My initial attempts to design gossip for the state network were heavily informed by my previous designs for a new transaction gossip network. In my previous post, I cover a design that allows nodes to participate in gossip without requiring them to process the full mempool of transactions.

Scalable Transaction Gossip [

Eth1.x Research

](/c/eth1x-research/37)

I'm imagining these as boring merkle proofs against the main account trie.

At a high level, our problem statement for transaction gossip is this (ignoring DOS/security requirements):

- transactions originate from all parts of the network.
- there are participants in the network that are inherently interested in receiving the full mempool (miners, front runners)
- there are participants in the network without the resources to process the full mempool (light nodes)

The solution proposed for transaction gossip uses a distance metric that we refer to as radius

which nodes can adjust up and down to raise/lower how much of the mempool they end up having to process. Nodes use a simple set of rules to manage their set of connected peers which results in a network togology. We refer to the "center" of the network as the nodes that have the largest radius values, and the "edge" of the network being the nodes that have the smallest radius values.

This solution works because of two main things:

First… we expect radius

values to vary widely and

we expect them to be relatively large. The wide variance comes from there being actors with inherent motivation to have a "full" radius, as well as actors that have "large" radius values. These nodes tie the edges of the network together.

Second… we expect radius values in general to be "large" with respect to the keyspace. According to Peter's recent post on the mempool, geth nodes default to 4k pending transactions, and there are as many as 40k-400k total transactions available at any given time. A lightweight node won't be expected to process 4k transactions, but they very well may easily process 5% of those. For this reason we expect radius values to typically fall into the range of 1%-100% of the total keyspace.

Applying the same design to state gossip (doesn't work)

My initial designs tried to apply this same design to gossip for the state network. This approach fails due to a few reasons.

First… radius values in the state network are going to have significantly lower variance and there is no expectation that there will be participants in the network operating at the "full" radius. The result is that we lose the "center" of our network which serves a critical role in connecting the edges.

Second… radius values are going to be extremely small. At 200GB of state, with the average node offering 100MB of storage, and a replication factor of 10, we can calculate a network size of 20k nodes. We can then expect each individual node to be storing 0.002% (1 / 20,000th) of the data.

These two changes fundamentally shift the network topology making the original tx gossip network designs non-functional.

Different goals than transaction gossip

If you recall, one of the goals for transaction gossip is for transactions to make their way to the "center" of the network where the miners are. The nodes at the edge of the network don't actually care much about seeing all of the transactions, or even seeing "any" transactions. They mostly just care about being able to broadcast their own transactions and have them reliably end up in a block.

In the state network, we have no center, and unlike tx gossip, our data flow is inverted. Our goals are instead to get data out to the edges of the network for storage.

And, unlike transaction gossip, where the messages originate from all over the network, in the state network, we expect new data to only originate from a small number of benevolent bridge nodes who are producing proofs and pushing them into the state network.

Relaying results in DOS issues and unatributable faults

One direction that I tinkered with was having nodes act as relays.

We expect every node to be interested in 0.002% of the data coming into the network which is a very small amount. I believe my conclusion holds under a number of different network models, but the easy one is to just model the connections between gossip nodes off of each node's routing table in the DHT. This gives us a network requiring log(n)

hops needed to get any piece of data to the nodes which need it in the network.

The problem that arises here is that if a node forwards data that turns out to not be interesting to any other node, but it takes more than one hop, the becomes an amplification vector. A malicious node can amplify a DOS attack by simply pushing unnecessary data over the gossip network.

Scaling things wayyyyyyy back to a dumb solution

I'm currently leaning heavily towards a "dumb" solution that doesn't attempt to solve this at the network level.

- A "small" number of benevolent state provider nodes will be generated proofs for new state data at each block.
- Each proof is expected to have around ~2k trie nodes. A subset of these nodes are new or updated data. Only this subset needs to be pushed into the network.

Given that we know individual nodes are expected to only care about 0.002% of the data, this means that on average, we can expect very little overlap in what nodes are interested in what data from each block. Basically, if a block has 2k new pieces of data, we can expect to need to push each of those pieces of data to completely different nodes. This means that for a single benevolent state provider to distribute the proof data for a new block within the block time would require them to be able to push out 2k independent proofs to 2k independent nodes every 15 seconds. This might

be possible, but it isn't going to be trivial, and if the proof sizes grow or network latencies go up a little, we could easily end up not being able to send out full proof data within the block time.

Luckily, we don't have to depend on only a single state provider. It's reasonable for us to expect there to be a small but not tiny number of state providers pushing in proof data. Under this model, we can design a system that has the emergent property of evenly distributing the load among the different state providers.

At each block, each state provider produces a proof. The provider will then sort all of the elements of that proof by their proximity to their own node_id

. Starting with the items that are closest, they can then lookup the node(s) that are interested in the data and gossip it into that part of the network. Under this model, the load is evenly distributed among the different state providers, and as they reach items that are further and further away from their own node_id

they should begin encountering decreased interest since other providers that are closer to that data will have already gossiped those items.

Ways that this could be improved/extended/optimized

We might

be able to optimize this a bit.

The structure of our network requires not only storing the leaves, but also all of the intermediate trie data. This means that splitting the block proof up into individual proofs for each of the leaves and nodes will result in a lot

of overlap between the various proofs. Examining the case for a leaf node, the proof for the leaf node also contains full proofs for all of the intermediate trie data.

A node that wishes to store a leaf node has an intrinsic interest in the intermediate nodes needed to get to the leaf also being available in the network. Without them, nobody is likely to ever request their leaf data as it won't be discoverable by walking down the trie since the intermediate data will not be present. We may be able to take advantage of this to spread out the responsibility for spreading data more widely throughout the network.

- benevolent state providers only

gossip the leaf data proofs.

- upon receiving a proof over gossip for content a node wishes to store, it will also seek out and push the "parent" proofs for the intermediate trie data.

This "recursive" process results in the benevolent state providers only needing to push the leaf data into the network, and delegating responsibility for spreading the intermediate trie data to those nodes who are interested in the leaf data. They

intern, push the proofs for the intermediate trie data one level up from their leaf data, and so on, until eventually, all of the nodes end up pushing the final state root into place.