

On Verifying Dynamical-Membership P2P Networks

THE AUTHORS

Drop few citations[1, 2] here to make it buildable.

1 Problem Definition

Formally modeling and verifying the most prevalent DHT protocol - Kademlia. Define the system states and transitions, define and verify the properties of Kademlia. The focus is on the dynamically-changed membership i.e. churning scenario that Kademlia (along with most p2p works) works with, which not only exposes challenging transitions but also affects the statements of properties.

The work may or may not deal with Kademlia specifically. If so, the contributions are

- Iterate the model into a more practical form compare to previous efforts (which are to be investigated), especially regarding the churning.
- Closely relate the verification with functional production in reality. In another word, make a system paper out of the idea.
- Optionally reasoning about Byzantine faults.

Alternatively, the work may be generalized into a universal logic and library for verifying churning networks. Either reusable facts or impossibilities would be useful. And again, reasoning about Byzantine faults is probably welcomed but not a must.

2 Rough Description of Kademlia

Most description probably also applies to other DHT protocols.

Kademlia provides two sets of services: peer discovery and key-value lookup. The former one contains less moving part and the reasoning about it may be heavily referenced for reasoning about the latter one, so we focus on the peer discovery service for now.

The visibility of the network. The p2p network contains a set of peers, each with its own ID and address. The first distinguishing feature of p2p networks is that peers by default cannot contact every other (currently online) peers, unless they know the addresses of the other peers. To bootstrap a peer i.e. upon the peer joining the network, a set of *seed peers* is hard coded into its program i.e. the IDs and addresses of the seed peers are known by every peer that ever exists. To bootstrap a p2p network, the system is initialized with the set of seed peers. Seed peers can be assumed to be never offline (at least not all of them simutenuously), and (if matters) to be always honest. They have somewhat the global view of the system as every joining peer must first contact them to properly join the network, but they usually run the identical protocol to all the other peers and e.g. do not remember every seen peer. This prevents the system to be centralized around them.

Peer discovery interface. It's not enough to just know and contact the seed peers, and Kademlia handles the rest. The peer discovery service provides single interface: $\text{FIND}(id, n)$. The interface returns *at most* n peer addresses that are *close* to id (more on the definition of closeness). If a peer learned the ID of the other peer externally and wants to contact the other peer, it can invoke $\text{FIND}()$ with $n = 1$ to find out the address. If some peers happen to invoke $\text{FIND}()$ with same id and n (*almost*) *at the same time*, the result *may be similar* on every peer. The soundness property will formalize this insight.

Peer ID space and distance. Kademlia (and most DHT protocols) defines *distance* of peer IDs that is respected by all peers. The $\text{DISTANCE}(id_1, id_2)$ returns an integer, and the larger the integer the more *distanced* the two IDs. ¹ Kademlia leverages three properties of the distance definition:

- $\text{DISTANCE}(id, id) = 0$
- $\text{DISTANCE}(id_1, id_2) = \text{DISTANCE}(id_2, id_1)$
- $\text{DISTANCE}(id_1, id_2) + \text{DISTANCE}(id_2, id_3) \geq \text{DISTANCE}(id_1, id_3)$

Peer local state. Each peer locally keeps a map from peer IDs to addresses of other peers. Kademlia carefully maintains the map so that the map contains many *close* peers and less *far* peers. In another word, every peer knows well about its local peers while know only a bit about peers far away. The function $\text{LOCALFIND}(id, n)$ can be invoked locally on the peers, which returns at most n peers that are the closest (respecting $\text{DISTANCE}()$) to id in the map.

State update. Peers may get to know about unseen peers through two ways: the replies of outgoing queries and the incoming queries. (The “query” refers to the only RPC of Kademlia.) The newly learned peer information is inserted into local map. If necessary eviction is conducted to keep the map a bounded size. The exact strategy is not described but the general idea is to keep denser information of closer peers.

FIND(id, n) implementation. Initialize peer candidate list with $\text{LOCALFIND}(id, n)$. Query the head peer of the list i.e. the peer that is closest to id based on current knowledge. The queried peer performs $\text{LOCALFIND}(id, n)$ and replies, merge the replied peers into candidate list. Repeat the query process until the top n peers in the list does not change anymore. Return them as the $\text{FIND}()$ result.

Notice that the protocol obeys the visibility rule described above. (Which is no surprise since Kademlia is a practical protocol.) All the query RPCs are made to the peers whose addresses are already known.

Bootstrap protocol. Initialize the local map with seed peers. Perform a $\text{FIND}(\text{self ID}, n)$ which serves dual purpose: populate local map with peer information that is centered around the peer itself; symmetrically announce and propagate self information to those queried peers so the peer itself becomes reachable from the further $\text{FIND}()$ happens in the network.

Refresh protocol is omitted from this note.

3 System Model

Extending an ordinary model for fixed membership network.

First with a marker of “active” peers. In the simplest form every peer active only once.

Second with a pair of peer activate/deactivate system transitions. In my opinion it is basically acceptable in practice to assume that peer activation (i.e. bootstrap) happens instantly, in another word the joining peer may interact with the network in whatever way it desired before any existing peer deactivates. The deactivation may be modeled as a broadcast message as in previous works.

The visible network abstraction. Unlike a fixed membership (or maybe “clique” assumption if I understand correctly) network where peers can freely contact every other peer in the network, under our setup a peer can only contact those peers whose addresses are known at the time to contact (and also haven’t left yet). In other p2p protocols this set of reachable peers depend on how well peer discovery service works, while in Kademlia this is somewhat self-referential.

¹Specific to Kademlia $\text{DISTANCE}()$ returns the XOR of two IDs that is interpreted as unsigned integer, but this detail is not particularly important.

4 Kademlia Properties

Many of the following I am not sure whether provable. If they are, probably demand some preconditions.

Soundness. FIND() should return peers that are either currently or previously active, but not the ones that have never been activated.

Two FIND() is considered equivalent if they are invoked with same arguments and there is no peer activate/deactivate event between the start and the end of them. Any two equivalent FIND() should return identical results, regardless of which peer invokes.

If two FIND() are equivalent except the n arguments are different, then the result of the one with larger n should be the superset of the other one.

If FIND() an ID of currently active peer, the peer should always be included in the result regardless of n argument.

(There should be some statements about the “closest” part but I cannot formalize them well for now.)

(There should be more statements that depends on the current active set of peers.)

(Anyway, I would like them to be proved in an ever-changing system setup instead of the salient network model.)

Termination. Every FIND() should be completed with finite rounds of RPCs. In fact the work achieves $\log N$ expected rounds where N is the number of active peers. I am not sure how to formalize the expected complexity.

Invariants/lemmas. Every peer should always be kept in the local map of *some* other nodes.

At any point we can construct a tree that contains all of active peers rooted by any active peer, and edge from a peer to all the peers in its local map.

A FIND() will not terminate before the top n closest peers are contacted, and will terminate *soon* after they are all contacted.

References

- [1] V. Buterin. A Next-Generation Smart Contract and Decentralized Application Platform. <https://ethereum.org/en/whitepaper/>, 2014.
- [2] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system. <https://bitcoin.org/bitcoin.pdf>, 2009.