Generalized Merkle DHT

Notation : `merkle_p_t`

Description : — name: Generalized Merkle DHT category: —

As blockchain technologies move beyond the "1.0" model of every node processing every transaction, and a more diverse ecosystem including "light clients" that achieve security by downloading only a small portion of the blockchain and extracting the rest of the data on-demand through hash-based authentication comes into play, and particularly in the long term as scalability models essentially turn $_{e}very_{n}odeintoalightclient, therearisestheneedtodevel$ $blockchain, applications$.

The kernel of the work is the distributed hash table, as pioneered by projects such as [Kademlia](http://pdos.csail.mit.edu/ petar/papers/maymounkov-kademlia-lncs.pdf) and implemented in BitTorrent and now IPFS. The distributed hash table uses a network where nodes are organized to point to other nodes in such a way that, given a value 'v' with a key 'k', a node knowing 'k' can quickly discover the location of the node that stores 'v'; usually we say 'k = H(v)' for some hash function (eg. SHA256, SHA3). Theoretically, such a DHT can be used to very easily enable light client support for $_{a}ny_{b}lockchainprotocolwithsufficientlyhighdegreesofMerkletreesupport, andevenuseonecodebaseforlightana$

However, this model has an important flaw: it requires 'log(n)' lookups in order to receive a proof. Hence, if the network latency is 500ms, and a tree has a depth of 10 nodes, then the total lookup and verification will take at least five seconds. Simpler light client protocols tend to avoid this obstacle by simply having a "request-response" model: light node A wants a proof for X, full node B provides $_{a}ll_{r}elevantMerklenodes, andlightnodeAverifies.TheDHTapproachhasthebenefitofextremegenerality; thede$

The protocol at base level is as follows: suppose there exists a program 'P' with an initial state 'S_0', $runinavirtualmachinewhichhasaccesstoan`INVHASH`("inversehash")opcode.`Pmaybeasin$ $-$', '$S < -S_0$'$andrepeatedlyundergoesthefollowingprocess$:

1. Run 'P' until it encounters either an end state (in which case exit) or an 'INVHASH' opcode with argument 'k', such that the node does not know a value 'v' such that 'sha3(v) = k'. 2. Let 'S_c'$bethecurrentstateafterthispoint.Set`S <$ $-S_c$'3.$Sendamessage`(P, S_c)`toanothernodeonthenetwork, using`k`asapointertofindingthatnode(eg.viatheKa$ $-$', $andruns`P`untilitencountersan`INVHASH`opcodewithargument`k`, suchthatthenodedoesnotknowavalue$ $k$'.$Itimmediatelyhaltsandrepliesbackwith`D'`5.Merge`D'`into`D`(ie.set`D < -DUD'`), andgotostep$1.

Essentially, the light client repeatedly asks a node on the network "run this program as long as you can until you find a value you cannot resolve, then reply back to me

with the values you did manage to resolve", and keeps on accumulating nodes in its hash until eventually it can locally run through the entire program.

This process takes:

* Linear time in the number of computational steps (as every step of execution, assuming honesty, is computed a maximum of exactly twice, once by the local node and once by the remote node) * Anywhere from constant to linear time in the number of 'INVHASH' executions, depending on what portion of all available key/value pairs every node has and how well they are clustered

It can be viewed as a superset of a DHT and Bitcoin-style SPV protocols, and is adaptable to state get operations, transaction proofs, receipt lookups, search operations, trie next/prev operations, proofs of non-membership, and generally any kind of computation on blockchain data. If implemented correctly it can be used for both Ethereum, Bitcoin proposals such as tree chains, and even parts of a decentralized search engine.

In order to prevent abuse, the program execution instance will naturally need to have a gas limit. There are two ways of doing this:

1. Have a hard fixed limit (eg. 1 million), and rely on the same anti-DDoS techniques that are normally used for messages that have a bounded response time. 2. Require each '(P, S)' message to specify a gas limit, and require either proof of work or a micropayment in exchange. If there is not enough gas, then simply return just as if you have encountered an 'INVHASH' you cannot resolve. However, this protocol may not be optimal when there is high uncertainty about when unresolvable 'INVHASH' operations will pop up. 3. The sending party creates a ticket which says "I am willing to pay X per step". The receiving party signs their reply message. The sender must then micro-pay after the fact. If the sender does not do so honestly, then the receiver can "go to blockchain court" and have the auditing contract verify the correct payment to be made, and if the correct payment was not in fact made force the sender to pay a fine.