The proposal/confirmation separation mechanism proposed by Justin a few days ago works as follows:

1. Proposers (a set that anyone can join) all create collations with headers, containing transactions, and broadcast the (signed) header (but not the body!). Each header also specifies a fee F, and we assume that the transactions in the collation have some total fee T which goes to the proposer.

2. The validator assigned to create a collation chooses a collation (generally, the one with the highest fee), an counter-signs it.

3. The proposer publishes the body

Only collations that are double-signed (by proposer and validator) and have an available body are eligible for inclusion in the canonical chain. The fee F goes to the validator regardless

of whether or not the collation gets into the canonical chain, though the transaction fees T only go to the proposer if the collation does get included into the canonical chain. This creates incentive alignment: proposers have an incentive to publish the body, and validators have an incentive to co-sign the proposal that pays the highest fee.

However, there is one problem. An attacker can create collation headers with fees higher than actual available transaction fee levels, and then never publish the correponding collation bodies. Normally, this would be equivalent to publishing very expensive transactions to spam the network and shut out legitimate users that cannot afford to outbid the attacker, an attack that can cause some damage though ultimately can't stop high-value transactions from getting in. Here, though, it's worse, because by not publishing collation bodies, an attacker can prevent the main chain from growing, thereby facilitating a 51% attack.

The problem arises for a simple fundamental reason: this mechanism allows an open set (proposers) to prevent validators' blocks from contributing to the chain's security

. One natural fix is conceptually simple: change the scoring rule of the chain to GHOST, and allow collation headers with unavailable bodies to contribute to a block's score. This actually fixes the problem; however, GHOST is hard to implement.

We create a simpler version of GHOST with the following scheme. We start off by creating a one-layer-restricted version of GHOST, where we follow the following rule: if a block has score N, and the block has k

children, then all children of that block have score N+k (and we can also interpret the "virtual score" of the block as being N+k-1). For example:

[

%5B1%5D%20--%20%5B4%5D%2C%5B1%5D%20--%20%5B4%20%5D%2C%5B1%5D%20--%20%5B4%20%20%5D%2C%5B4%5D%20--%20%5B6%5D%2C%5B4%5D%20--%20%5B6%20%5D%2C%5B6%5D%20--%20%5B7%5D%2C%5B4%20%20%5D%20--%20%5B5%5D

258×568

](https://yuml.me/diagram/scruffy/class/%5B1%5D%20--%20%5B4%5D%2C%5B1%5D%20--%20%5B4%20%5D%2C%5B1%5D%20--%20%5B4%20%20%5D%2C%5B4%5D%20--%20%5B6%5D%2C%5B4%5D%20--%20%5B6%20%5D%2C%5B6%5D%20--%20%5B7%5D%2C%5B4%20%20%5D%20--%20%5B5%5D)

This already gives us much of what we want; collations that are one hop away from being in some collation's ancestry chain contribute to that collation's score. A malicious proposer may be able to create a bunch of collations that are all missing bodies that are on top of the same head, but the legitimate head's score will continue accumulating at the same rate. However, it is hard to calculate; a block deep in history getting a child would lead to that block's score increasing, and then the score of every other descendant of that block.

So I propose a simplification. We add to collation headers a field (chosen by the validator), lookbehind_score_update

. We add to the storage record of each block a field, child_count

, that stores the number of children the collation has. When processing add_header

, we look back lookbehind_score_update

steps in its ancestry, and at each step (proceeding further to nearer in history), set collation.score = collation.parent.score + collation.parent.childcount

. Then at the end, we set collation.score = collation.parent.score + min(1, collation.parent.childcount)

and then collation.parent.childcount += 1

.

We may also wish to expose a function that allows the publisher of a header to retroactively recalculate the score of their header at any time, if they feel it is worth it to pay the gas fees to do so.

This allows the shard chains to continue to use a simple score-based fork choice rule, while incorporating enough elements of GHOST to ensure collations with missing bodies continue to contribute to security. It also solves the problem with direct uncle inclusion, where newer blocks are favored over older blocks because newer blocks can include older blocks as uncles but not vice versa.