This post gives a high level and informal construction of a UTXO blockchain where node resources scale sublinearly in all respects (storage, disk IO, computation, and bandwidth). The scheme can be applied more generally to an Ethereum-style blockchain but bandwidth would remain a bottleneck for full scalability (bandwidth scales linearly with public state diffs). A key ingredient are (non Merkle-) cryptographic accumulators. These accumulators are promising for the blockchain space as a whole because of their applicability at both the consensus layer and the application layer.

I need to post a disclaimer that I am not a cryptography expert and that everything below should be taken with a huge grain of salt. Having said that, it does seem like the approach below is a possible path towards finding the blockchain scalability holy grail. Thanks to Vitalik for challenging my ideas and encouraging me to make this write-up.

Background on accumulators

Merkle trees fit in a wider class of cryptographic accumulators that are space and time efficient data structures to test for set membership. Non-Merkle accumulators tend to fall into two classes: RSA accumulators and elliptic curve accumulators. There has been a fair amount of academic study of non-Merkle accumulators and they are used in several practical applications outside of the blockchain space. The discussion below focuses on RSA accumulators to give a bit of intuition, but an elliptic curve accumulator may well be more appropriate.

RSA accumulators are based on the one-way RSA function a -> g^a mod N

for a suitably chosen N

. The set {a_1, ..., a_n}

is compactly represented by the accumulator A = g^(a_1 * ... * a_n)

. The witness w

for an element a_i

is built like A

but skipping the a_i

exponent, and checking the witness is done by checking that w^a_i

equals A

. Adding elements b_1, ..., b_m

to the accumulator is done by exponentiating A

by the “update” b_1 * ... * b_m

, and likewise for the witness w

.

Notice that RSA accumulators are constant size (a single group element) and witness updates are cleanly “segregated” from the other set elements. Compare this to Merkle trees which are linear in size to the number of leaves, and where an update to one element will modify internal tree nodes which will “corrupt” Merkle paths (witnesses) for other elements. Notice also that updates to RSA accumulators are batchable, whereas Merkle tree updates are not batchable and take logarithmic time for each element, impeding sublinearity.

Non-Merkle accumulators can have all sorts of nice properties. They can be “dynamic”, meaning they accept both additions and deletions to the tracked set, which is something we need. They can be “universal”, where nonmembership can be proved in addition to membership. They can have optimal space/time complexities. They can be zero-knowledge. Having said that, it’s not all rosy and every scheme has its own trade-offs. For example, some constructions require a trap-door (like Zcash). The perfect accumulator for our needs may not be readily available in the literature (where the decentralised and fully open context is rarely assumed). The nitty-gritty detail is beyond the scope of this post

Construction

The UTXO set is kept track of using a non-Merkle constant-sized dynamic accumulator with segregated, efficient and batchable updates for both the accumulator and witnesses. In terms of maintaining state, fully validating nodes and mining nodes only have to keep track of the header chain (which contains batched and constant-sized accumulator updates). Everything else (UTXO set, transactions, blocks) is prunable. We are in the stateless client paradigm where transactions provide state and witnesses on a need-to-have basis, thereby relieving nodes of storing state (in our case, the UTXO set). Neither users nor nodes have to do the costly work of maintaining a Merkle tree. At this point we have sublinear storage and disk IO.

We next make use of SNARKs/STARKs to achieve sublinearity of computation and bandwidth. Instead of explicitly

disclosing transaction data (UTXOs, amounts, signatures, witnesses) we allow for transactions to contain only the final accumulator update (batched across UTXOs), as well as a succinct proof that the accumulator is valid. Here validity includes:

- Knowledge of witnesses for the UTXOs to spend
- Knowledge of valid signatures for the UTXOs to spend
- Money conservation (the sum of the amounts in the UTXOs to spend is less than the sum in the new UTXOs)

Notice that a single transaction can spend an arbitrary number of UTXOs (from different signers/owners) and create an arbitrary number of new UTXOs without explicitly communicating them to nodes. Instead, all the UTXOs are subsumed in the accumulator and the updates, and size (for bandwidth) and computation are both sublinear. Indeed, the transaction consists of the accumulator update and a SNARK/STARK, both of which sublinear (generally constant or close-to-constant) in size and in time (for the SNARK verification and accumulator update).

Data availability

The ideas above came as I was attempting to solvedata availability. The above scheme doesn't solve data availability, but side-steps significant chunks of it. In particular:

- Private state (e.g. Bitcoin UTXOs or Ethereum accounts) that only ever needs to be known by a single party does not need to be publicly available. My guess is that a large portion (90%?) of Ethereum's current ~10GB state falls in this category, or can be made to fall under this category by slightly tweaking the state management of individual contracts.
- Transaction data does not need to be publicly available because we use SNARKs/STARKs instead of fraud proofs.

In the context of an Ethereum-style blockchain, the only data that needs to be publicly available is state that at least two (non trusting) parties may require to make transactions. Think for example of a public order book where updates to the order book need to be known to the wider public to make trades. For such applications, the transaction (including the SNARK/STARK) needs to be extended to include the state diff. Notice that the state diff can be gossiped to the validator nodes and then immediately dropped after being checked against the SNARK/STARK. This leaves bandwidth as the final piece of the puzzle for full Ethereum scalability.

I am cautiously optimistic that bandwidth scalability can be solved convincingly. But even if not, in practical terms bandwidth is possibly the least pressing bottleneck. The capacity of a single node today is enough to support applications with significant amounts (think tens of gigabytes per day) of public state diffs, and Nielsen's law (50% bandwidth capacity increase per year) is showing no sign of stopping.