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

Petar Maymounkov and David Mazières
New York University
`http://kademlia.scs.cs.nyu.edu/`

Setting

Key/value pairs storage and retrieval

- Keys are unique
  $\Longrightarrow$ w.l.o.g. keys are uniformly distributed (160-bit) numbers (e.g. use hashing)
- Keys can have different store and / or retrieve popularity

DHT (Distributed Hash Table)

Constraints

- Any particular node can disappear at any time
- Nodes should be loaded equally (bandwidth and storage)

Goal

- Quick storage and retrieval, independent from node failures
- Minimize number of control messages

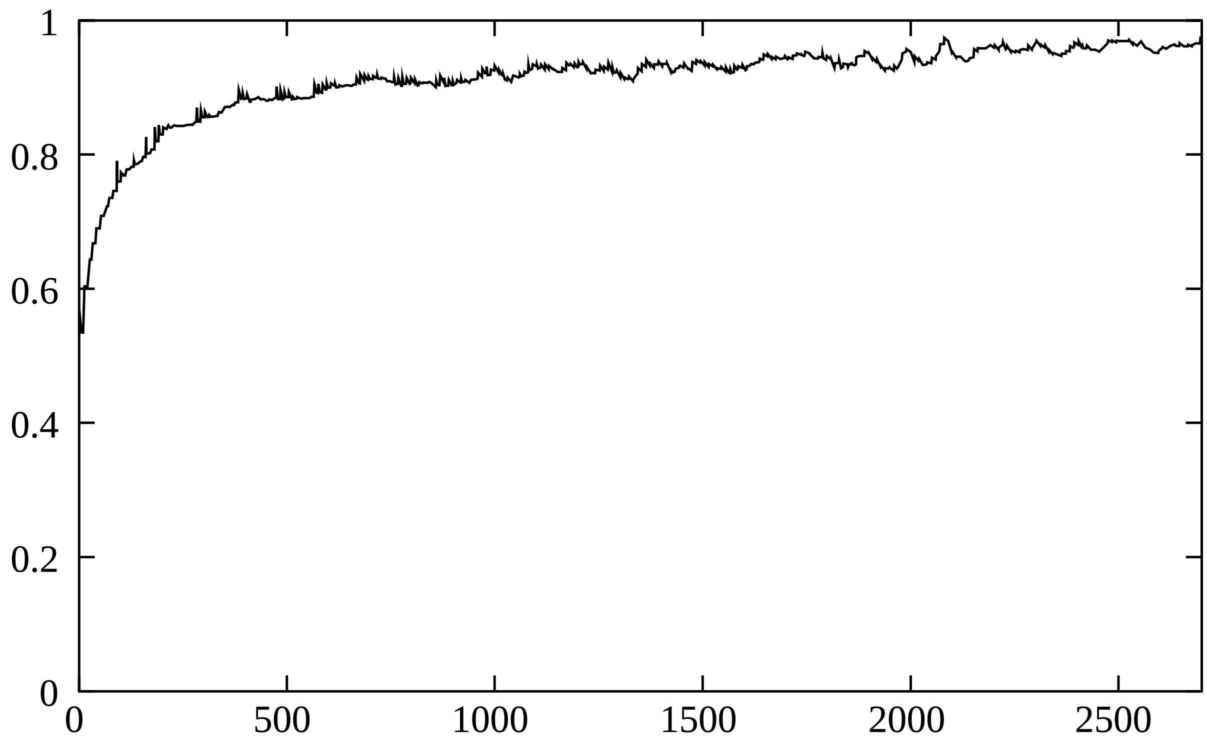
Node instability

Ideal case

- Once a node joins, it never leaves.

Realistic case

- A randomly selected online node will stay online for another 1 hour with probability 1/2.



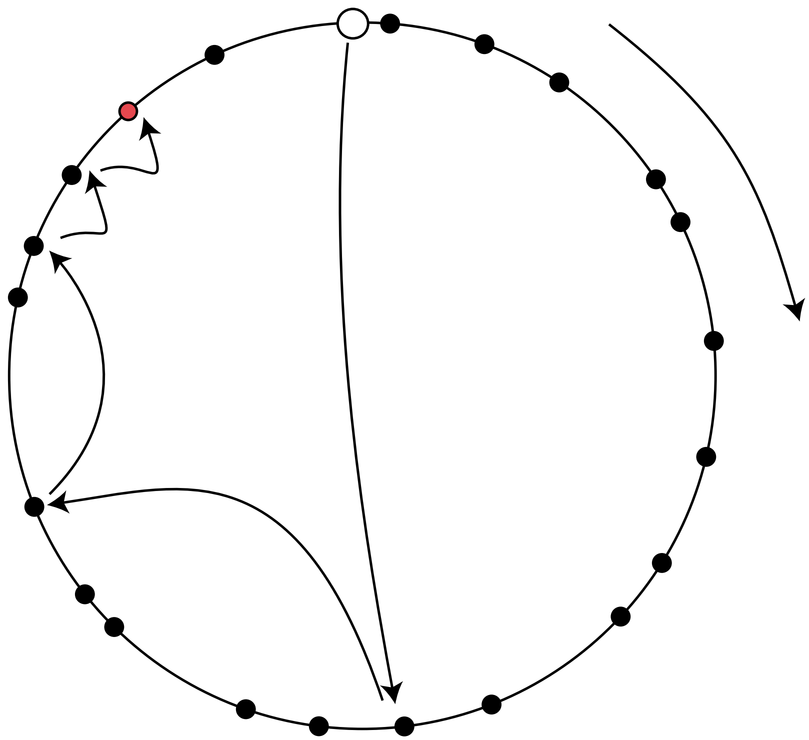
- Probability of remaining online another hour as a function of uptime. The x axis represents minutes. The y axis shows the the fraction of nodes that stayed online at least x minutes that also stayed online at least $x + 60$ minutes.

Common approach

1. Assign random (160-bit) ID to each node
2. Define a metric topology on the 160-bit numbers, i.e. the space of keys and node IDs
3. Each node keeps contact information to $O(\log n)$ other nodes
4. Provide a lookup algorithm, which finds the node, whose ID is closest to a given key.

   $\Longrightarrow$ we need a metric that identifies closest node **uniquely**
5. Store and retrieve a key / value pair at the node whose ID is closest to the key

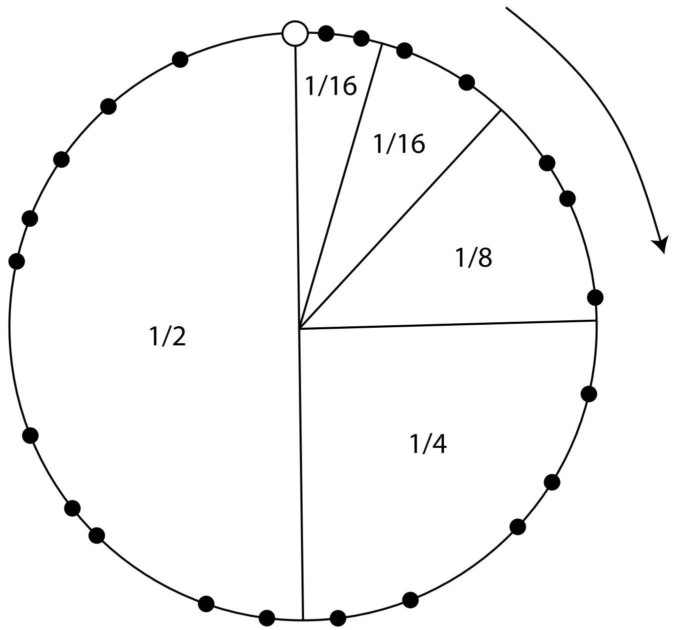
Chord lookup



Each step halves the topological distance to the target.

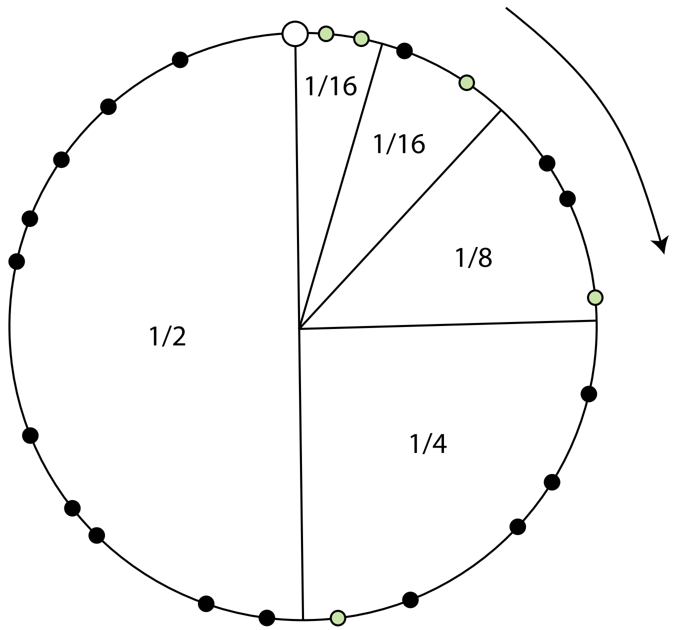
So we have expected $\log n$ hops to the target.

Chord routing table basics



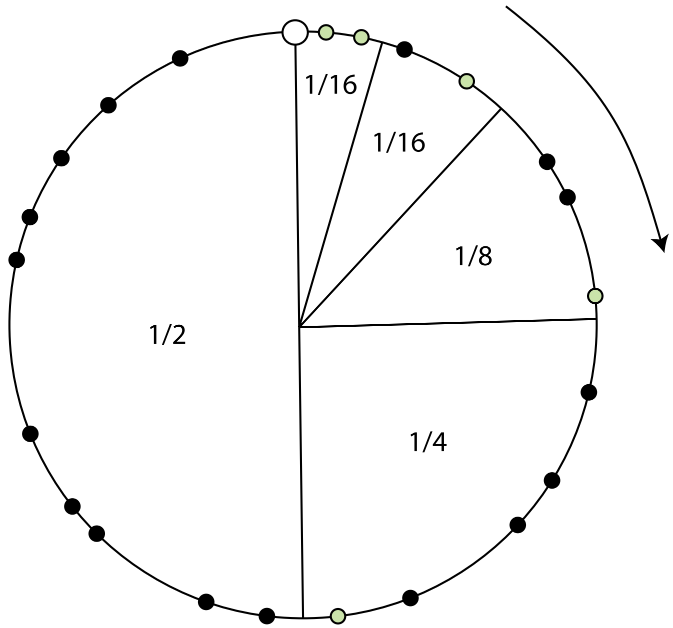
- Contacts in logarithmically distributed regions of the ID space

Chord routing table rigidity



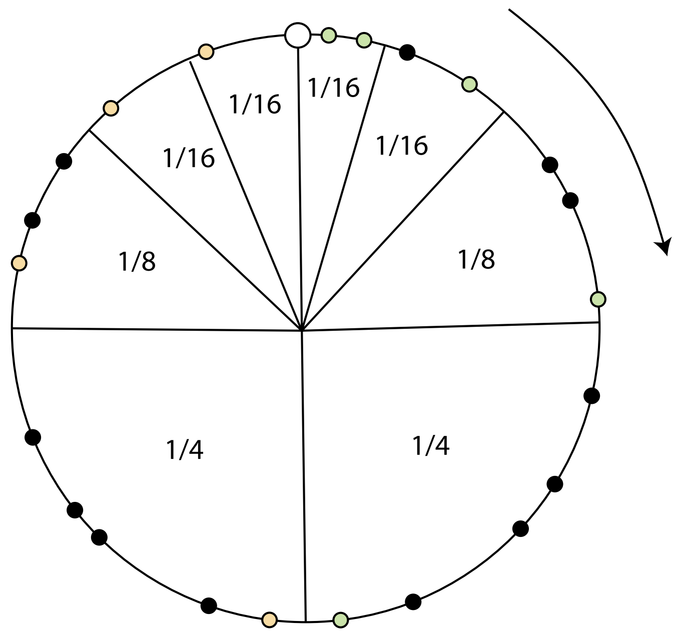
- Rigidity
  - Complicates recovery from failed nodes and routing table
  - Precludes proximity-based routing

Chord discrepancy



- In- and out- distribution are exactly opposite
  - Prevents from using incoming traffic to re-enforce routing table

Fixing Chord has drawbacks

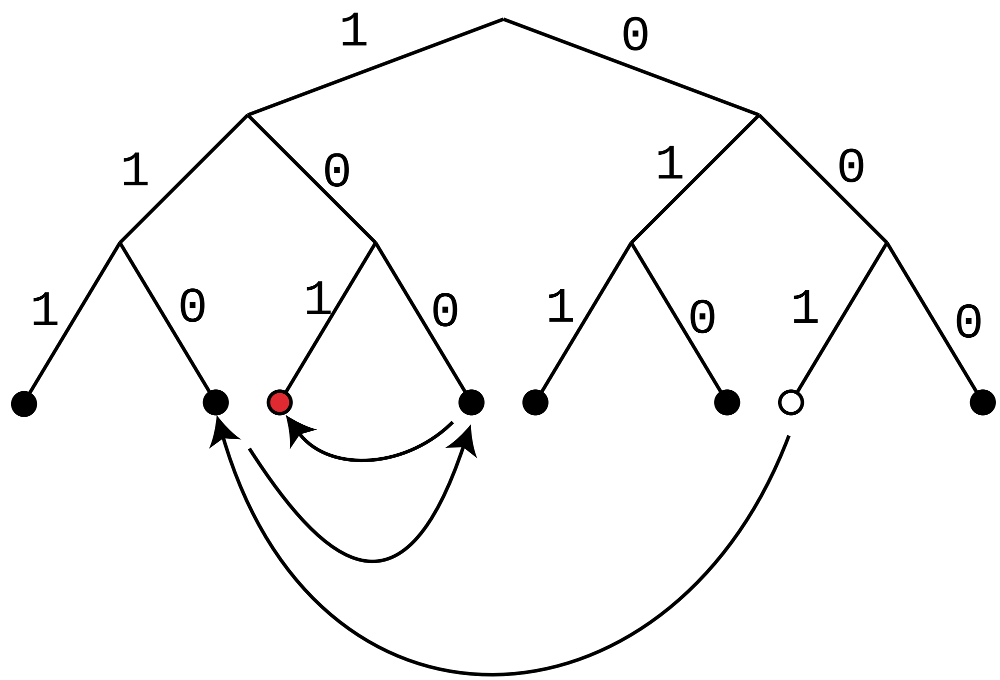


- Bi-directional routing table has drawbacks
  - Doubles routing table size
  - Doubles number of control messages

Kademlia: a peer-to-peer system

- Flexible routing table
  - Allows to benefit from proximity-based routing
  - So relaxed, that maintenance is minimal
- In- and out- distributions are the same
  - Network re-enforces itself
- Just $\log n$ contacts (not counting redundancy)

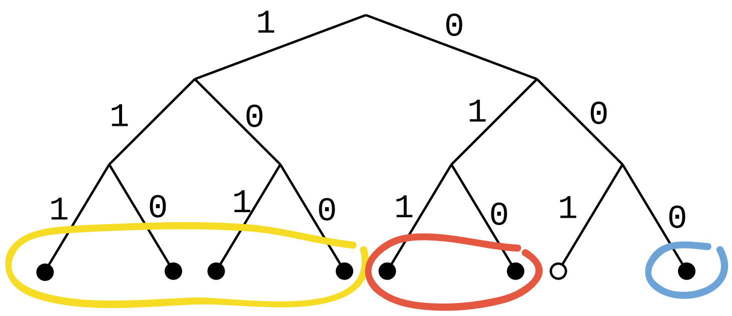
Overarching idea



Every hop brings us in a smaller subtree around the target.

Can forward requests to any node in the appropriate subtree.

Idea: routing table



- No more rigidity: can have any contact in a subtree
- In- and out- distributions are the same
- Routing table size is still $\log n$
- Why do we need a topology?

The XOR topology

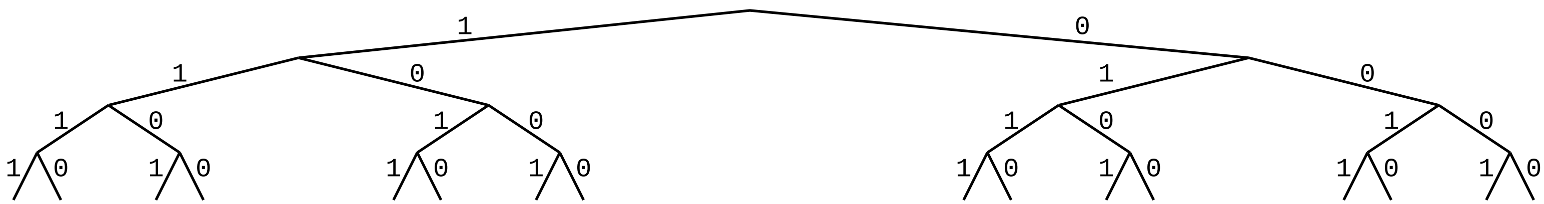
- **Definition:** $d(X, Y) = X \oplus Y$
- **Intuition:** Differences at higher order bits matter much more than differences at lower order bits.

  $\underline{\mathbf{0}}10\underline{\mathbf{1}}01$

  $\underline{\mathbf{1}}10\underline{\mathbf{0}}01$, distance is $4 + 32 = 36$

- **Geometric intuition:** Nodes in the same tree are much closer together than they are with nodes in other subtrees.

Complete XOR tree of 5-bit numbers



Points in the same subtree are much closer together than they are with points in other subtrees.

Data Structures

Contact

- A pair of node ID and IP:UDP_port

k-bucket

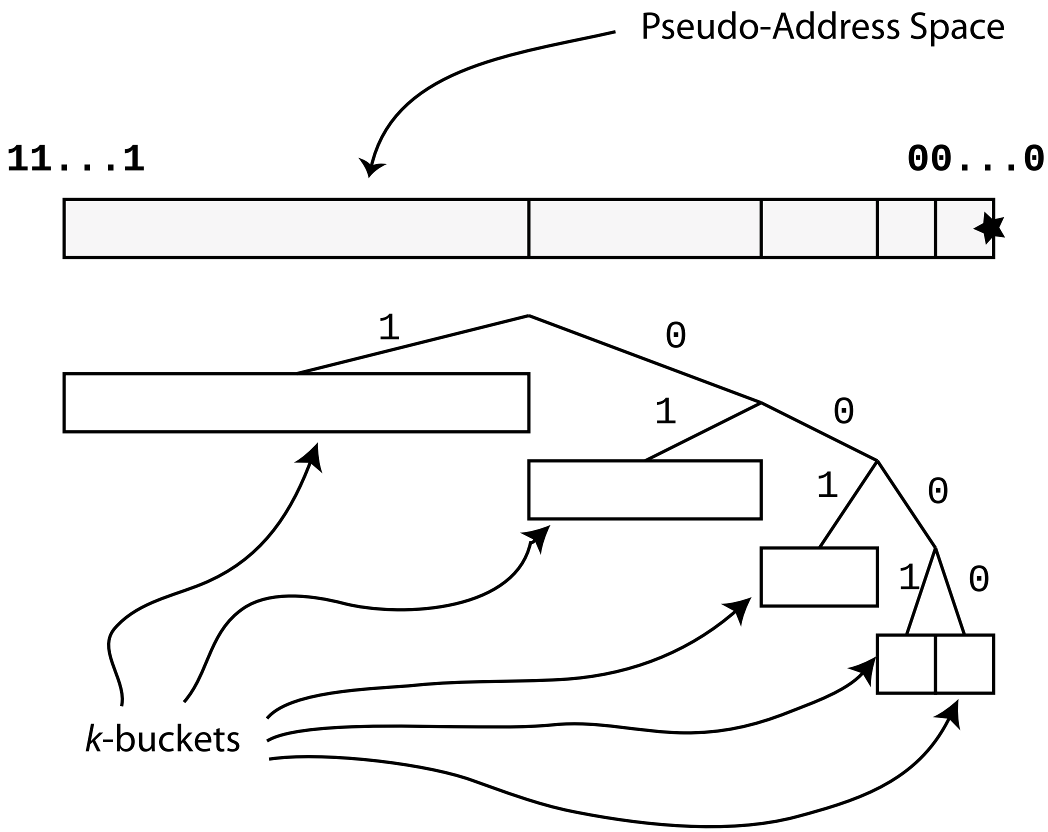
- A container for no more than k contacts (we use $k = 20$)
- Operations place contact and remove contact

Routing table

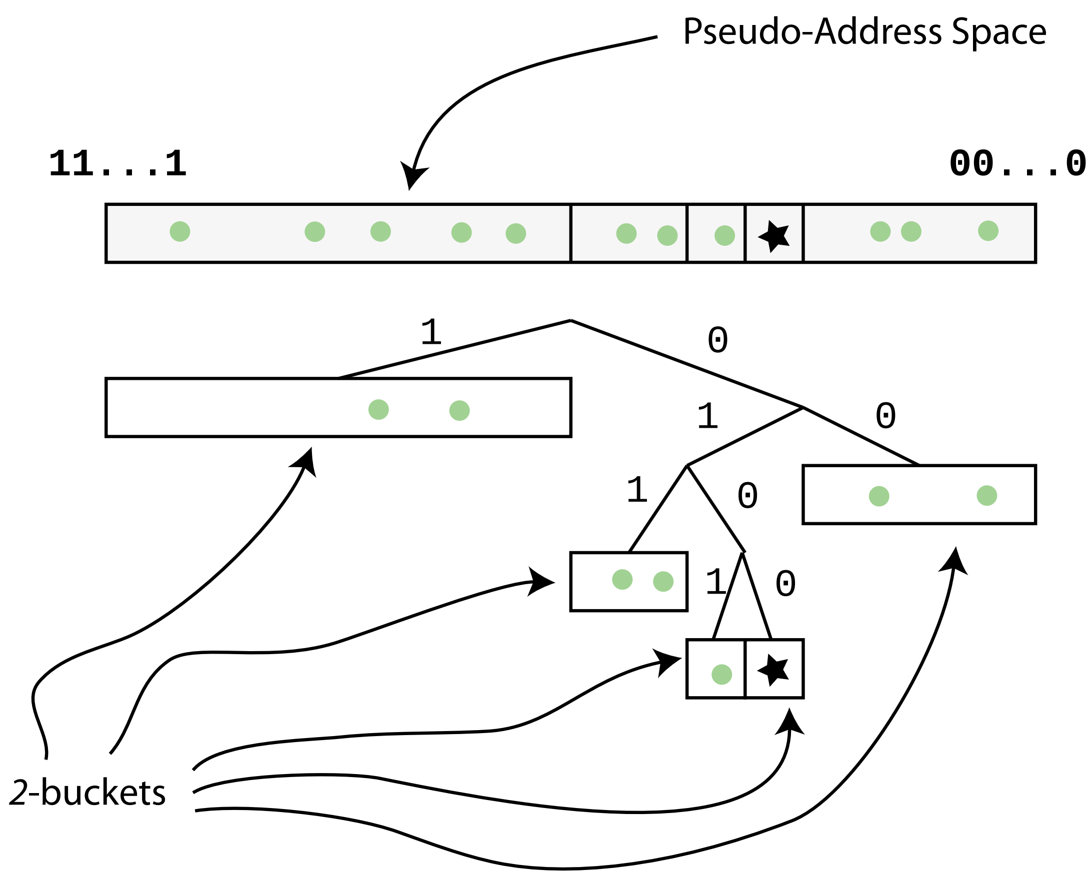
- Operations place contact and remove contact
- A constrained tree of k-buckets
- Each bucket responsible for a range of the node ID space

Routing Table Data Structure

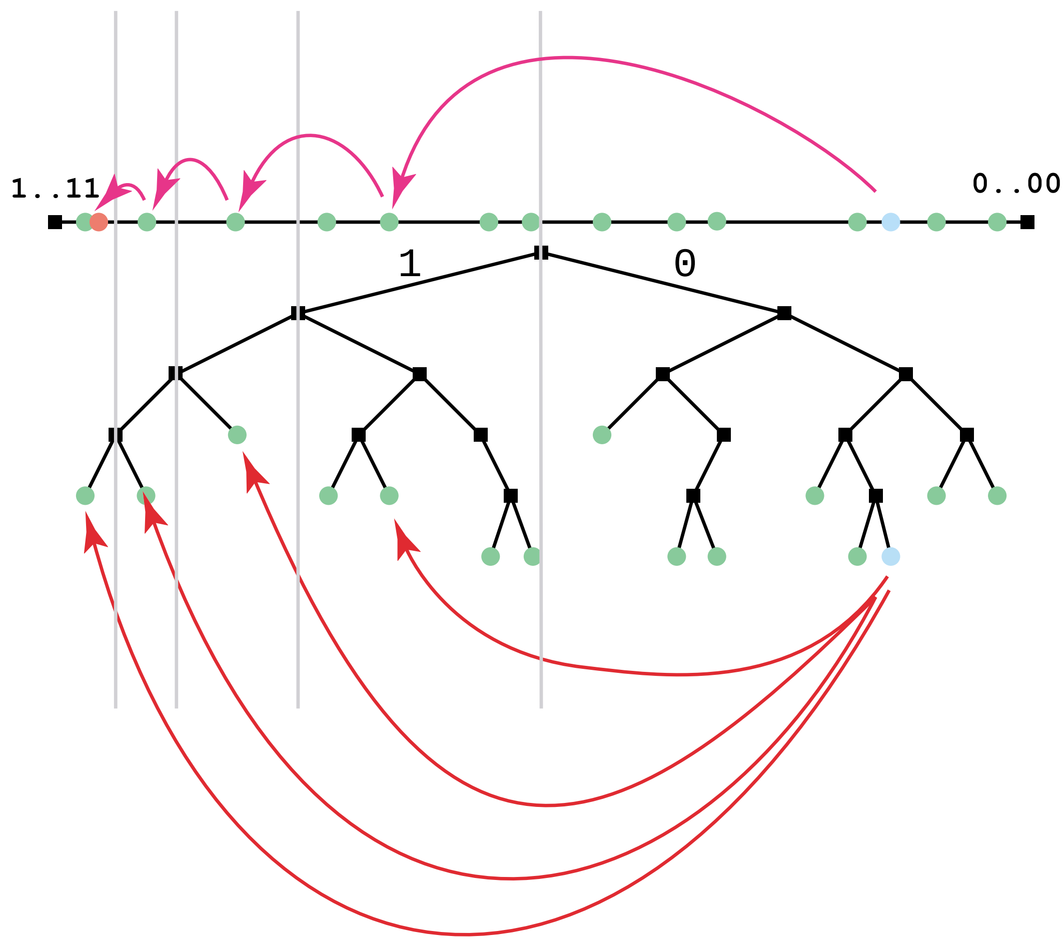
(for node, whose pseudo-address is 00...0)



Routing Table Data Structure



Simple lookup



Lookup algorithm skeleton

- **Goal:** Find the k nodes closest to a given target $T \in \{0,1\}^{160}$
- **RPC:** $\textsf{find\_node}_n(T)$ returns all contacts from the (first non-empty) k-bucket in n's routing table that is closest to T
- **Lookup:**

$$
\begin{aligned}
n_o &= \text{ourselves (the node that is performing the lookup)}\\
N_1 &= \textsf{find\_node}_{n_o}(T)\\
N_2 &= \textsf{find\_node}_{n_1}(T)\\
\cdots\\
N_l &= \textsf{find\_node}_{n_{l-1}}(T),
\end{aligned}
$$

  this completes when N_l contains no contacts that haven't been called already

- n_i is any contact in N_i

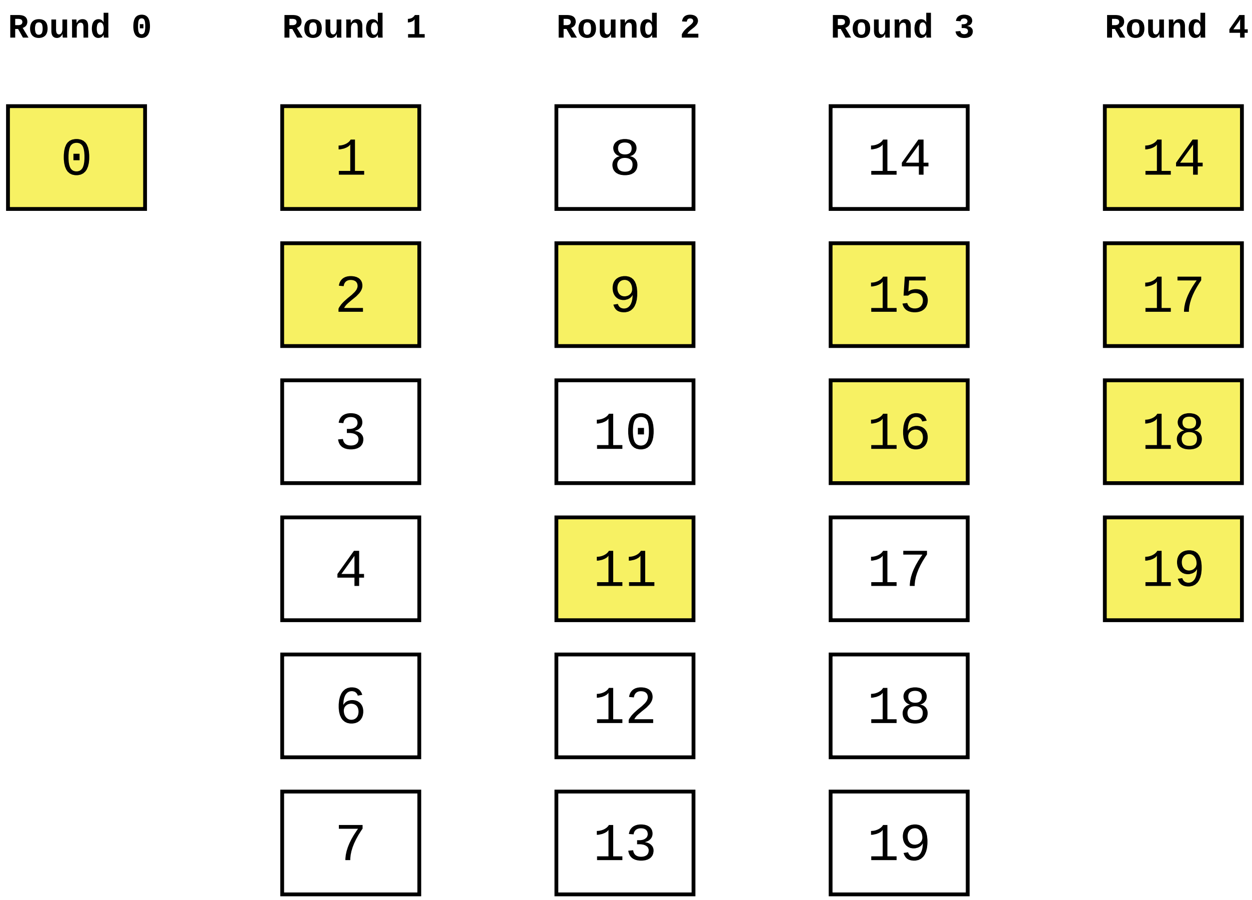
How lookup works?

- On every step, the metric distance between n_i and the target reduces by an exact factor of 1/2.
  $\Longrightarrow$ (abstractly) every step reduces the pool of candidates by an expected factor of 1/2.
- Consequent calls to $\mathsf{find\_node}_n(T)$ fetch the result from ever smaller-range k-buckets.

Concurrent lookup

- : Trade bandwidth for lower latency lookups
- **Goals:**
  - Route through closer/faster machines
  - Avoid delays due to timeouts on offline contacts
- **Idea:** Perform $\alpha > 1$ calls to $\textsf{find\_node}_n(T)$ in parallel.

Asynchronous Lookup



Why lookup works?

Routing table invariant

- The routing table always contains the k closest to ourselves nodes
- A k-bucket is only empty if there are no nodes in its range

Contact accounting

- Whenever we use a contact that doesn't respond within a given timeout, we remove it from the routing table
- **As a general rule:** every node places a contact to each node that makes an RPC call to it in its routing table
- Due to XOR topology's **symmetry**, the distribution of nodes that call us is going to be the same as the distribution of contacts that we need for our routing table
- **Formally:** the probability of being contacted by someone at a distance $l \in [2^i, 2^{i+1}]$, $i \geqq 0$, from us is a constant, independent of i

Joining, Leaving and Refreshes

Node join:

- Borrow some contacts from an already online node
- Lookup self
- Cost of join is $O(\log n)$ messages

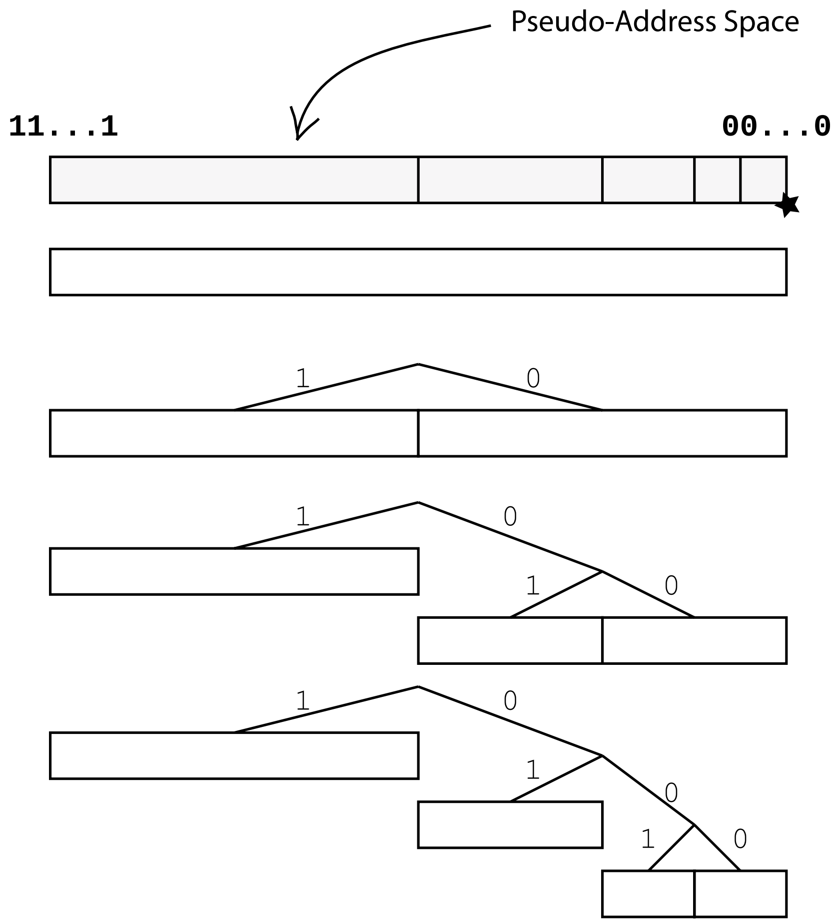
Node leave: no action

- Very useful for modem connections that may disconnect multiple times during a long online session

Hourly k-bucket refreshes (only if necessary)

Routing Table Evolution

(for node, whose pseudo-address is 00...0)



Key-Value Pairs

- **Invariant:** Be able to find the key-value pairs on one or more of the k nodes closest to the key
- Publishing and searching is like a lookup

Key/Value Invariant

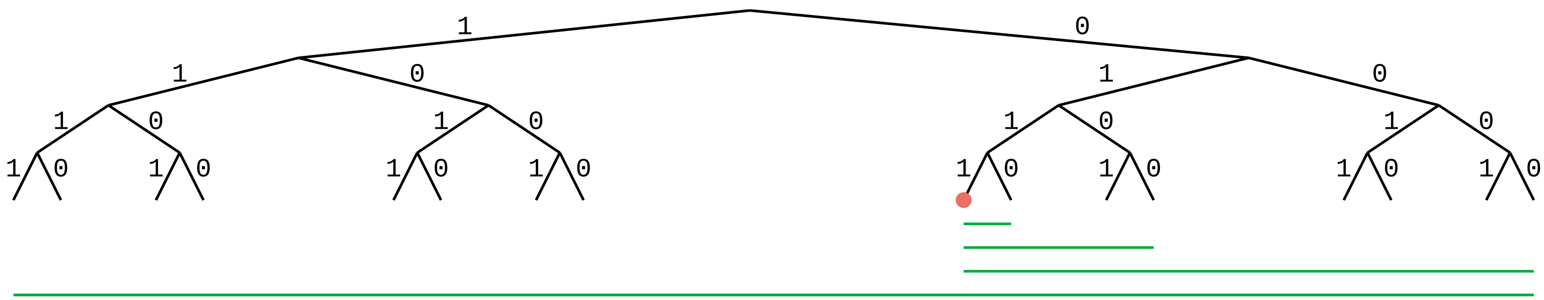
- Joining nodes are immediately noticed by their closest neighbours, and the appropriate key / value pairs are replicated to them.
- Re-enforce invariant every hour
- Expected Retainment Time (of a key / value pair) is 2^k hours.

Topological caching

Search caching

- When a key starts getting popular, replicate it to more nodes around its location.
- When searching for a key, stop the lookup as soon as we get a result.

Caching principles



Overpopular nodes

- Nodes tend to be seen only by nearby nodes
- Hard-limit on requests prevents over-popularity
- Flip-side: Natural separation between very-long-staying nodes and short comers.

Conclusions

Novel topology:

- **Symmetry:** If $d(X, Y) = d(Y, X)$. Helps reduce control messages.
- **Uniqueness:** For every $X \in \{0, 1\}^{160}$ and $l \in \mathbb{N}$ there is unique $Y \in \{0, 1\}^{160}$, such that $d(X, Y) = l$. Identify key location uniquely.
- **Unidirectionality:** For a fixed X there are 2^i Y's for which $d(X, Y) \leqq 2^{i-1}$. Makes caching efficient.

Asynchronous lookup: avoids slow links

Further directions

- Non-unique keys.
- Node heterogenity (nodes of different strengths)
- Network heterogenity (take advantage of fast intranets)
- Security models against node, key and lookup attacks.