Building up on previous work from @barryWhiteHat about rollup diff compression for the Reddit's airdrop scaling use case, I've researched how can we further compress the data in practice for the airdrop use case, while using the Reddit's airdrop as an example. I've written a long text with the all the details of the work, and the following is TL;DR version of it:

The commitments sent from L2 to L1 are usually already highly compressed by the rollup system. We refer to this compression as compression on the state level. What we're focusing on is how can we further compress the commitment size for the airdrop use case or any similar use case, where the business logic of the application is sending large amounts of similar data (amounts) to large amounts of users (addresses). In other words we have explored how we can further compress the commitment size by compressing the data on the application level. Data of this form can be highly compressed using standard data compression techniques. Additionally we've explored the idea of achieving constant compression, where the result is independent of the size of the original airdrop data, by referencing previous rounds of airdrop distributions. As a case-study for this problem the Reddit community points airdrop data for the r/CryptoCurrency and r/FortNiteBR subreddits was used. It is an example for complex, multiple round airdrop distribution with a lot of data. By using batching and data grouping, as well as strategies involving run-length encoding compression, we've managed to achieve linear compression of 61.68% (r/FortNiteBR) and 50.89% (r/CryptoCurrency) relative to the size of the original dataset data. Compression that yields constant results relative to the input data is not feasible for the dataset we've explored currently, but long term it might be possible under probabilistic assumptions. Current findings are that by using data compression algorithms the commitment data for certain use cases can be compressed by large amounts which contributes to reduction in gas usage and overall cost reductions. There are still a lot of things unexplored for applying this and similar strategies such as: exploring more data-compression strategies and algorithms and exploring other use cases and applications eligible for this kind of compression.

The conclusion is that using batching and data compression strategies, as well as using RLE compression algorithms it is possible to decrease the app data size (and thus gas usage) drastically. Also, by relying on probabilistic assumptions in the long term compression that produces constant size results might be achievable.

I've also written a node.js library for running experiments using these algorithms which I need to refactor and publish and I also plan to run this in practice on Arbitrum's testnet.

You can read the full document here:

docs.google.com

[

](https://docs.google.com/document/d/16FIPeg_HiMzXkVzrVUJLE0ryN9uOS8V4m-U0YwKCVGQ/edit?usp=sharing)

Rollup diff compression - application level compression strategies to reduce the...

Rollup diff compression - application level compression strategies to reduce the L2 data footprint on L1 Abstract The high level overview of the problem we're trying to solve is reduction of the size of the commitments sent from L2 to L1 for the...

Any comments are highly appreciated.