Payload-timeliness committee (PTC) – an ePBS design

by mike & francesco – based on discussions with potuz & terence – july 6, 2023 \cdot h/t Potuz for coming up with the idea of not giving the builder block explicit fork-choice weight, but rather using a committee to decide on the timeliness of the payload. This document is the result of a number of iterations on this concept. \cdot tl;dr; We present a new design for ePBS called Payload-Timeliness Committee (abbr. PTC). We include: a high-level overview, the new honest attesting behavior, and an initial analysis of the properties and potential new attack vectors. This document omits the formal specification, which will follow as we gain confidence in the soundness of this design. \cdot Many thanks to Danny, Barnabé, Caspar, Toni, Vitalik, Justin, Jon, stokes, Jacob, Aditya, and Chris for relevant discussions.

New concepts

- payload-timeliness committee (PTC) – a subset of the attestation committee from each slot that votes on the builder's payload release.
- full, empty, and missing blocks – a full block has a valid ExecutionPayload that becomes canonical (results in an update to the EL state). An empty block is a CL block does not have a canonical ExecutionPayload (does not update the EL state). A missing block is an empty slot that becomes canonical. With block-slot voting, missing blocks can have fork-choice weight.
- payload-timeliness (PT) votes – the set of votes cast by the PTC.

The PT votes for slot N are only used by the proposer and attesting committee in slot N+1. The PT votes for slot N determine the proportion of the fork-choice weight given to the full vs. empty versions of the slot N block. * The PT votes for slot N are only used by the proposer and attesting committee in slot N+1. * The PT votes for slot N determine the proportion of the fork-choice weight given to the full vs. empty versions of the slot N block.

Note – Throughout this document, we describe block-slot voting as a prerequisite for PTC. However, we can make use of the existing voting mechanics and treat ancestral block votes in slot N as votes for the missing version of the slot N block. For clarity in the examples, we describe missing as part of the competing fork, but adding pure block-slot voting may not be necessary in practice.

Design overview

We first present a minimal description of the new design. Note that this does not include the full specification of behavior and is intended to present the high-level details only.

Old slot anatomy

Presently, the 12 second slot is partitioned into the following three phases. Figure from Time, slots, and the ordering of events in Ethereum Proof-of-Stake.

upload_0ad62632f94d94974f544f1b20a6c9ff1577×565 67.3 KB

Flow

1. At t=0 block proposed by the elected PoS validator.
2. At t=4, the attestation deadline, the attesting committee for slot N uses the fork-choice rule to determine the head of the chain in their view and makes an attestation.
3. At t=8 the aggregate attestations are sent.
4. At t=12 the next proposer builds on whatever head they see according to their fork-choice view.

New slot anatomy

The new slot contains an additional phase for the PT votes to propagate.

upload_516b236ef25496ffda5566b3db21218a2038×533 97 KB

Flow

1. The slot N block that is published at t=t0 is the CL block proposed by the elected PoS validator. This block contains a builder bid, but no ExecutionPayload.
2. The attestation deadline at t=t1 is unchanged. The entire attesting committee for slot N uses the fork-choice rule to determine the head of the chain in their view and makes an attestation.
3. The broadcast of aggregate attestations for the slot still begins at t=t2. Additionally, the builder publishes their execution payload if they have not seen an equivocation from the proposer (the builder does not need to have a full view of the attestations).
4. At t=t3 the PTC casts their vote for whether the payload was released on time.
5. At t=t4 the slot N+1 proposer publishes their block, building on either the full or empty block based on the PT votes

and attestations that they have seen.

Note – Timestamps are not specified because we can adjust the duration of each phase based on performance testing. The key details are what happens in the four phases and at the boundaries.

Note 2 – Depending on the size of the PTC, we may need a fifth slot phase to aggregate the votes of said committee. Larger committees may need aggregation, but are more secure, which is the design tradeoff.

Block production

The full block is assembled in two phases. First, the CL (beacon) block is produced with a commitment to a builder bid. Any double-signing of this block represents a proposer's equivocation. Once the builder has observed the CL block, they produce their ExecutionPayload to construct the full version of the block. The PTC casts votes on the timeliness of this object in their view.

upload_16a2060853c41b28b83f27c03a64d5421389×880 105 KB

The slot N committee votes on the CL block, while the slot N PTC observes the payload release, which informs the fork-choice rule in the subsequent slot. We propose that the PTC is a strict subset of the slot N attesting committee (e.g., the first 1000 members).

Honest attesting behavior

Consider the honest attesting committee at t=t1 of slot N+1. Assume that there was a block in slot N and the slot N+1 proposer built a child block (it extends w.l.o.g. to missing blocks in either slot). The slot N+1 block could either build on the empty or full block from slot N. Similarly, the PTC cast their votes on full or empty based on the payload-timeliness at t=t3 of the previous slot. The attesting committee in slot N+1 assigns weight to the full vs empty block respectively by using the proportion of PT votes from slot N.

Examples

Assume the slot N block is canonical and received 100% of the slot N attestations. Let W denote the total weight of each attesting committee. PT indicates the payload-timeliness vote and boost is the 40% proposer boost given to each proposer block.

Case 1

upload_fb7fdc053bc36a887afec78cf836f4e21124×815 21.2 KB

The slot N PTC is split, with 51% voting for the full block and 49% voting for the empty block. The slot N+1 proposer builds on the empty block and is given proposer boost, which is equivalent to 40% of the weight of the attesting committee. The attesting committee at N+1 now sees a split and has to choose between attesting to N+1 or N+1,missing. The PT votes divide the attestation weight of the slot N attestations, so we have,

weight(N+1) = 0.49W + 0.4W = 0.89W,

while,

weight(N+1,missing) = 0.51W.

Thus, the attesters vote for N+1 (the top fork).

Case 2

upload_0a5354622de22951feca56b75174ae951149×836 22.4 KB

Here we have 100% of the PT votes agreeing that the block should be full. Now the attesting committee for slot N+1 votes against the proposer, and for N+1,missing because

weight(N+1) = 0.4W < weight(N+1,missing) = W.

In this case, the PTC from slot N overrules the slot N+1 proposer, who clearly built on the wrong head. The bottom fork wins.

Case 3

upload_121a77b1cbbd1cc377195d2f496d43ee1164×823 21.5 KB

In the worst case, the slot N+1 attesters see this split view. Here

weight(N+1) = 0.7W = weight(N+1,missing),

and they have to tie-break to determine which fork to vote for. The key here is that this split is difficult to achieve because you need 70% of the PTC to vote for full, while the proposer builds on empty. More on this in “Splitting considerations” (below).

Analysis

The properties specified below were initially presented in Why enshrine Proposer-Builder Separation? A viable path to ePBS. We modify honest-builder payload safety to the weaker condition of honest-builder same-slot payload safety.

Properties

1. honest-builder payment safety – if an honest builder is selected and their payment is processed, their payload becomes canonical.
2. honest-proposer safety – if an honest proposer commits to a single block on time, they will unconditionally receive the payment from the builder for the bid they committed to.
3. honest-builder same-slot payload safety – if an honest builder publishes a payload, they can be assured that no competing payload for that same slot will become canonical. This protects builders from same-slot unbundling. Note: This property relies on a 2/3 honest majority assumption of the validator set.

Non-properties

1. honest-builder payload safety – the builder cannot be sure that if they publish the payload, it will become canonical. The builder is not protected from next-slot unbundling (the builder is not protected from that in mev-boost either, as length-1 reorgs happen regularly). More on this in “Splitting considerations” (below).

Note – In terms of CL engineering, making the transition function able to process empty blocks is important. Because empty blocks will not update the EL state, they must exclude withdrawals.

Builder payment processing

Builder payments are processed during the epoch finalization process (open for discussion, could just be on a one-epoch lag). The payment takes place if both of these conditions are satisfied:

1. The builder ExecutionPayloadHeader is part of the canonical chain (i.e., the CL block for that slot is not missing). This includes two cases:

The corresponding ExecutionPayload is also part of the canonical chain (the happy-path) (i.e., the CL block for that slot is full). The builder ExecutionPayloadHeader is part of the canonical chain even if the corresponding ExecutionPayload is not (consensus that the builder was not on time) (i.e., the CL block for that slot is empty). 1. The corresponding ExecutionPayload is also part of the canonical chain (the happy-path) (i.e., the CL block for that slot is full). 1. The builder ExecutionPayloadHeader is part of the canonical chain even if the corresponding ExecutionPayload is not (consensus that the builder was not on time) (i.e., the CL block for that slot is empty). 1. There is no evidence of proposer equivocation.

A builder who sees an equivocation can get the validator slashed. Any slashed validator will not receive the unconditional builder payment. 1. A builder who sees an equivocation can get the validator slashed. Any slashed validator will not receive the unconditional builder payment.

Differences from other designs

1. The payload timeliness determines how to allocate the fork-choice vote, but it cannot create a separate fork.
2. The payload view determines how the subsequent attesting committee votes.
3. The builder is never given a proposer boost or explicit LMD-GHOST weight.
4. The unconditional payments are handled asynchronously, after enough time has passed for equivocations to be detected and the corresponding validators to be slashed.
5. The PT votes only inform the attesting behavior of the subsequent slot. Similar to proposer boost, the effect is bound to a single slot.

Splitting considerations

“Splitting” is an action undertaken by a malicious participant to divide the honest view of the network. In today’s PoS, proposers can split the network by timing the release of their block near the attestation deadline. Some portion of the network will see the block on time, while other will vote for the parent block because it was late. Proposer boost is the mechanism in-place to give the next proposer power to resolve these forks if the weight of competing branches is evenly split. The PTC introduces more potential splitting vectors, which we present below.

Proposer-initiated splitting

First, consider the case where the proposer splits the chain in an attempt to grief the builder.

upload_79750da5a56789643d5cf7ab16a1e2fa1929×1086 262 KB

The sequence of events is as follows:

1. The slot N proposer releases their block near the attestation deadline, causing a split of the honest attesting set. 50% of the honest attesters saw it on time and voted for the block, while the other 50% did not see it on time and thus voted for a missing block.
2. The builder of the header included in the block must make a decision about releasing the payload that corresponds to this block. The block is either full or empty based on that decision.
3. The slot N+1 proposer resolves the split by building on the missing, full, or empty head (any of the blue blocks). Because the proposer will have a boost, the fork is resolved.

Now consider the builder's options.

1. Publish the payload – If the missing block becomes canonical, they published the payload, but it never made it on-chain (bad outcome). Otherwise, the full payload became canonical (good outcome).
2. Withhold the payload – If the empty block becomes canonical, the unconditional payment goes through, but they aren't rewarded by the payload being included (bad outcome). If the missing block becomes canonical, then they didn't needlessly reveal their payload (good outcome).

In summary, either publishing or withholding the payload can result in a good or bad outcome, depending on the behavior of the slot N+1 proposer (which is uncertain even if they are honest).

Key point 1 – By not giving fork-choice weight to the builder, we cannot protect them from the proposer splitting in an attempt to grief. However, the builder can be certain that their block will not be reorged by a block in the same slot, so they can protect their transactions by bounding them to slot N.

Key point 2 – Today, such splitting is possible as well; it just looks slightly different. If the proposer intentionally delays their publication such that the next proposer might try to reorg their block using the honest-reorg strategy, the mev-boost builders have no certainty that their block won't be one-block reorged. Indeed, we see many one-block reorgs presently. See Time, Slots for more context.

Builder-initiated splitting

As hinted at above, builders can try to grief the slot N+1 proposer into building on a fork with a weak PT vote by selectively revealing the payload to a subset of the PTC.

upload_240b0eb8bf0b71269cb92e2bfaa2d7e11139×815 21.6 KB

In this case, the N+1 proposer is going to get orphaned by the N+1,missing block because there was such a disparity in the PT votes. Specifically, if the builder can get the proposer to build on the full or empty block which is the opposite of what the PTC votes for, then they orphan the block. Let N,empty be the block that the proposer of N+1 builds on (w.l.o.g.), and let \beta denote the proportion of the PTC voting for N,full. If \beta > 0.7, then the N+1 validator will be orphaned. In other words, the proposer needs to agree with at least 30% of the PTC to avoid being orphaned.

Spitting conclusion

Neither of these splitting conditions seem too critical. The proposer-initiated splitting is already possible today with mev-boost and builder-initiated splitting will be difficult to execute if the PTC is sufficiently large. Any ePBS design involves some additional degrees of freedom for network participants (by enshrining the builder role), and this design is no exception. Further analysis on the probability and feasibility of splitting will be an important part of the full evaluation of the PTC design.