# When the Levee Breaks

This document proposes a method of encouraging competitive proving on the Aztec network whilst incorporating measures to prevent a monopoly of said proving network.

## Scope

Our starting point for this proposal is that a sequencer has achieved a ranking high enough that they deem it worth proving and publishing a block. We have to consider that in order to prevent a sequencer from being excluded from the proving network, a sequencer must have the option of proving a block themselves. This proposal does not inhibit a sequencer from doing this.

## Summary

The basic principle of this proposal is to encourage aggresive price-driven competition between provers in order to drive down proving cost. Then a 'Proving Levy' is applied to the sequencer based on the prover/s that have contributed to the chosen proofs and the size of their contributions to previous blocks. The goal of this proposal is to create a network of provers that is large enough to ensure a monopoly can't be achieved but no so large that excessive redundant resource raises overall costs. Additionally, it attempts to reduce the overall administrative cost of orchestrating and compensating proving.

## Staking and Network Participation

Provers submit their address to a staking contract and are required to stake a minimum quantity of a designated currency to earn the right to have their proofs included within a block. This address is where fees for proof inclusion will be transferred. Provers can generate proofs for as much or as little of a block as they choose.

Staking and subsequent fee payments could occur either on L1 or L2.

## Prover Groups

At this stage it should be explained that it is our expectation that proving participants would likely be entities with significant financial and computational resources and would seek to prove entire blocks against multiple staked addresses.

## Request for Tender

A sequencer submits a request for tender for proving a proposed block via the L2 P2P network. This tender will include the ID of the un-proven block, the VRF proof and the maximum fee that the sequencer is willing to pay for the fully proven block. Provers can then make a judgement as to whether they wish to generate proofs for inclusion in the proposed block based on all of the available data:

- The ranking position of the block proposal
- The availablility of the transaction data
- The availability of their proving resource
- The fee offered for proving
- The proving charge currently levied against the prover (see below)

## Proof Proposal Submission

If a prover decides they wish to prove a block they will submit a proof proposal. This will include:

- The total fee charged by the prover
- The set of proving addresses contributing to the proof
- The number of proofs being contrbuted by each prover address

## Proving Levy Computation

The sequencer takes the proof proposal and passes it to an L2 contract function. The contribution of each prover in every block is stored as public data in the contract. The function will compute the proving levy to be applied to the sequencer for using the given proof proposal and update the contract storage with the new state of prover contributions. This proving levy computation will need to be proven by a 'Proving Levy Circuit' and the proof proposal will need to account for proving this circuit.

If the sequencer is happy with the resulting proving levy/total proving cost they will submit a proving request back to the prover.

## Proof Generation

The prover goes ahead and generates the rollup proof and the proving levy proof. The prover addresses and their associated fees must be included as public inputs to the proof. This ensures that the prover must be paid in order for the final block to be considered valid.

Once the seqencer has received a complete set of proofs that it is happy with it can go ahead and submit the rollup.

## Fees and Proving Levy

The Proving Levy is a charge based on the prover addresses used by the sequencer for a given block proof. It is designed to be a function of the contibutions those addresses have made to prior blocks. An example of a proving levy is given by the following formula, but of course a different formula could be used.

Here we simply compute the number of individual proofs produced by a single proving address over the last ‘B’ blocks as a proportion of the total number of proofs over that same number of blocks. We then subtract a threshold before scaling by a specified function. This results in a percentage to be applied to the ‘Leviable Portion’, the portion of the sequencer’s fee against which the charge is made.

$PC = \sum_{b \in [0, \ldots, B - 1]} pc(PB - b)$

$TC = \sum_{b \in [0, \ldots, B - 1]}tc(PB - b)$

$PE = \sum_{p \in [1, \ldots, P]}\max(\frac{PCp}{TC} - Th, 0) M$

$PL = BW * LP * PE$

PB = Proposed block number (block number of new block)

B = Number of blocks over which to compute levy

pc = Prover’s contribution to block in question (number of proofs)

tc = Total proofs contained in block in question (number of proofs)

PC = Prover’s total contribution over block history (number of proofs)

TC = Total proofs generated over block history (number of proofs)

P = Number of provers contributing to proposed block

Th = Threshold above which a proving levy is applied, as a fraction

M = Proving levy multiplier

PE = Prover excess (The fraction above the threshold that the prover has contributed to prior blocks)

BW = Block reward

LP = Leviable portion (The portion of the block reward subject to the proving levy)

**Worked Example:**

The sequencer receives 2 proof proposals with the following prover contributions over the previous B blocks. The threshold is 2% and the prover excess multipler is 10. The leviable portion is 60%.

**Proposal 1:**

| Prover Address | Percentage Contribution To Prior Blocks | Prover Excess |
|---|---|---|
| 1 | 1.2 | |

| | | |
|---|---|---|
| | | 0 |
| 2 | 5.1 | 3.1 * 10 = 31 |
| 3 | 5.7 | 3.7 * 10 = 37 |
| 4 | 4.1 | 2.1 * 10 = 21 |

Total Prover Excess: 88%

Prover Levy: 0.88 * 0.6 = 0.528 = ~53% of the sequencer’s fee is forfeited.

**Proposal 2:**

| Prover Address | Percentage Contribution To Prior Blocks | Prover Excess |
|---|---|---|
| 5 | 0.7 | 0 |
| 6 | 0.2 | 0 |
| 7 | 1.9 | 0 |
| 8 | 2.1 | 0.1 * 10 = 1 |
| 9 | 2.5 | 0.5 * 10 = 5 |

Total Prover Excess: 6%

Prover Levy: 0.06 * 0.6 = 0.036 = ~4% of the sequencer’s fee is forfeited.

Proposal 1 contains contributions from proving addresses that have made outsized contributions to prior blocks. As a result the sequencer would forfeit a much higher proportion of their fee for using proposal 1.

## Resource Consumption

In order to keep administrative costs to a minimum, we would seek to limit the maximum number of prover addresses that can contribute to a block to say 8. This would result in a maximum of 9 public data state writes for each block, 1 write per prover contributing to the block and 1 write for the total number of proofs used by the block.

The proving levy circuit will need to consume a lot of public state for the calculation of individual contributions over the configured block history.

There would be a maximum of 8 fee transfers required and these would either be on L2, proven by the prover levy circuit or on L1 executed by the Rollup Contract. Additionally, the burning of the proving levy would be performed in the same way.

## Known Issues

The system of staking as a way of determining ‘who is proving’ is obviously not without problems. If we computed the levy over 10 blocks and accepted up to 8 proving address per block then simply ‘buying’ 80 proving addresses with 80 times the staking requirement would ensure you could artificially force your proving levy to 0 by rotating addresses. When you factor in the prover threshold you could actually get away with less than 80 in this case. There may be a better solution to staking for this. Or alternatively there may be a combination of staking requirement, prover excess multiplier, threshold etc. that would make it economically unattractive to accumulate staking addresses and the resulting capital it would require.