Here I propose a new approach to syncing the Ethereum state data that I'm tentatively naming: "Merry-Go-Round Sync", or MGR sync. MGR aims to provide a new mechanism for syncing the Ethereum state that exhibits bittorrent style swarming behavior, and mitigates against other uses of the state syncing network primatives.

At a high level, MGR operates by enumerating the full state in a predetermined order and gossiping this data among the clients which are actively syncing. For a client to fully sync it needs to "ride" one full rotation of the merry-go-round.

This is aimed at just being a rough write-up. It doesn't get into specifics in many places, mainly because there are no firm specifics at this stage, only the high level concept and rough ideas of how it would work.

State Enumeration

MGR requires a well defined way to enumerate the state data such that:

1. Minimize duplication: During a single revolution of the MGR there should be very little or near zero duplication of data.
2. Zero coordination: There should be no coordination required for clients to know what part of the state is being distributed at any given time.
3. Flexible with-respect-to state size: For a chain with a very small state, each revolution of the MGR should be very quick. For a chain with very large state, each revolution of the MGR would be longer.

No obvious solution has been posed that fits both of these criteria well. Here is my best idea thus far.

Use the block hash

In this model we would have some concept of a sync epoch. At each epoch boundary, the block hash of the boundary block would determine what location in the trie enumeration should begin at. The state data would be enumerated starting at that location in the trie and working outwards until the next epoch boundary.

Mitigating against re-sending duplicate data when there are two epochs with block hashes that are close together in the trie would require something like tracking and broadcasting what ranges of the trie each client has fully synced.

This approach does not require any coordination for determining where to enumerate the trie, but it does require some level of coordination for clients to keep track of which of their peers has which data. This might be acceptable.

This approach seems to scale well with different state sizes.

One point of note is that under this model there may not be a clear definition of "one revolution" because two clients may be able to ingest data at different rates, meaning that they would take different amounts of time to fully enumerate the trie since the slower client would make less progress during each epoch. This might actually be a good thing.

Networking

This likely belongs in a separate protocol than the existing DevP2P ETH

. For the sake of simplicity we'll assume this is a new DevP2P protocol MGR

. Currently we would require the benevolent participation of fully synced nodes.

We would need something like the following commands.

Status

Allow clients to broadcast information to each other. This should probably contain forkid

as well as meta data about what parts of the state the client has already synced

StateData

A data packet that includes a provable chunk of the state. Assuming we use the witness spec, it may be that the "top" level of the tree is only sent around occasionally and that sub-trees are the primary unit of distribution in between re-broadcasting of the top levels of the tree

This message would be gossiped around the network…

Getting a full

picture of the state

One problem that arises with any state syncing approach is that the state is constantly changing which makes it difficult to get a full picture of the state within the garbage collection window that most clients use to discard old trie data.

Beam sync addresses this, however, for clients to rely on beam sync we will need witnesses propagating around the network.