Since rollups are expected to generally run with few proposers than that of decentralized L1's such as Ethereum, we should consider methods to safeguard user transactions from the potential censorship by sequencers.

In this post, we will briefly explore two methods of forcing transactions, namely allowing users to either compel the sequencer to include their transactions or bypass the sequencer entirely, inserting the transaction themselves.

Furthermore, there should be some guarantees to the user, that he can do so in bounded time. It don't suffice that there might come a sequencer around at some time that will not censor him, as it then becomes a probability of inclusion and not a guarantee.

This mechanism becomes particularly relevant when we consider rollup-governance that has, or could potentially gain, some control over the collateral held in portals for bridged assets. In such scenarios, it is important that users can force an exit (or a 'ragequit') from the rollup more quickly than the rollup can upgrade. This would ensure that any user who disagrees with an upgrade retains the freedom to leave. (Image below related)

[

1920×1910 70.8 KB

](https://europe1.discourse-cdn.com/business20/uploads/aztec/original/1X/bfcf2988381024544e0262fe46fe957e9a0665e4.jpeg)

Beyond time guarantees, the knowledge, cost and interaction of performing the ragequit should be minimized, as it will otherwise become a burden too large for most users to bear. For example, if rage-quitting requires that the user spend resources worth 1000$ to leave, smaller users don't have an option and they survive at the will of the governance. Similarly, if significant technical skills are required, some users might not be able to exit timely simply due to lack of required skills and time to learn them.

In short, it makes sense for a user to rage-quit when:

- Liveness is hit because of the censoring sequencer
- Safety is hit because sequencer is censoring exits before an upgrade

Note on CRLists

I think they can be useful for a probabilistic improvement to transaction inclusion, but this proposal strive to propose a method that can give the user a guarantee that the transaction will be included. See references:

- Inclusion Lists
- https://notes.ethereum.org/@fradamt/forward-inclusion-lists

How can rage-quitting be supported?

Currently, almost all platforms out there has some upgradability mechanism that can upgrade instantly or more quickly than rage quit can be performed. See L2BEAT – The state of the layer two ecosystem for details.

As noted earlier, there is really two paths:

- Let users insert blocks themselves
- Let users force sequencer to include them

Below we will go over the two directions, and how it could work in the context of Aztec.

Users propose blocks to bypass censoring sequencers.

This idea is based of the method we have previously used for Aztec Connect, and letting users build entire blocks and force them through if needed.

In the Aztec Connect world, this was supported with an escape hatch

. The escape hatch was essentially a time-window that ocurred periodically where anyone could publish a rollup (no-one ever did it, but they could have done so). When the escape hatch was open, there was absolutely no requirements to who could push new blocks (as long as they had a valid proof).

This lack of access control meant that anyone censored could use the window to force an exit, but also meant that the it becomes a gas and compute auction between the parties that want to publish.

Recall that a block proof must build on the previous state s

. Meaning that any time a block is updated to s'

anyone in progress of proving on top of s

have to restart ther work on s'

. An issue that arise from this is that a sequencer who publishes a block just before the hatch opens, could prepare multiple blocks in parallel and publish them in short interval making it unlikely that anyone else can get anything through while the hatch is open. For later reference, lets call this “spamming the hatch”

.

Nevertheless, the approach also have some clear benefits, for one only the access control is handled differently between hatch or not, the circuits are entirely the same and need no extra logic to handle the force inclusion. Blocks need to satisfy the exact same rules as always. Also, if the censored entity have sufficient with compute power so they can get around the “spamming the hatch” attack or if the attack is not enacted, they can include their transaction even if ALL sequencers are against them. With variable size blocks, the workload for the censored entity can be reduced greatly to a level where the can probably prove it themselves.

- No heavy modification to the circuits
- Theoretical ability to force through even if no sequencer will process your transactions.
- Possible to aggregate a lot of censored transactions at once. In the case of malicious upgrade this can be used to help mass exits.
- Possibly require many resources by censored party to “win”, essentially probabilistic

. Unclear how to mitigate “spamming the hatch” attacks.

Visualization of sequencers preparing blocks and proofs in parallel to spam such that you cannot use the escape hatch until they have upgraded and stolen your money.

Sequencer are forced to include transaction

This idea is based off of what most other platforms utilise when ready. The idea is relatively straight forward, the average user do not have the resources or skills to propose a full block, and might not know how to find external markets for this. To handle the issue, we want to have a way that require minimal resources from the user, and is easy to go to.

The high level outline is simple:

[

8218×3676 951 KB

](https://europe1.discourse-cdn.com/business20/uploads/aztec/original/1X/b55989d7e7cfa46a271236505abc240eec38bebe.jpeg)

1. The user builds a transaction + proof on device
2. The user post the transaction + verify the proof on chain
3. The sequencers are forced to include the transaction in a block within some timelimit to progress the rollup, after the limit they can only propose blocks that include the censored tx, otherwise revert.
4. If there are still “pending forced transactions” at the time of an upgrade, these must be executed before it is possible to upgrade. Sequencers can still chose to censor rage-quits, but it will stall the entire rollup without ability to upgrade the contract as long as censoring keeps on.
5. If there are still “pending forced transactions” at the time of an upgrade, these must be executed before it is possible to upgrade. Sequencers can still chose to censor rage-quits, but it will stall the entire rollup without ability to upgrade the contract as long as censoring keeps on.

However, there is a few notes and caveats that we need to address for this to work.

First, we are verifying the proof when it is posted on-chain for a relatively simple reason, dealing with reverting private kernels is not fun, so this check ensures that the proof at least is valid.

Second, while the proof can be valid, it might not be based off of a state that exists in our historic roots tree. When this is the case, it gives us a weird scenario, where it is not clear to the rollup circuits whether the user is lying or the sequencer is!

To work around that and ensure that the user is using a valid state, we can store the historic tree root at every block in a mapping mapping(uint256 blockNumber => bytes32 blockHash)

and then check that the public input used by the transaction is in this set. For more information about historic data tree consult tracking issue 849.

Third, even if the proof is valid on a valid historic state, the transaction could fail if it is emitting a nullifier that is already in the set, or doing a public function call that is reverting. While “reverting because of public execution” is also possible for “normal” transaction, the duplicate nullifier cases would normally not be included in the block.

We need to support an edge case, where a transaction with a valid proof that double-spend nullifiers can be included BUT

with no effect on the state!

This means that it will pay no fee! For forced transactions, these are paying a much higher fee for inclusion on L1 and only if sequencers are censoring them, so it seems fair that the sequencer forego some fee when censoring.

Non-state altering transactions

The edgecase for duplicate nullifiers might work fine for non-forced txs as well, it is essentially the sequencer burning gas to blast data on chain, for normal transactions he would simply not include it.

The benefit of this approach is that it is much easier for the user to force his transactions through, he builds the proof as he normally does and then he goes to the L1 contract to publish it instead. Publishing it on L1 might be much more expensive that if he had not been censored, but hopefully not completely mindblowing

. The user don’t have to battle against sequencers and sophisticated users to include his own block in the escape hatch window.

We are not sure about the exact cost atm, if we are setting up a special contract to support this, there might a wrapped version of the kernel proof that is using a proof system that is cheaper to verify on chain.

A downside of the method, is that sequencers can completely stall the rollup to censor transactions forever, unless the sequencer mechanism support opening an escape hatch if no blocks have been proposed for X time (this is similar to what Arbitrum does). In this case, it essentially combines the two solutions, while using L1 for tx-to-include for the non-sequencer produced block.

- The technical skill required by the user is minimal, and only little interaction is needed
- Sequencer selection don’t have to address periodic escape hatch, but only when no blocks have been produced for a prolonged period of time
- Cannot easily aggregate many censored transactions like was possible in full block force pushes (each must go to L1).
- Increases the cost of every rollup by ~25K for the extra storage needed to add historic roots.
- Needs modifications to circuits for nullifier edge case, but this should be manageble and likely require anyway for reverts due to public calls.

Conslusion

I think the second option of forcing inclusion (and then then allowing direct blocks in unresponsive) is the better of these solutions as it can provide users with a guarantee that they can rage-quit if needed. Also, the burden on the user rage-quitting is much smaller and the effect of censoring much higher for the censoring sequencers (stall entire rollup and lose all fees while censoring).