#ASLI ENGINEERING

Kademlia - a P2P Distributed Hash Table

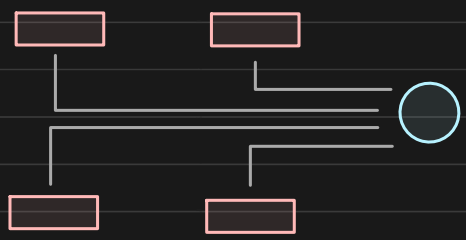
SWIPE

BY

ARPIT BHAYANI

Kademlia - a pure P2P Distributed Hash Table

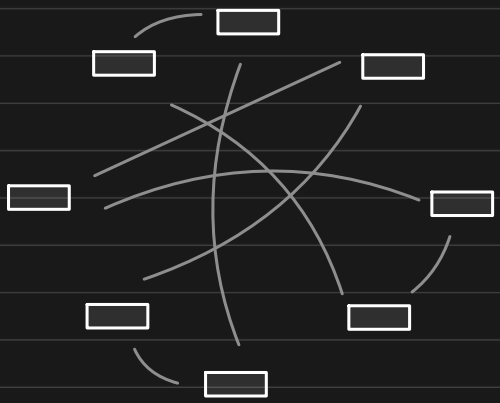
To get information about peers, a node in the BitTorrent network talks to Tracker.



Having a central entity is still prone to attacks and failures

So, can we do a pure p2p network, without Tracker?

Kademlia



Say, we have a gigantic set of KV pairs that one node cannot store or handle.

Hence we have to distribute

Hence it is called a Distributed Hash Table

1. How do we distribute?
2. How would a node know how to find a KV?
3. How to gracefully handle nodes joining/leaving?
4. How to do this without a central entity?

Representation

Every node (machine) participating gets a unique 160b (20B) ID.

The unique ID can be

- explicitly assigned
- implicitly derived ← for P2P

The data that is stored across the network is also hashed and identified by 160b ID

↳ KV pair

* This is a generic DHT, nothing specific to BitTorrent

In the context of BitTorrent, the only thing that changes is the kind of information (reachable peers) stored on the node.

Ownership

key $k_1 \rightarrow h_{k1}$ Node $N_1 \rightarrow h_{N1}$

The node that is closest to the key, owns the key. * not a ring

$k_i \in N_j \mid d(h_{k_i}, h_{N_j})$ is minimum for $\forall j$

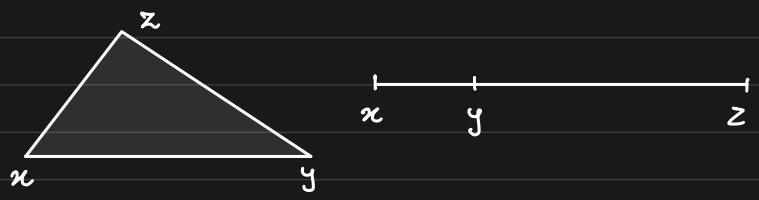
Distance metric

In order to find the "closest" node we need a distance metric that quantifies the closeness

For any non-euclidean geometry

Requirement from a distance metric

1. $d(x,x) = 0 \quad \forall x$ ——— distance to self = 0
2. $d(x,y) > 0$ if $x \neq y \quad \forall x,y$ ——— distance to other is +ve
3. $d(x,y) + d(y,z) \geq d(x,z)$ ——— triangle inequality



Shortest distance b/w the two points is a straight line connecting them

For two nodes / keys in our Kademlia distribution, the distance metric is

$d(x,y) = x \oplus y$

↓

Bitwise XOR of 160 bit IDs, interpreted as integer

Satisfies all the 3 requirements

1. $d(x,x) = x \oplus x = 0$
2. $d(x,y) = x \oplus y > 0$
3. $d(x,y) + d(y,z)$
   $= (x \oplus y) \oplus (y \oplus z)$
   $= x \oplus z = d(x,z)$

Visualizing Distance

For simplification, say we work with 4-bit ID i.e. nodes and key are given 4-bit IDs

Node / key	ID	Bit Representation
N_1	15	1111
K_A	6	0110
N_2	5	0101
K_B	13	1101

$$d(K_A, N_1) = 0110 \oplus 1111 = 1001 = 9$$

$$d(K_A, N_2) = 0110 \oplus 0101 = 0011 = 3$$

Hence, key K_A is owned by node N_2

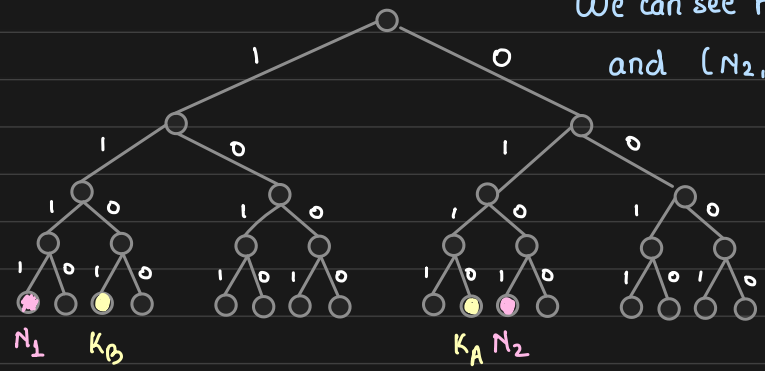
A pattern we see is that the

> Common the prefix, smaller the distance

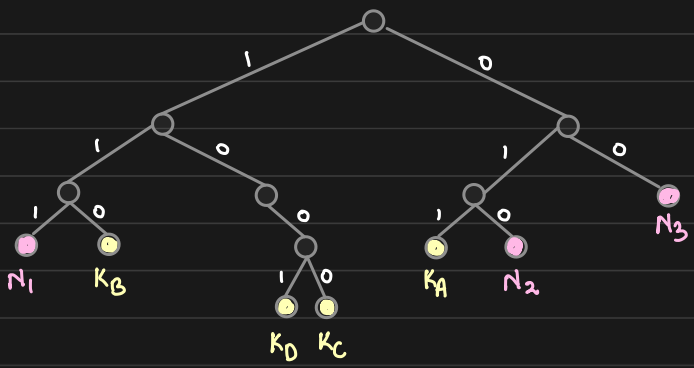
↗ the bits that are same would XOR to 0

So, IDs sharing the same prefix are "closer".

Hence, we visualize this as a "trie"



We can see how (N_1, K_B) are "close" and (N_2, K_A) are "close".



Instead of creating a complete binary tree, we create the paths as needed.

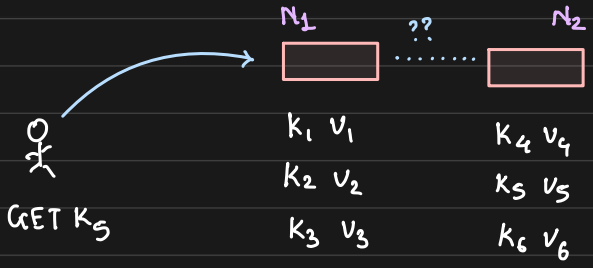
Instead of creating complete path we carve it till it minimally disambiguates

N_1	15	1111			
K_A	6	0110	K_C	8	1000
N_2	5	0101	K_D	9	1001
K_B	13	1101	N_3	1	0001

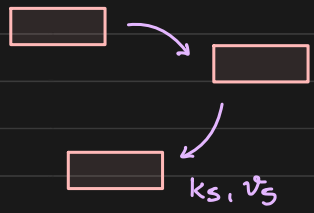
Routing

Given that there is no central entity to hold the addresses of all the nodes

How would one node access the KV on the other



Every node in the network would need to keep track of a few nodes, and hope they keep track of others, and so on.



Eventually we would have covered the entire network

Peer nodes that each node keep track of cannot be Random.

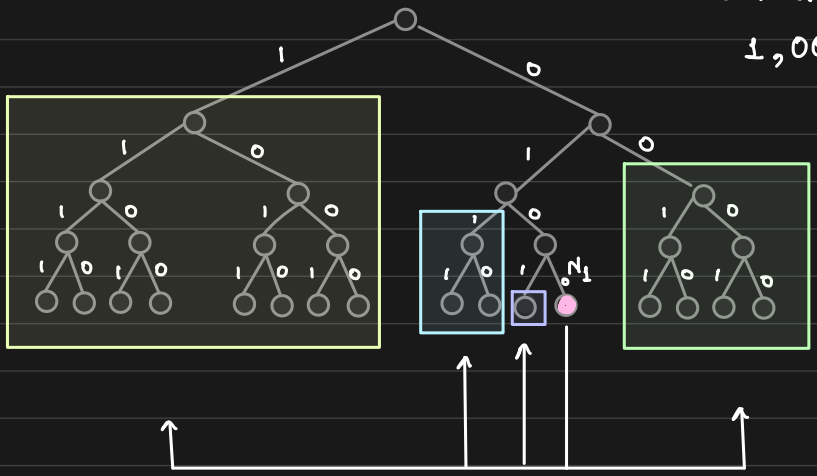
↳ as we need guaranteed convergence quickly.

So, what should be our routing strategy, that ensures

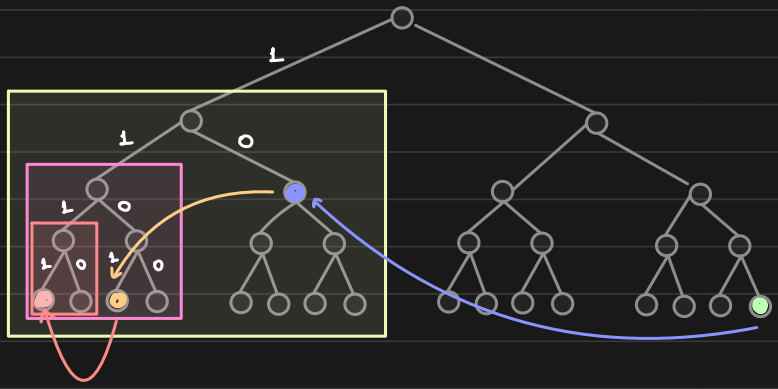
Core Idea: Every node knows at least one node in each subtree that it is not part of.

Routing table of node N1 should have contact in the 4 subtrees

1, 00, 011, 0101



If every node in the network keeps track of at least one node in each subtree, we would converge to the desired node in $\log n$



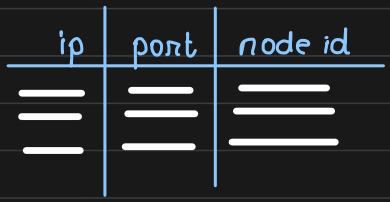
Say, N_1 (0000) wants to reach N_2 (1111) that it does not have a direct connection with. So, it will leverage intermediate nodes in the routing table

$N_1(0000) \rightarrow N_A(1000) \rightarrow N_B(1101) \rightarrow N_2(1111)$

would have one entry for subtree 1···	would have one entry for subtree 11···	would have one entry for subtree 111···

Thus, we see how XOR based distance metric ensure we would always converge (without ever digressing)

Thus each node only has to keep track of a small subset of nodes and the routing takes care of converging to the target node.



Communication happens over UDP and routing table holds

node id → < ip , udp port >

As the routing converges when every node has a few contact in every subtree that it is not part of,

the problem statement reduces to making fault tolerant

k-buckets

Every node, for each subtree, holds k entries

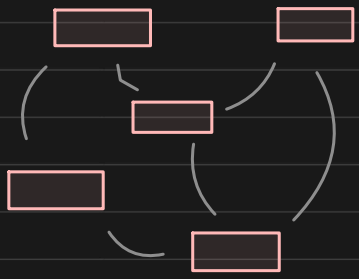
Each k-bucket is sorted by time last seen
↳ most recently seen at the tail

a typical k is 20, ie. for each subtree, each node holds 20 contacts.

Updating routing table

When a node receives any message from other node in the network, it updates its appropriate k-buket with the node id.

1. entry is always added at the tail
2. entry is always created at the tail



if the k-bucket is full
↳ node pings the least-recently seen node (at the head)
↳ if no respond, then evict and insert new node at the tail
↳ if respond, new node is discarded & first node is moved to the tail

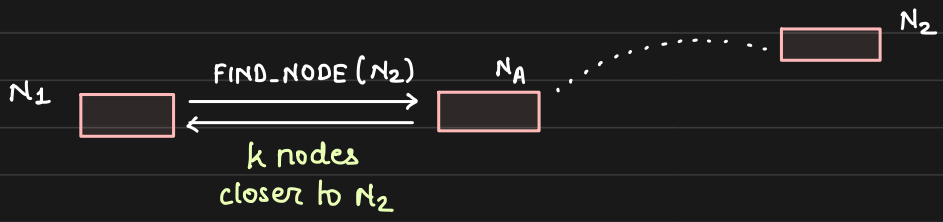
It is observed, that if a node is online for a long time, it would continue to remain online in the future k-bucket algorithm exploits this.

Communication Interface

Every node part of Kademlia exposes 4 RPC

PING : Probes a node to see if it is online

FIND_NODE : The node returns <ip, port, nodeid> for the k nodes it knows about that are closer to the requested node



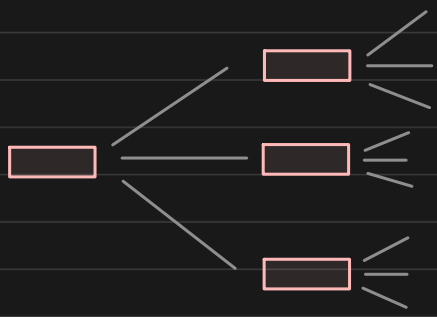
FIND_VALUE : It is like FIND_NODE but the machine holds the key, it would return the stored value.

Note: the intermediate nodes do not forward the request they just return the nodes through which we could reach the target

The lookup continues until we reach the target and complete the desired action.

STORE : Instructs a node to store <k, v> pair

To store a <k,v> pair, a node locates k-closest nodes and sends them STORE RPC



Each node re-publishes to keep the k,v alive in the network.

Original publisher republishes every 24 hours and nodes store them with 24 hour expiration

* The implementation of STORE varies as per the use-case
  - ↳ single-copy / multiple-copies
  - ↳ expiration / no expiration
  - ↳ reads / write responsibilities

  } Torrent may have different implementation than digital certificates.

Performance optimization

cache the k,v pair throughout the chain (with LRU eviction)

if a node goes down, neighbouring nodes would already have the k,v pairs