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

Petar Maymounkov and David Mazières
New York University

Credits

Amir H. Payberah (amir@sics.se)
Seif Haridi (haridi@kth.se)

KADEMLIA, is currently used by many (not only P2P) applications

Core Idea

KADEMLIA, as CHORD is another DHT,
Distributed Hash Table, i.e.
a DB of the shape : KEY—>VALUE
FULLY DECENTRALIZED

Definition

Kademlia is a peer-to-peer (key-value) storage and lookup system

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

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

Main concepts:

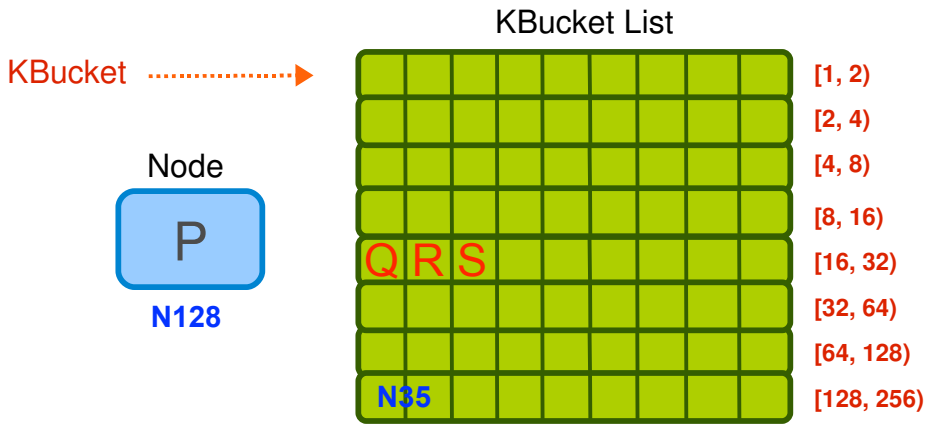
1-Binary tree topology i.e. nodes are leafs in a binary-tree, 2-tree-like routing, 3- XOR metric space, 4- SHA for keys and nodes (to ensure load balancing), 5- Values memorised more than once and on “closest nodes”, 6- Fast stabilization, 7- Fast lookup O(log N), 8- Designed with concurrency in mind…

ROUTING TABLES ARE (A BIT) MORE RICH THAN CHORD !!!!

Core Idea - 1

- Kbucket: each node keeps a list of information for nodes of distance between 2^i and 2^{i+1}.

In this slide-set:
the logical space is 2^8 = 256, i.e.
N1 … N256 potential nodes

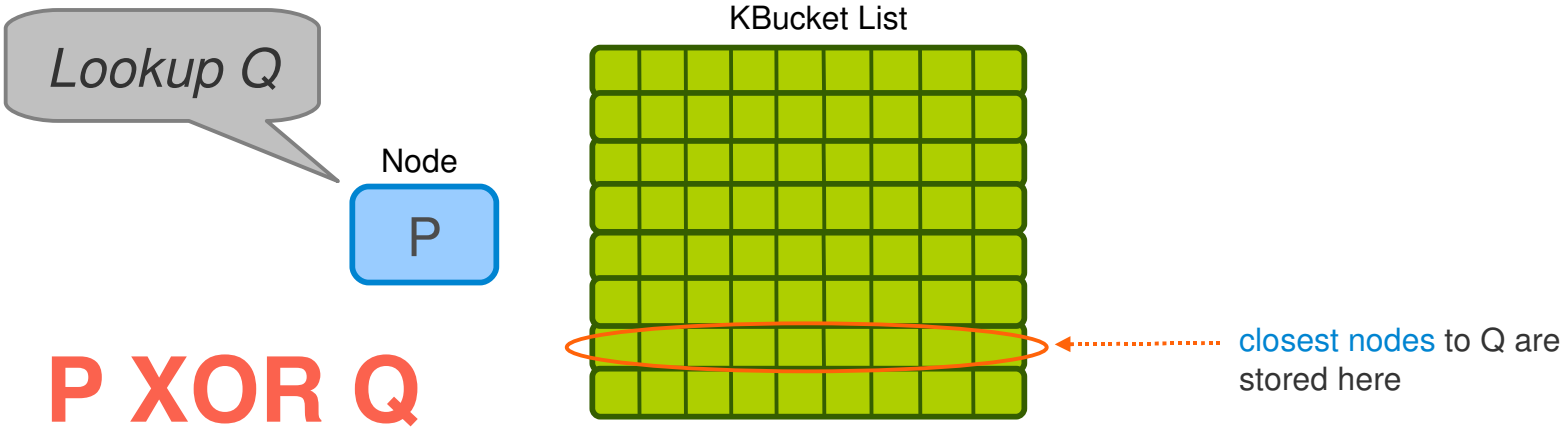


Every KBucket of P contain nodes ex: Q,R,S

if and only if P XOR {Q,R,S} ∈ [2^i,2^(i+1))

Core Idea - 2

Key K and nodes N are hashed with SHA1/2/3 function. Value V associated to key K is stored in the “closest” nodes Ns to K. We use XOR metric and a built-in replication factor



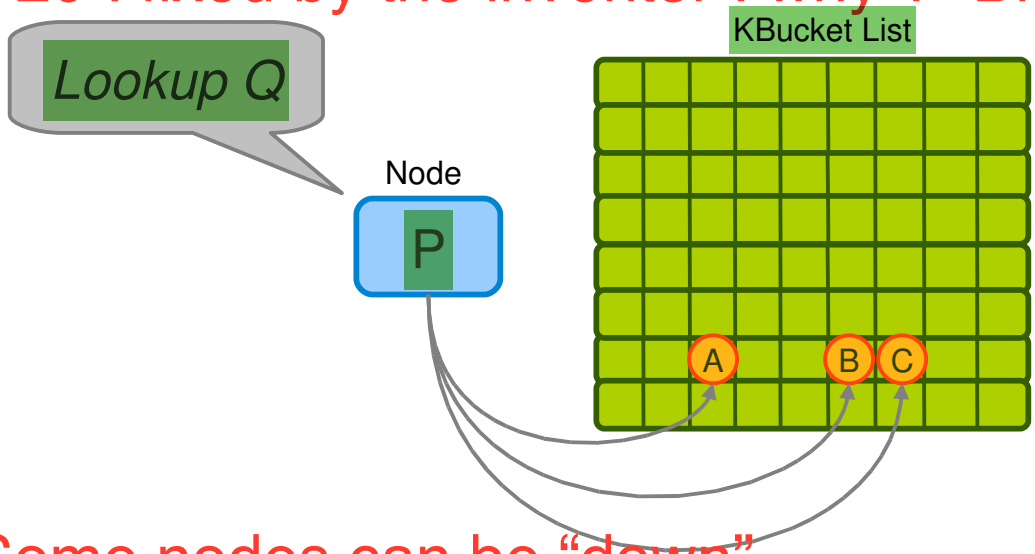
- Closest nodes in ID space

RPC P.FIND_NODE(Q) Core Idea - 3

k = length of every KBucket list entry

k = replication factor, k nodes memorise the value V of the key K

k = 20 : fixed by the inventor : why ? “Bravo les inventeurs”



P XOR Q in [64,128)

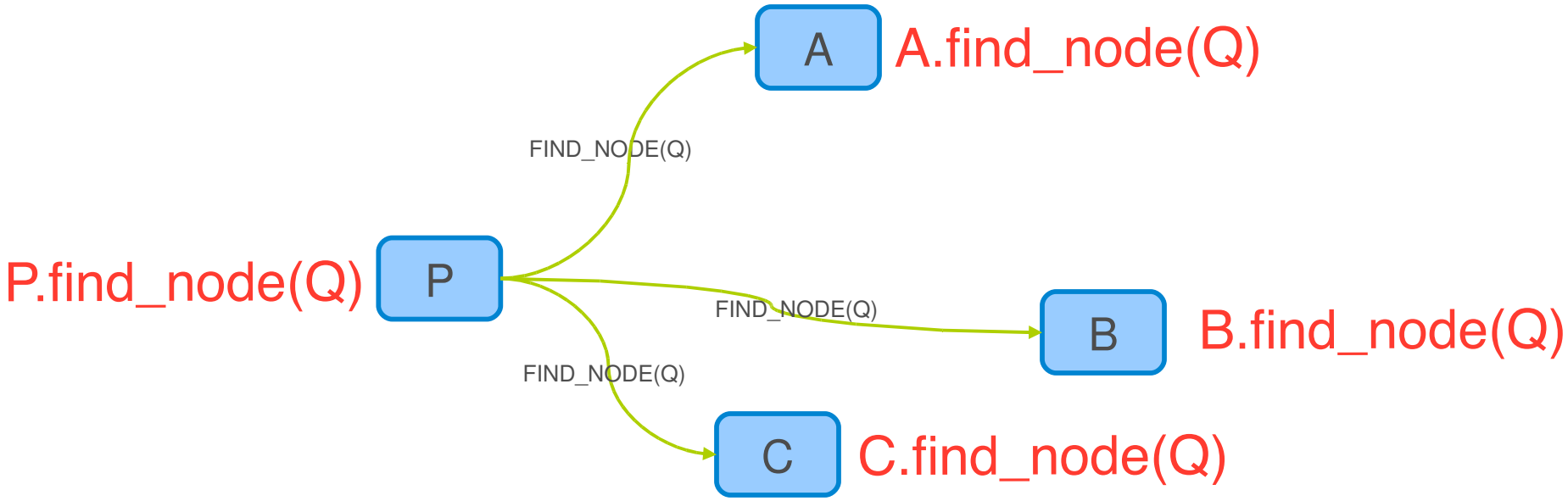
Some nodes can be “down”

Kademlia code is concurrent by design, because we launch the lookup on “alpha” nodes

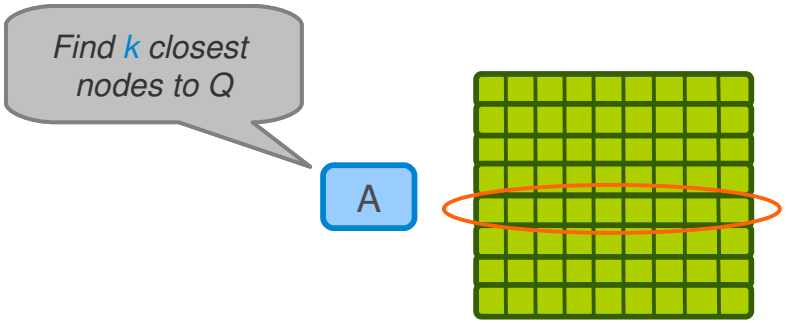
... and select α nodes from the appropriate kbucket

alpha = 3 : fixed by inventors : why ? “Bravo les inventeurs”

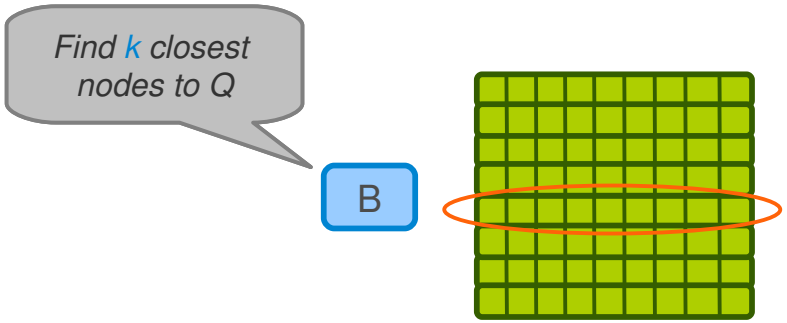
Core Idea - 4

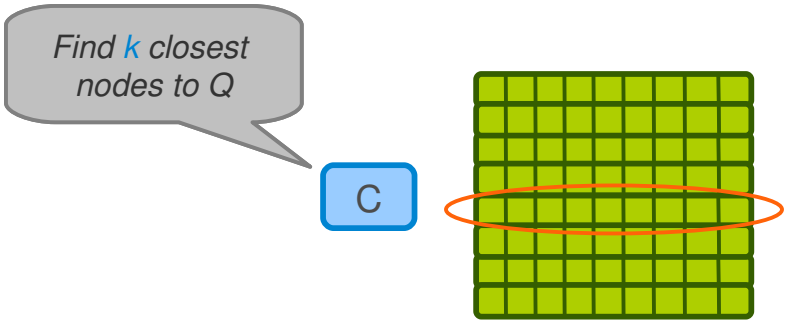

A
A.find_node(Q)
FIND_NODE(Q)
P.find_node(Q)
P
FIND_NODE(Q)
B
B.find_node(Q)
FIND_NODE(Q)
C
C.find_node(Q)

Core Idea - 5



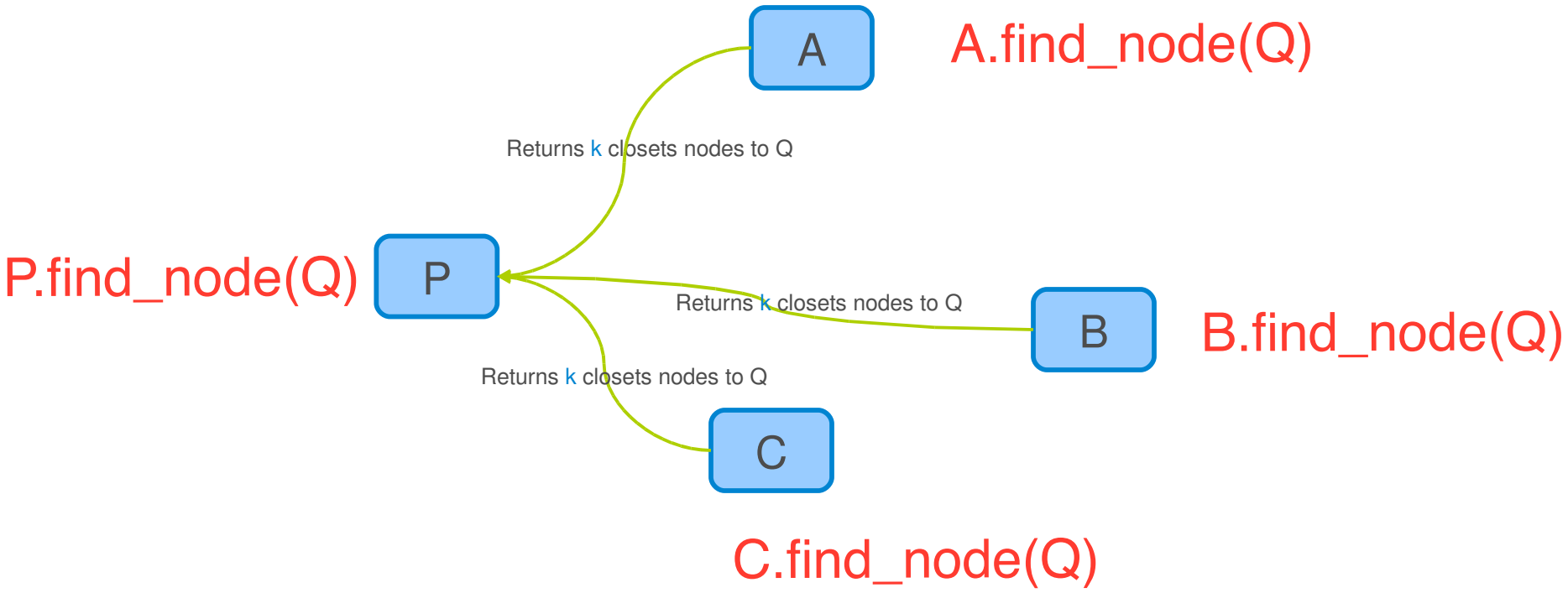
because alpha = 3
Kad run 3 times the procedure _.find_node(Q) on A,B, and C





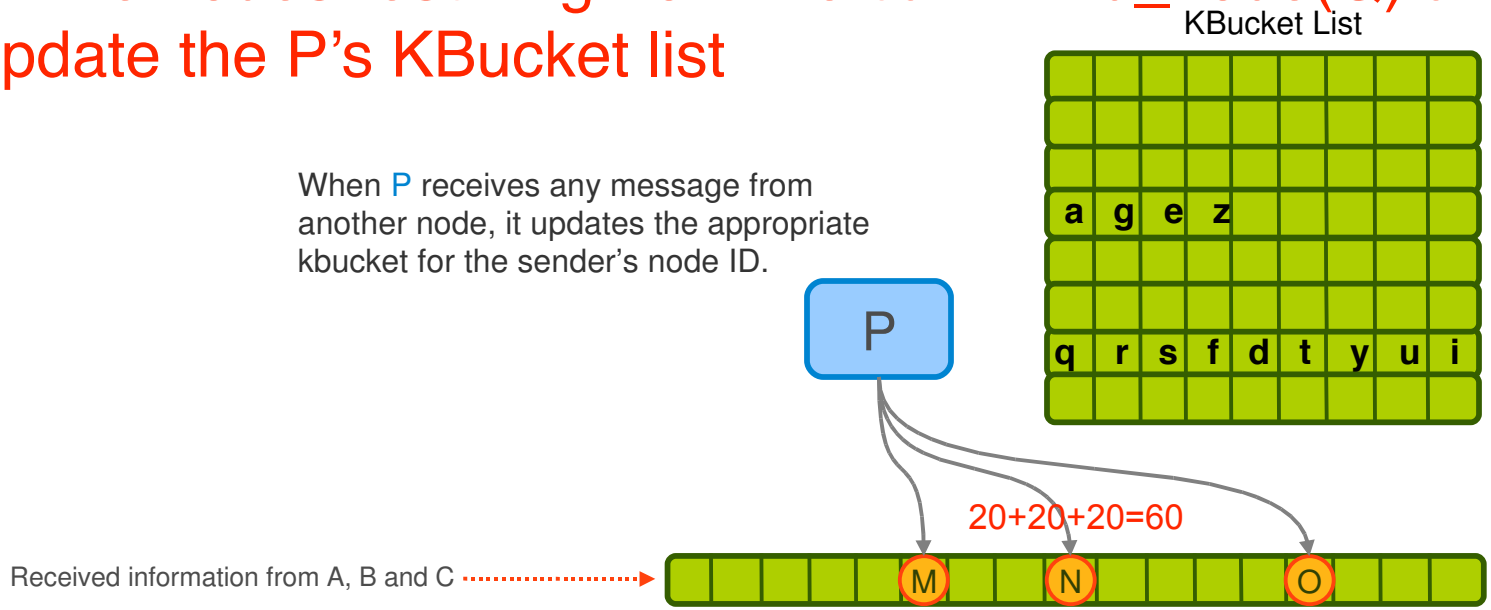
find_node( )

Core Idea - 6



Core Idea - 7

Innovative stabilisation w.r.t Chord: “Learning from lookup”:
All the nodes resulting from the call P.find_node(Q) are used to update the P’s KBucket list

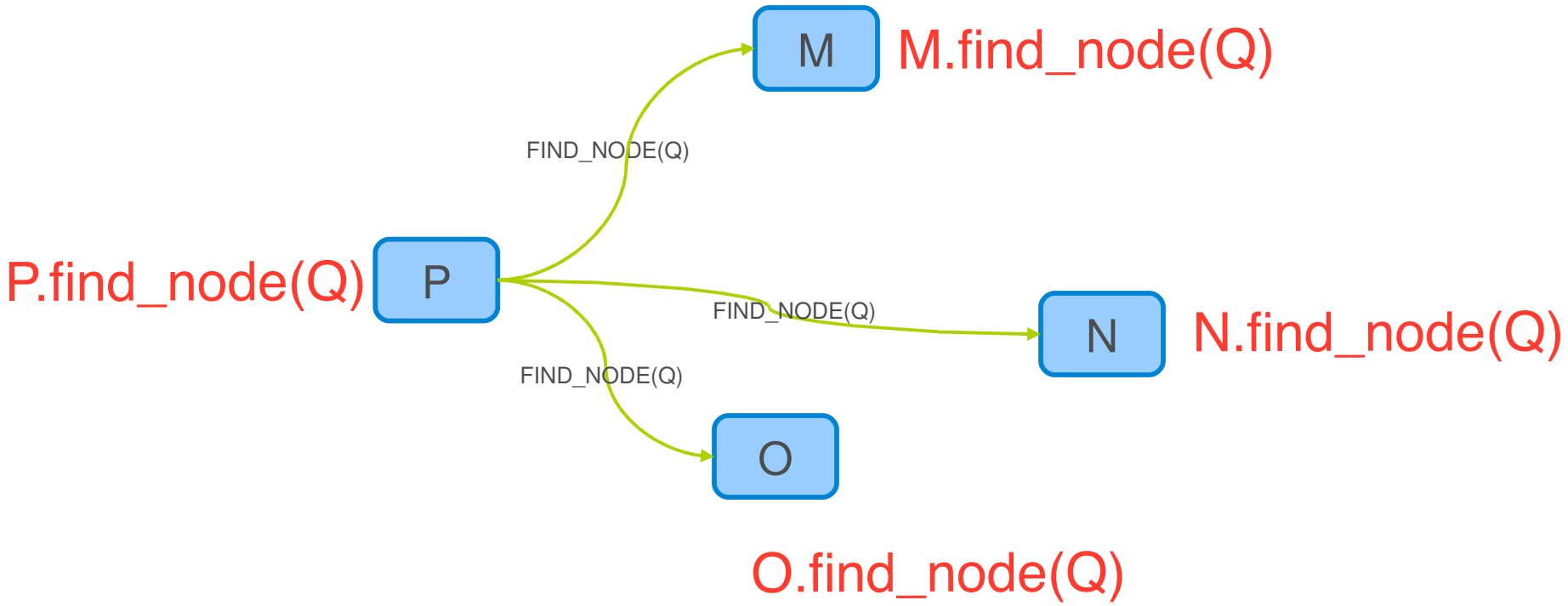


- P refresh its Kbucket list with new nodes and list the results of the alpha calls in a vector of length max 3 times k

... again select α nodes from the received information

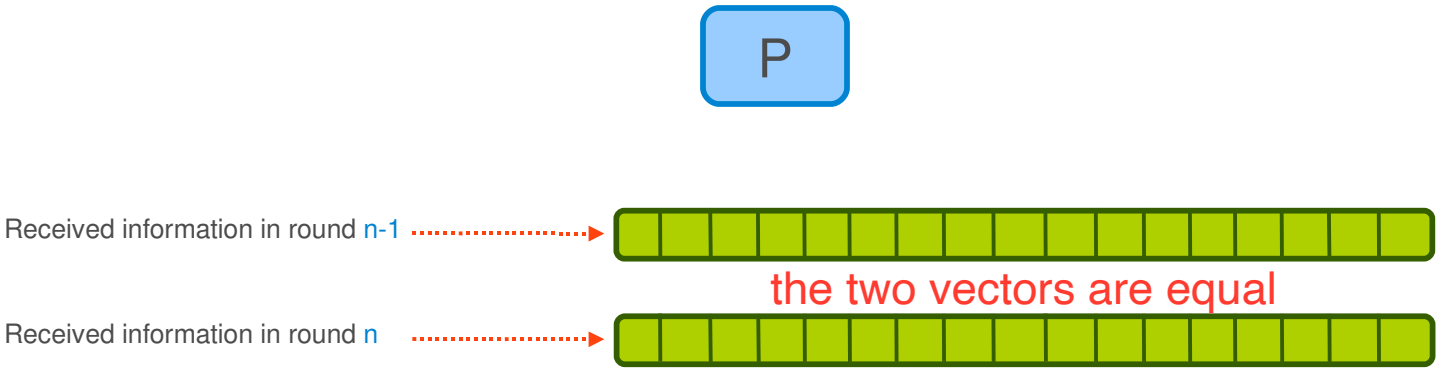
- P pick alpha (3) nodes from this vector and run again “alpha”
  {M,N,O}.find_node(Q) IN PARALLEL !!!!
- Question: when this process END? (aka what is the exit condition?)

Core Idea - 8


M
M.find_node(Q)
FIND_NODE(Q)
P.find_node(Q)
P
FIND_NODE(Q)
N
N.find_node(Q)
FIND_NODE(Q)
O
O.find_node(Q)

Core Idea - 9

Exit condition : when the “round” n-1 will be equal to round n!

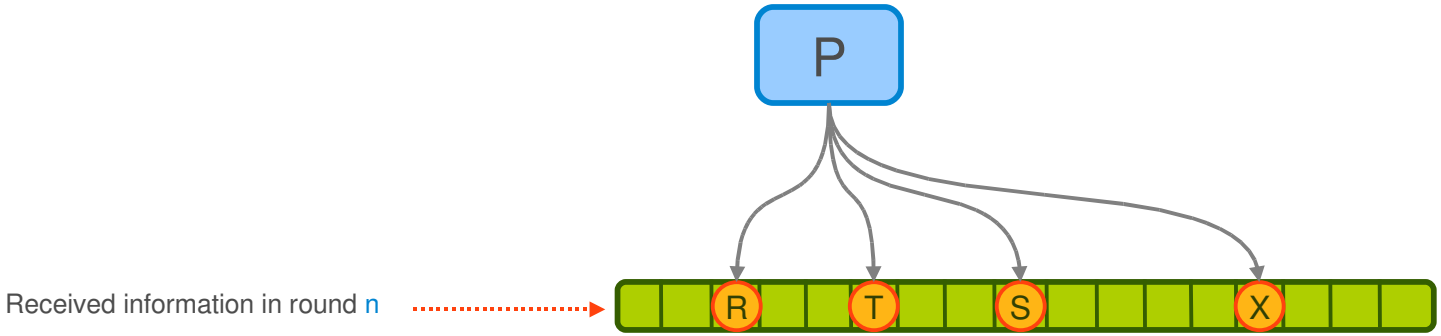


Repeats this procedure iteratively until received information in round n-1 and n are the same.

Core Idea - 10

… finally (finally) P run *for the last time *
R.find_node(Q) and T.find_node(Q) and S.find_node(Q) and X.find_node(Q) because that nodes *were not yet solicited*

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



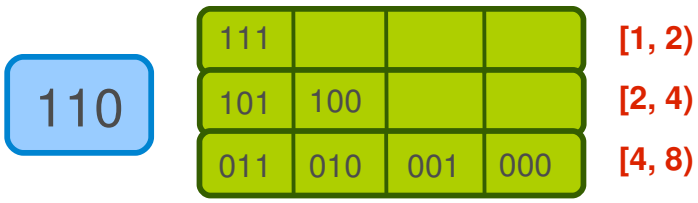
and finally P.find_node(Q) end with output
Q if that node if found in KAD OR
a list of the the closest “k” nodes to Q, otherwise.

END OF KAD PROTOCOL

Let's Look Inside of Kademlia

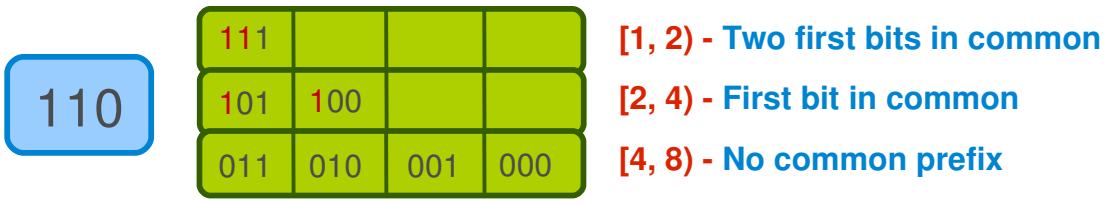
Node State

- 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

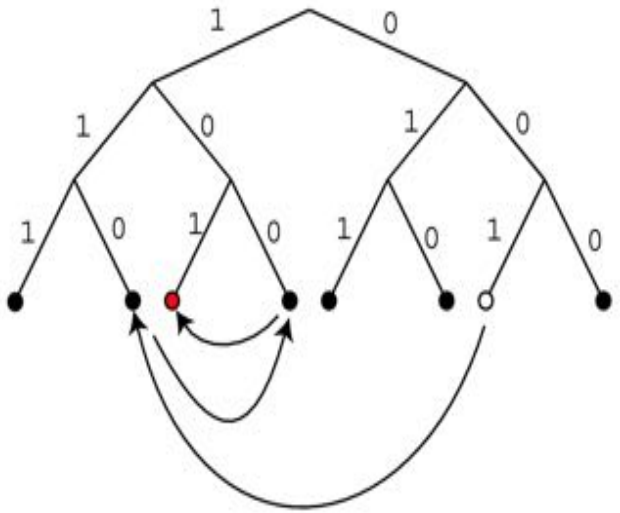
- The <key, value> data is stored in k closest nodes to the key.

In real Kademlia implementations,
every <key,value> has a replication factor of k = 20

???? W H Y *K=20* HAS BEEN CHOOSEN ????

from the paper:
“ k=20 is chosen such that
any given k nodes are
very unlikely to fail
within an hour of each other ”
VERY KOOL INTUITION !!!!!

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)

001.find_node(101)
001 XOR 101 = 4 -> select 110
110.find_node(101)
110 XOR 101 = 3 -> select 100
100.find_node(101)
100 XOR 101 = 1 -> return 101! bingo

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.

Is very "selective" to remain in a bbucket :-)
only ACTIVE nodes could stay, otherwise they can be "kicked off"

Maintaining Kbucket List (Routing Table)

- Buckets will generally be kept constantly fresh, due to traffic of requests travelling through nodes.

MORE TRAFFIC MORE FUN !!

- When there is no traffic: each peer picks a random ID in kbucket's range and performs a node search for that ID.

PROPERTIES

- Easy table maintenance. Tables are updated when lookups are performed, due to the XOR symmetry a node receive lookups from the nodes that are in its own table

- Fast lookup because of alpha-parallel searches (at the expense of increased traffic)

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.

P.join(Q)

« Node P wants to join KADEMLIA via node Q »

```
begin
    P XOR Q = n;
    Insert Q in the n in [2^i,s^(i+1)) kbucket list of P;
    Run a P.find_node(P);        ***yes P find itself***
end
```

Leave And Failure

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

PING simply call FIND_NODE on a node in its bucket table

P.PING(Q) = P.FIND_NODE(Q).

no burocracy in case of leave and failure !!

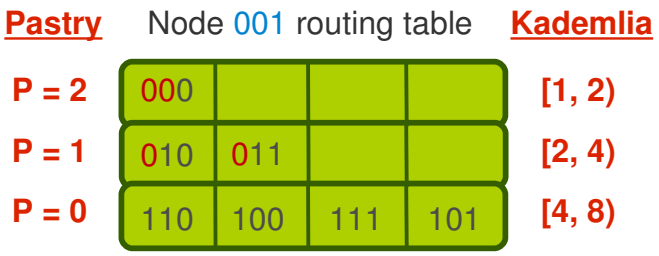
Kademlia and other DHTs

Kademlia vs. Chord

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

Kademlia vs. Pastry

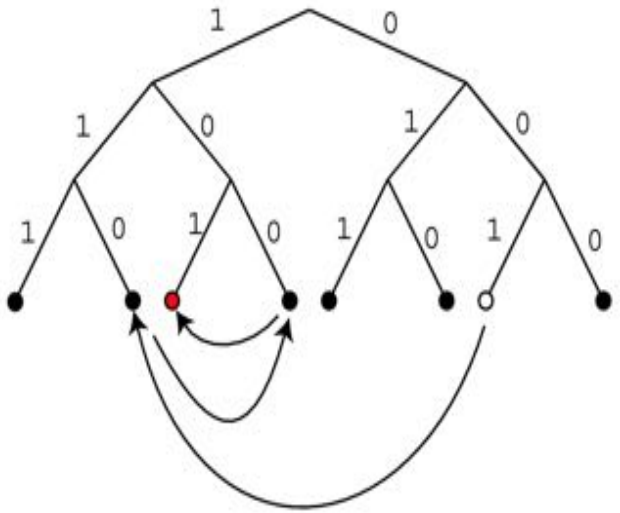
- like Pastry
  - The same routing table.



- Unlike Pastry
  - $\alpha = 3$ by default in Kademlia, while $\alpha = 1$ in Pastry.

DONE!

A Page To Remember



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

- [1] Maymounkov, P. and Mazières, D. 2002. Kademlia: *"A Peer-to-Peer Information System Based on the XOR Metric"*. In Revised Papers From the First international Workshop on Peer-To-Peer Systems (March 07 - 08, 2002). P. Druschel, M. F. Kaashoek, and A. I. Rowstron, Eds. Lecture Notes In Computer Science, vol. 2429. Springer-Verlag, London, 53-65.

Question?