TL;DR

: A proposal to make sure that the state of the “stateless” Execution Environment is maintained in an inexpensive and scalable manner. I discuss why I think the Proof of Custody construction is actually the best choice here, and show that this can probably be done quite cheaply, which means that “deposit as rent” will provide a viable construction.

Background

On Ethereum 2.0, shard data chains are planned to be almost stateless. This is important because it allows validators to be reassigned between shards frequently at low cost, which can only be achieved if shard state is of the orders of kilobytes. If shard state were much more than that, then the load on validators would increase a lot – the cost of downloading the current state of the shard to be validated could easily outstrip the cost of downloading and verifying a block. Keeping a state of the size of just one shard block essentially doubles per validator load.

Data availability checks and proof of custody

The current Eth2.0 design includes two different methods to ensure that, upon finality, the shard block data is always available: data availability checks and the proof of custody (1, 2)

- Proofs of Custody

are a cryptoeconomic scheme that incentivizes validators to have and keep data (currently used for block data, but other data such as execution traces has been suggested)

- Data availability checks

can allow an external party to verify that the data is on a (neutral) Peer-to-Peer network and can be downloaded with very high probability

The concern about Execution Environment state

The current plan is for each execution environment to have only 32 bytes of state per shard, which is just enough to store one state root in the form of a 256 bit hash; in most cases, this will be used to store the root of a much larger state, and all access to this state will then have to come with complete witnesses allowing validators to verify all inputs and changes.

The shard blocks will essentially give a changelog to the state, and this is secured via both proof of custody and data availability checks. However, keeping the actual state is still an open problem.

Suggested solution

Choosing the right tool

We use both the proof of custody and data availability checks as mechanisms to ensure shard block data availability. Proof of custody ensures that the entire set of validators attesting actually has the data (not just some subset boosted by lazy validators), and data availability means everyone can check that it is also available for download and couldn’t be withheld even if the entire committee colluded to do so.

At first sight, it may seem that we need to employ the same solution for the Execution Environment state. However, there is a difference in that there is almost no conceivable attack that comes from EE state withholding:

- Unlike shard block data, there is no problem that we need to ensure the availability in order to enable fraud proofs. We already know that the state root is correct, as every shard block (which is proven available) will include the complete data necessary to verify the transition. It is thus not possible to use this as an attack vector to introduce an incorrect state transition (thus stealing or printing money).
- The only remaining attack would be to “freeze” someone’s assets or contract by “forgetting” that part of the state. Note that an attacker can never directly gain from such an action (as contracts/accounts always expand and never deny capabilities), so it seems like a very unlikely form of attack. What makes it worse for the attacker is that the process of trying to forget the state in itself would ensure that that part of the state would have to be brought into the current chain, thus ensuring its availability as part of shard block data availability/custody for a long time (at least 9 days). The victim or anyone else would thus have plenty of time to snapshot it, making it very unlikely that anyone would ever actively try this.

This means that freezing assets by forgetting EE data is not a credible attack vector. We can instead focus on the problem that some data might just plain be forgotten by accident. The good thing is that we can prevent this using the proof of custody

, which stops validator laziness.

Fortunately, the proof of custody is much cheaper to compute than data availability roots, and do not require any change in the fork choice rule. They also scale linearly with the data size.

Possible scheme based on proof of custody

Here is a suggested scheme based on the proof of custody:

- For each EE+shard pair, a validator can choose to be a state custodian. The EE can pay its state custodian a reward
- When a state custodian is in the persistent committee of a shard, it has to sign a custody root per epoch for each EE it declared itself a custodian. If it skips the custody root, a small penalty is exacted

Each EE can thus set its own pricing scheme for validators to keep the state, and would probably try to target a safe number of validators (say 100/shard). There is thus a price discovery mechanism built into the system.

Performance/costs

Validators have an incentive to become state custodians if paid more than the costs to perform this duty. Luckily all operations are linear, so I will give an estimate per 10 GB of state data (per shard):

- Cost of downloading state (once per day on persistent committee change): $1
- Cost of computing custody root: ca. 1.5h core hours/day (20s/custody root): $0.05
- Cost of storing data of 9 shards (custody period), 90 GB: $0.30

(prices estimated using https://calculator.s3.amazonaws.com/index.html)

This means it will probably be possible to find enough people volunteering to join the scheme at around $2/day reward for 10 GB of state per shard (note that this is likely an upper bound – this is more than current expected returns per validator, so it is very likely that EEs will bid much less than this and still get enough custodians).

So if an EE were targetting 100 validators per shard, the total cost would be on the order of 64 * 100 * 2 = 4.6M$. It is very likely that in practice it would be much less than that as the node with the cheapest access to storage would be the ones determining the supply, and those are likely unlimited broadband home nodes which can very cheaply add 100GB of storage or might even have this available on their operating system disk and not using it at the moment.

Centralization risk

One big risk with this scheme is that since most of the cost is in storing data and downloading state, it is much more advantageous for large stakers to do this compared to smaller ones, as they would be able to compute several custody roots from the same state copy and thus collect multiple rewards. Note that this problem already exists without this mechanism – once a staker runs a supernode of 64 validators, costs don't grow any further but rewards do. However, in this case the staker will fool the EE into believing that there are more independent copies of the state than there really are, which is not ideal and may also compromise the pricing mechanism.

A full solution to this would be by changing it from a Proof of Custody protocol to a Proof of Replication.

Layer 2 implementation

It is possible to implement this scheme purely in layer 2. However, this would be quite wasteful as each EE would have to make state custodians lock up assets to secure it; which would also mean having to pay a much higher price for this service.

However, we could create a more flexible system where EEs get a way to slash validators as well (if the validator consented to this; which again they would expect payment for). That would allow the concrete mechanism to be built inside the EE rather than at layer 1. However, this would allow creating mechanisms to use PoS stake in DeFi applications etc., so it is probably not a good idea to make it that general.

Charging contracts

A nice property of this system is that it allows price discovery for state storage, and so we don't have to pay inflated prices in order to be safe.

An EE can then use a rent mechanism to charge its contracts for the state storage.

Unfortunately, contract level state rent has terrible UX. The very low price that I expect for this at reasonable state sizes

(probably only ca. 1% of consensus costs) suggests that there may be ways to ensure this other than rent. An EE may instead chose to limit its state to a reasonable size using a “deposit as rent” mechanism, and choose to pay the rent out of slightly increased transaction fees; or we could introduce a global 0.01% interest rate on Ether held by EEs that they can use to pay this (or use in another way if state is not needed). The question is how an EE would handle the case that this isn’t enough to cover storage. It would be nice to have a graceful purge mechanism in this case.

Here is how such a mechanism could work in a deposit-as-rent EE (DAR EE):

- Introduce a global 0.01% interest rate for top-level ETH held by EEs
- The DAR EE chooses a deposit rate DR

in Wei/byte

- Any contract that has a deposit less than DR

Wei per byte of storage it consumes, can be evicted by poking (a receipt will be produced that allows it to be reinstated for a fee)

- The DR

is readjusted automatically or through some governance process so that the global interest rate covers about twice the custody fees (to allow for some safety margins in case of fluctuations). The rest of the fees can be kept by the EE and for example used to incentivize development on that EE

Now let’s see what a reasonable DR

would be: Let’s assume that 25% of Ether is inside an EE and we agree that we want to use a 0.01% inflation rate to pay for state. Then that EE would receive about 625,000 USD (@250

USD/ETH) from the interest rate mechanism, thus being able to fund 1.35 GB shard state using our relatively conservative cost estimate above. Thus a DR

of 25,000,000 ETH/(64*1.35e9 Bytes) = ca. 300,000 GWei/byte would be safe (though, using the above mechanism, it is likely to be lower as many accounts will contain much more ETH than state). Using this conservative estimate, a user account with 96 bytes storage (32 bytes for address, nonce and balance) would need a minimum balance of 0.03 ETH. In reality, I expect that it would be \frac{1}{10}

–\frac{1}{100}

of that.

Extension using Proof of Replication

A version of this that forces large stakers to keep several copies instead of just one can be constructed by using aProof of Replication instead of the Proof of Custody. This would also somewhat mitigate the centralization risk, but only if storage costs would dominate the bandwidth costs, which currently does not appear to be the case. It would also ensure that more copies of the state exist, but since those copies aren’t necessarily independent (they could be all on the same hard drive), it isn’t clear if that this actually gives any additional security. So a Proof of Replication would only be worthwhile if it can be achieved very cheaply.

Appendix: Current custody construction

The numbers for the computation above are based on the Proof of Custody construction that is based on the “Hash then PRF” construction using a Polynomial UHF and then the Legendre PRF. Concretely, for a key K

and 256 bit data blocks interpreted as integers d_0, \ldots, d_{n-1}

, the Proof of Custody bit is defined as

b = \ell_p\left(\sum_{i=0}^{n-1} K^i d_i + K^n\right)

where

\ell_p(x) = \left\lceil\frac{1}{2}\left(1 + \left(\frac{x}{p}\right)\right)\right\rceil

is the Legendre symbol normalized to a bit and p

is a 256-bit prime. Note that only a single Legendre symbol evaluation is needed, meaning that almost all the computation is computing a polynomial evaluation. This can be done using two multiplication per term in Horner’s scheme. While I don’t

have concrete benchmarks for this yet, my estimate is that one 256 bit multiplication modulo a well-chosen prime (we have almost no restrictions) can be done in ca. 30 ns on latest generation Intel processors, and optimistically in as little as 10 ns. The computation time estimates above are using conservative 30 ns. This is much faster than, say, a sha256 evaluation.