Optimistic rollups, and some other applications, assume that parties can always get a transaction included on Ethereum within some challenge period

, which is conventionally seven days. The challenge period needs to be long enough to resist censorship attacks where a malicious actor tries to keep certain valid Ethereum transactions from being included on the chain.

I'll describe below how we might be able to dramatically reduce the typical challenge period, by taking advantage of the fact that some types of censorship are more detectable under proof-of-stake than they were before. I'll describe how to build a "censorship oracle" in a contract, which says either (a) there has not been censorship over the last N blocks with high confidence, or (b) no conclusion can be reached. Then rollup protocols can accept their L2 state assertions more quickly if the oracle is reporting no censorship. If the oracle remains unsure, the rollup protocol would still wait up to seven days.

Types of censorship

There are two types of censorship to worry about: block-building and forking (sometimes called "weak" and "strong" censorship, respectively).

Block-building censorship is when block builders refuse to include certain transactions in their blocks. Forking censorship is when validators collude to fork the chain so that blocks containing certain transactions are reorged away and don't become part of the canonical chain.

Of these, forking censorship is the more difficult one to cope with, because it is repeatable. Block-building censorship fails if even one block is made by a non-censoring builder, but forking censorship can be repeated reliably if there are enough colluding validators. So forking censorship is the main reason for the seven-day challenge period in use today.

Detecting (the absence of) forking censorship

The good news is that in proof-of-stake Ethereum, forking censorship leaves detectable effects: blocks that are forked away will show up as empty slots in the consensus chain. And a contract can detect the number of empty slots over an interval by comparing the change in block number to the number of slots (which is the change in timestamp divided by 12 seconds).

Of course there are other reasons why a slot might be empty, but at least we know that if few slots are empty, then few blocks could have been forked away. In particular, if we know that N non-censoring blocks were created over some time period, but fewer than N blocks are missing, then we can conclude that some non-censored block was included in the chain.

If we assume, for example, that at least 10% of validators will produce non-censored blocks, then each slot will propose a non-censored block with 10% probability, and over a long enough time period the number of non-censored blocks proposed will be close to 10% with high statistical confidence. If fewer than 5% of blocks are missing during that time, we'll be able to conclude with high confidence that some non-censored block was included.

The rest is a matter of statistical calculation.

Formula and examples

Assume that each slot is assigned to a non-censoring validator with probability p

. Then the probability of seeing k

or fewer non-censored blocks in n

blocks is equivalent to the probability of getting k

or fewer heads when flipping a biased coin that comes up heads with probability p

.

The cumulative distribution function is

$$\mathrm{Pr}(X \leq k) = \sum_{i=0}^k {n \choose i} p^i (1-p)^{n-i}$$

which we can calculate numerically in practical cases.

For example, this implies that with n = 688

and p = 0.1

, we can conclude that $\mathrm{Pr}(X \leq 34) < 10^{-6}$

.

In other words, if we see 34 or fewer missing blocks out of 688 slots, we can conclude that a non-censored block was included with very high confidence. 688 slots is about 2 hours, 18 minutes.

Alternatively, if we see 4 or fewer missing blocks out of 225 slots, we can conclude that a non-censored block was included with very high (10^{-6}

) confidence. 225 slots is 45 minutes.

Observed rate of missing blocks

Over a recent series of 500,000 blocks, 3346 blocks were missing, a rate of 0.067%. We can create a more aggressive test, testing whether fewer than, say, 0.1% of blocks were missing.

Adding a non-forking assumption

The analysis above assumed that a fixed percentage of validators would not build censoring blocks, but it made no additional assumptions about forking censorship attacks. If we assume additionally a bound on the (stake-weighted) fraction of validators that will participate in forking censorship, we can get tighter bounds.

The reason this helps is that the above analysis assumed that a forking adversary can censor just exactly the blocks they choose. But if the forking collusion is limited, the collusion will sometimes be forced to suppress a run of multiple consecutive blocks, in order to get rid of one targeted block. Accounting for this can give us a more sensitive test—one that can infer non-censorship with high confidence after a shorter delay.

But it's also a more complicated analysis. We're planning to follow this post with a separate, longer piece that analyzes what is possible by adding non-forking assumptions.

Proposed Implementation

A censorship oracle, implementing a procedure like the one described above, could be deployed as a mainnet contract for anyone to use. Here is a strawman interface for such a contract:

interface CensorshipOracle { function testParameters(uint64 percentNoncensoringValidators, uint64 inverseConfidenceLevel,) pure returns (uint64, // test duration uint64, // max missing blocks allowing test to pass);

```
function startTest(
    uint64 percentNoncensoringValidators,
    uint64 inverseConfidenceLevel,
) returns (bytes32, uint64, uint64);

function getTestInfo(
    bytes32 testId,
 ) view returns (
    uint64,  // percent non-censoring validators
    uint64,  // inverse confidence level
    uint64,  // test start timestamp
    uint64,  // test result available timestamp
    bool,    // test has finished
    bool     // (test has finished) && (non-censored block was included)
 );

function finishAndGetTestInfo(bytes32 testId) returns (
    bytes32 testId,
 ) returns (
    uint64,  // percent non-censoring validators
    uint64,  // inverse confidence level
    uint64,  // test start timestamp
    uint64,  // test result available timestamp
    bool,    // test has finished (will be false if result not available yet)
    bool     // (test has finished) && (non-censored block was included)
 );

}
```

Thanks to @potuz and @terencechain

for their contributions to this work and feedback on this post.