Lessons from Cosmos Hub

If you observed how Cosmos Hub performed their upgrade from version 1 to version 2, and then from version 2 to version 3, you would know that it was essentially done via re-starting the blockchain from a fresh genesis. Upon the upgrade, node operators had to shut down their nodes, then generate a snapshot of the state of Cosmos Hub, and then effectively use that snapshot as a genesis for launching a new blockchain, from block 1.

Now anyone who wants to join Cosmos Hub, needs to acquire genesis for CosmosHub-3, download all the blocks of CosmosHub-3 (but not CosmosHub-1 or CosmosHub-2) and replay them.

Can we “relaunch” Ethereum 1?

Lets look at a hypothetical application of such method in Ethereum, where we have a really large blockchain (150-160Gb), as well as fairly large state (40-100Gb depending on how you store it). The obvious gain of such “relaunch” would be that the new joiner nodes will need to start with the 40Gb genesis state and not with 150 Gb worth of blocks. But downloading 40 Gb genesis is still not a great experience.

State in Ethereum is implicit, only its merkle root hash is explicit

Let us now assume that we can make these 40 Gb implicitly stored “off-chain” and only the root hash being used as the genesis. And let us also start with the empty state. How do we get transactions access part of the implicit state?

Bear in mind that even now the 40 Gb is implicit and the exact way of acquiring it is an implementation detail. You can run through all 10 million blocks to compute it, or you can download its snapshot via fast sync, or warp sync, or you can even copy it from someone’s external disk and then re-verify. Although the state is implicit, we assume that the block composers (mining pools usually) have access to that implicit state and will always be able to process all transactions. What we are removing is the assumption that all other validating nodes have the access to that implicit state to check that the transactions in the block are valid and the state root hash presented in the block header matches up with the result of the execution of that block.

Isn’t it Stateless Ethereum?

If you followed Stateless Ethereum at all, you may recognise that this is exactly what we are trying to do there - leave the assumption that block composers have access to the implicit state, but remove the assumption that all validating nodes have the same access. We propose to do it by obligating the block composers to add extra proofs in the block, and we call these proofs “block witness”.

Proofs in the blocks vs proofs in the transactions?

When people first learn about this, they assume that these extra proofs are indeed provided by the transaction senders, and become part of transaction payload, but we have to explain to them that no, it is block composers’ job. But later we discovered that transactions will have to include some extra proof. Namely they will need to prove that the sending address has enough ETH to purchase the gas for this transaction, and also for all other transactions from this account but with lower nonces. They might also need to prove the nonce of the sending account, so that the node can figure out if there are any nonce gaps and therefore potential DDOS attack via a storm of non-viable transactions. More stringent checks can be done, but knowing the ETH balance and the nonce of the sending account is the necessary (but perhaps not sufficient) info for most of the DDOS-protection heuristics.

Critique of proofs in the transactions

Let us now imagine that we do, in fact, want transaction senders to include proofs for all the pieces of state that the transaction touches. This would be nice because it would simplify our work on trying to charge extra gas for witnesses. The main critique of this is usually to do with Dynamic State Access (DSA, as opposed to Static State Access - SSA). If transaction goes to a particularly complex smart contract, specifically, if it involves lots of nested calls to other contracts, it might be that the items of state that this transaction is going to touch, is hard to pre-calculate. And it can even be used to “trap” a user by front-running their transactions and making them fail due to insufficient proofs.

ReGenesis provides mitigation

The concern about DSA can not be easily solved completely, but it can be sufficiently mitigated to the point that users will very rarely see the inconvenience, and never get permanently “trapped” into never being able to achieve their desired state transition. The mitigation relies on the extra rule that any proof provided with a transaction, which checks out against the state root (but is not necessarily sufficient for the transaction to be successful), becomes part of the implicit state. Therefore, repeated attempts by the user to execute the transaction will keep growing the implicit state, and eventually, it will succeed. Any attacker that tries to “trap” the user, will have to come up with more sophisticated ways to redirect the transaction’s state access outside of the implicit state, and eventually, the attacker will fail.

As the implicit state grows from nothing (just after the “relaunching”) to include more and more of the actively accessed state, the proofs that transactions need to provide will shrink. After a while, most transactions won’t even need to attach any proofs, only the ones that touch some very old and “dusty” parts of the state.

We can keep doing it

I call this “relaunch” reGenesis, and it can be done regularly to ease the burden on the non-mining nodes. It also represents a less dramatic version of Stateless Ethereum.

Doing ReGenesis repetitively would simplify the architecture of Ethereum client implementations. It can mostly obviate the need for more advanced snapshot sync algorithms. If we perform ReGenesis every 1m blocks (which is roughly 6 months), then the state snapshots as well as blockchain files can be made available on BitTorrent, Swarm, IPFS. We cannot do it now because the state changes every 15 seconds, not every 6 months. If client implementations can cope with replaying 6 months worth of blocks, then we do not need very sophisticated snapshot algorithms. Consequently, complexity of Ethereum implementations will go down.

Downsides

I have not yet explored lots of them, but there are three that I already see:

1. Users may need to have access to the full implicit state to create transactions. I actually see this as a fair compromise
2. Users may need to repeat transactions (due to Dynamic State Access) until they eventually achieve the desired state transition
3. Some roll-up techniques (that utilise the blockchain data for data availability) may break if the network effectively “archives” all the blocks prior to the ReGenesis

Please consider the discussion open!

UPDATE: here is a link to the “ReGenesis Plan” document:https://ledgerwatch.github.io/regenesis_plan.html