See previous ideas on this topic: Proposer/block builder separation-friendly fee market designs

Sequence of events in a slot pair

- Right before 0 seconds - exec header publishing: anyone can publish an exec header, which contains an exec block hash, a bid, and a signature from the builder
- 0 seconds - beacon block deadline: beacon block must include the winning exec header
- 0-2.67 seconds - attestations on beacon block: only one committee attests to the beacon block
- 8 seconds - intermediate block deadline: the winning builder publishes an intermediate block, consisting of the exec block body and as many attestations on the beacon block as they can find
- 8-10.67 seconds - attestations on intermediate block: the remaining N-1 committees attest to the intermediate block
- 10.67-13.33 seconds - aggreation of intermediate block attestations
- 13.33-16 seconds - next exec header publishing

If a beacon block is missing, the next slot is switched to be for a beacon block instead of an intermediate block.

In diagram form

1162×277 12.9 KB

Key intuitive properties

- From a fork choice perspective, the system can be described as a beacon chain just like the current one, except with uneven committee sizes and with a (block, slot) fork choice. The only difference is that some of the blocks are only there to select proposers for the block right after them. This simplifies analysis.
- A committee in between each step helps to ensure that each step is “safe” and reduces vulnerability to abuses by single actors

Safety properties for builders

At the bid publishing step, builders see the head, and know whether it’s safe or unsafe (a head could be unsafe if there are lots of disagreeing or missing attestations).

- If a head is safe, the head cannot be reverted barring a 45%+ attack, significant amounts of slashing, or extreme network latency. In this case, builders can feel confident bidding safely.
- If the head is unsafe, there is a risk that the chain could be reorged after they release their body, “stealing” their MEV opportunities. In this case, builders see this risk and can decrease their bids to claim a risk premium for this risk.

At time of intermediate block publishing, there are two cases:

1. The beacon block has not been published. In this case, the attestation committee will have voted against the block, and so the intermediate block producer (aka. the builder) can safely not publish, and will not be penalized.
2. The beacon block has been published. In this case, the intermediate block has a “proposer boost” slightly greater in size than the entire attestation committee, so if the builder publishes, their block will win in the eyes of the remaining N-1 attestation committees.

This ensures that if the attestation committees are honest, and latency is not extremely high, the builder is guaranteed to:

1. Get included if they publish
2. Not be penalized if they do not publish because the beacon block header is missing

The builder has a period of ~5.33-8 seconds during which to publish. They can safely publish as soon as they see the beacon block; however, they may want to wait until they see more attestations, as they get rewarded for including attestations (attesters that get included also get a reward themselves). They are free to negotiate the tradeoff between (5.33 second window, get attestation inclusion reward) and (8 second window, get no attestation inclusion reward) as they wish.

Beacon chain spec change sketch

Proposer index definition

- Set get_random_proposer_index(state: State) to what get_beacon_proposer_index(state) returns today.
- Add state variables chosen_builder_index and chosen_exec_block_hash. If a slot is empty, set state.chosen_builder_index = NO_BUILDER (a constant equal to 2**64 - 1). If a slot contains a beacon block which contains a BuilderBid, set:

state.chosen_builder_index = builder_bid.message.builder_index state.chosen_exec_block_hash = builder_bid.message.exec_block_hash * state.chosen_builder_index = builder_bid.message.builder_index * state.chosen_exec_block_hash = builder_bid.message.exec_block_hash * Define get_beacon_proposer_index(state: State) as follows:

If state.chosen_builder_index == NO_BUILDER, return get_random_proposer_index(state)

Otherwise, return state.chosen_builder_index * If state.chosen_builder_index == NO_BUILDER, return get_random_proposer_index(state) * Otherwise, return state.chosen_builder_index

Conditions on bid-carrying block

- If state.chosen_builder_index == NO_BUILDER, the block is required to contain a BuilderBid and may not contain an ExecBody. The builder_bid is required to pass the following checks, whereval = state.validators[builder_bid.message.builder_index]:

bls.Verify(val.pubkey, compute_signing_root(builder_bid.message), builder_bid.signature) val.activation_epoch == FAR_FUTURE_EPOCH or val.withdrawable_epoch <= get_current_epoch(state) val.balance >= builder_bid.bid_amount * bls.Verify(val.pubkey, compute_signing_root(builder_bid.message), builder_bid.signature) * val.activation_epoch == FAR_FUTURE_EPOCH or val.withdrawable_epoch <= get_current_epoch(state) * val.balance >= builder_bid.bid_amount * Add a balance transfer to the processing logic:

val.balance -= builder_bid.bid_amount state.validators[get_beacon_proposer_index(state)].balance += builder_bid.bid_amount * val.balance -= builder_bid.bid_amount * state.validators[get_beacon_proposer_index(state)].balance += builder_bid.bid_amount * Change get_committee_count_per_slot to take inputs (state: BeaconState, slot: Slot) (instead of epoch). If a slot has state.chosen_builder_index == NO_BUILDER, the committee count should return 1.

Conditions on exec-body-carrying block

- If state.chosen_builder_index != NO_BUILDER, the block is required to contain an ExecBody and may not contain a BuilderBid. The exec_body is required to pass the following checks:

hash_tree_root(exec_body) == state.chosen_exec_block_hash eth1_validate(exec_body, pre_state=state.latest_exec_state_root) * hash_tree_root(exec_body) == state.chosen_exec_block_hash * eth1_validate(exec_body, pre_state=state.latest_exec_state_root) * Add to the processing logic:

state.latest_exec_state_root = exec_body.post_state_root * state.latest_exec_state_root = exec_body.post_state_root * The get_committee_count_per_slot should return (get_epoch_committee_count(epoch) - state.committees_in_this_epoch_so_far) // (slots_remaining_in_epoch) * If state.chosen_builder_index != NO_BUILDER, set state.chosen_builder_index = NO_BUILDER, regardless of whether or not there is a block.

Notes

- Reduce slot time to 8 seconds (remember: 1 exec block will come every 2 slots)
- All beacon blocks, including bid-carrying and exec-carrying, should get a proposer boost in the fork choice.
- Fork slot should be changed to (block, slot).

Possible extension: delayed publishing time with a fee

If the intermediate block builder does not publish during slot N, no bundle is selected in slot N+1. The entire proposer sequence gets pushed back by one slot (so the slot N+1 proposer becomes the slot N+2 proposer, etc), and a new random proposer is chosen for slot N+1. The builder gets another chance (so, an extra 12 seconds of slack) to publish. The slot N+1 exec block cannot contain any high-value consensus transactions (ie. slashings). However, they get penalized block.basefee * block.target_gas_limit.

The intuition is that they are delaying their exec block by one slot and prepending it with an empty exec block, so they need to pay for that slot. The proposer sequence being delayed ensures that delaying one’s exec block is not useful for stealing future proposal rights in times when they are high-value.

Possible extension to shards

1158×397 24.9 KB