It's my first time posting a top level thread here, so I apologize if I in any way violate decorum. I've been thinking about this idea mostly in the context of an app-specific rollup for our lending protocol, but hope it may be generally relevant, and will appreciate any and all feedback.

TLDR

post only state root, no calldata

Detail

What if, instead of using Ethereum as a data availability layer by posting the full state as calldata, a rollup posted only a state root to mainnet? The main benefit is reducing the amount of data stored on Ethereum, and thus the cost to users transacting on L2. Even with EIP-4844, blobspace ain't free.

The main risk is a data withholding attack, where the proposer posts a valid state root, but withholds the full data from other rollup nodes, in order to monopolize future block production or take funds hostage. To prevent this, honest nodes must challenge any state update which no peer can provide data for. Arbitrum-style interactive fraud proofs can be used to force the proposer to disclose the full state on mainnet, but will still result in a failed challenge if the root was valid, and so it is important that the cost of challenge is low even in the case of failure.

If the cost of a failed challenge is low, it's possible to grief honest proposers by forcing them to pay for posting all the state data to mainnet to defend against the challenge, even though they correctly propagated the state data peer to peer. The cost of making the challenge must be proportional to the cost of defending to ensure it's not viable to attack honest proposers in this way.

At worst, if an attacker can spend $1 to cost an honest proposer $1, they can force that proposer to give up and let their block revert. Then, a new honest proposer can bid, and unless the attacker can repeat their attack for the the set of all potential honest proposers, which includes everyone with funds on the rollup, they can't cause permanent downtime. It's possible another term could be added, where the cost of challenge goes up when it has been too long since a valid block was finalized. In this way, it's easy to challenge a dishonest proposer, but impossible to halt state transitions for long.

More optimistically, if nodes propagate data peer to peer, they can decide on their own data backup and accessibility solutions, and users would be well served to store the data they need for their own state transitions locally. In an app-specific context, I've thought about encoding the rollup state quite differently than the EVM does to optimize for this. All state relevant to a given user account can be encoded into the same hash, and thus it can be much easier for a user to validate changes to their own account without knowing the global state (ie, confirm you received the amount of tokens you wanted in a swap, without needing to worry about specifically from where).

Conclusion

I'm curious to hear all thoughts, and also appreciate links to related work showing where I may be reinventing the wheel. Unlike a normal optimistic rollup, where it's easy to determine if the calldata submitted matches the mainnet state root, and if both are valid, it's impossible to know from the state root alone if an update is valid, hence the need for careful thought around the economics of the challenge period and griefing.