Special thanks to Danny Ryan, Ben Edgington, Corver Roos, Abhishek Kumar and Dhruv Bodani for feedback and review.

Ethereum's beacon chain consensus spec has always been very carefully designed to enable multi-party compute solutions, such as distributed validator technology (DVT), without being opinionated as to their implementation architectures. Examples of this include the adoption of BLS signatures, and the use of the Legendre proof of custody method in a former design. Algorithms that hinder MPC have been studiously avoided.

However, there is one place in the spec that currently limits our options. While being theoretically MPC-friendly, the current method for the selection of aggregators from committees prevents us from implementing DVT as a convenient middleware solution.

The following is an overview of the aggregation duty, followed by a description of how it impacts the implementation of Distributed Validator Technology, and a proposal for the Consensus Layer implementers that would enable middleware-based distributed validator clients to co-ordinate validator clients to perform aggregations on the correct topics.

The Aggregation

To help scale the beacon chain protocol, each committee needs to randomly select a small subset of its members that will aggregate the attestations from the whole committee. The determination a validator must undergo to figure out whether it has been selected to produce an aggregation duty is determined in the following manner:

def is_aggregator(state: BeaconState, slot: Slot, index: CommitteeIndex, slot_signature: BLSSignature) -> bool: committee = get_beacon_committee(state, slot, index) modulo = max(1, len(committee) // TARGET_AGGREGATORS_PER_COMMITTEE) return bytes_to_uint64(hash(slot_signature)[0:8]) % modulo == 0

The important piece of this for DVs is hash(slot_signature)[0:8]

.

The slot_signature

is calculated as a validator private key signing the current slot.

def get_slot_signature(state: BeaconState, slot: Slot, privkey: int) -> BLSSignature: domain = get_domain(state, DOMAIN_SELECTION_PROOF, compute_epoch_at_slot(slot)) signing_root = compute_signing_root(slot, domain) return bls.Sign(privkey, signing_root)

Ultimately, this process is used to randomly but predictably allocate an appropriate number of validators to the duty of aggregation across every committee and slot, this duty is one of the fundamental scalability improvements that allowed proof of stake Ethereum to reduce from 1,500 ether minimum stake to 32 ether.

DV Architectures

There are two broad approaches to implementing DVs; as a middleware to existing validators, or as a standalone validator client implementation.

The middleware

BLS signatures are homomorphically additive, you can combine them in an untrusted operation. This fact allows you to build a 'non-custodial' distributed validator, for lack of a better analogy. Once you can coordinate existing validators to all sign the same object, their resulting signatures can be combined into a group signature by a middleware client and subsequently broadcasted to a beacon node.

[

Distributed Validator Middleware

1200×627 167 KB

](https://ethresear.ch/uploads/default/original/2X/5/5b33bcb79790ef0fa7df65ecca69367b5eb3e0fd.png)

In my opinion this architecture is important as it allows distributed validator technology to be additive

to existing validator clients, rather than competitive.

A standalone validator client

BLS signatures may be additive, but coordinating independent software clients to all behave in unison is difficult and may not be practical. The alternative is to implement a distributed validator client as a full validator client rather than a middleware, with direct or indirect control over the private key, meaning either the private key shares are used by the client, or a remote signer is directed by the distributed validator client to sign what the DV client instructs.

This approach is more practical, in that you have control over when you ask remote signers to produce randao_reveal

values required for block production, and you can ask each signer to produce a slot_signature

for their key share, which can then be aggregated, and hashed, to determine whether this distributed validator must perform an aggregation. It can also then direct a remote signer to sign an aggregation, something a middleware cannot force a downstream validator client to do either.

The problem

A recent episode of Uncommon Core addressed in a really frank manner, the systemic risk posed by the fact that 80% or more of Ethereum mining pools were running the flashbot team's mev-geth. Stephane Gosselin and Danny Ryan amicably discuss the rearchitecture of mev-boost into a non-critical sidecar for a validator to run alongside their beacon client to provide higher value blocks when the validator is selected to propose the next block on the chain.

In a similar vein, we at Obol think hard about how Distributed Validator Technology can be value-additive rather than competitive to client teams. Our intention is that our client charon, will support connecting a standard validator client to it, or a remote signer, to offer people the options that make sense for how they would like to deploy a distributed validator. However we think whether DVT propagates into the staking ecosystem as a middleware, or as an alternative client choice has important implications for the anti-fragility of the space as a whole.

It is our view that if we have validator clients implement DV protocols, instead of choosing which DV client to use as a middleware, we may end up in one of two outcomes;

- We run the risk of fragmenting or partitioning our burgeoning client diversity with many clients adopting a variety of incompatible DV implementations that causes homogeneity issues between validator clients and distributed validator protocols. Or;
- We rush the adoption of a canonical DV spec, that may not be optimal, and we won't have the freedom to innovate on models and designs as much as a result of ossifying too early.

For these reasons, we at Obol are keen to encourage DVT to thrive as an optional middleware rather than as a standalone validator client / standard protocol, not unlike how mev-boost is moving to a value-add sidecar model rather than an alternative client choice. However this outcome might need the continued support of the Consensus Spec community to facilitate this architecture. We aspire to work with the spec writers, and the client teams to gradually clear the way to allow middleware based DVs to perform as optimally as a standard validator implementation.

A middleware friendly spec

With the above said, I return to aggregations. Aggregations are not currently directly rewarded nor punished in protocol, rather their absence would indirectly affect the participation rate for a slot. The problem as it stands is, with a middleware based distributed validator, each validator client, which has just one share of the overall group private key, calculates their slot_signature

with their own private key share rather than the group private key, which they then hash and end up deciding on incorrect occasions and committees to perform aggregations.

Danny further isolated the issue by highlighting that in the prepareBeaconCommittteeSubnet, the validator client just says true/false

as to whether the beacon client should do the work of preparing for an aggregation later in the slot. We considered that a validator client could potentially pass the slot_signature

they have calculated to the beacon client for the beacon client to verify. Receiving these slot_signatures from each validator client would enable a middleware like charon to reconstruct the group slot_signature, which the clients could then hash and see concretely whether their distributed validator is required to perform an aggregation.

Determining the correct occasion for a DV to aggregate is one aspect of the challenge, the second is prompting a validator client to ask for an aggregate attestation to sign later in the slot, we propose the following.

In the interest of not changing any existing code, we would add a new endpoint

POST /eth/v1/validator/is_aggregator

With a request body like:

[{ "validator_index": "1", "committee_index": "1", "committees_at_slot": "1", "slot": "1", "slot_signature":

"0x1b66ac1fb663c9bc59509846d6ec05345bd908eda73e670af888da41af171505cc411d61252fb6cb3fa0017b679f8bb2305b26a285fa2737f175668d0dff91cc1b66ac1fb663c9bc59509846d6ec05345bd90
}, {...},]

This request would return an array of booleans, telling the validator client whether the given slot_signature's mean they have been selected to aggregate. Think of it as inverting the current dependence such that a validator client now trusts the beacon client to inform it of its duty to aggregate. We would then feature flag this behaviour behind a --distributed-mode

flag, which when enabled, would make a validator client ask it's beacon node via this endpoint whether it is due to aggregate or not, rather than calculating it itself. In practice, calls to this endpoint would be intercepted by the charon middleware, and once a quorum of validators in the cluster have submitted their slot_signatures, the cluster can calculate
bytes_to_uint64(hash(group_slot_signature)[0:8]) % modulo == 0

and inform all validators they must aggregate later in the slot by returning true

from the is_aggregator

request.

Conclusion

This post is designed to source feedback, concerns, suggestions and support for middleware based distributed validators, and the addition to the beacon API and opt-in modified validator client behaviour we are proposing. The next steps will be to incorporate the gathered feedback before opening PRs on the beacon API and consensus specs repos and seeking adoption on a consensus implementers call once ready. Thanks for reading.