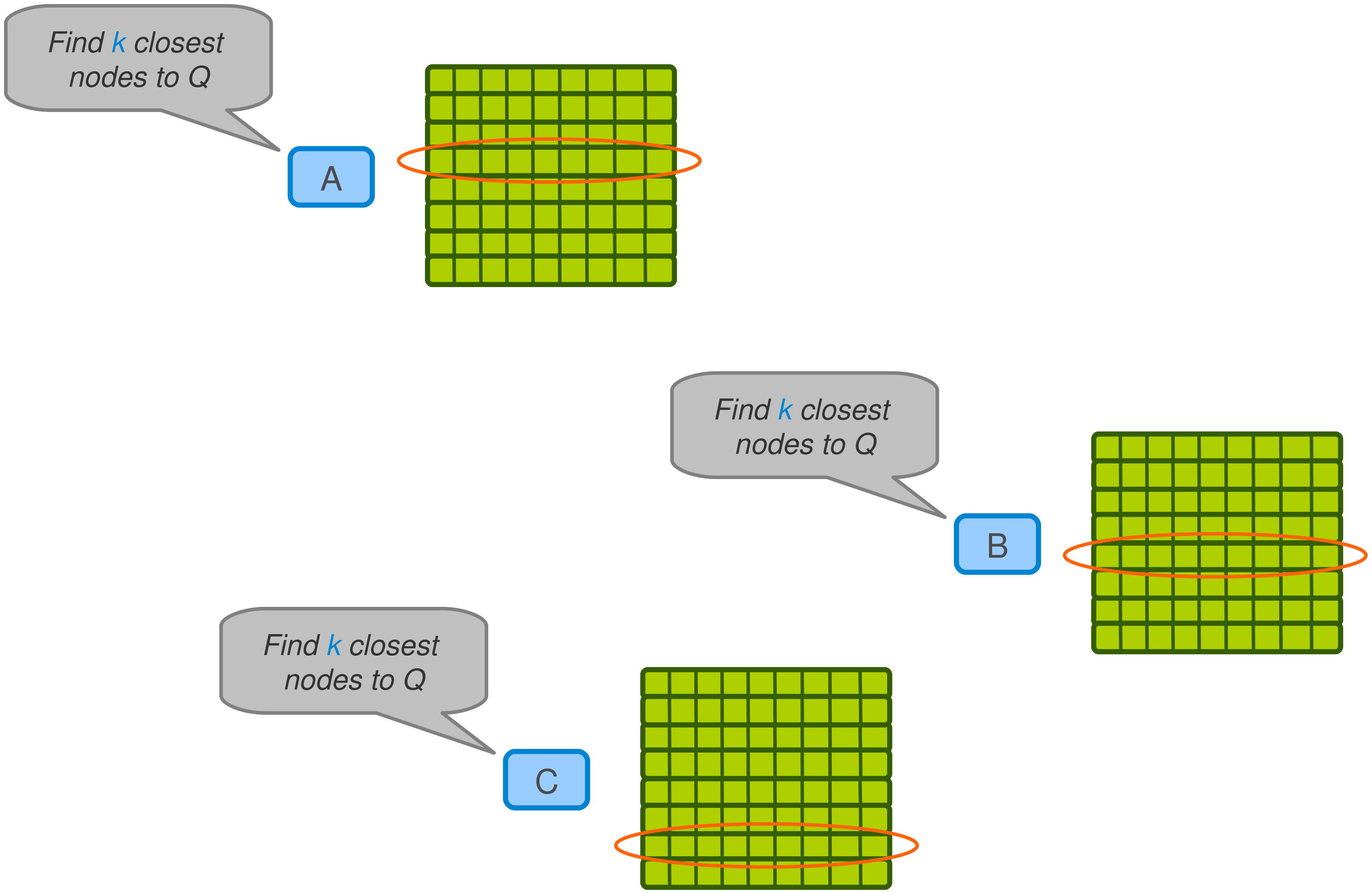


... and select α nodes from the appropriate kbucket

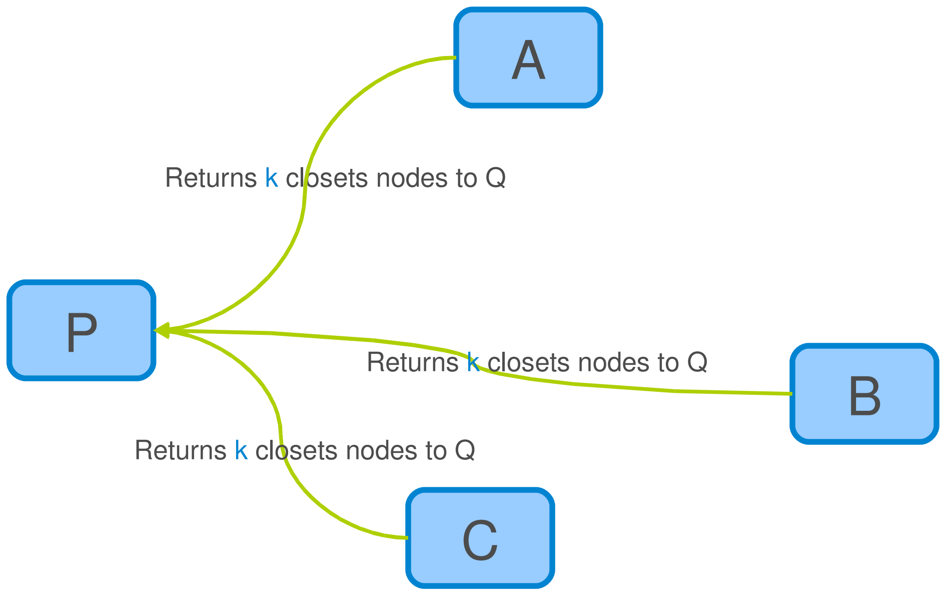
Core Idea - 4



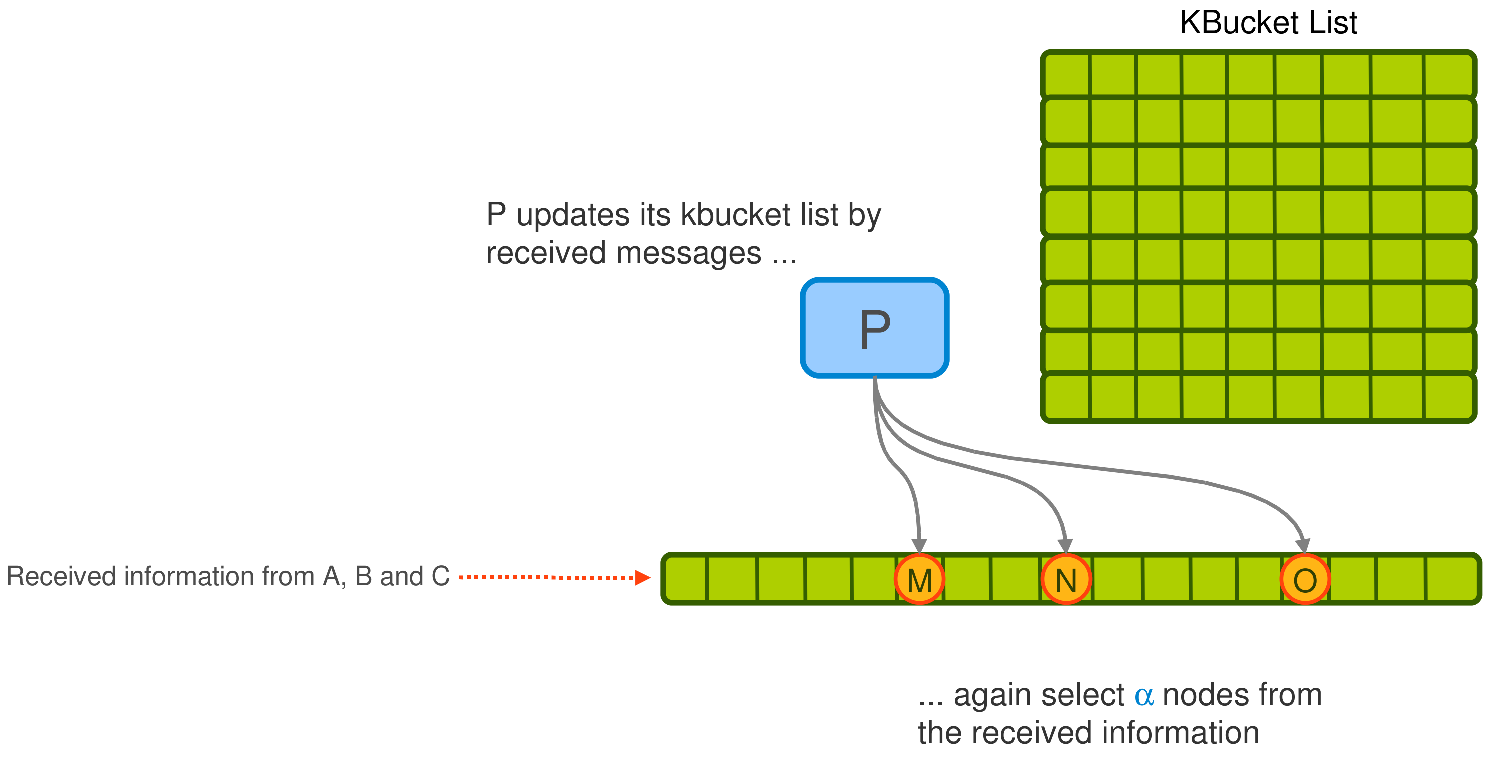
Core Idea - 5


Find k closest nodes to Q
A
Find k closest nodes to Q
B
Find k closest nodes to Q
C

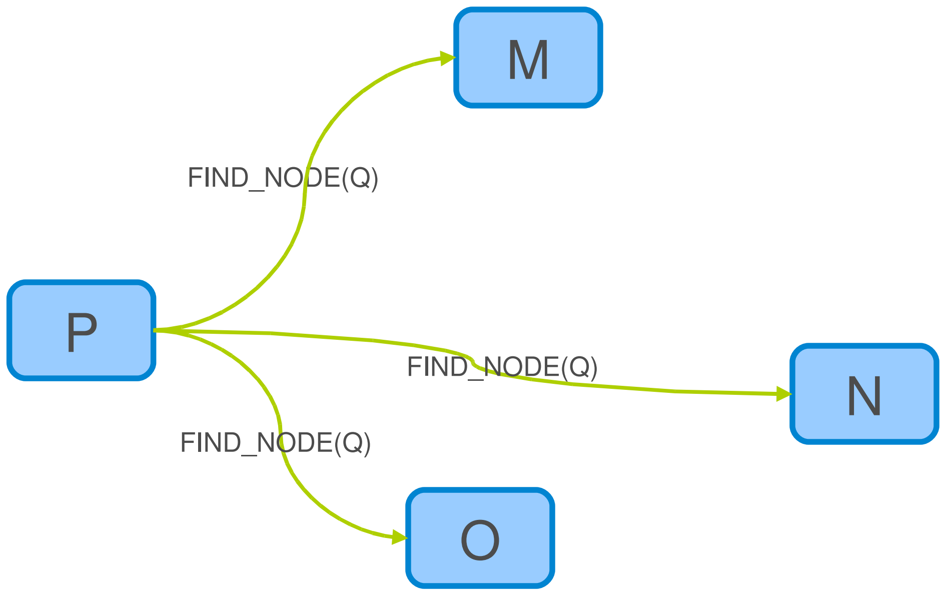
Core Idea - 6


A
Returns k closets nodes to Q
P
Returns k closets nodes to Q
B
Returns k closets nodes to Q
C

Core Idea - 7


KBucket List
P updates its kbucket list by received messages ...
P
Received information from A, B and C
M
N
O
... again select α nodes from the received information

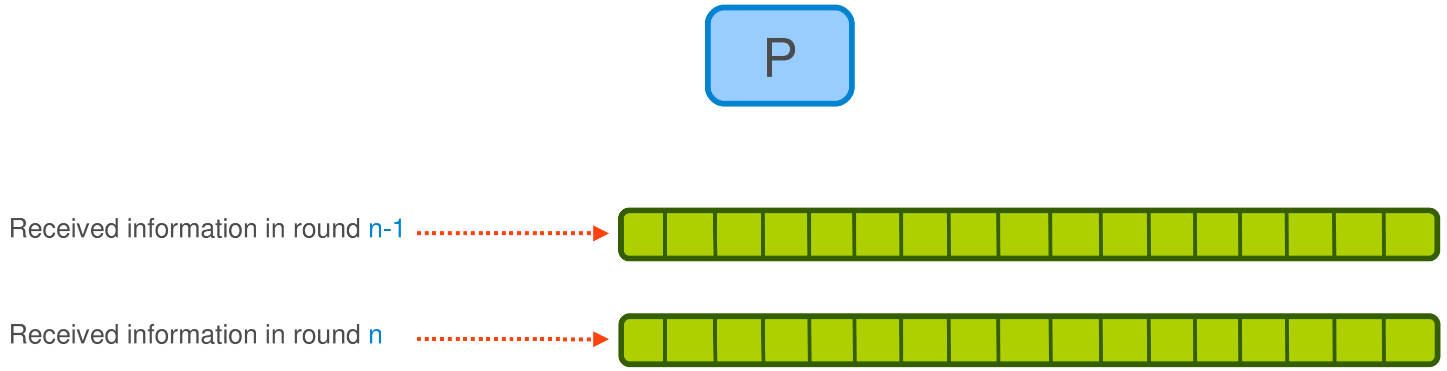
Core Idea - 8



Core Idea - 9

Repeats this procedure iteratively until ...

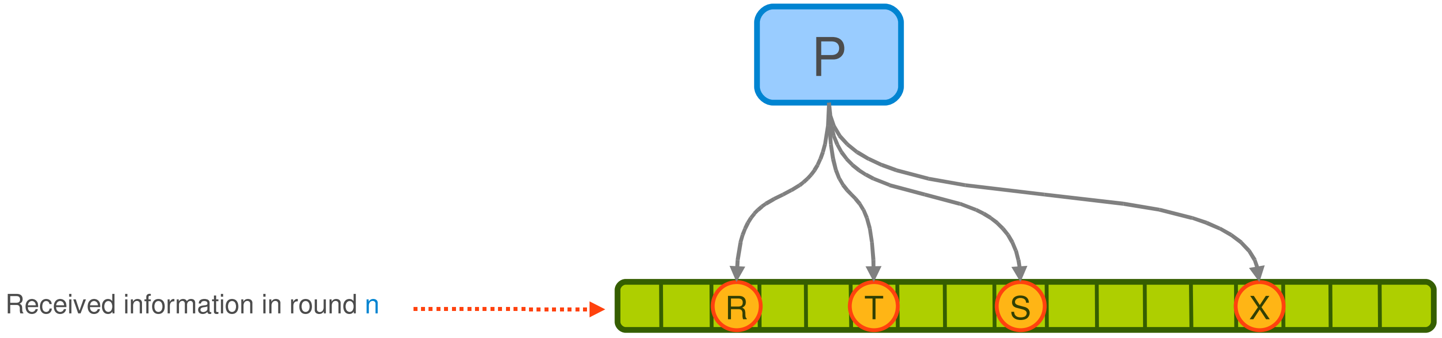
Core Idea - 10


P
Received information in round n-1
Received information in round n

... until received information in round n-1 and n are the same.

Core Idea - 11

P resends the FIND_NODE to k closest nodes it has not already queried ...



Let's Look Inside of Kademlia

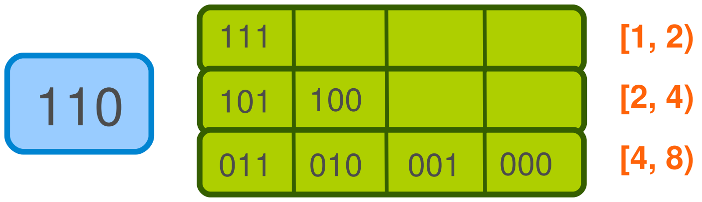
System Description

- Each Kademlia node has a 160-bit node ID.
- To publish and find <key,value> pairs, Kademlia relies on a notion of distance between two Ids.
- Distance between id1 and id2: d(id1, id2) = id1 XOR id2
  - If ID space is 3 bits:

$$\begin{aligned} d(1, 5) &= d(001_2, 101_2) \\ &= 001_2 \text{ XOR } 101_2 \\ &= 100_2 \\ &= 4 \end{aligned}$$

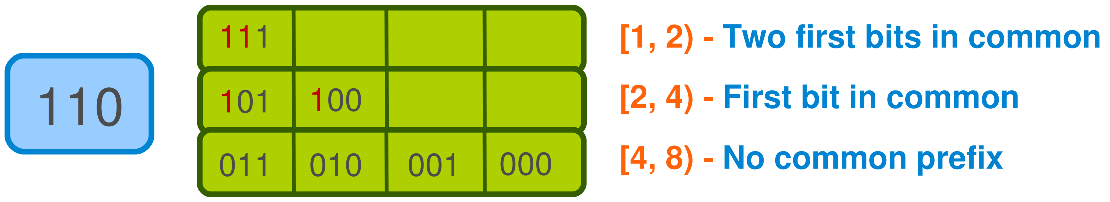
Node State

- Kbucket: each node keeps a list of information for nodes of distance between 2^i and 2^{i+1}.
  - $0 \le i < 160$
  - Sorted by time last seen.



Node State

- Kbucket: each node keeps a list of information for nodes of distance between 2^i and 2^{i+1}.
  - $0 <= i < 160$
  - Sorted by time last seen.


110
111
101
100
011
010
001
000
[1, 2) - Two first bits in common
[2, 4) - First bit in common
[4, 8) - No common prefix

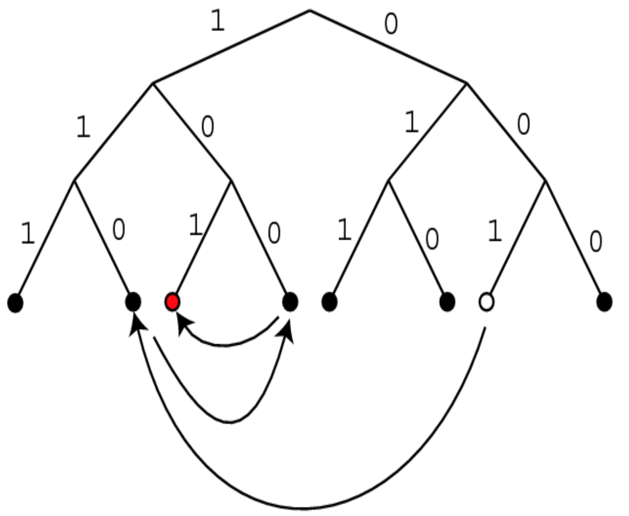
Kademlia RPCs

- PING
  - Probes a node to see if it is online.
- STORE
  - Instructs a node to store a <key, value> pair.
- FIND_NODE
  - Returns information for the k nodes it knows about closest to the target ID.
  - It can be from one kbucket or more.
- FIND_VALUE
  - Like FIND_NODE, ...
  - But if the recipient has stored they <key, value>, it just returns the stored value.

Store Data

- The <key, value> data is stored in k closest nodes to the key.

Lookup Service



Step1 001

000				[1, 2)
010	011			[2, 4)
110	100	111	101	[4, 8)

Step2 110

111				[1, 2)
101	100			[2, 4)
011	010	001	000	[4, 8)

Step3 100

101				[1, 2)
111	110			[2, 4)
001	000	010	011	[4, 8)

Maintaining Kbucket List (Routing Table)

- When a Kademlia node receives any message from another node, it updates the appropriate kbucket for the sender's node ID.
- If the sending node already exists in the kbucket:
  - Moves it to the tail of the list.
- Otherwise:
  - If the bucket has fewer than k entries:
    - Inserts the new sender at the tail of the list.
  - Otherwise:
    - Pings the kbucket's least-recently seen node:
    - If the least-recently seen node fails to respond:
      - it is evicted from the k-bucket and the new sender inserted at the tail.
    - Otherwise:
      - it is moved to the tail of the list, and the new sender's contact is discarded.

Maintaining Kbucket List (Routing Table)

- Buckets will generally be kept constantly fresh, due to traffic of requests travelling through nodes.
- When there is no traffic: each peer picks a random ID in kbucket's range and performs a node search for that ID.

Join

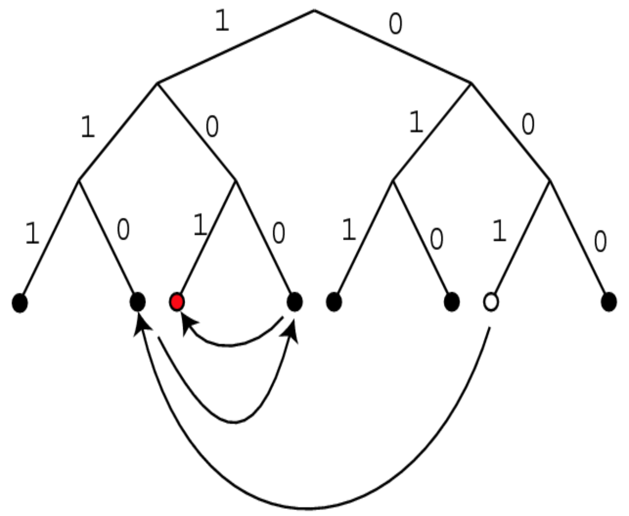
- Node P contacts to an already participating node Q.
- P inserts Q into the appropriate kbucket.
- P then performs a node lookup for its own node ID.

Leave And Failure

- No action!
- If a node does not respond to the PING message, remove it from the table.

DONE!

A Page To Remember



Step1 001

000				[1, 2)
010	011			[2, 4)
110	100	111	101	[4, 8)

Step2 110

111				[1, 2)
101	100			[2, 4)
011	010	001	000	[4, 8)

Step3 100

101				[1, 2)
111	110			[2, 4)
001	000	010	011	[4, 8)

Question?