Overview

This is an exploration of the various ideas and concepts around improving the time for rollup nodes to initially synchronise their state to the latest state of the rollup. Hereinafter, this improved process can be referred as “Snap Sync”-ing for rollups, drawing inspiration from the similarly named L1 process. This document outlines a general idea how snap syncing can work for rollups and dives deeper into the roles and needs of various rollup actors.

Snap synchronisation for rollup nodes is an important concept due to the promise of the rollups to achieve a throughput orders of magnitude higher than L1. While the rollups promise that anyone observing the data availability layer will have all the data to re-execute and synchronize the rollup, in practice this will quickly become almost prohibitively slow mechanism. This slowness can lead to centralisation through consolidation of the data into few entities specialising in continuously syncing the L1.

Snap synchronisation aims to enable nodes to synchronise by means of getting pre-processed information from other actors of the system, but duly verifying it themselves.

Conceptual Idea

Below is a conceptual idea with 4 steps that can be used for snap synchronisation for rollups. It outlines how a fresh rollup node can synchronise to the tip of the rollup.

1. The syncing node notes a recent (ideally not the last) finalised block and its blockhash/state root from the L1 smart contract of the rollup.
2. The node connects to one or multiple peers and downloads the state at the noted finalised block.
3. The node verifies the blockhash/state root against the one noted from L1 in step 1.
4. The node continues syncing the subsequent finalised (if applicable) and virtual state from L1.

Deep dive

The concept above is deceptively simple. As per usual though, the devil is in the details. The rollup specific peculiarities of snap syncing start to appear when you start considering the goals and incentives of the various actors in the rollup system. Below are several important considerations for the Snap Syncing design that provide more input in order to suggest a more complete conceptual solution.

The role of follower nodes

The major actors in a rollup are the sequencers and the provers (Optimistic rollups can consider their validators such a provers).

Sequencers job is to get user transactions (somehow) and sequence them into L1. They get paid for the service of posting rollup (ideally valid) sequences. Provers take the data and produce proof of the validity of a given sequence. The provers get paid for submitting valid proofs in L1.

Ultimately, both of these actors incentives are to be paid for their specific actions. Sequencers get paid for sequencing L2 state modifying transactions. In order for the sequencers to do their job, they only need the recent virtual state and some version of a mempool (private or public). They do not need to be concerned (read store) at all with past verified or not state.

Provers get paid for the production of proofs. They only need the sequence information from the L1 DA layer. They too do not need to be concerned (read store), with the past state.

This leaves quite a huge vacuum for the end-users - there is no actor whose job is to serve RPC requests for reading current or historical state. While it is common for rollup designers to think about the follower nodes as “dumbed down” version of the sequencers, actually, one can now argue, that they are a completely separate actor with a separate goal - to serve the users of the network.

Reasoning from the L1 nodes, it is your own “Full node” that you should be asking for state reads. One can reason that the equivalent of “your own full node” is “your own follower node. It is follower nodes that need to save some or all historical state of the rollup, in order to be able to serve (your) rpc request for historical data. Some very very common historical operations are - “Give me the emitted events from contract X”, “Give me the transaction receipt for my transaction hash”. No other actor currently needs to serve any of these.

Types of follower nodes & types of syncing

[

Snap Follower

1144×426 63.5 KB

](https://ethresear.ch/uploads/default/original/2X/0/024136fe5a8477f8e3f55ab849593ecd54bdbc46.jpeg)

The naive way to sync a follower node is to download the L2 sequences from the L1 DA layer and re-execute them until caught up with the tip. As stated before, this will quickly be impractically slow to be the only

mode of synchronisation. Therefore this is a possible but insufficient version of a follower node. Hereinafter, we will call nodes that are completely syncing from the L1 DA and storing the complete state an Archival Follower Node

.

On can reason that multiple use-cases would require nodes to sync quite faster than what the archival follower nodes will be able to offer. These are practical needs that can be triggered from the normal operation of the rollup, its crypto-economics, or purely to quickly seize an (MEV?) opportunity. Such nodes can connect to an archival follower node and use the “Conceptual Idea” outlined above to sync to the tip of the rollup.

Diving deeper, if a follower node only downloads the latest verified state from the archival follower and syncs to the tip forward (through re-executing virtual state), it will largely not meet the requirements that we have asked the follower nodes to fulfil. They will not be able to answer queries about historical state of any block (recent or not) prior to the last finalisation checkpoint. This outlines the need for the follower nodes to synchronise from a state prior to the last finalised state.

Two separate decisions can be made from the standpoint of follower node operators. These decisions are somewhat similar to the modes that L1 nodes employ for chain synchronisation.

First, a follower node might choose a fixed historical verified sequence - checkpoint - as a starting point and sync forward from there. Much like Geth in snap-syncing mode only stores the last 128 blocks, a Snap Follower Node

can sync starting from the last X (ex. 32) verified sequences. Furthermore it can choose to only keep at any time the state for the last X (ex. 32) form the verified tip. Snap follower nodes will be able to serve queries for recent state and blocks of the rollup. The X parameter needs to be chosen carefully in order to ensure the right balance between the historical data needs of the majority of the users and the storage requirements of the snap follower node.

Second, a follower node might choose to snap sync, in a similar manner to the Snap Follower Node, but also keep the state for multiple/all other verified sequences - checkpoints. We can call this node a Full Follower Node

. It is important to be noted that the full follower node does not need to re-execute the transactions between the old historical checkpoints, but only save the state for them.

This mechanism allows the full follower node to serve requests for much older state of the rollup. The full follower node can see the block information requested, calculate the previous synchronised and verified checkpoint, and only re-execute the transactions of the next sequences until the desired block. This approach is not as intensive on storage space like the Archival nodes, but is also not as efficient and quick to respond on queries, due to the need of data download and re-execution.

All three types of follower nodes have their own specific set of tradeoffs and are optimal for the various usecases of their users.

Sequencers and Provers need follower nodes too

As stated above the sequencers and provers have their own needs of state data. Similarly to users, they need to somehow synchronise to their desired point in the rollup. This, essentially, means that the sequencers and provers need a mechanism to quickly sync to the latest finalised state and continue on from there. Ironically, this actually means that they need some form of follower node in order to quickly sync. (Who is dumbed down sequencer now, huh?)

Both sequencer and provers looking to snap sync and start sequencing, can connect to a follower node, download the latest finalised state, verify it and continue syncing the virtual state.

Further Research Avenues

Diving deeper in the snap synchronisation mechanism, some practical issues arise. These need to be accounted for and require further research and ideation.

Update: Reusing existing synchronisation mechanisms in the L1 execution nodes can solve both of these.

Mechanism of Serialising State

In the text above we hide the complexity of the checkpoint data transfer behind “connecting and downloading the state”. Under the hood, however, this means that there needs to be a defined protocol for serialisation, partitioning and transfer of the state of the rollup at any given checkpoint.

Mechanism of downloading the state

While any node looking to snap-sync can connect to “a” follower, this poses a liveness and availability problem. Synchronising from multiple followers in parallel, in a “torrent”-like manner can speed up the snap sync process through parallelisation and enable further verification and validation of the downloaded data.

Conclusions

Snap synchronisation is paramount for mitigating centralisation vectors towards infrastructure providers. While on the surface the process of snap syncing is somewhat straightforward, when one considers the various incentives, roles and needs of the different actors in the network, some interesting nuances appear.

The process of synchronisation showcases how the (commonly misunderstood)

role of the follower nodes is a special and important for the rollup ecosystems, and needs to be carefully designed. Diving deeper into the follower nodes, various mode of synchronisation for follower nodes appear based on the needs of their operators.

Last but not least, the analysis above showcases how the follower nodes are important to the other actors of the rollups - sequencers and provers. This makes the follower nodes important for the robustness of the system as a whole.

RFC

Any comments on the analysis above are welcome. Any ideas, suggestions and/or contributions towards the “further research avenues” are welcome.