Content note: preliminary research. Would love to see independent replication attempts.

Code: https://github.com/ethereum/research/tree/master/correlation_analysis

One tactic for incentivizing better decentralization in a protocol is to penalize correlations

. That is, if one actor misbehaves (including accidentally), the penalty that they receive would be greater the more other actors (as measured by total ETH) misbehave at the same time as them. The theory is that if you are a single large actor, any mistakes that you make would be more likely to be replicated across all "identities" that you control, even if you split your coins up among many nominally-separate accounts.

This technique is already employed in Ethereumslashing (and arguably inactivity leak) mechanics. However, edge-case incentives that only arise in a highly exceptional attack situation that may never arise in practice are perhaps not sufficient for incentivizing decentralization.

This post proposes to extend a similar sort of anti-correlation incentive to more "mundane" failures, such as missing an attestation

, that nearly all validators make at least occasionally. The theory is that larger stakers, including both wealthy individuals and staking pools, are going to run many validators on the same internet connection or even on the same physical computer, and this will cause disproportionate correlated failures. Such stakers could

always make an independent physical setup for each node, but if they end up doing so, it would mean that we have completely eliminated economies of scale in staking.

Sanity check: are errors by different validators in the same "cluster" actually more likely to correlate with each other?

We can check this by combining two datasets: (i) attestation data

from some recent epochs showing which validators were supposed to have attested, and which validators actually did attest, during each slot, and (ii) data mapping validator IDs to publicly-known clusters

that contain many validators (eg. "Lido", "Coinbase", "Vitalik Buterin"). You can find a dump of the formerhere, here and here, and the latter here.

We then run a script that computes the total number of co-failures

: instances of two validators within the same cluster being assigned to attest during the same slot, and failing in that slot.

We also compute expected co-failures

: the number of co-failures that "should have happened" if failures were fully the result of random chance.

For example, suppose that there are ten validators with one cluster of size 4 and the others independent, and three validators fail: two within that cluster, and one outside it.

There is one co-failure here: the second and fourth validators within the first cluster. If all four validators in that clusters had failed, there would be six

co-failures, one for each six possible pairs.

But how many co-failures "should there" have been? This is a tricky philosophical question. A few ways to answer:

- For each failure, assume that the number of co-failures equals the failure rate across the other validators in that slot times the number of validators in that cluster, and halve it to compensate for double-counting. For the above example, this gives \frac{2}{3}

.

- Calculate the global failure rate, square it, and then multiply that by \frac{n * (n-1)}{2}

for each cluster. This gives (\frac{3}{10})^2 * 6 = 0.54

.

- Randomly redistribute each validator's failures among their entire history.

Each method is not perfect. The first two methods fail to take into account different clusters having different quality setups. Meanwhile, the last method fails to take into account correlations arising from different slots having different inherent difficulties

: for example, slot 8103681 has a very large number of attestations that don't get included within a single slot, possibly because the block was published unusually late.

[

882×227 11.8 KB

](https://ethresear.ch/uploads/default/original/2X/c/c34994d5a8fe647e304a91b98a72361eeb532c48.png)

See the "10216 ssfumbles" in this python output.

I ended up implementing three approaches: the first two approaches above, and a more sophisticated approach where I compare "actual co-failures" with "fake co-failures": failures where each cluster member is replaced with a (pseudo-) random validator that has a similar failure rate.

I also explicitly separate out fumbles

and misses

. I define these terms as follows:

- Fumble

: when a validator misses an attestation during the current epoch, but attested correctly during the previous epoch

- Miss

: when a validator misses an attestation during the current epoch and also missed during the previous epoch

The goal is to separate the two very different phenomena of (i) network hiccups during normal operation, and (ii) going offline or having longer-term glitches.

I also simultaneously do this analysis for two datasets: max-deadline

and single-slot-deadline

. The first dataset treats a validator as having failed in an epoch only if an attestation was never included at all. The second dataset treats a validator as having failed if the attestation does not get included within a single slot

.

Here are my results for the first two methods of computing expected co-failures. SSfumbles and SSmisses here refer to fumbles and misses using the single-slot dataset.

Fumbles

Misses

SSfumbles

SSmisses

Expected (algo 1)

8602090

1695490

604902393

2637879

Expected (algo 2)

937232

4372279

26744848

4733344

Actual

15481500

7584178

678853421

8564344

For the first method, the Actual

row is different, because a more restricted dataset is used for efficiency:

Fumbles

Misses

SSfumbles

SSmisses

Fake clusters

8366846

6006136

556852940

5841712

Actual

14868318

6451930

624818332

6578668

The “expected” and “fake clusters” columns show how many co-failures within clusters there “should have been”, if clusters were uncorrelated, based on the techniques described above. The “actual” columns show how many co-failures there actually were. Uniformly, we see strong evidence of “excess correlated failures” within clusters: two validators in the same cluster are significantly more likely to miss attestations at the same time than two validators in different clusters.

How might we apply this to penalty rules?

I propose a simple strawman: in each slot, let p

be the current number of missed slots divided by the average for the last 32 slots. That is, p[i] = \frac{misses[i]}{\sum_{j=i-32}^{i-1}\ misses[j]}

. Cap it: p \leftarrow min(p, 4)

. Penalties for attestations of that slot should be proportional to p

. That is, the penalty for not attesting at a slot should be proportional to how many validators fail in that slot compared to other recent slots

.

This mechanism has a nice property that it’s not easily attackable: there isn’t a case where failing decreases

your penalties, and manipulating the average enough to have an impact requires making a large number of failures yourself.

Now, let us try actually running it. Here are the total penalties for big clusters, medium clusters, small clusters and all validators (including non-clustered) for four penalty schemes:

- basic

: Penalize one point per miss (ie. similar to status quo)

- basic_ss

: the same but requiring single-slot inclusion to not count as a miss

- excess

: penalize p

points with p

calculated as above

- excess_ss

: penalize p

points with p

calculated as above, requiring single-slot inclusion to not count as a miss

Here is the output:

basic basic_ss excess excess_ss
big 0.69 2.06 2.73 7.96
medium 0.61 3.00 2.42 11.54
small 0.98 2.41 3.81 8.77
all 0.90 2.44 3.54 9.30

With the “basic” schemes, big has a ~1.4x advantage over small (~1.2x in the single-slot dataset). With the “excess” schemes, this drops to ~1.3x (~1.1x in the single-slot dataset). With multiple other iterations of this, using slightly different datasets, the excess penalty scheme uniformly shrinks the advantage of “the big guy” over “the little guy”

.

What’s going on?

The number of failures per slot is small: it’s usually in the low dozens. This is much smaller than pretty much any “large staker”. In fact, it’s smaller than the number of validators that a large staker would have active in a single slot

(ie. 1/32 of their total stock). If a large staker runs many nodes on the same physical computer or internet connection, then any failures will plausibly affect all of their validators.

What this means is: when a large validator has an attestation inclusion failure, they single-handedly move the current slot’s failure rate, which then in turn increases their penalty. Small validators do not do this.

In principle, a big staker can get around this penalty scheme by putting each validator on a separate internet connection. But this sacrifices the economies-of-scale advantage that a big staker has in being able to reuse the same physical infrastructure.

Topics for further analysis

- Find other strategies to confirm the size of this effect where validators in the same cluster are unusually likely to have attestation failures at the same time
- Try to find the ideal (but still simple, so as to not overfit and not be exploitable) reward/penalty scheme to minimize the average big validator’s advantage over little validators.
- Try to prove safety properties about this class of incentive schemes, ideally identify a “region of design space” within which risks of weird attacks (eg. strategically going offline at specific times to manipulate the average) are too expensive to be worth it
- Cluster by geography. This could determine whether or not this mechanism also creates an incentive to geographically decentralize.
- Cluster by (execution and beacon) client software. This could determine whether or not this mechanism also creates an incentive to use minority clients.

Mini-FAQ

Q

: But wouldn’t this just lead to staking pools architecturally decentralizing their infra without politically decentralizing

themselves, and isn’t the latter what we care about more at this point?

A

: If they do, then that increases the cost of their operations, making solo staking relatively more competitive. The goal is not to single-handedly force solo staking, the goal is to make the economic part of the incentives more balanced. Political decentralization seems very hard or impossible to incentivize in-protocol; for that I think we will just have to count on social pressure, starknet-like airdrops, etc. But if economic incentives can be tweaked to favor architectural decentralization, that makes things easier for politically decentralized projects (which cannot avoid being architecturally decentralized) to get off the ground.

Q

: Wouldn’t this hurt the “middle-size stakers” (wealthy individuals who are not big exchanges/pools) the most, and encourage them to move to pools?

A

: In the table above, the “small” section refers to stakers with 10-300 validators, ie. 320-9600 ETH. That includes most wealthy people. And as we can see, those stakers suffer significantly higher penalties than pools today, and the simulation shows how the proposed adjusted reward scheme would equalize things between precisely those validators and the really big ones. Mathematically speaking, someone with 100 validator slots would only have 3 per slot, so they would not be greatly affecting the penalty factor for a round; only validators that go far above that would be.

Q

: Post-MAXEB, won’t big stakers get around this by consolidating all their ETH into one validator?

A

: The proportional penalty formula would count total amount of ETH, not number of validator IDs, so 4000 staked ETH that acts the same way would be treated the same if it’s split between 1 validator or 2 or 125.

Q

: Won’t adding even more incentives to be online create further pressure to optimize and hence centralize, regardless of the details?

A

:The parameters can be set so that on average, the size of the incentive to be online is the same as it is today.