PSLE + SPBE

Public Single Leader Election (PSLE) + Secret Probabilistic Backup Election (SPBE)

This idea came up in conversation about DoS attacks with@vbuterin, @dankrad , and @protolambda spurred by @jrhea's recent packetology research.

Why single leader in eth2?

Single leader election for block proposal at each slot in eth2 (as opposed to multiple) serves to (1) reduce "extra work" and p2p message overhead and (2) reduce unnecessary forking that would be associated with "multiple" potential block heads at each slot. Having multiple potential heads at each slot would lead to disparate attestation votes within a committee, reducing aggregatability, ulimately increasing p2p message overhead and increasing requistite block sizes.

With the addition of shards in Phase 1+, multiple leaders on shard block proposals would compound this disparate attestation problem (BEACON_PROPOSERS_PER_SLOT * SHARD_PROPOSERS_PER_SLOT

options per slot), again leading to higher overhead but also reducing the probability that at a given slot, crosslinks are successful. If the number of shard proposers in a slot are SHARD_PROPOSERS_PER_SLOT

and naively each has the same probability of being selected, then attestations will roughly split across the different shard proposals and with any SHARD_PROPOSERS_PER_SLOT > 1

fail to get the 2/3 crosslink vote in most cases.

The downsides of single leader election

Both public and secret single leader election (PSLE and SSLE respectively) for block proposal has a distinct downside: a single point of failure for the liveness of a particular slot

.

This liveness failure can come in two flavors (indistinguishable by the consensus protocol):

1. The validator is simply not online, not well connected, or otherwise just doesn't produce a block
2. The validator has been specifically taken down by a malicious attacker

(1) is fundamental – if a validator is simply offline, that's it. The argument against (1) being a big concern is that there is a significant opportunity cost (~1/8th of total rewards) in being offline during a block proposal, but we still expect some amount of empty slots because of this.

Given the information leaked by validators (i.e. the broadcasting of frequent consensus messages and capabilities advertised in discovery) and thus de-anonymization for typical nodes is expected to be trivial in many cases, PSLE leaves block proposers susceptible to targeted DoS attacks near their assigned slot time. Although there are a number of extra-protocol tools at a block proposer's disposal (sit behind TOR, sit behind cloudflare/etc, use a sentry node architecture, etc), many of these solutions are not satisfying (e.g. higher latency, centralized dependences, higher infrastructure costs, etc), and this vector is of particular concern to hobbyists/home-stakers (a class of validator we hold has critical to the decentralization eth2).

(2) can be more generally solved via by making this leader election secret

(SSLE) and thus a targeted attack on a particular node(s) cannot performed without attacking all nodes, but SSLE is hard and not ready for production.

[Note! There has been some solid progress on SSLE, but constructions general require ZKPs so are not a prime candidate for layer 1 today

, but we are optimistic in the medium term

]

Proposed solution

Given that secret

single leader election is hard and not yet ready today and that (1) will always be a marginal a problem, we propose Public Single Leader Election (PSLE) + Secret Probabilistic Backup Election (SPBE)

as a hybrid solution to increase the network's resilience to attacked/offline block proposers. Even when SSLE is ready, SSLE+SPBE might still serve as additional resilience against (1).

The PSLE component operates exactly as block proposer leader election does today, whereas SPBE provides a backup in the case that the selected block proposer does not show up for their job.

SPBE probabilistically selects a set of backup proposers via a local secret that can be proven in the block proposal (e.g. a signature of the slot and the epoch's seed of randomness). This selection can be done in a very similar way to aggregation selection (see [is_aggregator

](https://github.com/ethereum/eth2.0-specs/blob/a019b03ae0cdc45024abdbae3b7b041841916975/specs/phase0/validator.md#aggregation-selection)) with a tunable parameter for target number of backups – TARGET_BACKUPS_PER_SLOT

.

Selected backup proposers create and broadcast a block partially through the assigned slot (e.g. 1/6th in eth2 Phase 0 config) if a block proposal from the public leader has not yet been locally seen. Nodes do not consider/broadcast early backups.

When tie-breaking in LMD GHOST, public leader proposals are given priority followed by backup block proposals and then ties broken by lexicographical order. In the normal case, the public leader submits a proposal on time, committees for that slot will see that proposal as the head and disregard any backups, naturally converging the fork choice on public leader blocks in the case that they are on-time/available. [remember, we don't consider attestations for slot S

until slot S+1

so proposals are 0-weight leaf nodes, giving tie break favor to the leader block proposal]

Concrete spec changes

Add backup_proposer_proof: BLSSignature

to [BeaconBlock

](https://github.com/ethereum/eth2.0-specs/blob/a019b03ae0cdc45024abdbae3b7b041841916975/specs/phase0/beacon-chain.md#beaconblock)

Modify proposer assertion in [process_block_header

](https://github.com/ethereum/eth2.0-specs/blob/a019b03ae0cdc45024abdbae3b7b041841916975/specs/phase0/beacon-chain.md#block-header) to allow for backup proposals:

def is_valid_leader_proposal(state: BeaconState, block: BeaconBlock) -> bool: return (block.proposer_index == get_beacon_proposer_index(state) and block.backup_proposer_proof == BLSSignature() # empty bytes)

def is_valid_backup_proposal(state: BeaconState, block: BeaconBlock) -> bool: # Similar to leader selection, backups are only known within current epoch seed = get_seed(state, epoch, DOMAIN_BEACON_PROPOSER) + int_to_bytes(state.slot, length=8) signing_root = compute_signing_root(seed, get_domain(state, DOMAIN_BACKUP_PROPOSER)) proposer = state.validators[block.proposer_index] return (# public leader cannot make backup proposal block.proposer_index != get_beacon_proposer_index(state) and bls.Verify(proposer.pubkey, signing_root, block.backup_proposer_proof)

)

def process_block_header(state: BeaconState, block: BeaconBlock) -> None: ... # Verify that proposer index is the correct index # -- REMOVE -- assert block.proposer_index == get_beacon_proposer_index(state) assert (is_valid_leader_proposal(state, block) or is_valid_backup_proposal(state, block)) ...

Remember which blocks are "leader" blocks in the fork choice store. E.g. add is_leader: Dict[Root, bool]

to store that tracks each blocks as a leader (True

) or backup (False

) proposal when added in [on_block

](https://github.com/ethereum/eth2.0-specs/blob/a019b03ae0cdc45024abdbae3b7b041841916975/specs/phase0/fork-choice.md#on_block).

Modify fork choice's [get_head

](https://github.com/ethereum/eth2.0-specs/blob/a019b03ae0cdc45024abdbae3b7b041841916975/specs/phase0/fork-

choice.md#get_head) to tie break first by is_leader

and secondly lexicographically.

def get_head(store: Store) -> Root: ... while True: ... # Sort by latest attesting balance with ties broken leader blocks then lexicographically head = max(children, key=lambda root: (get_latest_attesting_balance(store, root), is_leader[root], # tie break by leader root, # tie break by lexicographical sort))

Add backup duty to Validator guide. Backup blocks are broadcast at SECONDS_PER_SLOT // 6

into slot if valid proposal from leader is not yet seen.

Modify [process_randao

](https://github.com/ethereum/eth2.0-specs/blob/a019b03ae0cdc45024abdbae3b7b041841916975/specs/phase0/beacon-chain.md#randao), [process_attestation

](https://github.com/ethereum/eth2.0-specs/blob/a019b03ae0cdc45024abdbae3b7b041841916975/specs/phase0/beacon-chain.md#attestations), and [slash_validator

](https://github.com/ethereum/eth2.0-specs/blob/a019b03ae0cdc45024abdbae3b7b041841916975/specs/phase0/beacon-chain.md#slash_validator) to use block.proposer_index

(rather than get_beacon_proposer_index

) for the proposer lookup as get_beacon_proposer_index

is only for the leader. This would require changing the function signature of each to ensure that info is passed in.

Discussion

Eager backup proposers

We might worry that it becomes rational for a backup proposer to eagerly create and broadcast backup proposals at the start of any selected slot rather than waiting to see if a backup is actually needed. This primarily hurts the network in terms of bandwidth – ~1+ TARGET_BACKUPS_PER_SLOT

blocks per slot instead of 1

--, but due to the modifications of the fork choice, does not induce a higher degree of forking when the leader is live.

Additional constraints on in the gossipsub beacon_block

topic can be added to ensure that early backup blocks (before 1/6 slot time) are not propagated and if a leader block is seen, are totally dropped. With the majority of the network following such a propagation rule, then in the normal case when the leader is live and latency is low, even if backup proposers attempt to disseminate blocks, they will be dropped within a hop or two.

Attacker backup proposers

We might worry that it becomes rational for a backup proposer to attack

the leader to increase the backup proposer's chance for inclusion in the canonical chain.

Creating this additional attack vector/incentive internal to the validator is a legitimate concern. Without SPBE, the set that wants to disrupt proposer liveness is generally limited to outside attackers, whereas SPBE creates a new intra-validator incentive to analyze.

A few things to note

1. This type of attack on SPBE degrages to a live chain, which is nice.
2. The incentive to perform such an attack is arguably low because the chances of actually getting the backup proposal are a fraction of TARGET_BACKUPS_PER_SLOT

so in expecation this is only worth some fraction of already relatively small (compared to attesation duties) block reward

1. To further reduce the additional attack incentives of backup proposals, we can make backup proposals only receive PROPOSAL_REWARD // BACKUP_DISCOUNT_FACTOR

where BACKUP_DISCOUNT_FACTOR >= 2

. As you increase this number, backup proposals become more of an altruistic action to help the liveness of the network (I'd assume most honest validators would be willing to perform) rather than an individually profitable action with potential incentives to disrupt the normal operation.

That said, attacks that can be profitably conducted by backup proposers are of primary concern in PSLE+SPBE and should be further discussed and analyzed.

Consensus complexity and phase 0 delivery

Although I want to get this potential algorithm out there for discussion, I would argue for not

introducing SPBE in Phase 0 as to not disrupt Phase 0 delivery. If

targeted DoS becomes a real problem on mainnet, we have this as an option in our back pocket. In addition, stating PSLE+SPBE as a public option might serve to discourage such an attack from the start.

Depending on further discussions, observed mainnet attacks, and the progress of SSLE, we can consider this for integration in a subsequent phase/fork.