RRDHT

The Relaxed, Agent-Centric DHT Model for Quick Peer Discovery

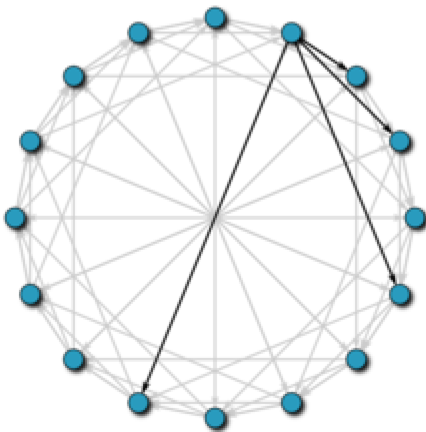
1 - Abstract

This document will attempt to describe a dynamic model for distributed peer discovery. This model should allow individual nodes to tune in real time to changing network conditions, while maintaining healthy and efficient communication.

We say "relaxed" because it is not as strictly organized as some of the other options in the following solution comparison.

2 - Solution Comparison

2.1 - Chord

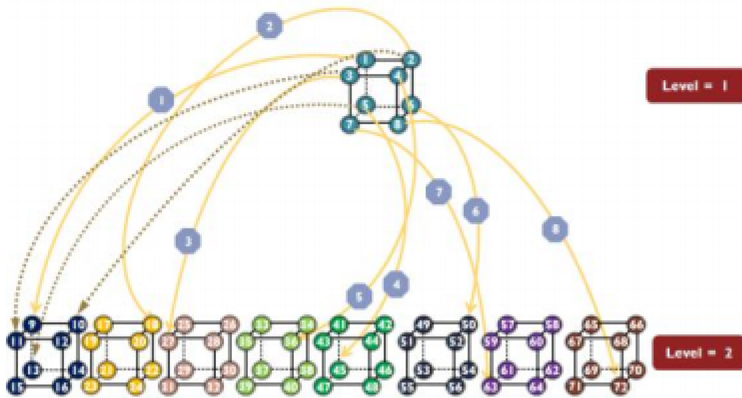


https://en.wikipedia.org/wiki/Chord_(peer-to-peer)

In a chord network, as nodes come online they distinguish their position within the ring based on their identity. These nodes identify themselves to the nodes immediately in front of and behind themselves, and become a part of the ring chain. As nodes go offline, they try to notify their connections, but in the case of failure, their connections will notice the lack of connectivity and relink themselves.

Discovery requires messaging the closest node in your "finger list", then messaging the closest node in their "finger list" and so on.

2.2 - HyperCube

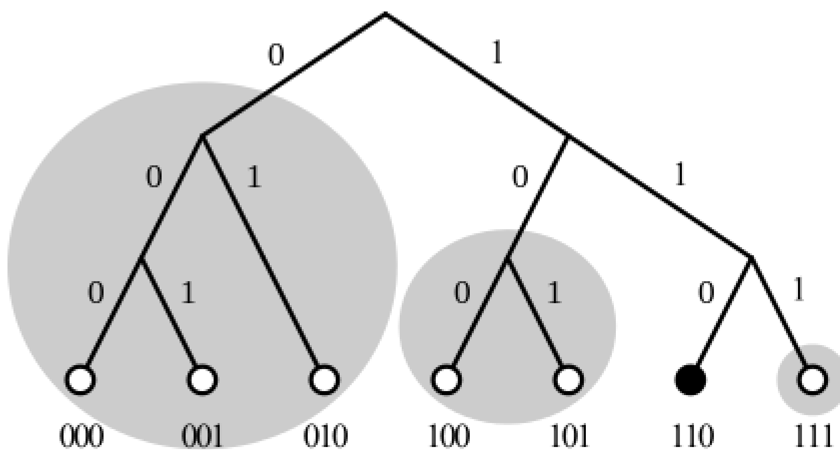


https://pdfs.semanticscholar.org/a229/61657804a5cdbb09b0c450dd9ee20733076e.pdf
https://hal.archives-ouvertes.fr/hal-00916734/document

In HyperCube, as nodes come online, they simply take the next space in the tree structure. As nodes leave, vacancies are created that can be filled by future arriving nodes. In the dire case of too many nodes leaving, the existing nodes can reorganize.

Discovery is a fairly straight-forward navigation question once you know the effectively randomly assigned position identifier of a node.

2.3 - Kademlia

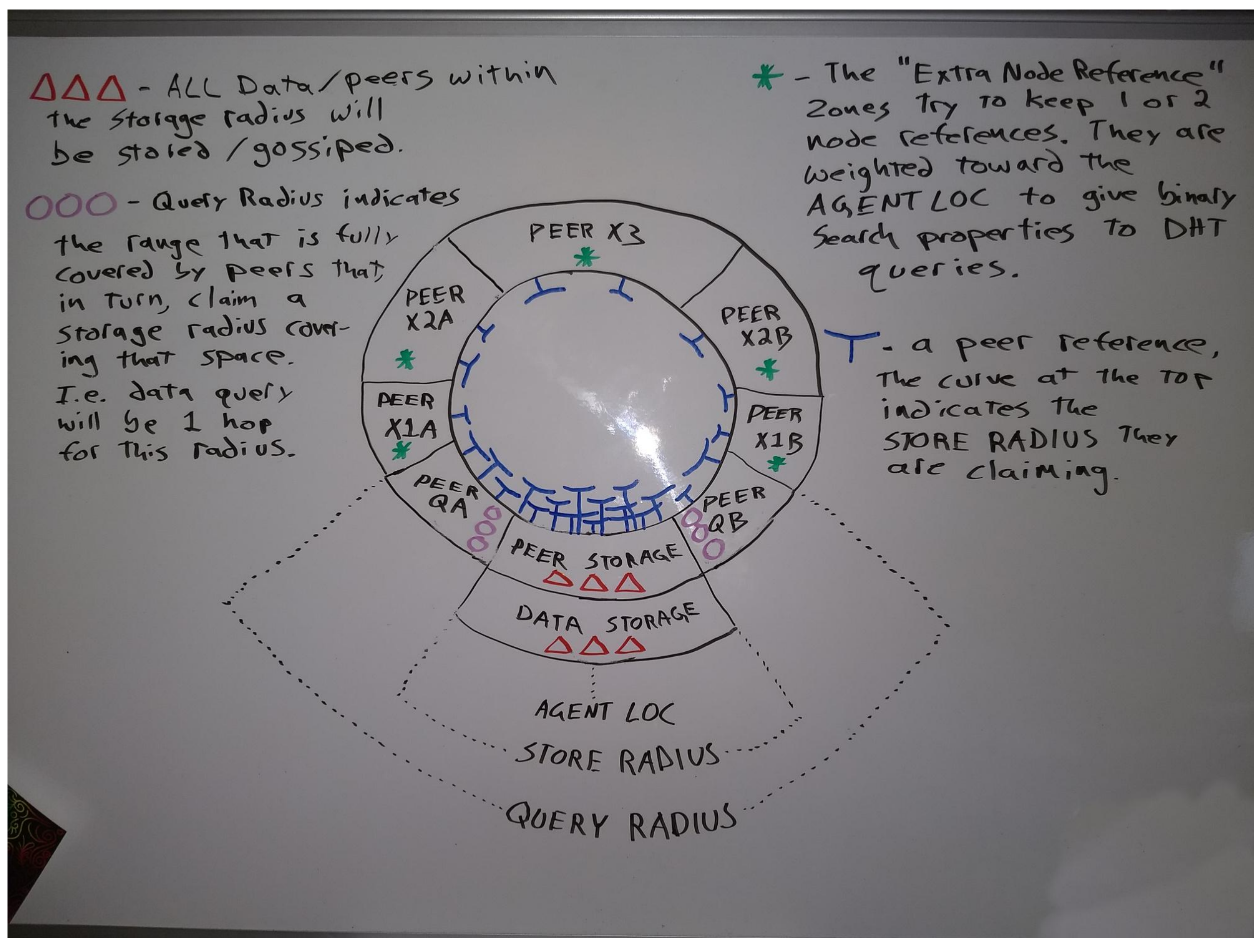


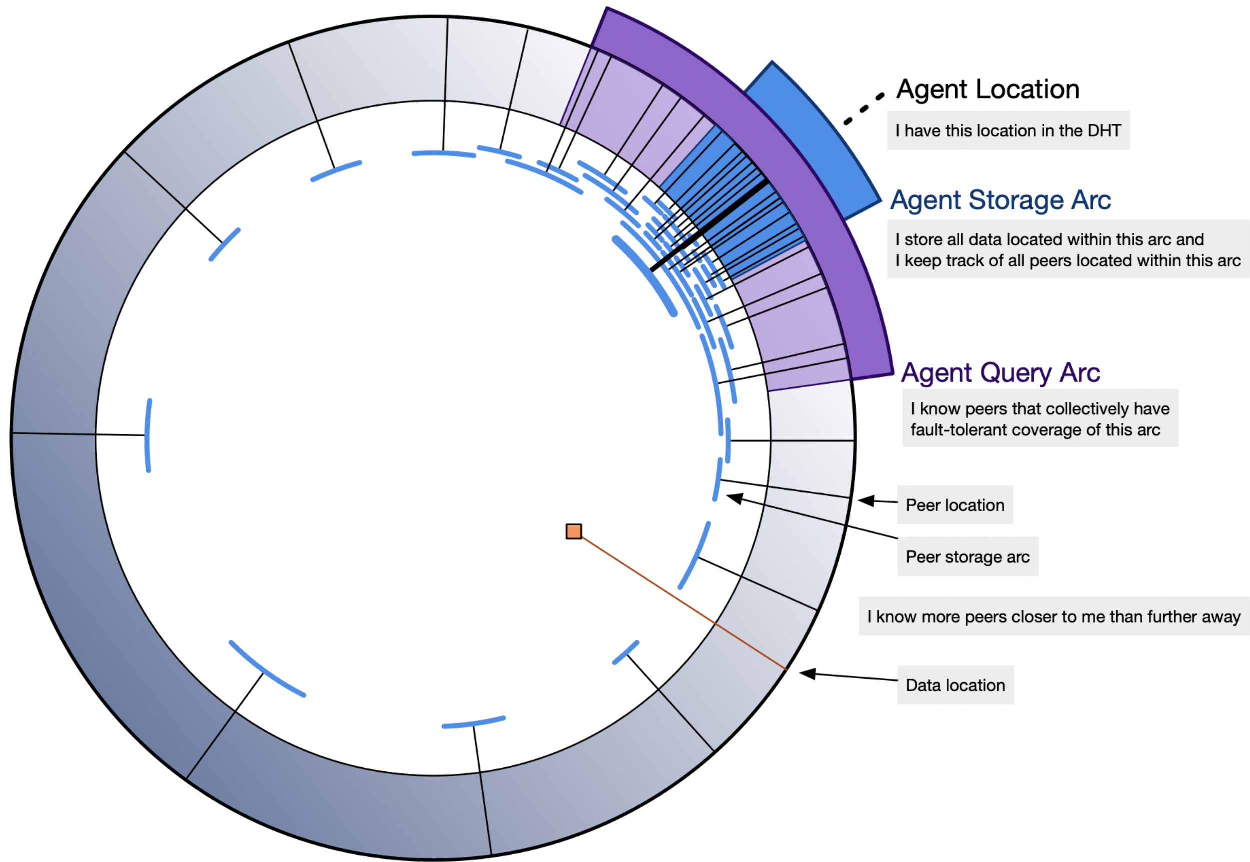
https://en.wikipedia.org/wiki/Kademlia

In Kademlia, nodes are organized into "k-buckets" according to the binary digits of their identity. The "distance" as measured by the xor of two identities determines the relative closeness of another node, and a lopsided binary tree effectively means you maintain references to more nodes closer to you than those further away.

Discovery requires making a query to a node you know about as close to your target identity as possible (based on the xor "distance"). That node should theoretically have references to more nodes in that particular neighborhood and can get you closer. Repeat as needed.

2.4 - RRDHT - The Short Version (for Comparison)





In RRDHT, nodes self-elect and publish separate “store arc”, and “query arc” values, based on a wrapping integer “agent loc” (collisions are valid). Nodes store and maintain all data within their “store arc”. The full range of a node's “query arc” must be covered at least once by a known reference to another node's “store arc”. Nodes also maintain references to nodes outside their “query arc”, organized in a manner lending binary search properties to the dht.

Discovery likely is only a single hop; a node probably already has a reference to another that is storing the requested data. If not, discovery requires making a query to a set of known nodes with an agent loc closest to the target. These nodes follow the same procedure, returning their results. Repeat until you have a set of nodes that claim to store the data.

3 - RRDHT in Detail

3.1 - Agent Identities and Agent Locs

A node should be identified by the public side of a cryptographic signing keypair.

This public key directly or a hash of this key to increase distribution shall serve as a node's “Agent Identity”. A node's “Agent Loc” can be derived by compressing this binary “Agent

Identity", into 4 bytes by applying an xor operation to every successive four bytes of the identity.

```
let hash = b"fake hash fake hash fake hash...";
let mut loc: [u8; 4] = [0; 4];
loc.clone_from_slice(&hash[0..4]);
for i in (4..32).step_by(4) {
    loc[0] ^= hash[i];
    loc[1] ^= hash[i + 1];
    loc[2] ^= hash[i + 2];
    loc[3] ^= hash[i + 3];
}
```

Online Execute In Playground

3.2 - Arc

Given a node agent loc of 42 (interpreted as a decimal uint32), an arc of 2 would indicate any peer or data with a loc in the range 40 - 44 (exclusive so that 0 can indicate no arc).

Node agent locations are intended to wrap to form a circular addressable space so that, given a node agent loc of 4294967295, an arc of 2 would indicate any peer or data with a loc greater than 4294967293 or less than 1.

3.3 - Store Arc

A node must actually have stored the data for a full arc before claiming that storage on the dht. Thus, all nodes enter the network with a store arc of 0. Once they have queried / downloaded all data to satisfy an arc of 1 (namely, any other peer agent loc or data loc that exactly matches THIS node's agent loc), they can publish a store arc of 1, and so on.

3.4 - Query Arc

For any loc within a node's query arc, that node must be able to produce a reference to a node with a store arc that overlaps that loc. Therefore, a node's query arc will always be equal to, or greater than a node's store arc (because for that range the node can always reference itself).

Similar to the store arc, a node must have references that cover the full range of the query arc before publishing that arc to peers.

3.5 - Extra Node References

Nodes will also choose to keep references to some number of additional nodes outside their query arc. When a node discovers such a reference, it will decide whether it should be kept in

favor of existing references.

This algorithm gives preference to closer nodes, and could be something like the following:

- Given the loc space remaining outside the query arc
- Call a 34% sized zone exactly in the center zone “X3”
- Call two adjacent 22% sized zones “X2A” and “X2B”
- Call the remaining two 11% sized zones “X1A” and “X1B”

Nodes will track up to 2 peers in each zone. If they already have two peers, a quality algorithm will decide which to keep based on responsiveness, size of store, query arc, and other metrics.

Imagine a worst-case scenario: a DHT network with 4 billion nodes. The network stores so much data that all nodes choose to only index an arc of 1 and keep a query arc of 2. A worst-case query should be O(log n) / roughly 22 hops.

But individual node references do not take up that much memory space, so nodes could, in fact, store a great deal more references than the above algorithm, and publish a much wider query arc than 2. These factors greatly reduce the number of hops to query. In most real-world applications, it should be trivial to achieve full query arc coverage, thus reducing the hops for any query to 1.

3.6 - Network Bootstrapping

Consider two nodes discover each other before connecting to an otherwise functional and healthy network. They each see that there are only two total nodes on the network, and that it is trivial to index and query the entire space.

Now they connect to the real network. Care must be taken to ensure they do not publish that they are indexing the entire DHT.

Assuming the good statistical distribution of our identity tags, any node should be able to estimate the entire datum count of the network. Upon connecting to a node with a wildly different count estimate, a node should reset its indexing and query arc to zero.

Nodes also will not claim any arc without first knowing that they can see resilience factor (“R”) count nodes within the arc they are attempting to store (or query). For example, if the resilience factor is 25, they will not publish either a storage or query arc greater than zero, until they have expanded enough that 25 other peers would be within their store arc.

3.7 - Publishing Data / Gossip

3.7.1 - Push

Publishing data requires talking to a node that claims responsibility for storing that data. That storing node must then make a best effort to publish that data in an exponential manner, preferably using a protocol with very little overhead, such as UDP. This node already knows what other nodes should be storing this data, as that is part of its store arc. It publishes to a number of nodes >= 2 (preferably greater than, due to UDP unreliability). If those nodes do not already have that datum, they re-publish in turn.

3.7.2 - Pull

When a node is initially syncing the network to achieve a store arc, and continuously afterward to maintain consistency, nodes should gossip with other nodes which overlap, at least partially, the same store arc. These nodes should compare nodes, gathering data each is missing.