For context see:

1. History, state, and asynchronous accumulators in the stateless model
2. A cryptoeconomic accumulator for state-minimised contracts
3. Batching and cyclic partitioning of logs

TLDR

: We describe a log accumulator with two layers of batching, significantly improving the concrete efficiency of Merkle Mountain Ranges (MMRs) and multi-MMRs (3MRs). The construction has all-round exceptional concreteness, potentially making it an ideal log accumulator for Ethereum stateless clients. In particular witnesses are only ever updated once

.

Construction

Every shard is endowed with two buffers storing 32-byte hashes:

1. Bottom buffer:

Fixed size with 2^n

entries labelled 0, 1, ..., 2^n - 1

1. Top buffer:

Variable size increasing linearly with collation height

Log accumulation is done as follows:

1. The logs produced in the collation with height i

are batched into a Merkle tree with the log batch root placed in the bottom buffer with entry labelled i

modulo 2^n

.

1. When the last entry in the bottom buffer is updated the bottom buffer is itself batched into a Merkle tree with the root appended to the top buffer.

Notice that log witnesses only ever get updated once when the bottom buffer reaches its last entry. We call the initial witness upon log creation (the log batch Merkle path) the “pre-witness”. We call the final witness created by concatenating the pre-witness with bottom buffer Merkle path (n

hashes) the “permanent witness”.

The size of the accumulator is 32*(2^n + h/2^n)

bytes where h

is the collation height. For concreteness we set the collation interval to 8 seconds and we set n = 13

. The bottom buffer has fixed size 250kB. The top buffer grows linearly, but will take 51 years

to reach 750kB. So for all practical purposes, the accumulator can be considered to have size < 1MB.

Because we now have a concept of permanent witnesses, these can be safely stored (e.g. in cold storage) alongside private keys and left unattended arbitrarily long. Note that it is actually sufficient to store the pre-witness because the bottom buffer Merkle path can be reconstructed by SPV clients if log batch roots (one per collation) are placed in collation headers.

Notice also that this accumulator is dead-simple to reason about and implement, and induces marginal storage and CPU overhead for fully validating nodes.

Conclusion

We may have found the ultimate log accumulator for Ethereum! It delivers single-update witnesses (at deterministic time, within 2^n*8

seconds = 18 hours) and SPV recovery from pre-witnesses. It has a low complexity spec and implementation, and introduces marginal overheads for fully validating nodes.