Censorship on L1 and Inclusion List

Definitions

Proposer: Actor that signs the block. Same as validator.

Builder: Actor that constructs the block.

Local block: Block is constructed using local mempool. Not using mev-boost.

Builder block: Block is constructed using mev-boost.

Bid: The amount builder is willing to pay proposer if the block remains canonical on chain. Simple eth transfer.

Defining censorship

Before we start, let's define censorship in its simplest terms, when a user sends a transaction with more than sufficient base fee and tip, the transaction never gets included on the chain. There are weak and strong censorship. This article excludes the reasoning behind censoring and assumes it's just a binary decision by the proposer or builder. In the rest of this article, we will refer censorship as weak censorship and see if we can improve upon that and avoid the last defense, where the last defense for censorship through the social layer. Defending through the social layer is extremely messy, client teams need to update their code and make releases under time constraints. This is why optimistic rollup has a 7-day withdrawal delay under happy case. Can we do better against weak censorship today?

Types of censorship

Weak censorship: Some proposers may refuse to include your transaction in their block. However, eventually, one proposer will include it, requiring you to wait for some time. This introduces eventual liveness, but the downside is a degradation in user experience.

Strong censorship: Proposers and attesters refusing your transaction as part of their canonical chain and constantly attempt to reorg your transaction away from the chain, necessitating a social response to hard-fork the censoring parties. We hope this will never happen.

Where does censorship exist?

From a protocol's perspective, a block gets proposed today through the proposer because it's signed by proposer, but typically, what happens is the block is constructed by the builder and facilitated by the relayer. This interaction happens outside the protocol. ~90% of the proposers outsource block construction to external builders today

Let's look at these two cases:

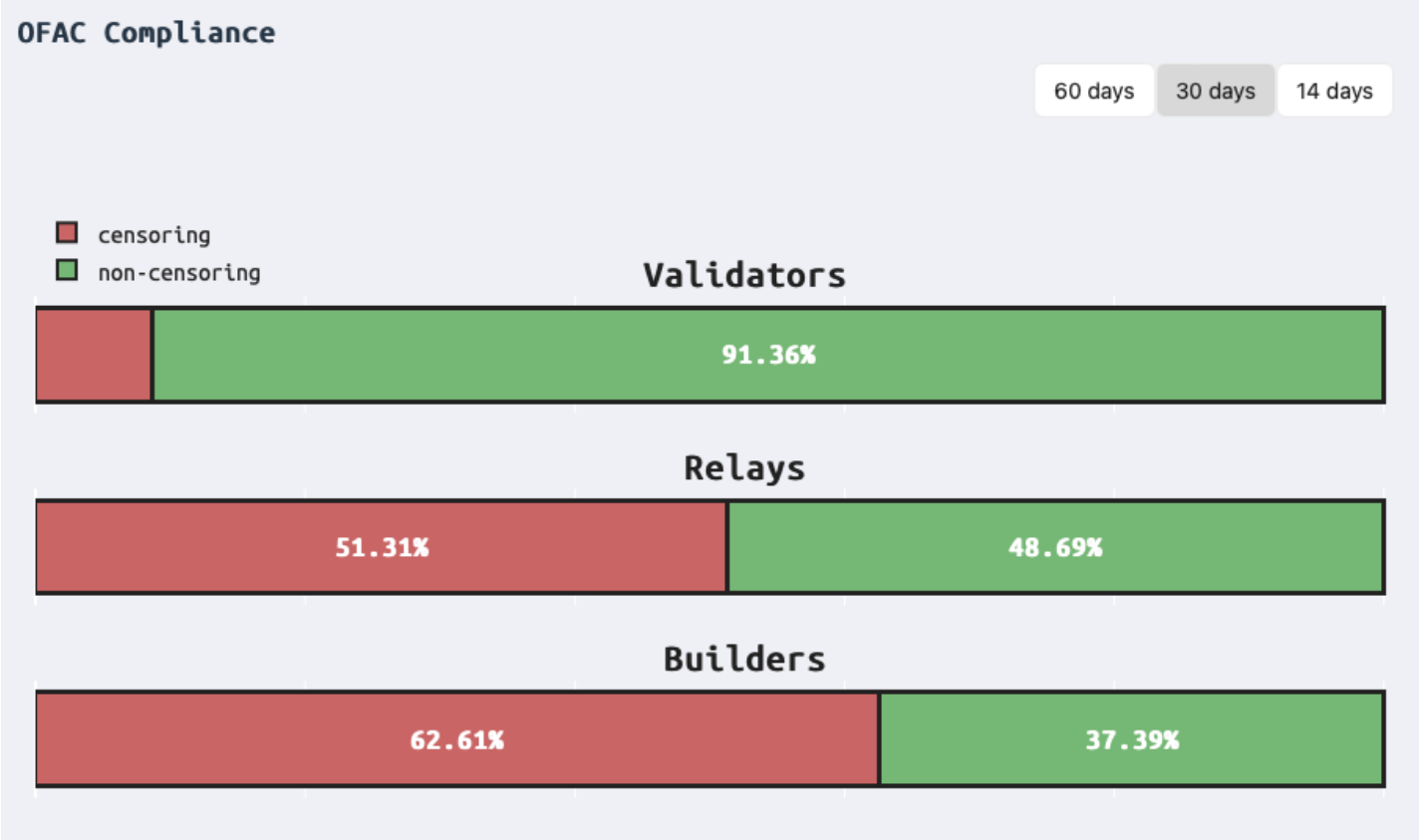
1. Censoring proposer.
2. Non-censoring proposer, but using censoring relayers or censoring builders.

1 or 2 both result in a censoring block. Now, why we think case 2 occurs more frequently than case 1 and why it is more pressing to fix. Given these two options to a non-censoring proposer:

1. Propose a less profitable local block.
2. Propose a more profitable builder block by using relayers and builders but knowing your block could be censored.

Historically, it's been proven that builder blocks are more profitable than local blocks. 90% of the proposers today are choosing option 2 over 1 because they are rational or care less about censorship[1]. Based on the below stats we observe 7% of the proposers censoring, 52% of the relayers censoring, and 63% of the builders censoring.

With probability theory, a non-censoring proposer could get a censoring block 82% of the time by treating all builders and relayers with equal weight. The uncertainty is not ideal for multiple reasons. One reason is it makes detecting forking based censorship hard, as in the future, L2 may implement its censorship oracle for faster settlement.[2]



http://censorship.pics

[1]: non-censoring proposers could choose to use a non-censoring relayer and non-censoring builder. It's unclear what % this is, but I imagine it's low.

[2]: One example is the implementation of a censorship oracle in the reducing challenge times in rollups article which counts missing blocks and a percentage of honest validators.

What can the protocol do?

Currently, the protocol does not allow a proposer to mix its local mempool transactions with the builder's block simultaneously. As a non-censoring proposer, you dont have the option to choose to proposing your own block and using builder's block, and It's one or another. The fundamental limitation is the lack of optionality. One thing we can address immediately to increase the non-censoring block rate is to allow proposers to include their local mempool transactions in parallel with the builder block.

What does the protocol want?

I think at least we can agree on the following premise: to improve censorship of the network, the easiest bang for our buck is to allow the proposer to propose its block while still using the builder block. There will be some proposers choose not to submit an inclusion list, which is fine. It's the same limitation we have today, where 9% of the proposers are censoring, and we will not change that until a more invasive protocol changes, like encrypted mempool.

What does the protocol *really* want?

The protocol wants robust censorship resistance. Allowing the proposer to include its transactions with the current slot builder. The question is who submits the final version of block? Today, the relayer submits the final block, so to make everything work, the proposer will send the inclusion list to the builder beforehand. But If these transactions are part of the builder's filtered list, the builders will refuse to build your block. Rational proposers may not opt into this regime to use the inclusion list as it limits its builder options, which may result in less MEV. We are back to square one.

Instead of forcing the inclusion list to the current slot builder, we can change the incentive by allowing the proposer to force the inclusion list to the next slot proposer/builder. This change imposes stronger censorship resistance than the previous model unless the same entity proposes two slots in a row. Given 15% entity, it proposes two slots in a row ~162 times per day.

From an implementation perspective. Enforcing the same slot inclusion list and the next slot inclusion list does not make that big of a difference. The same slot may not require a hard fork but simultaneously requires changes across the entire software stack.

Other solutions also exist today

As a user, how do you force a transaction without touching the builder? There are several paths:

1. Increase the priority fee of your transaction when it's greater than the builder's bid. The consensus client will choose a local block with your transaction in it. Live chart on how much the builders are bidding currently.
2. EL clients may use the builder override flag during execution API interactions with the CL client. This relies on EL clients to implement a censorship heuristic through the local mempool and signal to the CL client if active censorship is detected. In such instances, the CL client will utilize the local block. As of today, the EL clients heuristic is not implemented but CL clients do support them.

A few variations of inclusion list designs

We transitioned from 1 to 2. However, there are other designs in the wild that may be interesting to study, as detailed below.

1. Proposer enforces inclusion list to current slot builder
2. Proposer enforces inclusion list to next slot builder[3]
3. Concurrent proposers of a slot
4. Attester committee can force include transaction

[3]: It's currently allowing 16 transactions per inclusion list and a max of 2M gas per transaction. The inclusion list has its gas limit, meaning the next slot builder has to satisfy this constraint, or the block can not become the canonical head.

Finally

In this article, I argue that addressing censoring builders and relayers is far more impactful than censoring proposers. The true power of the inclusion list isn't just to force censoring parties to include your transaction. It enables non-censoring proposers to use builders while including transactions from the local mempool. Transaction inclusion under the next slot regime is more effective than the current slot regime. More updates to come! :star: