Transaction Submission on Based Rollups

much thanks to mike and others for review

~tl;dr~

Based rollups take advantage of the sophisticated L1 MEV infrastructure to sequence L2 blocks. L2 proposers submit transactions to L1 that they retrieve from L2 users. The design of the system of transaction submission (from users to proposers) is a critical part of the rollup design. This post explores the trade-offs between different ways to submit transactions to the rollup.

Introduction

Based Rollups are more decentralized than a single-sequencer rollup, but there are other problems that need to be solved. One of these problems is the transaction submission system. The transaction submission system is the way that users send their transactions to the rollup and the way that rollups discover user's transactions. I have not seen any public discourse on transaction submission to Based rollups so I decided to write this brief post to highlight some concerns and hopefully start some discussion.

Transaction Submission Designs

Taiko, a Based Rollup, uses a public mempool controlled by a centralized RPC for the transaction submission. This design is simple and easy to implement, but it has some drawbacks. The centralized mempool is a single point of failure and a public mempool can lead to toxic MEV extraction, a problem that doesn't exist in centralized sequencer rollups.

Other transaction submission designs may include:

- Some sort of decentralized RPC and mempool. This would includeusing the existing L1 p2p network/mempool with a different message type.
- A private mempool that is only accessible to the rollup proposers. The mempool could be controlled by a centralized server or a decentralized network.
- Private orderflow where users submit transactions directly to competing proposers akin toFlashbots Protect
- With Based Preconfirmations preconfers may advertise their own API endpoints and offer some sort of sandwich protection (if a preconfirmed tx is not executed as preconfed) within some sort of enforcement mechanism (e.g. restaking or trust)

Concerns about a single RPC

A single centralized RPC server is a single point of failure. If the server goes down, the rollup will stop processing transactions even though proposers may still be able to produce blocks. This is a significant risk for the rollup's liveness.

A single centralized RPC server is also a single point of control. An operator of an RPC server has the power to censor transactions, just like centralized sequencers can. Although Based Rollups seem to be more censorship resistant, centralized mempools have the exact same censorship resistance properties as centralized sequencers

(without workarounds like private orderflow).

In the current Taiko rollup design, the centralized RPC is the only entity with write access to the mempool. This means that competing RPCs cannot exist without alternative means of transaction submission to proposers. This is the reason why a RPC server failure is a significant risk to the rollup's liveness.

Concerns about a public mempool

A public mempool (like the Ethereum mainnet mempool) is a gold mine for MEV searchers. Unsuspecting users may be front-run, sandwiched, or otherwise exploited by MEV searchers. Users are not prepared to deal with MEV on L2s because Non-Based Rollups do not have public mempools (and therefore no frontrunning or sandwiching).

This is a significant UX risk for the rollup.

One alternative to a public mempool is a private mempool. A private mempool is only accessible to the rollup proposers. Unfortunately, this system eliminates the permissionlessness of sequencing/proposing L2 blocks. As this is one of the key benefits of Based Rollups, such a design seems suboptimal.

Another alternative is a decentralized mempool. A decentralized mempool is a mempool that is controlled by some sort of decentralized network. The Ethereum mainnet mempool is a decentralized mempool. A decentralized mempool has the key property that anyone can submit transactions to it. A decentralized mempool is, by definition, public so it has the same MEV risks as a public mempool. Because of this, a decentralized mempool may be best alongside a seperate private orderflow system.

Private orderflow

Private orderflow is a system where users submit transactions directly to competing proposers. This system is implemented on Ethereum by services like Flashbots Protect. Private orderflow completely eliminates the MEV risks of a public mempool. Unfortunately (as mentioned in many other Ethereum research articles) it also may lead towards a more centralized ecosystem. This is because proposers with more resources can more easily collect transactions from users (e.g. in a payment for orderflow system). Private orderflow is also permissioned, which may be a significant drawback for a rollup. Competing proposers that are not favored by a popular private orderflow service will likely be competed out of the market.

Conclusion

The transaction submission system is a critical part of the rollup design.

The current Taiko rollup design uses a centralized RPC server to control the mempool. This design has significant risks to the rollup's liveness and UX. The rollup community should work towards a decentralized mempool or some other transaction submission system.

We have come a long way on 'Frontrunning the MEV crisis" on Mainnet, but the same problems will exist in current Based Rollup designs. I believe that instead of tolerating toxic MEV extraction, Based Rollups should actively works towards a better future.

A postscript on Based Preconfirmations

These same problems still exist with Based Preconfs because users (read: wallets)still need to send preconf requests somewhere.

If all rollups start to use Based Shared Sequencing, the problem of MEV extraction starts to diminish. It is still unclear to me how users can protect themselves from frontrunning in a Based Shared Sequencing model. Will this be a default feature?

As George Spasov from LimeChain wrote:

Is it even viable to have preconfirmers? Various practical issues can lead to a worse UX for the user.

Preconf relays may be the solution to this UX problem but also sound overcomplicated to me.

This is the step that still needs to be explored:

Step 1. User sending a transaction and request for preconfirmation

The flow starts with the User

signing a transaction with a preconfirmation request (PR

) to one or multiple preconfirmers

. Ideally this is a new transaction type carrying not only the EIP1559 base fee and tip, but also an additional preconfirmation tip (PR tip

). The rollup transition function should make sure that the user has enough balance to cover the base fee and tips.

– [RFC] Preconfirmations Flow Exploration

Please DM me if you want to discuss.