

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 that of Leaf Sync (Swende [2019]).

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 $\mathbf{k} \in \mathbb{B}_{32}$,

$$(1) \quad \pi \in \mathbb{Y} \wedge \|\pi\| \leq 64$$

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

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

(\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} , [account⁰: \mathbb{B}_{20} , account¹: \mathbb{B}_{20} , ...]] Request storage trie sizes as of block #**blockAtLeast** or newer.

StorageSizes [+0x21, reqID: \mathbb{N} , blockNumber: \mathbb{N} , [numLeaves⁰: $\mathbb{N}|\emptyset$, numLeaves¹: $\mathbb{N}|\emptyset$, ...]] 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 \emptyset instead of the number of leaves for accounts it does not have enough information about.

GetNodeData2 [+0x22, reqID: \mathbb{N} , blockAtLeast: \mathbb{N} , [account⁰: $\mathbb{B}_{20}|\emptyset$, prefix⁰: \mathbb{Y} , prefix¹: \mathbb{Y} , ...], [account¹: $\mathbb{B}_{20}|\emptyset$, prefix⁰: \mathbb{Y} , prefix¹: \mathbb{Y} , ...], ...] Request state or storage trie nodes as of block #**blockAtLeast** or newer. The empty list \emptyset 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 keys¹. 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.

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

NodeData2 [+0x23, reqID: \mathbb{N} , blockNumber: \mathbb{N} , [node₀⁰: \mathbb{B} , node₁⁰: \mathbb{B} , ...], [node₀¹: \mathbb{B} , node₁¹: \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 \emptyset returned instead of a node means that the peer does not have enough information about the node requested.

GetSubtries [+0x24, reqID: \mathbb{N} , blockAtLeast: \mathbb{N} ,
 [account⁰: $\mathbb{B}_{20}|\emptyset$,
 [prefix₀⁰: \mathbb{Y} , fromLevel₀⁰: \mathbb{N} , subtrieHash₀⁰: $\mathbb{B}_{32}|\emptyset$],
 [prefix₁⁰: \mathbb{Y} , fromLevel₁⁰: \mathbb{N} , subtrieHash₁⁰: $\mathbb{B}_{32}|\emptyset$],
 ...
],
 [account¹: $\mathbb{B}_{20}|\emptyset$,
 [prefix₀¹: \mathbb{Y} , fromLevel₀¹: \mathbb{N} , subtrieHash₀¹: $\mathbb{B}_{32}|\emptyset$],
 [prefix₁¹: \mathbb{Y} , fromLevel₁¹: \mathbb{N} , subtrieHash₁¹: $\mathbb{B}_{32}|\emptyset$],
 ...
],
 ...
]

] Request state or storage subtrie leaves along with proof nodes as of block #**blockAtLeast** or newer. The empty list \emptyset instead of the account address signifies state rather than storage trie. **fromLevel** specifies the number of upper nodes to be excluded from the proof in case the chain has not moved ahead (reply block is not newer). **subtrieHash** is a means to avoid resending leaves that have not changed.

Subtries [+0x25, reqID: \mathbb{N} , blockNumber: \mathbb{N} ,
 [
 [[node₀₀⁰: \mathbb{B} , node₀₁⁰: \mathbb{B} , ...], tooManyLeaves₀⁰, [key₀₀⁰: \mathbb{B}_{32} , val₀₀⁰: \mathbb{B} , key₀₁⁰: \mathbb{B}_{32} , val₀₁⁰: \mathbb{B} , ...]_{opt}],
 [[node₁₀⁰: \mathbb{B} , node₁₁⁰: \mathbb{B} , ...], tooManyLeaves₁⁰, [key₁₀⁰: \mathbb{B}_{32} , val₁₀⁰: \mathbb{B} , key₁₁⁰: \mathbb{B}_{32} , val₁₁⁰: \mathbb{B} , ...]_{opt}],
 ...
],
 [
 [[node₀₀¹: \mathbb{B} , node₀₁¹: \mathbb{B} , ...], tooManyLeaves₀¹, [key₀₀¹: \mathbb{B}_{32} , val₀₀¹: \mathbb{B} , key₀₁¹: \mathbb{B}_{32} , val₀₁¹: \mathbb{B} , ...]_{opt}],
 [[node₁₀¹: \mathbb{B} , node₁₁¹: \mathbb{B} , ...], tooManyLeaves₁¹, [key₁₀¹: \mathbb{B}_{32} , val₁₀¹: \mathbb{B} , key₁₁¹: \mathbb{B}_{32} , val₁₁¹: \mathbb{B} , ...]_{opt}],
 ...
],
 ...
]

] Reply to **GetSubtries**. 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 \emptyset (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 and only if **blockNumber** = **blockAtLeast**. (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 keys² and values. The peer may only return either all leaves of the subtrie or nothing. In case of **tooManyLeaves** the leaves should not returned³.

Proof nodes in **Subtrie** replies give us a means to detect faulty or malicious peers. Note that state trie replies do not inline storage tries unlike Leaf Sync.

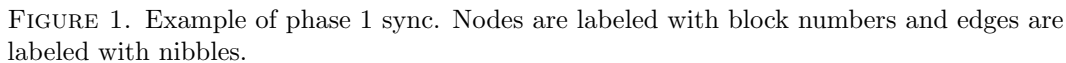
TODO: is block number OK given chain reorgs?

4. SUGGESTED FULL SYNC ALGORITHM

Here we suggest a possible algorithm for full state and storage snapshot synchronisation using the protocol specified above. We describe a modus operandi where the seeder replies with its most recent data, and the

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

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



After all trie levels from the root down to the penultimate level are up to date, the leecher determines which nodes at the ultimate level $d_2 - 1$ have to be refreshed. For such nodes it sends a **GetSubtrie** request with a prefix of size d_2 corresponding to that node. The last nibble is set arbitrarily. **subtrieHash** is set in order to avoid leaf resending in case the leaves the leecher obtained during phase 1 are still valid. Given a reply the leecher finds out which of the nibbles are still valid and which have to be refreshed with **GetSubtrie**. Repeating the process for each stale node at the last level $d_2 - 1$ concludes the algorithm for state trie synchronisation.

⁵ d_2 is small enough so that we can reasonably assume that (almost) all nodes in question are branch nodes; see Akhunov [2019b].

The algorithm for storage sync is similar for large storage tries. Its parameters d_1 and d_2 are optimised based on the storage size as described in the next section. Small storage tries can be obtained in bulk requesting the zero prefix (meaning the entire trie) for a number of them in one go. `GetStorageSizes` provides a means to finding out storage sizes.

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

$$(3) \quad S(R) = o + ||R_b||b + ||R_l||l$$

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 $\pi \cdot 0, \dots, \pi \cdot 15$ of size $d+1$ (not necessarily all 16 of them). In the first case we receive a reply of the size, on average,

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

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

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

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

$$(6) \quad 16^d(16b + 15o) \geq 15tl$$

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

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

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/@akhunov/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>.
- Martin Holst Swende. Leaf sync, March 2019. URL https://notes.ethereum.org/kphcc_CKT4a5sUs_zWVe1A.
- Gavin Wood. Ethereum: A Secure Decentralised Generalised Transaction Ledger, December 2018. URL <https://github.com/ethereum/yellowpaper>.