I think it's been difficult to follow some of the conversations about stateless ethereum, because there are so many different proposals and it's difficult to think about how they tie together. I hope that having a set of concrete named proposals will help with that. To that end, here's a Least Ambitious Proposal, a Little Baby Stateless Ethereum, Kittenless:

1. Stateless clients look just like full nodes today, except they do not download the state trie. In order to sync they must download the header chain and block bodies, as full nodes do today. (It's not really "stateless", it's "less state")

2. In order to support stateless clients a new version of the eth

protocol is launched, eth/66

. Most messages are the same, but it does not have GetNodeData

. State is fetched using a different mechanism, described below.

1. When miners create a block they also create a witness for that block. The witness specifies a subtree of the state from the previous block. It contains every part of the state tree which is read from or written to by the transactions in the block. Block headers include an additional field, the hash of the witness.

2. There's a new block propagation rule, peers will only propagate blocks which are accompanied by a witness. (In the eth/66

subprotocol the NewBlock

message includes a field which contains the witness)

1. Transactions look almost exactly as they do today. However, nodes do not accept themunless they're accompanied by a witness proving the sending account. An affordance for this is made in eth/66

.

1. Somehow, we migrate the account trie from the patricia tree to a binary trie. This will reduce witness sizes.

2. Somehow, we merkelize contract bytecodes. Witnesses only include the parts of the bytecode which are actually touched. This also reduces witness sizes.

3. We add a gas cost. Transactions must pay an amount proportional to the number of accounts and storage items and bytes of code they touch. This gas cost is set such that the witnesses are bounded to a reasonable size, at most 1MB?

4. There's a new subprotocol, state/1

, which nodes can use to fetch chunks of state (Geth calls these tiles) from each other. Both full nodes and stateless clients use this protocol to fetch state, stateless clients just fetch less state.

1. In the state/1

subprotocol each node advertises a list of chunks which they store, and the protocol provides a method of looking up which nodes store a given chunk. Nodes can request chunks from each other. All chunks are sent along with proofs which can be used to verify them. As blocks are added to the chain each node updates the chunks they store locally. The witness for each block is enough to update chunks. However, nodes must be careful to properly handle reorgs. A witness is not sufficient to undo

changes to a chunk, when updating chunks the node must store some additional data it can later use to undo changes, if necessary.

1. There is a social convention that nodes which have fetched a chunk must advertise it until they've served it to at least one other node. Full nodes will automatically follow this rule, they'll always serve the entire state. Stateless clients might provide some kind of feedback to the user: "The current ratio of uploads to downloads is below 1, please continue running the node until the ratio is above 1". Stateless clients will not support any features likely to break this invariant. Most notably:

2. eth_call

does not work unless you are running a full node, or your stateless client already has access to all the state necessary to service the request. Stateless clients do not attempt to fetch missing state from the network. eth_call

is a fundamentally expensive operation and if you want to do it you should pay someone to do it for you.

1. There's a second social convention: When fetching state, nodes prioritize fetching state from stateless clients (which which aren't advertising "I have everything"). This reduces the load on the full nodes, who are only asked to serve state which no stateless clients are serving.

You can see that even though I've called this "Least Ambitious" some of these bullets are still rather ambitious. Am I missing anything that would prevent this from working? Is there a way to simplify it even further?