RED QUEEN'S SYNC PROTOCOL FOR ETHEREUM

ANDREW ASHIKHMIN & ALEXEY AKHUNOV

ABSTRACT. TODO: abstract.

> "A slow sort of country!" said the Queen.
> "Now, here, you see, it takes all the running you can do, to keep in the same place. If you want to get somewhere else, you must run at least twice as fast as that!"
>
> Lewis Carroll, Through the Looking-Glass and What Alice Found There

1. INTRODUCTION

In Red Queen's Synchronisation Protocol for Ethereum 1x seeders reply with data as of their most recent block. That results in an inconsistent trie on the leecher initially ("phase 1"), which we patch later ("phase 2"). The idea is similar to Leaf Sync.

TODO: mention the sync failure problem Akhunov [2019a] and the needs of light clients like Mustekala. Inspirations like BitTorrent, Parity's warp sync, Firehose Sync, Light Client Protocol.

N.B. Snapshot synchronisation rather than from genesis.

2. NOTATION

We mostly follow the conventions and notations of the Yellow Paper Wood [2018], for instance $\mathbb{Y}$ denotes the set of nibble sequences. We use the letter π for prefixes of state or storage trie keys $\mathbf{k} \in \mathbb{B}_{32}$,

$$\pi \in \mathbb{Y} \, \wedge \, ||\pi|| \leq 64 \tag{1}$$

A key matches a prefix iff all their first nibbles are the same,

$$\mathtt{MATCH}(\mathbf{k}, \pi) \equiv \forall_{i < ||\pi||} : \mathbf{k}'[i] = \pi[i] \tag{2}$$

($\mathbf{k}'$ is a sequence of nibbles, while $\mathbf{k}$ is a sequence of bytes.)

3. PROTOCOL SPECIFICATION

We propose the following 3 request/reply operative pairs.

GetStorageSizes [+0x20, reqID: $\mathbb{N}$, blockAtLeast: $\mathbb{N}$, [account[0]: $\mathbb{B}_{20}$, account[1]: $\mathbb{B}_{20}$, ...]] Request storage trie sizes as of block #`blockAtLeast` or newer.

StorageSizes [+0x21, reqID: $\mathbb{N}$, blockNumber: $\mathbb{N}$, [numLeaves[0]: $\mathbb{N}|\varnothing$, numLeaves[1]: $\mathbb{N}|\varnothing$, ...]] Reply to `GetStorageSizes`. Returns storage trie sizes as of block #`blockNumber` $\geq$ `blockAtLeast`. The elements returned must strictly match the accounts requested. The peer may return the empty list $\varnothing$ instead of the number of leaves for accounts it does not have enough information about.

GetNodeData2 [+0x22, reqID: $\mathbb{N}$, blockAtLeast: $\mathbb{N}$, [account[0]: $\mathbb{B}_{20}|\varnothing$, prefix$_0^0$: $\mathbb{Y}$, prefix$_1^0$: $\mathbb{Y}$, ...], [account[1]: $\mathbb{B}_{20}|\varnothing$, prefix$_0^1$: $\mathbb{Y}$, prefix$_1^1$: $\mathbb{Y}$, ...], ...] Request state or storage trie nodes as of block #`blockAtLeast` or newer. The empty list $\varnothing$ instead of the account address signifies the state (rather than storage) trie. Note that this operative is similar to `GetNodeData` from Ethereum Wire Protocol PV63, but it uses prefixes rather than hashes as node keys[1]. TODO: prefix–node correspondence is trivial for branch nodes, not so much for leaf or extension nodes. TODO: prefix encoding consistent with the Yellow Paper.

Date: March 2019.

[1]For a justification see Péter Szilágyi's comment at ETH v64 Wire Protocol Ring.

NodeData2 [+0x23, reqID: $\mathbb{N}$, blockNumber: $\mathbb{N}$, [$node_0^0$: $\mathbb{B}$, $node_1^0$: $\mathbb{B}$, ...], [$node_0^1$: $\mathbb{B}$, $node_1^1$: $\mathbb{B}$, ...], ...] Reply to GetNodeData2. Returns trie nodes as of block #blockNumber $\geq$ blockAtLeast. The nodes returned must strictly match the prefixes requested. The empty list $\varnothing$ returned instead of a node means that the peer does not have enough information about the node requested.

GetLeaves [+0x24, reqID: $\mathbb{N}$, blockAtLeast: $\mathbb{N}$,

  [$account^0$: $\mathbb{B}_{20}|\varnothing$,

    [$prefix_0^0$: $\mathbb{Y}$, $fromLevel_0^0$: $\mathbb{N}$, $subtrieHash_0^0$: $\mathbb{B}_{32}|\varnothing$],

    [$prefix_1^0$: $\mathbb{Y}$, $fromLevel_1^0$: $\mathbb{N}$, $subtrieHash_1^0$: $\mathbb{B}_{32}|\varnothing$],

    ...

  ],

  [$account^1$: $\mathbb{B}_{20}|\varnothing$,

    [$prefix_0^1$: $\mathbb{Y}$, $fromLevel_0^1$: $\mathbb{N}$, $subtrieHash_0^1$: $\mathbb{B}_{32}|\varnothing$],

    [$prefix_1^1$: $\mathbb{Y}$, $fromLevel_1^1$: $\mathbb{N}$, $subtrieHash_1^1$: $\mathbb{B}_{32}|\varnothing$],

    ...

  ],

  ...

] Request state or storage subtrie leaves along with proof nodes as of block #blockAtLeast or newer. The empty list $\varnothing$ instead of the account address signifies state rather than storage trie. fromLevel specifies the number of upper nodes to be excluded from the proof. subtrieHash is a means to avoid resending leaves that have not changed.

Leaves [+0x25, reqID: $\mathbb{N}$, blockNumber: $\mathbb{N}$,

  [

    [[$node_{00}^0$: $\mathbb{B}$, $node_{01}^0$: $\mathbb{B}$, ...], $tooManyLeaves_0^0$, [key_{00}^0: $\mathbb{B}_{32}$, val_{00}^0: $\mathbb{B}$, key_{01}^0: $\mathbb{B}_{32}$, val_{01}^0: $\mathbb{B}$, ...]$_{opt}$],

    [[$node_{10}^0$: $\mathbb{B}$, $node_{11}^0$: $\mathbb{B}$, ...], $tooManyLeaves_1^0$, [key_{10}^0: $\mathbb{B}_{32}$, val_{10}^0: $\mathbb{B}$, key_{11}^0: $\mathbb{B}_{32}$, val_{11}^0: $\mathbb{B}$, ...]$_{opt}$],

    ...

  ],

  [

    [[$node_{00}^1$: $\mathbb{B}$, $node_{01}^1$: $\mathbb{B}$, ...], $tooManyLeaves_0^1$, [key_{00}^1: $\mathbb{B}_{32}$, val_{00}^1: $\mathbb{B}$, key_{01}^1: $\mathbb{B}_{32}$, val_{01}^1: $\mathbb{B}$, ...]$_{opt}$],

    [[$node_{10}^1$: $\mathbb{B}$, $node_{11}^1$: $\mathbb{B}$, ...], $tooManyLeaves_1^1$, [key_{10}^1: $\mathbb{B}_{32}$, val_{10}^1: $\mathbb{B}$, key_{11}^1: $\mathbb{B}_{32}$, val_{11}^1: $\mathbb{B}$, ...]$_{opt}$],

    ...

  ],

  ...

] Reply to GetLeaves. Returns subtrie leaves with proofs as of block #blockNumber $\geq$ blockAtLeast. The subtries returned must strictly match the prefixes requested. If the peer does not have information regarding a particular subtrie, it should return the empty list $\varnothing$ (e.g. []) rather than [nodes, tooManyLeaves, leaves] for it. The nodes returned are the upper nodes of the trie down to the subtrie root, so that it is possible to verify that the leaves do belong to the Merkle Patricia trie in question. The first fromLevel upper nodes must be skipped. (If fromLevel = 0, then the nodes must start with the root node.) tooManyLeaves is a boolean flag ($0 = false$, $1 = true$) indicating that the subtrie requested contains too many leaves. TODO: how many is too many? The leaves are represented as the list of their keys[2] and values. The peer may only return either all leaves of the subtrie or nothing. In case of tooManyLeaves the leaves should not returned[3].

Note that state trie replies do not inline storage tries unlike Leaf Sync.

TODO: is block number OK given chain reorgs?

4. Suggested Full Sync Algorithm

TODO: top-level trie with branch nodes only; nodes track blocks; phase 1, phase 2.

---

[2]It is feasible to return suffixes rather than full keys given that prefixes are known, but we deem the performance gain to be insignificant.

[3]In that case the peer must not return an empty list as that would imply that there are no leaves matching the given prefix.

5. Performance Analysis

For this analysis we assume that all tries are well balanced. We also assume that all top nodes up to a certain trie level i are branch nodes, not leaf nor extension nodes. This is a reasonable assumption if i is not too big—see Akhunov [2019b].

Some notation (TODO: internal notation consistency + cross-check against the Yellow Paper):

o – `Leaves` reply overhead in bytes.

b – size of a branch node in bytes.

l – average leaf size in bytes.

$||R_b||$ – number of nodes in reply.

$||R_l||$ – number of leaves in reply.

Thus the size of a reply, assuming the average leaf size, is

$$S(R) = o + ||R_b||b + ||R_l||l \tag{3}$$

TODO: RLP changes the formula slightly.

t – total number of leaves in the trie.

TODO: optimal phase 1 depth.

Now let us find the maximum size d of the request prefix π that makes sense to use when we are catching up (phase 2 of the sync). Let assume that only one leaf that matches π has changed. (If we know that there are no changes, there is no need for a sync request.) That is a reasonable assumption if we are not too many blocks behind, there are not that many changes per block, and d is sufficiently large. (For instance, if we are 100 blocks behind, and there are 500 leaf changes per block, then $d \geq 4$ will suffice.) Consider two options: request the prefix π or send requests with prefixes $\pi \cdot 0, ..., \pi \cdot 15$ of size $d+1$ (not necesseraly all 16 of them). In the first case we receive a reply of the size, on average,

$$S = o + ||R_b||b + \frac{t}{16^d}l \tag{4}$$

For the second case we need to send at most two requests, as the first reply will give us the information to identify which nibble has changed. (With the probability $\frac{1}{16}$ the second request is not necessary.) The combined size of those 1 or 2 replies, on average, is

$$S' = \left(1 + \frac{15}{16}\right)o + (||R_b|| + 1)b + \frac{t}{16^{d+1}}l \tag{5}$$

It does not make sense to prefer requests with longer prefixes if $S \leq S'$. Solving this inequality, we obtain

$$16^d(16b + 15o) \geq 15tl \tag{6}$$

In other words, it does not make sense to use prefixes longer than

$$\left\lceil \log_{16} \frac{15tl}{16b + 15o} \right\rceil \tag{7}$$

TODO: convergence analysis.

6. Conclusion

TODO: conclusion.

References

Alexey Akhunov. Looking back at the Ethereum 1x workshop 26–28.01.2019 (part 1), January 2019a. URL https://medium.com/@akhounov/looking-back-at-the-ethereum-1x-workshop-26-28-01-2019-part-1-70c1ebd93266.

Alexey Akhunov. Looking back at the Ethereum 1x workshop 26–28.01.2019 (part 2), February 2019b. URL https://medium.com/@akhounov/looking-back-at-the-ethereum-1x-workshop-26-28-01-2019-part-2-d3d8fdcede10.

Gavin Wood. Ethereum: A Secure Decentralised Generalised Transaction Ledger, December 2018. URL https://github.com/ethereum/yellowpaper.