Background

The UTXO data model is more conducive for parallelism compared to the accounts data model, but still suffers from the issue that in order to full validate a transaction, state lookups must be performed. Without state rent, the ever-growing chain state will make transactions more and more expensive to validate. Even with state rent, this is a problem.

Suggestions to alleviate this issue boil down tostateless clients, which use some sort of accumulator

over the state. Accumulators are a fixed-size representation of a set, from which you can efficiently insert and remove elements, and verify proofs of membership (accumulators may have other properties, or may not have all these properties, but these are the important ones). Transactions then need a proof of inclusion that they are spending TXOs that are unspent, which can be verified against the accumulator. Thus nodes need only store a fixed-size accumulator rather than an enormous amount of state. Unfortunately, accumulators with all the desirable properties without a trusted setup don't exist at this time.

Of note is that storage

of the state is less of an issue than constant lookups into the state, which must be done for each transaction. This is especially problematic when the state is modified, as the data structure must be locked, preventing parallelism.

Proposal

Objective: We want a way to completely validate transactions without ever doing state lookups. Ideally we also want to completely validate blocks without ever doing state lookups. The only time state lookups should be performed is when a state transition is applied (i.e.

, when a block is fully validated and ready to be executed against the state).

The key data structure used in this proposal is theSparse Merkle Tree (SMT), a Merkle accumulator. Efficiently implemented, it has O(log(N))

average-case insertion and removal costs and proof of inclusion size, where N

is the number of state elements (not the number of leaves).

Nodes store the chain state in a SMT, which is committed to in the block header. Transactions specify a block hash (and implicitly or explicitly height h

) and an inclusion proof against the state at h

. Transactions must be included within D

blocks (a system parameter) or they are considered invalid.

State deltas

are constructed for each block. The delta of a block is simply the set of UTXOs that are removed and the set of UTXOs that are added to the chain state for that block. This can be done without state lookups.

Transactions in a block are validated with the following steps:

1. If the block that includes this transaction is past the timeout D

, the transaction is invalid and so is the block.

1. For each input, if the input has an incorrect proof of inclusion, then the transaction is invalid. This can be done by only looking at block headers from at most D

blocks ago.

1. For each input, check that the input has not been spent between the proof of inclusion block and the tip. This can be done by only looking at state deltas from at most D

blocks ago.

1. The rest of the transaction (sum of outputs <= sum of inputs, etc.) can be validated in isolation, so long asUTXO IDs are constructed using all the necessary information, such as amount and owner.

Once a block is fully validated, its state transition (the state delta for that block) can be applied. This is the only time chain state is touched. As such, all other operations can be done independently and in parallel. State deltas can be saved to disk as the ordered UTXOs (leaves of the SMT) and re-constructed as needed.

Proof of correctness by induction is left as an exercise to the reader.

The only other time that the chain state is accessed is when generating inclusion proofs. In practice, however, the set of nodes that will generate inclusion proofs for wallets (Infura, Electrum servers, at-home nodes) and the set of nodes that produce blocks is disjoint, so this won't have any effect on block producing node performance or non-mining full node performance.

Cost

In-memory, state deltas for the past D

blocks must be kept. In the UTXO data model, the state delta for a block is basically at most the size of the block itself, so this is equivalent to storing D

blocks in-memory. In practice this is easy to do.

Inclusion proofs for each input will increases the size of transactions by a small factorProofs can be batched however, and transactions that don't include optimal proofs can be rejected, so in practice this isn't an issue.

On-disk, state deltas may be saved for each block. Since state deltas are no bigger than blocks themselves, this means disk storage will at most double. As archival storage of history is not an issue, neither is this. Alternatively, since the state delta for a block can be re-constructed using only that block, they don't have to be stored at all and can be re-constructed on the fly if needed.

Choosing Parameters

The timeout for transactions D

can just be the usual number of confirmations needed to consider a block finalized (i.e.

, D=15

for Ethereum and D=6

for Bitcoin), possibly with a factor of safety. Nodes should have safeguards against automatic re-orgs longer than D

anyways, so in practice state deltas will never have to be re-constructed from disk.

Comparison to Other Proposals

This proposal is similar to the stateless client proposalhere, which is for an accounts data model chain, though it is much simpler and actually implementable in practice.

This proposal bears some similarity toUtreexo, though unlike Utreexo it is light-client friendly and incentive-compatible.