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

Based on slides by Amir H. Payberah (amir@sics.se)

BitTorrent™

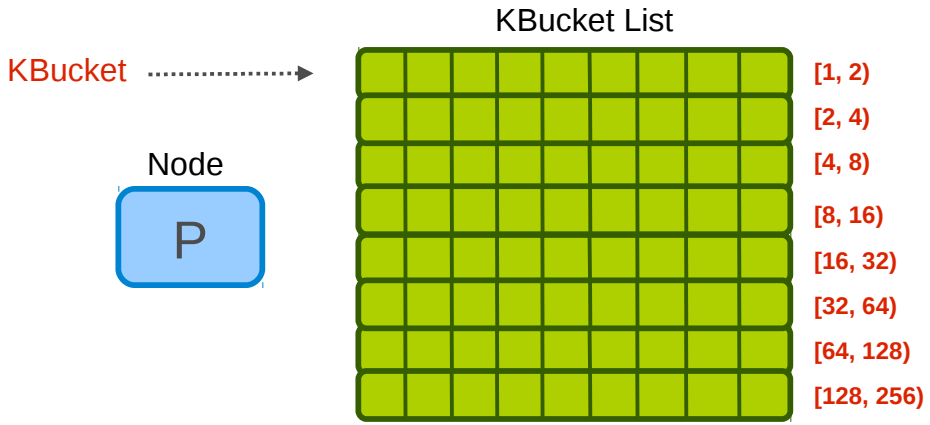
Kademlia Basics

- Kademlia is a key-value(object) store.
- Each object is stored at the k closest nodes to the object's ID.
- Distance between id1 and id2: d(id1, id2) = id1 XOR id2
  - If ID space is 3 bits:

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

Kademlia Routing Table

- Kbucket: each node keeps a list of references to nodes (*contacts*) of distance between 2^i and 2^{i+1} for i=1 to i=N.
- Each Kbucket has max k entries.

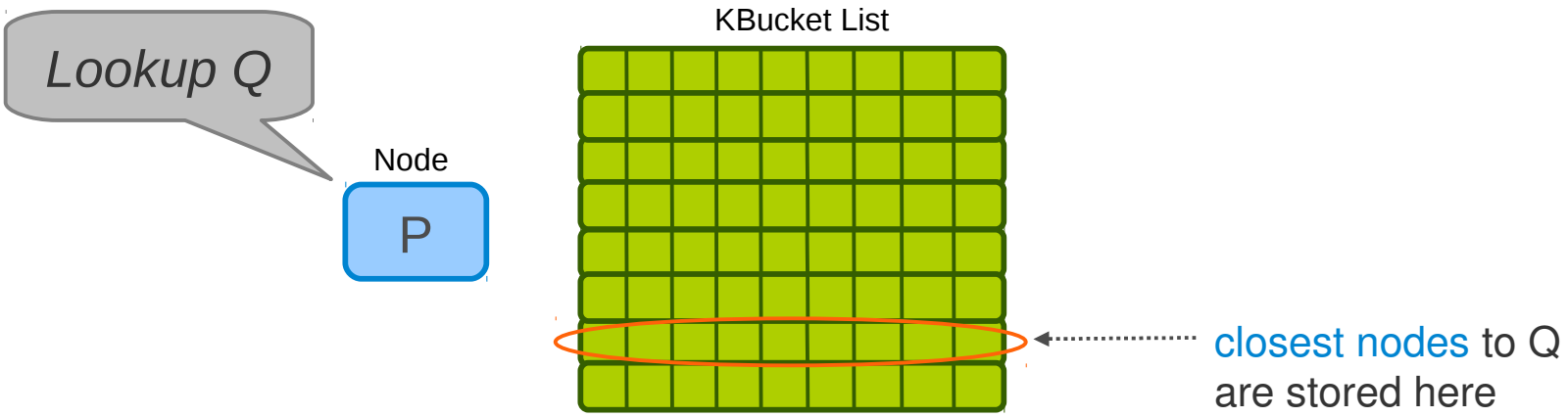


Kademlia Tuning Parameters

- B is the size in bits of the keys used to identify nodes and store and retrieve data; in basic Kademlia this is 160, the length of an SHA1 digest (hash).
- k is the maximum number of contacts stored in a Kbucket; this is normally 20.
- alpha (α) represents the degree of parallelism in network calls, usually 3.

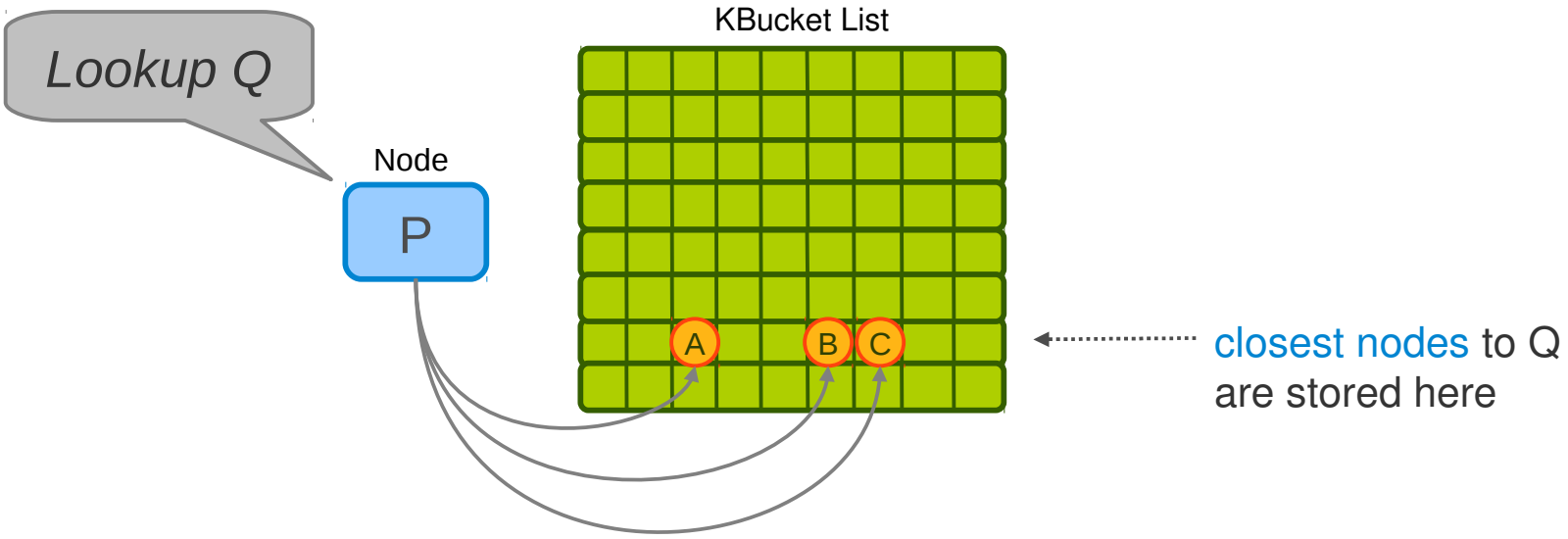
- Other constants used in Kad:
  - tExpire = 86400s, the time after which a key/value pair expires; this is a time-to-live (TTL) from the original publication date
  - tRefresh = 3600s, after which an otherwise unaccessed bucket must be refreshed
  - tReplicate = 3600s, the interval between Kademlia replication events, when a node is required to publish its entire database
  - tRepublish = 86400s, the time after which the original publisher must republish a key/value pair

FIND_NODE in Kademlia



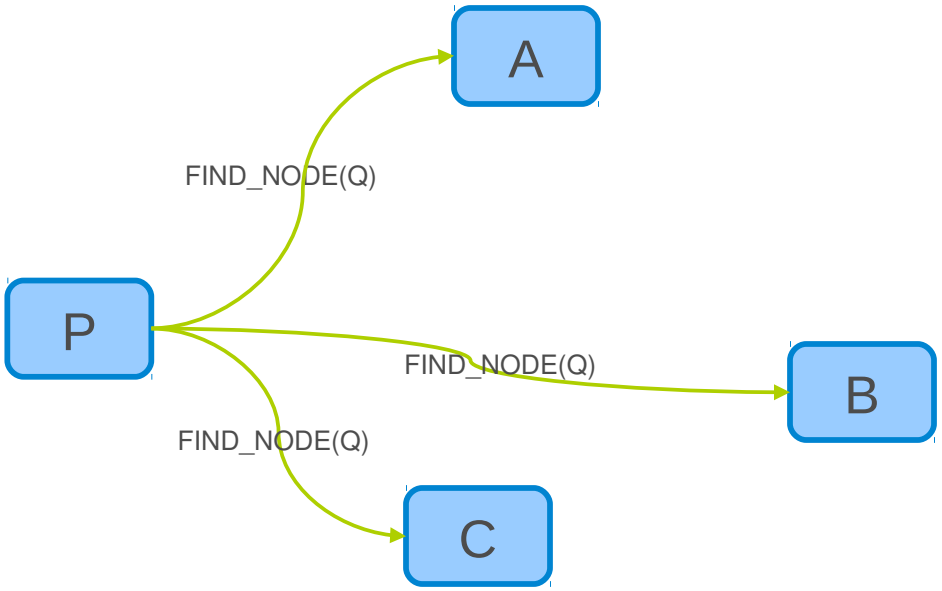
- Closest nodes in ID space

FIND_NODE in Kademlia

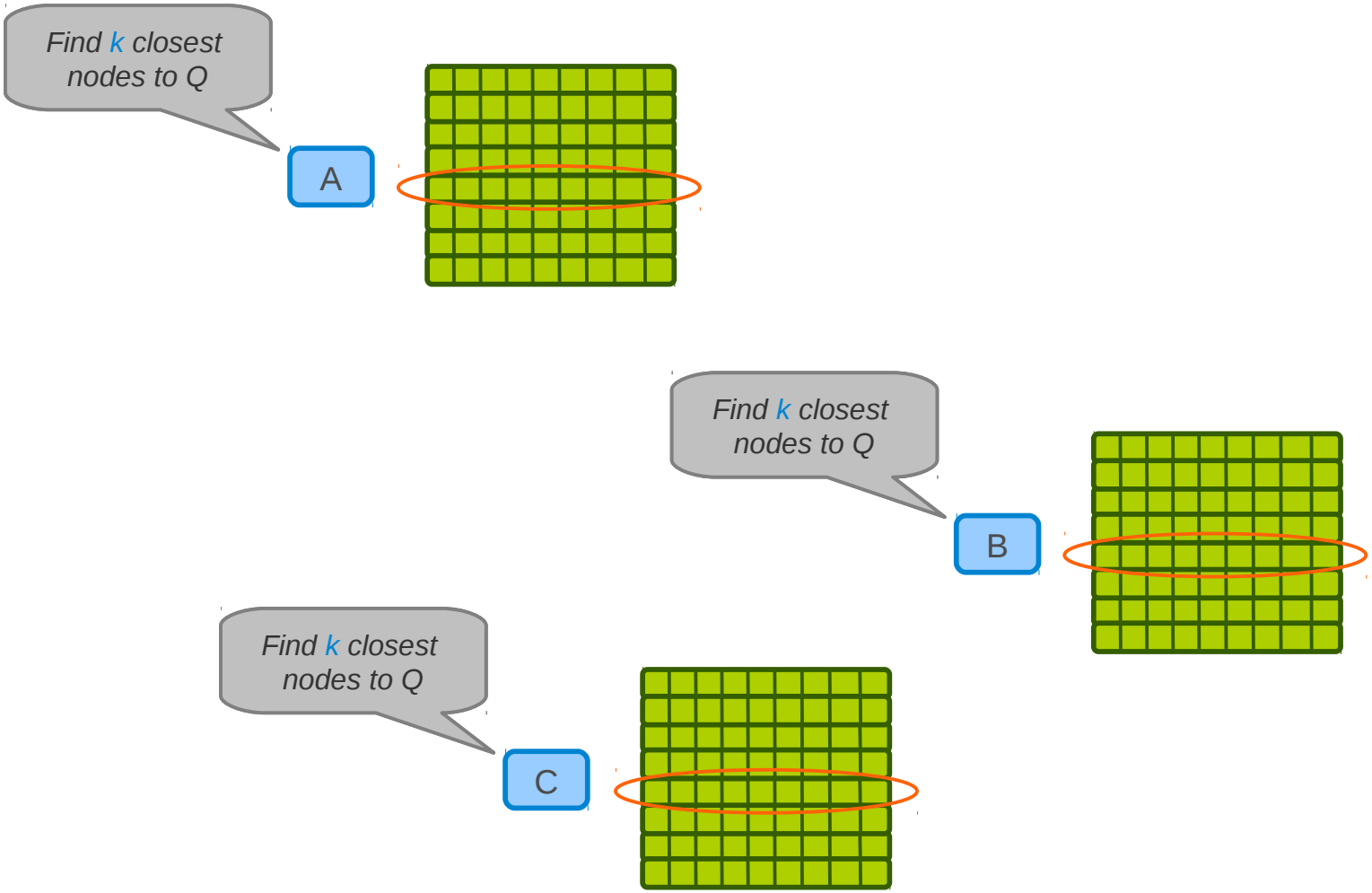


... and select α nodes from
the appropriate kbucket

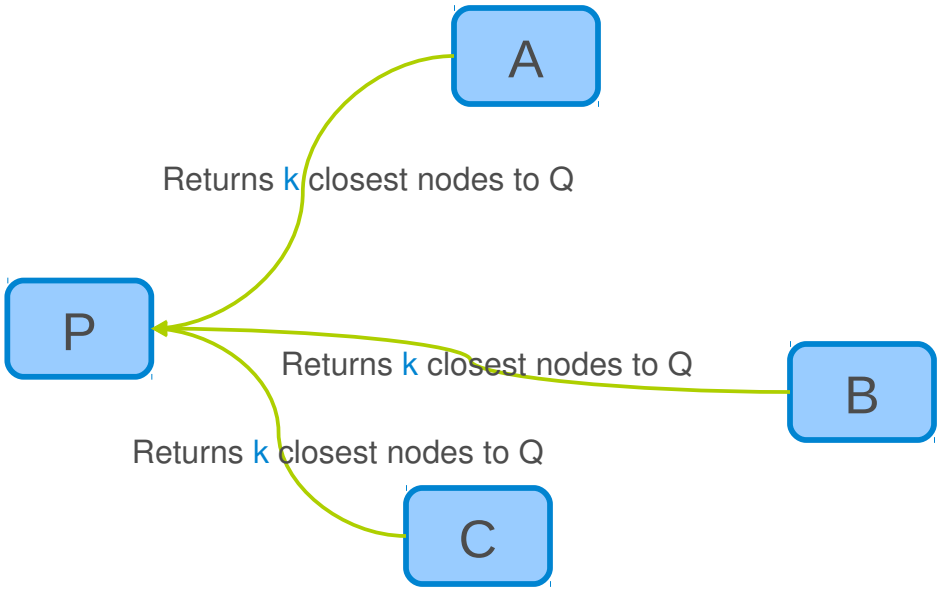
FIND_NODE in Kademlia



FIND_NODE in Kademlia

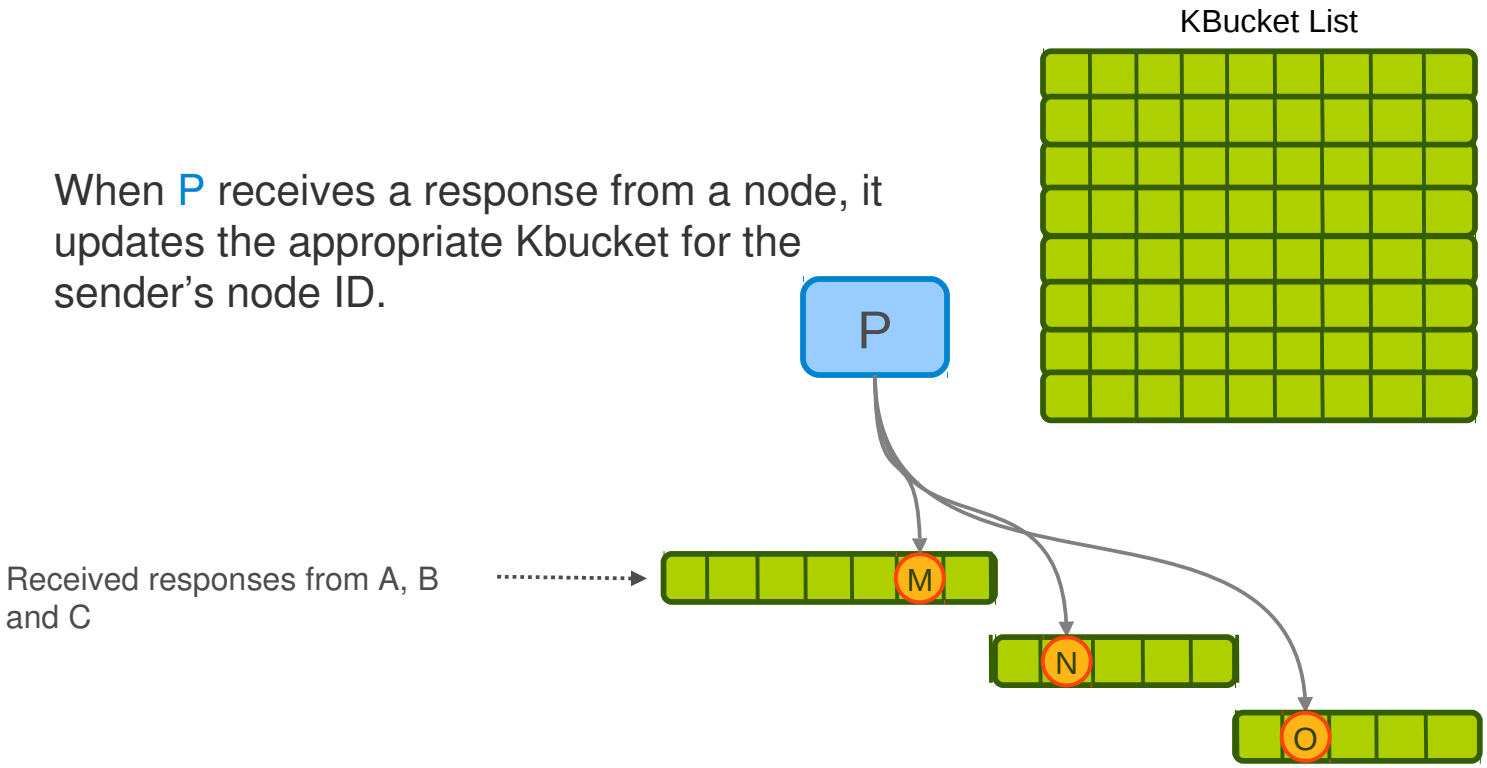


FIND_NODE in Kademlia



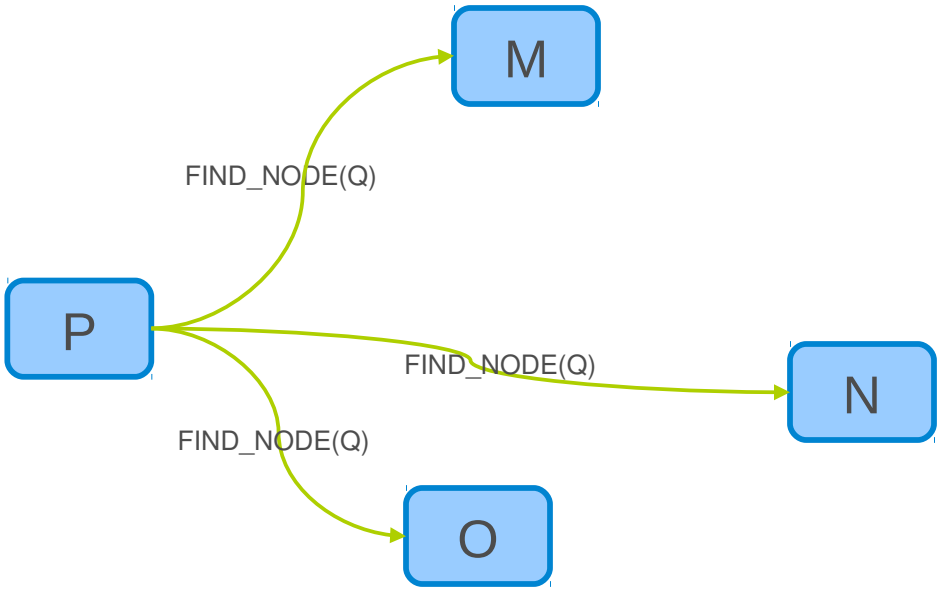
FIND_NODE in Kademlia, Update Kbuckets

When P receives a response from a node, it updates the appropriate Kbucket for the sender's node ID.

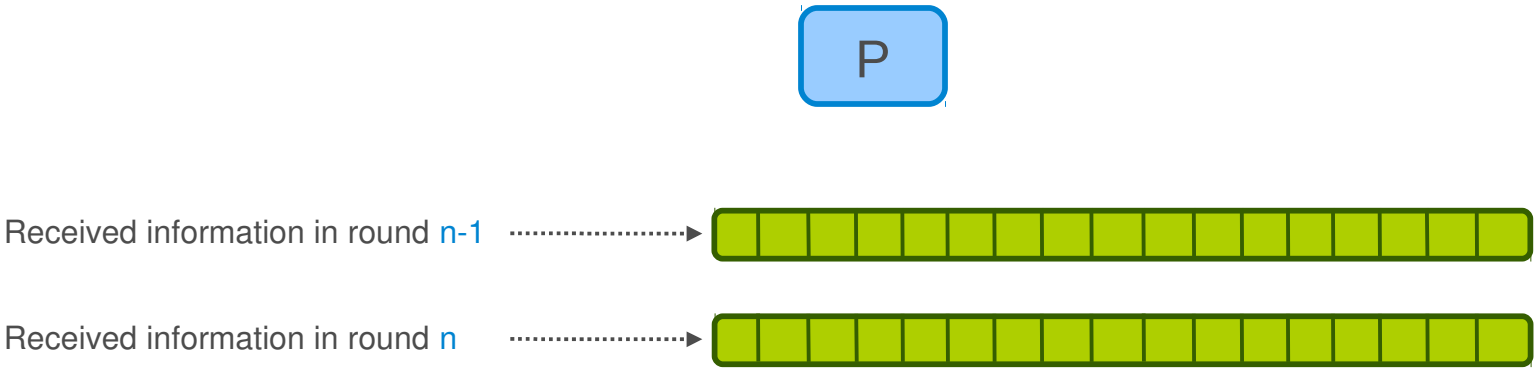


P issues up to α new requests to nodes it has not yet queried from the set of nodes received in the responses

FIND_NODE in Kademlia



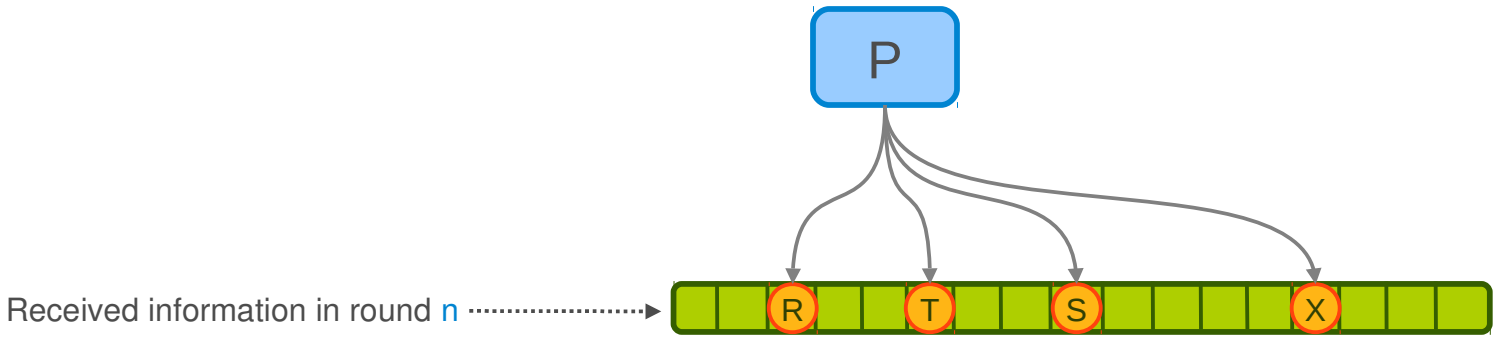
FIND_NODE in Kademlia



Repeats this procedure iteratively until received information in round n-1 and n are the same.

FIND_NODE in Kademlia

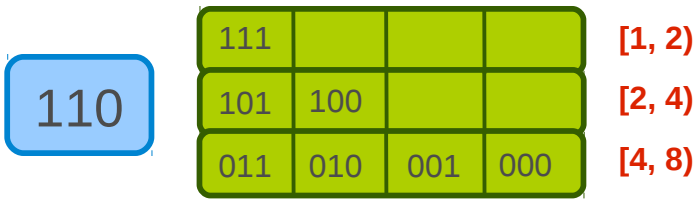
P resends the FIND_NODE to k closest nodes it has not already queried ...



Let's Look Inside Kademlia

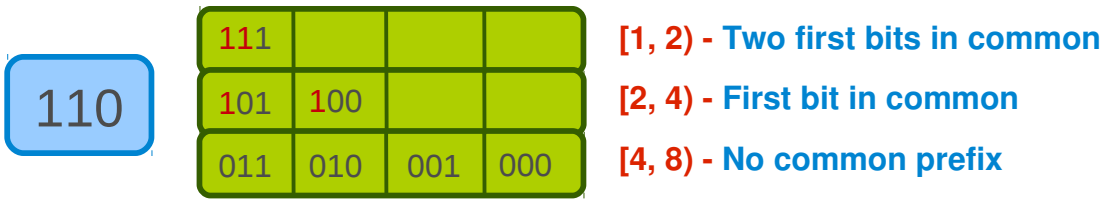
Node State

- 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.



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.



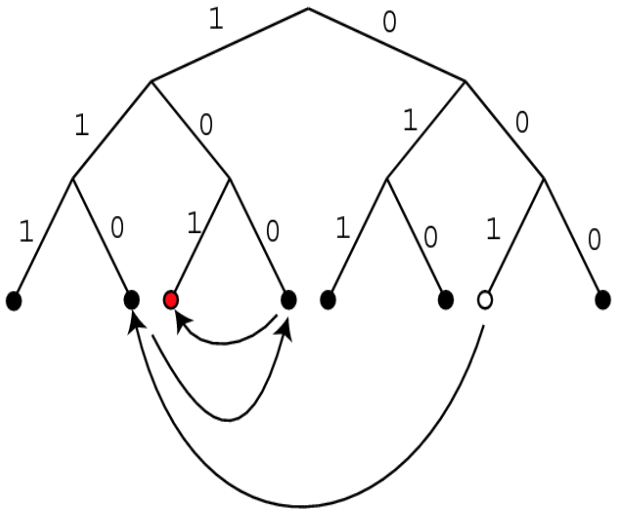
Kademlia RPCs

- 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		[4, 8)

Step2 110

111				[1, 2)
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.

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.

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 should 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 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

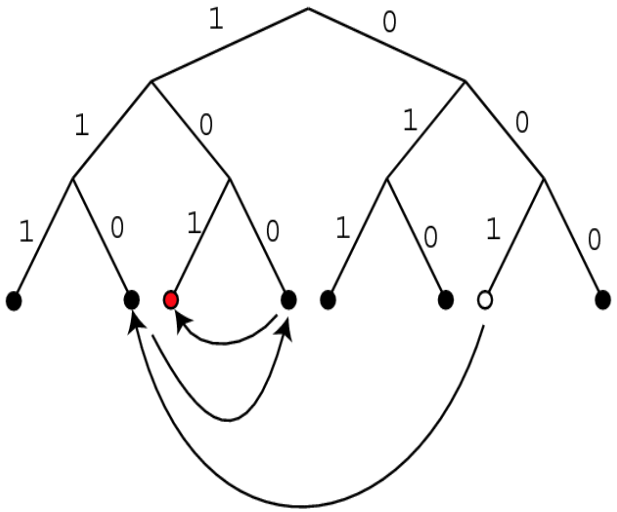
- No action!
- If a node does not respond to the PING message, remove it from the table.

Kademlia vs. Chord

Kademlia vs. Chord

- like Chord
  - When $\alpha = 1$ the lookup algorithm resembles Chord's in term of message cost.
- Unlike Chord
  - XOR metric is symmetric, while Chord's metric is asymmetric.

Summary



Step1 001

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

Step2 110

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

Step3 100

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

References

- Kademlia Specification
  - http://xlattice.sourceforge.net/components/protocol/kademlia/specs.html
- Petar Maymounkov and David Mazieres, "Kademlia: A Peer-to-Peer Information System Based on the XOR Metric", IPTPS '02
  - http://www.cs.rice.edu/Conferences/IPTPS02/109.pdf
- Daniel Stutzbach and Reza Rejaie, "Improving Lookup Performance over a Widely-Deployed DHT", INFOCOM '06
  - http://www.barsoom.org/~agthorr/papers/infocom-2006-kad.pdf
- Raul Jimenez, Flutra Osmani and Bjorn Knutsson, “Sub-Second Lookups on a Large-Scale Kademlia-Based Overlay”, P2P '11.
  - http://people.kth.se/~rauljc/p2p11/jimenez2011subsecond.pdf