One challenge with current rollup protocols is that if you are a light client of the rollup (ie. you're not processing the transactions in the rollup yourself, you're talking to third parties to get proofs of any part of the state you care about), you have no way to generate a Merkle proof of your rollup account balance to exit the rollup without either (i) talking to a third party or (ii) processing the rollup's entire history yourself.

This is not a large problem in practice, as we expect there to be many state-providing intermediaries and only one honest intermediary is required to assist an exit, but especially if we want to guarantee the ability to exit a rollup quickly even in extreme scenarios, it seems desirable to try to improve on this.

Understanding the status quo

A rollup is a sequence of on-chain batches

; each batch contains a list of state updates

(eg. "account 482 now has balance 808000"), a post-state root

(the root hash of the new rollup state after applying those updated), and an attestation

claiming that the post-state root really is the result of applying the updates to the previous state root. In an optimistic rollup, the attestation is a signature from an account that has ETH or other assets deposited, which could be confiscated if they signed an incorrect claim (which can be later proven on-chain via a fraud proof), in a ZK-rollup, it's a SNARK directly proving the claim.

To withdraw from a rollup, the user needs to generate the current state. They can do this by starting from the beginning, and sequentially processing all state updates (note: there's no need to compute any hashes while doing this; just keep a table of the whole state). At the end, they generate the entire Merkle tree, allowing them to withdraw.

Suppose that a rollup has n

transactions-per-batch, has been in existence for t

batches, and has state growth factor r

(this is just the portion of activity in the rollup that is creating new accounts, as opposed to editing existing accounts; we're assuming an eth1-like rollup with no substantial state clearing). The above algorithm requires O(nt)

"logistical" operations (searching for a key in a map, editing values, doing arithmetic; all very cheap) and \approx ntr

hashes. This is actually surprisingly cheap; you can compute ~1 million hashes per second on a consumer laptop, so a 1200-tx-per-batch (100 TPS) rollup running for a year (~2.5m batches) would require ~3 billion logistical operations and 300 million hashes assuming r=0.1

. So far, ~5-10 minutes to generate the current state and be able to withdraw.

Things get trickier when the rollup state size exceeds RAM size, as O(nt)

disk accesses would also be required. Disk accesses are unfortunately only possible in the low tens of thousands per second, and so here unfortunately generating the state starts to take hours or longer.

Proposal

Instead of each rollup batch containing one post-state root, we split up the state into \frac{n}{32}

sections, where n

is the expected transaction count per batch; the batch must include the post-state root of each section. Note that we can still only save one root on-chain in the state; we just need the batch attestation to also attest to the \frac{n}{32}

section roots. Additionally, state updates in the batch must be themselves split up by sections.

Now, generating an exit witness (a Merkle branch for an account) can be split up into two parts: (i) the part of the branch proving the section root is part of the global state, which can be done using only the section roots, and (ii) proving that the account is committed to in the section root.

(i) is trivial, requiring only \frac{n}{32}

hashes (reminder: n

is transactions-per-batch, so this would at most be ~1000).

(ii) requires the same procedure as in the status quo: clients scan through all historical batches, compute the state and then finally generate the Merkle tree. However, instead of concerning themselves with the entire history, clients can look only at the portion of the batch that deals with the section their account is in, compute only that state, and Merklize only the state.

Now, let's look at client overhead: on average, each section deals with 32 transactions, so the logistical overhead is O(32t)

, and the hashing required is O(32tr)

: both constant

in the level of usage (!!!). The additional overhead added by this scheme is one hash per 32 transactions, so roughly one byte added to each transaction in the rollup (this could be reduced further by reducing the section count to eg. \frac{n}{256}

for one-bit-per-transaction data overhead). Realistically, the main overhead for the client will be that of re-fetching the historical data, if the client does not store history (as most eth1 clients likely won't in the future).

Extension: limiting history lookup

A rollup with some state clearing procedure (rent or otherwise) to allow a fixed state size could also require batches to provide on-chain specific accounts (eg. if the state size is D

, the k\ mod\ D

'th account in the state would be published as part of the k

'th batch). This ensures that clients looking to withdraw would only need to go back a finite amount of time in history. Alternatively, the state clearing procedure may well explicitly require state objects to be “poked” once every eg. 6 months to remain part of the state; in that case, we get this benefit “for free” already and there's no need for an extra procedure to deliberately walk through and re-publish accounts from the state.