On one of the Stateless Ethereum calls we had a great question from@cdetrio about the relation between the transaction propagation and witnesses. The question is this: If we do not assume that Ethereum nodes hold any state, how would they verify transactions before propagating them? Here is the context: when a transaction arrives to an Ethereum node from a peer, this node needs to perform some basic verifications on it before sending it further:

1. Extract the sending account from the transaction (sender

), then read the ETH balance of sender

from the current state (sender.balance

), and verify that the sender has enough ETH to at least pay for gas: sender.balance >= tx.limit * tx.gasprice

.

1. Extract the sending account from the transaction (sender

), then read the nonce of the sender

from the current state (sender.nonce

), as well as the nonces of other transactions from sender

currently in the transaction pool. Verify that there is no nonce gaps, i.e. there is no such number nonce_gap

, such that sender.nonce < nonce_gap < tx.nonce

, and there are no other transaction in the transaction pool (tx'

), such that tx'.nonce = nonce_gap

.

These basic checks stop the node from acting as an amplification vector for basic spamming attack that can take down critical nodes like mining pools. Looking at the checks, one should be easily able to come up with such attacks:

1. Unfunded transactions - create a lot of transactions coming from lots of generated accounts with no ETH on them.
2. Funded transactions that will never be valid. Take an account with some ETH in it, and create a lot of transaction with very high gas price, but with a nonce gap, which means that none of the created transactions will be able to get into the block, and be stuck in the transaction pool.

There can be refinements to these verifications, for example, if there are multiple transactions from the same sender

, without nonce gaps, we can require that sender.balance >= tx1.limit*tx1.gasprice + tx2.limit*tx2.gasprice + ...

, in other words, sender

has enough ETH to pay for all outstanding transactions. Other refinement could be based on a minimum viable gas price.

One common theme in all these verifications is that they require access to one item in the current state, to read sender.balance

and sender.nonce

. And if the node is “pure” stateless, where will it take this information from? That was the question introduced on the call

.

On the same call, the suggested solution was to require transactions to carry the data about their sender.balance

(as an extra field, perhaps), and having sender.nonce

implied (meaning for the tx to be mineable, there needs to be equality tx.nonce == sender.nonce

). And, additionally, the transaction would carry the merkle proof, or witness, certifying the correctness of sender.balance

and sender.nonce

. Such merkle proof might be a bit large (if we assume 8 levels in the state trie saturated, and 15 sibling hashes on each level, the proof size is around 8 * 15 * 32 = 3840

- not the end of the world, but still large, and adding 61k gas to the transaction cost, if we assume 16 gas per byte).

Switch to binary trees would reduce the such of such merkle proof to around 32*32 = 1024

(or 16k gas).

Another interesting detail about this merkle proof inside a transaction is that the proof would have been constructed based on some state root which was in the past. And, therefore, to verify this proof, the Ethereum node will need access to the relevant segment of the header chain.

ReGenesis is the middle ground between hypothetical “pure” stateless nodes and fully stateful nodes we have now. And here is the main implication on the sender

merkle proofs. Between two “resets” (ReGenesis events) of the state, any sender

will only need to include the merkle proof of its balance and nonce until its first transaction made it into a block. This is because under the rules of ReGenesis, any state data attached to the transactions with correct proof, gets added to the “active state”, and this active state survives until the next ReGenesis event.