[

banner

1792×1024 745 KB

](https://ethresear.ch/uploads/default/original/2X/6/62d6a99d68e97088bdf157a582a55d8b5cccb38b.jpeg)

ePBS design constraints

Summary

We characterize the problem that enshrining proposer-builder separation into the core Ethereum protocol tries to solve. We identify a basic suite of features essential for any viable solution. Subsequently, we explore additional design considerations that lack consensus. By accepting certain constraints, the range of potential solutions becomes significantly limited.

Table of Contents

1. Introduction

A Hackmd.io version of this document can be found here, it may render better.

This document examines the various considerations and compromises involved in developing a framework to incorporate proposer-builder separation (PBS) into Ethereum's core protocol. It serves dual objectives: firstly, to demystify the current progress in PBS research for a broader audience, and secondly, to establish a consensus on design constraints among the core developer community related to this issue.

There are three different sets of constraints that shape the problem at hand, those are

1. What is the problem itself?
2. What would be the minimum set of requirements that a design would have to satisfy to be considered a solution?
3. What are the trade-offs considered in extra implementation details for proposed solutions?

This document navigates through topics that inherently involve a high degree of subjectivity, sparking discussions that can be influenced by personal and business interests. It doesn't shy away from presenting my own perspectives on these matters. Whenever possible, I articulate these viewpoints in the first person*, striving to clearly delineate them while also presenting counterarguments. Encouragingly, there appears to be broad agreement on points 1) and much of 2), with the majority of debates focusing on point 3). Regarding specific design decisions, multiple valid perspectives exist, leading to divergent outcomes and influencing related design considerations, each with its own set of constraints. Although I anticipate consensus on the definitions and constraints outlined in sections 2 and 3, the same cannot be assumed for the discussions in subsequent sections. To streamline what could otherwise become an exhaustive list within a complex decision tree, I express my personal preferences on each design trade-off, thereby imposing constraints and advancing the discussion. Nevertheless, I aim to thoroughly examine the trade-offs, including the advantages and drawbacks, at each decision point.

- It has been many years of writing documents in the third person so this article will not be consistent in this regard.

As the discussions progress over the next few months, our goal is to finalize the design and reach consensus on the implementation specifics of point 3). However, the immediate aim of this document is to concentrate on points 1) and 2), offering clear and as formal as possible definitions to minimize subjective elements in the solution design process.

Remarks.

- The reader will surely have read about ePBS in the context of MEV and the future of financial markets. While I recognize that the decisions that we make when implementing ePBS will have profound financial impact on some of the players involved in MEV extraction, there will be little to no mentions of MEV in this text. The problem that ePBS sets out to solve is a problem of trust, nothing more and nothing else. It is not a problem about transaction ordering nor who gets to place transactions in special places. MEV, and specially MEV stealing, plays a role in trying to understand the motivation for certain forkchoice attacks. One could argue that MEV is the reason d'etre for PBS, which may well be (although statelessness will require it anyway), but here we are trying to understand how to enshrine PBS and we do not need to deal with why PBS exists, but rather what are the current problems in its design and how do we solve them.
- This document prioritizes the "how" of implementing ePBS over the "why." It aims to articulate the specific problem ePBS intends to address—though some may interpret the acknowledgment of this problem as part of the "why"—and delineates the essential constraints any solution must meet.

This document laid out as follows. In section 2 we describe the main problem, In section 3 we describe the main constraints that any solution should satisfy, we believe these constraints are widely accepted. In section 4 we describe some design decisions that are not universally agreed upon. We make some choices along the way to obtain a few more constraints that

may help guide finding a unique solution. In section 5 we describe constraints that while not strictly necessary, they would appear from some of the design choices in section 4. In section 6 we describe briefly some little nuances with forkchoice, these are just implementation details of the sort of complexity that we need to account for.

Acknowledgments.

I would like to thank @n0ble

, @terencechain

and @barnabe for useful feedback.

2. Current status of PBS: MEV-Boost

To address point 1) mentioned in the introduction and effectively describe the challenge of integrating proposer-builder separation (PBS) into the protocol, it's crucial first to comprehend what PBS entails and its current implementation. This document includes a concise overview of PBS to ensure it remains self-contained; however, we presuppose the reader has a basic understanding of the subject. Extensive literature exists on PBS, and for those seeking a foundational introduction, we recommend referring to [1]. This preliminary context sets the stage for a deeper exploration of enshrining PBS within the protocol, focusing on identifying and resolving the inherent challenges.

[

ethereum_block

610×934 122 KB

](https://ethresear.ch/uploads/default/original/2X/b/bead979188ab846799808677344f89e657b64b3b.png)

Credit: chainstack

In current Ethereum PoS, blocks are handled by two related layers that, while intricately connected, are meant to be encapsulated and isolated as much as possible. These are the execution layer

(EL) and the consensus layer

(CL). Blocks are gossiped and transmitted through a P2P network that is handled by the CL. They consists of mainly two parts, a consensus envelope that carries all the needed information to decide what the current status of the blockchain is and within it, an execution payload

(AKA simply a block

) which consists mainly of the actual transactions that are to be executed by the Ethereum virtual machine. The CL unwraps this payload and submits it to the EL for validation.

The execution payload itself has market value, as it is the information contained within it that will be included in the blockchain and is accessible to smart contracts. Users of the network pay to include data within this payload. The ordering of the transactions in this payload is the source of market extractable value

(MEV) and specialized builders

and searchers

produce bundles

of transactions with a prescribed order that profit from this. For an introduction to MEV see[2]

Before the merge, miners

used to solve cryptographic puzzles in order to gain the right to produce such a block. Since the merge and the switch to PoS, validators

are assigned duties for each slot, and on every slot (that is a unit of 12 seconds currently) a validator is chosen to be the proposer

of a block. That is, a particular validator is chosen and this validator has the right to broadcast a block, containing both the consensus information and within it the execution data, to the Ethereum network. Validators are rewarded for the inclusion of their block. The rewards depend on many factors, like how many attestations

(votes for previous blocks in the consensus layer) the block contains, whether or not the block contains any slashing

(penalizations for badly behaved validators in the consensus layer), but by far the biggest reward comes from the execution

data, either directly by fees paid by users to include their transactions, or indirectly because of MEV exploited within the block.

Economically rational validators (hereby called proposers

) are thus inclined to outsource the construction of their execution payload to a specialized external agent (hereby called builder

) and this is what leads to PBS. The mechanism by which this is implemented today is rather involved, but at a high level the key ingredients to understand are as follows:

- Lack of trust 1: builders can't trust the proposer with their payload. If they simply give their payload, the proposer can simply steal the information (MEV) included in it. Validators have gone to extreme cases in order to steal this information [3].
- Lack of trust 2: Proposers can't trust builders with their blocks. If proposers simply give a pre-signed block for the builder to complete assembling it, they have no guarantee of inclusion. Thus the builder could collude with the next slot proposer to steal the rewards of the current slot.
- Trusted intermediary: Since the two parties involved do not trust each other, they both interact with a trusted intermediary, the relay

. This entity obtains the pre-signed block from the proposer, the pre-built execution payload from the builder, assembles the full block and broadcasts it to the p2p network.

[

Mev Boost interaction

1724×1056 116 KB

](https://ethresear.ch/uploads/default/original/2X/2/2f7db65d0b476fe891f891d1671bfb77cb9ea4e6.png)

credit: Mike Neuder

The full interaction of this implementation is rather involved. A simplified version is as follows

1. At the start of the slot, the proposer requests bids from the trusted relay. Bids consist of a hash of the block (a cryptographic commitment to the block contents) and a value promised as payment to the proposer.
2. The relay gathers bids from different builders and then submits the best one to the proposer when requested.
3. The proposer chooses the best received bid and constructs the full consensus part of the block. In place of the execution payload it places the hash in the bid. This is called a blinded block

. The proposer signs it and submits it to the relay.

1. The relay gathers the blinded block from the proposer, the corresponding payload with the right hash and replaces the hash in the block with the corresponding full payload. After assembling the full block, it broadcasts it to the network.
2. Communication between the proposer's CL client and the relay is done through a middleware called MEV-Boost[4] that forwards the proposer's requests to several relays at once.

2.1. The essential players

A consequence of the current implementation of PBS is the existence of the trusted intermediary whose sole purpose is to bridge communications between entities that don't trust each other. This is the very same problem that blockchains were created to solve in the first place. We are now in conditions of giving some formal definitions so as to start framing the problem we want to solve and answer 1) in the introduction.

Proposers

Within the beacon chain, a validator is identified by the following structure[5]

class Validator(Container): pubkey: BLSPubkey withdrawal_credentials: Bytes32 # Commitment to pubkey for withdrawals effective_balance: Gwei # Balance at stake slashed: boolean # Status epochs activation_eligibility_epoch: Epoch # When criteria for activation were met activation_epoch: Epoch exit_epoch: Epoch withdrawable_epoch: Epoch # When validator can withdraw funds

We can now define what we mean by proposer

:

During the current slot N

, and assuming the beacon state at the head of the network is given by st

such that st.slot = N

. The proposer

is the owner of the BLS private key that corresponds to the st.validators[idx].pubkey

where idx

is the result of get_beacon_proposer_index(st)

.

We could in principle give a weaker definition of proposer, for example if the BLS holder wanted to outsource the construction of the CL block itself to another entity, we would need terminology to separate the above defined proposer from the entity that actually builds the CL block. We will restrict however to PBS in which there is one entity as defined above and it is an in-protocol tracked validator.

Builders

There are some designs that allow for a formal definition of builder as an in-protocol tracked entity. However, without committing to a particular design we can't give such a formal definition. We thus adopt an ill-defined description for the time being.

We assume the following situation. During the current slot N

, the proposer sells the rights to construct the execution payload within his block to a separate entity. The mechanisms for payments, guarantees, etc. are left as implementation details. The only requirement is that in any form of PBS, the proposer wants to sell this right in exchange for some form of payment. The entity that builds the execution payload is called the builder

.

[

seller

1792×1024 554 KB

](https://ethresear.ch/uploads/default/original/2X/a/a8d4b72db4e2064326b45e6bbf2b1567d606a97e.jpeg)

2.2. What is the problem itself?

We can now state the problem thus answering 1) in the introduction.

The main problem is that in the current implementation of PBS in Ethereum

1. A proposer that wants to sell his right to build a payload must

trust an intermediary.

1. A builder that wants to buy the rights to build a payload must

trust an intermediary.

Point 2) has a partial solution: since builders are not currently protocol-enforced, and relays aren't even necessary for PBS, builders can always be that intermediary, thus removing the trust. These builders/relays are called vertically integrated

. A better framing for 2) is thus:

2'. An untrusted

builder that wants to buy the rights to build a payload must

trust an intermediary.

Emphasis is here in the word must

, we will see below some nuances to these issues. There is no analog of 2' for 1), however, a proposer always has the

option to not sell his rights and thus is not forced to trust an intermediary. This is an important property that we will explicit here for future reference

(Self-Building)

A proposer may, if so desires, not sell his right to build a payload. In other words, a proposer can always be his own builder.

There are other problems that are typically connected and mentioned in the context of (e)PBS: these include

- Censorship resistance [6]
- Monopolization/Cartelization.
- Single point of failure (in the MEV-Boost software)

Each of these problems come with separate design space for solution, while it's valid to characterize that these problems are associated to PBS, I would say that all these are secondary and ultimately are caused by the trust assumptions of the main problems 1) and 2). The mere existence of the forced

trusted intermediary is the problem to solve.

3. Enshrining PBS

Once we agree on the problem we want to solve, then we can start searching for a solution to this problem. The purpose of this section is to lay out the first constraints that such solutions should satisfy and that are widely agreed upon. We will move on to less clear options later in the text.

Given the statement of the problem, it becomes clear that a solution would have to satisfy:

1. A proposer that wants to sell his right to build a payload may do so without having to trust any entity.
2. A builder that wants to buy the rights to build a payload may do so without having to trust any entity.

However, while the word trust

in the statement of the problem in the previous section does not cast any doubts about its semantics, here, this word is loosely defined so as to indicate a solution. We must thus specify a sequence of parameters that would unequivocally satisfy 1. and 2. above.

3.1. Builder safety

Let us start with builder safety. A property of Ethereum blocks is the following. During the current slot N

, the head is the block at slot M < N

, if the proposer of N

constructs his block based on M

and this block is seen timely by honest validators, then the block N

becomes the head of the chain*. This could be taken as some form of "proposer safety", it roughly says that if you are an honest proposer and act according to the honest validator guide, your blocks will be included on-chain.

- This wasn't always the case due to FFG considerations that have been fixed since.

Something similar happens in the current implementation of PBS via MEV Boost. We have seen the interaction between the proposer and the relay in Section 2. The proposer, in his final request for a payload sends the pre-signed block to the relay. Assuming it is timely and that the relay and builder act honestly

his block will be included on chain. Similarly for the builder, the builder sends the full payload to the relay, and assuming it is timely and the proposer and the relay act honestly

his block will be included on-chain. There are trust assumptions both ways here. The proposer trusts both that the builder will reveal its payload and that the relay will assemble and reveal the payload. The builder trusts that the relay has gotten the full signed block from the proposer and that the relay will assemble and broadcast the full block. Modulo the trust assumptions that were already in place as described in Section 2.2, if the builder behaves honestly, his block will be included on chain. We can't provide a rigorous definition of builder safety since we don't even have a definition of builder (we have seen already that given such a proper definition would depend on the implementation details). But we can take therefore the above description as our definition

(Builder reveal Safety)

An honest builder that reveals his payload timely during his turn will have his block included on chain.

Of course this is only a local view of the chain. When we say “will have his block included on chain” we mean that his block will become immediately the canonical head of the chain. This does not imply that the block is finalized, it may be reorged by different conditions afterwards that are external to the builder/proposer in turn. This also holds for the discussion above for regular blocks produced without an external builder.

[

betting

1792×1024 1010 KB

](https://ethresear.ch/uploads/default/original/2X/8/88404ffc9128d92ea2e86d015462074176de5d9a.jpeg)

3.2. Unconditional payment.

This one is easier to define, the sole purpose of PBS is to allow the proposer to sell the rights to broadcast a payload. We can take again a very loose definition

(Unconditional payment)

assuming the proposer and the builder agree upon a value for these rights, and the proposer acts honestly, he should receive payment regardless of the builder’s actions.

The mechanisms for agreeing to a value and the actual definition of the proposer acting honestly would depend on implementation. In the current MEV-Boost auction, builder’s bids contain a value promised to be paid to the proposer, and they are signed. The proposer agrees to sell by sending back a signed blinded block with the corresponding hash for the payload. The unconditional payment is guaranteed by the relay under the main trust assumptions.

The question of what does it mean for a proposer to act honestly is simpler to frame since proposers are always in-protocol entities, so we can simply refer to the current honest validator guide for this. At present, this means to broadcast a valid signed block (which in the context of PBS is a blinded block) timely.

All proposed mechanisms for ePBS rely on a commit-reveal scheme for the negotiation between the builder and the proposer. That is, the proposer gathers bids that contain a cryptographic commitment to a payload and a promise of payment. The proposer first broadcasts his CL block together with this commitment. The builder then later reveals (or not) his committed payload. Since we are not aware of any other mechanism to enshrine PBS, we will assume that such a commit-reveal is in place. In this conditions we can then specify then that the proposer acting honestly means that it has broadcast is CL block, with the builder’s commitment, timely.

Remarks.

- Notice that unconditional payment does not necessarily mean immediate payment. There are different design options on how the payment is actually implemented and these result in trade-offs with regard to complexity and timing. We will deal with these options in detail below, but briefly mention here two of them: if the payment is carried in the EL via a transaction to an address specified to the proposer, then the payment can be included directly in the builder’s payload. If on the other hand the payment is fulfilled in the CL by transferring staked ETH from the builder to the proposer, then a churn may need to be applied (eg if the builder is also a validator) to avoid transfers between slashed validators.
- @barnabe made me notice that my assumptions on commit-reveal of a cryptographic commitment are stronger than necessary, only a commitment to a particular builder is needed, there is no need to commit to a particular payload. This in fact is an extra design decision that is not considered in this document but does not change in any way the remaining of the document. Whatever choices are made for the fully committed payload, will remain valid if we only commit to a builder and then later let the builder decide the payload. In fact some of the designs like the signed message to notify of a purposely withheld payload, can be better implemented as an empty payload being revealed by the committed builder. Notice that this approach, in which the proposer does not commit to a particular payload, but rather any

payload from a specific builder, is equivalent to a slot auction instead of a block auction, just that it happens just in time and not in advance [19]. Also notice that this is still compatible with inclusion lists.

3.3. Proposer Safety.

The fact that the proposer reveals his block without any intervention from the builder (as long as the proposer has at least one valid bid to include he can broadcast a block) allows us to keep proposer safety as a required constraint of the system:

(Proposer safety)

If the proposer acts honestly and reveals his block timely, it will be included on chain.

In particular this forces the design to track at least three types of slots:

- Full slots: the CL block and the payload have been revealed
- Missed slots: the CL block and therefore the payload has not been included.
- Empty slots: the CL block has been included but the committed payload has not.

Remark.

proposer safety as required above can in principle be weakened, allowing for example to not include the CL block, but rather include enough information from it to enforce the unconditional payment. There is a trade-off between forkchoice complexity (dealing with empty slots) and specification complexity in order to keep the guarantee of unconditional payment.

3.3. Builder safety revisited.

The previous two sections dealt with proposer's safety regardless of the builder's actions. On the other hand, in Section 3.1 we are only guaranteeing the builder's safety under the assumption that the proposer has acted honestly and his block was seen on time. Unfortunately, in most commit-reveal systems and with the usual time constraints that the Ethereum slot has, the builder needs to reveal his payload without having BFT guarantees as to whether the proposer's block is included on-chain. That is, the builder's view of the head of the chain may differ from the majority of validators at the time of reveal. This is a fundamental problem that will manifest itself in several sections of this document. Let us analyze however a situation in which these views agree, even in the tautological case:

If the builder has auctioned but not bought the rights to construct the current payload, then he cannot be charged.

Although tautological, one must first understand how is it that we reach to this situation in a commit-reveal scheme. As we have seen multiple times: builders submit bids with commitments to payloads and a promise of payment. They only find out that a proposer has chosen their bid when the signed block from the proposer is revealed, containing their bid. If this block is never revealed, then the builder should have nothing to pay. The problem is that the network does not reach consensus that the block was never revealed until after

the builder was supposed to reveal his part. We can enforce the following obvious rule

(Builder withholding safety - preliminary)

The builder cannot be forced to fulfil a payment for a bid that was never included on-chain.

This doesn't however address a most common situation which is that of weak blocks. Honest validators currently, when they are about to propose a block during slot N

are allowed to reorg the block in slot N-1

if this block was seen late by the majority* of the network[7], that is, they will construct their block based on the block at slot N-2

.

- Currently the honest reorg property takes 20% as threshold by default to decide if a block is weak. This will have to be changed as mentioned below to a higher value.

This feature is currently optional, and some clients may or may not implement this. As such, during slot N

if the previous block was weak, it is the proposer of N

that gets to impose its view of the chain on other validators: either if he builds on top of N-1

or N-2

, his block N

is guaranteed to be the canonical head. Consider now this situation in the context of PBS: the builder of N-1

has by now seen a weak block (eg late, with almost no attestations for it) from the proposer, including the commitment to his payload. The builder is now at a conundrum; the final view of the chain will be decided by the proposer of slot N

. Should he reveal his payload or not? if he decides to reveal and the proposer of N

imposes his view of and reorgs the block, then we have already violated the builder reveal safety

rule of Section 3.1. On the other hand, what we have not addressed is what happens if the builder does not

reveal his payload? In this case if the proposer of N

decides to base his block on top of N-1

(which by the previous section is allowed to be “empty”) then the chain will have a commitment to a builder’s payload and thus because of the unconditional payment

guarantee of the proposer of N-1

, the builder would have to pay. This situation cannot happen currently with MEV-Boost, as there are no empty slots

. When the block is deemed to be weak, the relay will simply not assemble and broadcast the block. The builder loses the right to build a payload, but the slot is simply missed and no payment is enforced. Under the assumption of a common view of the network, the above rule can be strengthened as follows:

(Builder withholding safety)

If the CL block for the current slot has not been seen or has been seen late by the majority of the network and the builder decides not to reveal his payload, then he cannot be forced to pay.

This is not really a strengthening of the above rule but rather this forces us on another constraint of the system:

(Mandatory honest reorgs)

If the CL block of the current slot N

has been seen late by the majority of the network, then the next slot proposer has to reorg it, that is, base his block on top of the parent of N

.

Under this last rule, the previous two become equivalent.

Remark.

There are possible actual strengthenings of the builder withholding safety property. The desired property would be such that even if the builder has a different view of the chain than the majority of the chain, as long as these views do not differ by much, he is allowed to withhold a payload. For example this could be implemented by allowing the builder to submit a signed message saying “I am the builder of the committed payload, but I have seen a weak block”. Validators that see this message may want to assign to the parent block a higher weight by some margin, say a withholder boost

(WB). So if the next proposer is honest, he will reorg the current slot if it had less than (1 + WB)/2

of weight. We will see this idea further explored in Section 4.2.

3.4. Minimal trustless system.

Gathering the information from the current section, we have up to this point that a reasonable minimum set of constraints that an enshrined PBS system should satisfy are

- Builder reveal safety.
- Mandatory honest reorgs.
- Builder withholding safety.
- Unconditional payment
- Proposer safety.

These are already under an assumption that ePBS consists of a commit-reveal system in which the proposer gathers signed bids from builders by whatever means and reveals his blind block

before the builder reveals his payload.

3.5. No trusted advantage.

We conclude this section by articulating a requirement that, due to its broad phrasing, serves more as a guiding principle than a strict rule. A foundational solution for trustless building ensures that proposers have the liberty to sell their building rights to builders, and conversely, builders can purchase these rights without the need for trust. However, this framework

does not inherently restrict the use of trusted systems outside the protocol. In navigating our design decisions, we should consistently adhere to the following principle:

There should be no inherent advantage for proposers, to sell their blocks to builders off-protocol.

Not following this principle would quickly render the system useless as rational proposers would flock to off-protocol building and we would have added protocol complexity in vain. We will give an example of this principle when we talk about P2P effects and bid gossip topics below.

4. Implementation details.

In the previous section we have established a minimum set of constraints for an enshrinement system. Unfortunately these aren't enough to either be practical nor specify completely a solution. For example, it would be simple to guarantee the builder's reveal safety if we impose single slot finality and reaching BFT consensus before deciding if the CL block was present or not. This would inevitably require the slot time to be much larger than the current 12 seconds. These types of trade-offs are not part of the problem we set to solve itself, but rather are external constraints that we would like to continue satisfying. These constraints allow us to shape the solution that we are designing for, and if we gather enough of them, we may be able to tune in a single design.

4.1. Rounds of voting.

After the move to PoS, the Ethereum network adopted a rather complicated forkchoice algorithm [8,9] which is a combination of an actual fork choice rule with a finality gadget. At a high level, blocks are synced as follows.

1. A proposer produces and broadcast the block (with or without MEV Boost).
2. Attesters see the block and vote for it (or don't see it on time and vote for the parent).
3. Aggregators collect many of these attestations for the same block into one aggregate attestation

for more efficient verification.

1. Validators count these attestations and decide what their current head is.

The reason for 3) is that there are just too many attestations in the network to count in one single round, so the process is divided with this intermediate stage of aggregation. Questions like "has the block been received" or "is the block the current head", etc. depend on this attestation counting.

In ePBS, there are two different objects that are being revealed at different times, thus we can ask the same questions above for both the CL block and the corresponding execution Payload and in order to answer them, two rounds of voting should be counted. We see here the first trade-off

Either we aggregate the votes on both reveals or the slot time remains 12 seconds. Both cannot happen at the same time.

Remark.

This is not really a theorem but a strong belief. Aggregation requires at least 3 seconds for two rounds of broadcasting.

One argument to keep the slot time at 12 seconds is that this invariant leaks not only to CL and EL clients but into the application layer and tooling. I am aware of at least one contract that tracks slots by using the block timestamp [10]. Besides this, on the CL clients themselves, changing the slot time may be quite an invasive change gating every timing decision based on genesis time to before or after the fork. I will continue this write-up with the following design constraint

Constraint 1.

We keep the slot time at 12 seconds per slot.

This is not necessary indeed, but the only consequence of this constraint, in conjunction with the need to keep CL attestations aggregated, that I will assume is the following:

Constraint 2.

Payload attestations cannot be aggregated.

We have mentioned above that allowing for aggregation of both attestations and payload attestations (votes from validators as to whether the builder has acted honestly) would relatively easily solve for builder's safety. Constraining our design with the above two decisions make this problem harder to solve.

[

fc

1792×1024 465 KB

](https://ethresear.ch/uploads/default/original/2X/f/f167b33c9475afdffd76522b8c7668b9f85e3444.jpeg)

4.2. Forkchoice Weight.

This subsection includes perhaps the most contentious unresolved problem in any ePBS design. Assuming that we satisfy Constraints 1 and 2, what should be attesters for a payload be actually voting for?

The current way of voting in Ethereum is as follows. Validators are assigned to a particular slot to vote. During their slot, they wait until 4 seconds into the slot* and they vote at that time for the block that is the head of the chain in their view at the time. This can be kept exactly as is for ePBS. Validators during their slot will vote for their head view of the chain. Since they only need to validate consensus and not execution at this time, we can even deadline at 3 seconds instead of 4. But how about payload attestations? should the attester vote for its current head view or should it simply vote for the timeliness of the payload? should it also attest for the correctness of the payload?. Validators use the normal CL attestations to decide what their own head view will be. Should validators also count payload attestations to decide what their current head is?.

- This is simplifying, the honest guide indicates that validators should vote as soon as they see the block arrive.

There are several options here to decide, but given the Constraint 2, the committee size, and therefore the number of validators attesting for the builder's payload has to necessarily be restricted and much smaller than the CL committee. Thus far, given the current constraints our typical slot would look like this

1. At the beginning of the slot the proposer evaluates his view of the chain, chooses a bid and broadcasts his CL block.
2. 3 seconds into the slot validators attest to the presence of this block.
3. 6 seconds into the slot two events are happening in parallel.
4. The builder has been tracking attestations and reveals his payload.
5. Aggregators broadcast aggregates of all CL attestations.
6. The builder has been tracking attestations and reveals his payload.
7. Aggregators broadcast aggregates of all CL attestations.
8. 9 seconds into the slot a smaller committee attests for the payload (we have not decided how)

Thus the question is what do the payload attestations attest to and how are they counted by the proposer of the next slot and validators in general to decide what the head of the chain is?

(Block, Slot) voting.

A characteristic of Ethereum's existing fork choice mechanism is that any child of the current canonical head automatically assumes the canonical status, aside from exceptional cases involving self-reorganizing blocks, which are beyond the scope of this discussion. This transition occurs irrespective of the number of votes garnered by the current canonical head. If the chain's head is determined solely based on the CL block—that is, if it is designated as the head immediately following the CL block's revelation, without considering the status of its associated payload—then it becomes impossible to ensure the builder withholding safety

.

We have seen this already when we talked about mandatory honest reorgs

. A typical way of forcing mandatory honest reorgs is what is known as (block, slot)-voting. I will not get into details here, but it is sufficient to say that if a given block is seen timely by a fraction x

of the committee and late by 1-x

, then the votes in x

will count for the chain containing this block (that is both this block and its parent) but the 1-x

will count for the parent but not for the child. Also, if x < \frac{1}{2}

, equivalently x < 1-x

, validators will see the parent as the head of the chain and will vote as though the block never appeared. In particular, the next proposer will reorg the block. (Block, Slot) voting is not necessary for ePBS as long as we can enforce the constraint of mandatory honest reorgs

, however, for the sake of forkchoice computations below, I will assume that (Block, Slot) is in place, or rather that any block with less than 1/2

of the committee vote will be reorged by honest validators.

Remark.

The situation described above is not technically a reorg

, the late block is never included on-chain as the canonical head. In fact in this regard, (Block, Slot)

voting is better than making the honest reorg feature mandatory. Currently on Ethereum any proposer can freely cause a 1-slot reorg simply sending his block late, this may hurt certain services' UX.

PTC voting.

One possible design is to simply let the committee voting for the payload indicate whether the payload has been seen and its valid (we will see below some more nuances regarding this). That is, this committee has no forkchoice weight at all, they aren't capable of influencing the choice of the head block root, they simple assert if the head slot is empty or full with the nomenclature of Section 3.3.

This approach does not satisfy neither the builder reveal safety

nor the builder withholding safety

. Indeed, suppose the CL block for slot N

has received a fraction x

between \frac{1-PB}{2}

and \frac{1}{2}

of the committee vote, where PB

is the proposer boost that the proposer of N+1

will get for his timely block. This means that the parent block has received 1-x

(for simplicity we may assume no forks currently).

The builder does not see the CL block as head since 1-x > x

, and decides not to reveal his block, the proposer of N+1

can still base his block on N

and because

$$PB + x > 1-x \Leftrightarrow PB > 1 - 2x \Leftarrow PB = 1 - 2\frac{1-PB}{2} > 1-2x,$$

his block becomes canonical, thus violating the builder withholding safety

as long as PB > 0

. A similar analysis shows that if the block receives between \frac{1}{2}

and \frac{1+PB}{2}

, and the honest builder reveals his block, then the next proposer may decide to reorg the payload because

$$ PB + 1 - x > x \Leftrightarrow PB > 2x - 1 \Leftarrow PB = 2 \frac{1 + PB}{2} - 1 > 2x - 1, $$

thus violating the builder reveal safety

as long as PB > 0

.

We recall that proposer boost was included originally to mitigate balancing attacks and reorging attacks[11]. While it was shown that PB is not really a solution to balancing attacks, it does help reduce ex-anti and sandwich reorgs by validators that can split the view of the next proposer, so setting it to zero is not an option.

Usual voting.

We could allow the payload voting to actually decide what the current head state is by counting the votes directly as we count attestations in the LMD weight of forkchoice, but there is no known way of doing this without aggregation or weighing more the votes for the payload, that is, counting a payload vote by a multiple of the effective balance of the validator casting it. The problem of the latter approach is that builders are by nature a much more centralized set than validators, and by revealing or not, splitting the view of the attester's set, a reduced set of builders could cause a balancing attack.

Payload boosts.

Another option that we have mentioned above is to allow an honest builder to have some boost. Still in the context of PTC voting, suppose again the situation where the builder has seen a block have a fraction x<\frac{1}{2}

of the committee voting for it and decides to honestly withhold his payload. The builder can instead release a signed message signifying that he is withholding the release. Suppose the builder gets a withholding boost

(WB) for this message being seen by the PTC. This boost will count towards the parent block, thus the proposer of N+1

will be able to include his block based on N

only if

$$ PB + x > 1 - x + WB \Leftrightarrow PB + (2x - 1) > WB $$

Hence if WB \geq PB

then the next proposer cannot break the builder withholding safety

. On the other hand, as long as x > \frac{1}{2}

the builder cannot grief the proposer and thus proposer safety

is held.

Similarly, if we allow a reveal boost

(RB) for a payload that is revealed, we see that the next proposer is able to reorg a block that has received x > \frac{1}{2}

votes and the builder revealed, only if

$$ PB + 1 - x > RB + x \Leftrightarrow PB + (1-2x) > RB, $$

hence again as long as the RB \geq PB

the next proposer is not able to reorg the block.

If this boost is only valid during the previous slot of the proposal it diminishes the forkchoice attacks that builders can exert. This last boost was already proposed in [12] in a different context as LMD weight. In the context of PTC voting, the boost is applied if the PTC attests for the presence of the payload.

If combining PTC with payload boots, then the PTC needs to attest for three different outcomes:

1. No message from the builder has been seen on time.
2. The payload was withheld with a message from the builder saying so.
3. The payload was revealed timely.

The WB is applied if the PTC reaches agreement (say a threshold) in the second case, the RB is applied in the third case.

Remark.

One would think that having a WB would allow a builder to collude with the next proposer to facilitate a post-anti reorg. This has high chances of being the case, but at least not in the vanilla situation, that is if the block for slot N

has received x

of the committee and its parent 1-x

, then the block cannot be reorged as long as x \geq \frac{1+WB+PB}{2}

. This fraction would stand at 90% currently which is extreme, reducing (WB+PB)

to 40%

would still be a bit high with this fraction being 70%. Perhaps other mechanisms can be set in place like whenever WB

is applied then PB

is reduced.

4.3. Payment methods.

There are several ways of guaranteeing unconditional payments from the builder as well as different layers in which these payments can be carried out. Builders could include proofs of validity and proof of a transaction to the fee recipient of the proposer. They could escrow enough for a bid in a contract in the EL, etc. On the other hand, builders could just register as participants in the CL, either as validators or simply as different entities identified by a BLS address and with staked assets. There are various trade-offs between CL/EL for payments:

- Simplicity of design and implementation: having the builders be registered and staked in the CL, payment can be guaranteed immediately at the block header processing in the CL.
- Conversely, doing this in the EL requires some form of ZK proving the validity of the execution or a future inclusion of a transaction.
- Allowing payment on the EL does not require the builders to be staked, presumably allowing for a greater decentralized system of builders which simply submit bids when they see them available and escrow the funds just in time.

In my opinion, the simplicity of implementation and design outweighs any advantages that other systems may have. Specially in the presence of networking constraints that we will see below, it is very unlikely that any builder that is not being regularly polled by validators, will ever win a bid against dedicated builders with direct communication with proposers. Hence the code complexity that one saves by having the builders being in-protocol and payments handled in the CL inclines me to add another design constraint

Contstraint 3.

Builders are registered BLS addresses staked in the beacon chain.

This constraint simplifies much of our discussion as it allows for proper formal and rigorous definition of builder

along the lines of what we did for proposer in Section 2.1. In particular this allows a validator to trustlessly check that indeed a builder has the current balance available to fulfil the payment and accept/gossip the bid. Signature validation can also be done immediately at the gossip layer.

Having staked capital, specially if there is any non-trivial amount immediately forces upon us two extra constraints though:

Constraint 4.

Builders are allowed to be validators.

Constraint 5.

Max EB should be increased to the maximum value considered safe.

The first one is so that builders do not lose capital efficiency by having bid money stationary without earning interest. The second one is to not cap the maximum bid that can be processed by the system.

Remark.

Constraint 4 does not force builders to be validators, but only gives them the right if so they desire. This can be trivially implemented by having them as non-active validators.

Payment timing.

If builders are allowed to be validators, this immediately allows transfers of large value between validators in the form of large bids. This opens for a serious attack to the network if combined with slashings. As such, instead of immediate payment fulfilment this leads to another constraint

Constraint 6.

Payment is enforced by some form of churn so as to disallow large flows of stake.

A mechanism that is already in place with the Max EB increase proposal[13], is to use the deposit churn to enforce these payments. For most blocks except exceptional 3 digit ETH bids, payment would be immediate.

This is perhaps the principal drawback of handling payments in the consensus layer. If payments are fulfilled in the EL, they can be made immediate. The requirement for ZK mechanisms, new escrowing systems and the overall extra complexity, while at the same time the scarcity of blocks worth more than 1 ETH make for CL payments, delayed by churn, to be more appealing.

[

escrow

1792×1024 752 KB

](https://ethresear.ch/uploads/default/original/2X/b/b39a039525faacf285a85351c07e0dec5dd04996.jpeg)

Escrow.

Handling payments in the CL already takes care of escrowing funds to make sure the bid can be honored even if the payload is not revealed, guaranteeing unconditional payment

. If payment in handled in the EL, we can design a special purpose contract which can be proven against. Other mechanisms include top of block unconditional payments

[14] in a form of a transaction that is sent with the bid. However these types of payments would not satisfy the builder withholding safety

or make it require a refund in protocol.

A question that is raised by a design in which builders are validators, is that the need for large bids in certain blocks forces the existence of highly staked builders (and led ultimately to Constraint 5). However, this doesn't preclude small builders to be staked minimum amounts. Two natural questions are

Question 1.

Should the option of self-building be a requirement?

In a world in which builders are validators, requiring builders to be staked more than 32 ETH would preclude any current validator from self-building. This has several consequences. On one hand one could argue that it would remove for the most the notion of a trusted relay at least in its interaction with the proposer, in the sense that the signer of the payload can be identified as it can no longer be the proposer. This however takes the wrong approach to the design constraints we started above: instead of

A proposer that wants to sell his right to build a payload may do so without having to trust any entity.

it would seem that we would be aiming to "… must

do so without having…", something that ultimately can't really be enforced as proposers would always find ways of trusting parties off-chain if they so desire.

Although staking builders heavily presents advantages, the builder landscape remains predominantly centralized, with no signs of diversification, whether or not a substantial staking requirement is imposed. However, requiring builders to stake significant amounts enables the introduction of new slashing rules, including social slashing mechanisms. It could be argued that these measures may be somewhat ineffective, as the potential gains from a successful attack by a centralized building system could far outweigh any slashing penalties. There is a general agreement that requiring builders to stake amounts exceeding the 32 ETH entry-level for validators does not add value. This observation naturally leads to the following question:

Question 2.

Should builders be allowed to stake less than the ejection balance?

Implementing a small minimum stake requirement presents a mixed bag of outcomes. On the positive side, it enables smaller builders to engage with the network, particularly in blocks with minimal MEV, fostering inclusivity and diversity among participants. The primary challenge, however, lies in the need for exceptional handling of builders within the beacon state to address issues like quadratic leaks and extended periods without finality. During such times, ejecting validators becomes crucial to restore finality. A potential solution to this dilemma could involve allowing these builders to be exited validators who possess BLS withdrawal credentials (or a new withdrawal prefix) and maintain a low balance. Although these individuals cannot withdraw their stakes, they could still fulfill the role of builders, offering a pragmatic approach to integrating them into the ecosystem without increasing the spec complexity.

Personal note.

I was once strongly minded in staking builders heavily. I have changed my mind with regards to this given that it does not

bring large benefits. The right framing of the problem is about having the option to build trustlessly, not forcing it. I do think that we should strive to follow the no trusted advantage

principle of Section 3.5 and should severely limit the ways in which builders can affect the forkchoice status of nodes.

4.4. P2P and bid gossip

One of the problems that the MEV-Boost landscape is seeing is relay-builder vertical integration[15]. Latency is paramount to winning builder auctions, builders should adapt to send a bid up to the very last possible moment. If we only allow in-protocol bid to propagate through the p2p gossip network, we would directly violate the no trusted advantage

principle. This leads to another constraint

Constraint 7.

Proposers should be able to contact directly the builder and request bids. Builders should be able to open up public channels of direct communication to proposers.

This in fact would allow builders to be directly trustless vertically integrated relays. There would be no need for B&D investment and generation of trust as these would be already guaranteed in-protocol, thus in fact the builder will have an advantage to directly communicate with the proposer rather than dealing with another hop in the form of a relay. Relays today offer other services like bid cancellation

. These would also be available if the builder decides so in their own HTTP endpoint. CL clients would direct connections to a list of builders that would be user specified, or, for those that want to take a little extra latency, they could connect to external services that automatically update lists of builders. This would make the CL client themselves MEV-Boost, with all the benefits that this carries:

- Better code maintenance with access to a larger community of developers.
- Better testing environment reducing an extra coordination points when deploying devnets/testnets.
- Better user support as the CL client support system would directly cover their questions.

Of course this does not mean that we should remove a P2P layer for gossip as this allows for a larger set of honest watchover builders

to submit bids over P2P to set a base layer for the block value that the proposer would receive. Discouraging thus cartelization of the centralized builder set.

Bringing this further, we could have clients by default randomly every now and then (even user configurable) submit bids and prepare blocks automatically, guaranteeing liveness even if the full centralized builder set is offline and the proposers are not self-building at large.

[

censoring

1792×1024 907 KB

](https://ethresear.ch/uploads/default/original/2X/f/f7bd56c849d750b81b8bc97b743fdbf5e39ae39a.jpeg)

5. Relation with other designs.

5.1. Censorship resistance.

Given that the builder set is centralized by nature, several methods to mitigate censoring have been proposed. There is currently a proposal for forced inclusion lists being considered for Electra/Fujin [16] which has broad support and is mostly speced out so I refer to its design. I will here only comment on the relations with ePBS. Part of block validation for a CL block is the availability and validity of the IL. Part of the block validation of the payload is the satisfaction of the inclusion list. What happens when the CL block is included, guaranteeing availability for the IL in slot N

, but then the payload for N+1

, which should satisfy it, is not? In this case we could keep the IL satisfaction as a validity condition of any payload that is revealed descending from this block, or we could drop it and satisfy the next inclusion list. For example, if the CL block of N+1

was also included, and its builder has not revealed the payload, we have two ILs available, that of slot N

and that of slot N+1

. Dropping the IL from N

opens up to a censoring attack in which builders can simply censor transactions by not revealing one single payload. It is thus better to keep the IL satisfaction as a validity condition of the next block. This leads to a problem when there are multiple consecutive empty slots, as each CL proposer would include an IL and we would never catch up. We are thus faced with

Constraint 8.

If slot N

is empty

(thus not satisfying the IL from its parent block) then the proposer of N+1

should not

submit an IL, that is, its block validity is tied to the emptiness of the inclusion list within it.

This guarantees that if there is a chain of empty slots, only the last two ILs are available and still unfulfilled (that of the last full block and it's empty child). The network thus catches up eventually, and the first proposer after the first full block, is allowed to include a non-emtpy IL.

5.2. EL triggerable withdrawals.

If we adopt a design in which builders are staked in the beacon chain, we have seen from Constraint 5 that Max EB should be included, which in turn forces EIP 7002 which is already in track for Electra [17].

5.3. Delayed execution.

Some designs allow for the block in slot N

to only commit to the execution state of slot N-1

, so as to delay execution validation to the background, hence the current execution payload needs to be verified by the next slot.

This fits nicely with the question of whether the PTC should vote for timeliness and validity or just timeliness. In some regards, having the PTC only vote for timeliness, and requiring the parent block to be valid, is akin to importing an "optimistic block" into forkchoice, that may be removed in the future.

6. Some details on forkchoice.

In this section I just want to highlight some of the complexities of dealing with the presence of empty slots in forkchoice and LMD accounting. There's an interplay between the builder/proposer safety rules of Section 3 and the problem of forkchoice implementation complexity. A more detailed description (currently outdated) can be found in [18].

6.1. Payload reorgs.

Since we haven't agreed on a design for payload attestation accounting this section is highly speculative. I just want to highlight some of the features that many of these designs will have to deal with. On each block, two separate state transitions are carried out, once when importing the CL block and another when importing the EL block. One could represent these two blocks as separate nodes in forkchoice, but given that you cannot have the second without the first, it is better to keep them as a single node, with a marker of full/empty depending on the payload attester's voting (there are other designs where each of the two options would be kept with independent accounting according to how many payload attestations have been received, I am taking here the binary approach of PTC-like voting). For full blocks N-1 <- N

, the proposer of N+1

may

- continue the canonical chain and build on top of (N, full)

,

- try to reorg the payload of N

and build on top of (N, empty)

,

- try to reorg completely N

and build on top of (N-1, full)

,

- …

This, in addition with (Block, slot)

voting or similar as we have indicated as mandatory to satisfy the builder withholding safety

and possibly the payload boosts WB and RB make forkchoice LMD accounting a bit difficult. One way of counting would be as follows. Votes cast by attesters during slot N

for the CL block revealed during slot N

, only support descending chains that agree with the PTC decision in that slot. As an example consider the following situation

In this example the proposer of N+1

is attempting to reorg the payload from N

but not the CL block. Assume for simplicity that the full committee of N

has voted for the CL block N

(thus making it pretty hard to reorg). If the PTC has voted for the presence of the payload, then these votes will support the chain containing (N, Full)

but not the chain containing N+1

. Under these conditions, validators would see N+1

with weight given by PB, vs the head (N, Full)

with weight given by the full committee of N

plus the addition of RB, making it head.

In the same situation as above, votes during slot N

that have not seen N

on time would be counted for (N-1, Full)

but not for N

because of (Block, Slot)

voting. These votes would therefore not count for either versions of N+1

nor N

.

There may be some degenerate cases in which the view of the builder is not consistent with those of the committee attestters. This can happen in the case of split views. Consider for example a situation as above in which the voters of N

did not see it on time but just barely (say 49% saw it on time and there is no payload boosts involved). Suppose the builder saw the block on time and decided to reveal his payload and the PTC members saw the block with a commitment to this builder's payload so they voted to it's timeliness. The PTC committee would have voted for (N, Full)

attesting for the timeliness of the payload. We have seen above that this can be resolved in favor of the builder by including a RB that lasts into the next slot. But even with it, depending on its value, there may be split views.

There is another issue with forkchoice implementation complexity: the votes for N-1

should not support N

during N+1

, but how about in future slots if the chain of N

(either empty or full) advances?

6.2. Checkpoints.

One of the most complicated aspects of forkchoice is the interface between LMD and FFG. With regard to this, checkpoints are among the most important objects, for example an honest validator would only consider for head, descendants of its current justified checkpoint. A checkpoint (R, E)

currently consists of the data of a block root R

and an epoch E

. Given a block B

descending from the block with root R

, one can easily verify if it is a descendant of the checkpoint (R, E)

. One checks if in the chain containing B

the block with root R

is the last block before or at the beginning of the epoch E

.

With ePBS the situation changes as there are now two possible forkchoice nodes with root R

, that with and without a payload present. In principle a checkpoint should consist of (R, E, payload_present)

where payload_present

is a boolean indicating if the slot was full or empty. The actual beacon state is different after performing the state transition of including or not the payload. However, checkpoints in forkchoice are mainly used to obtain effective balances of validators, and in any case are a purely consensus construct. It is highly plausible that no safety concerns are included by considering the checkpoint being still a pair (R, E)

and omitting the payload presence. This in turn simplifies greatly forkchoice specification and implementation.

References

- [1] Proposer-builder separation | ethereum.org
- [2] Maximal extractable value (MEV) | ethereum.org
- [3] Equivocation attacks in mev-boost and ePBS
- [4] GitHub - flashbots/mev-boost: MEV-Boost allows Ethereum validators to source high-MEV blocks from a competitive builder marketplace
- [5] https://github.com/ethereum/consensus-specs/blob/1a33bf8a00bb593d61a9128c929b8dcacef0b437/specs/phase0/beacon-chain.md#validator
- [6] State of research: increasing censorship resistance of transactions under proposer/builder separation (PBS) - HackMD
- [7] https://github.com/ethereum/consensus-specs/pull/3034
- [8] [1710.09437] Casper the Friendly Finality Gadget
- [9] [2003.03052] Combining GHOST and Casper
- [10] Reducing challenge times in rollups
- [11] Change fork choice rule to mitigate balancing and reorging attacks
- [12] Two-slot proposer/builder separation
- [13] Increase the MAX_EFFECTIVE_BALANCE – a modest proposal
- [14] Relays in a post-ePBS world

- [15] Relay-Builders and Vertical Integration in MEV-Boost | by Aestus Relay | Medium
- [16] EIP-7547: Inclusion lists - Core EIPs - Fellowship of Ethereum Magicians
- [17] EIP-7002: Execution layer triggerable exits
- [18] https://github.com/potuz/consensus-specs/blob/f31929acd92efea3d3ac92ce7a8c6ea05bb2472c/specs/_features/epbs/design.md
- [19] Block vs. Slot Auction PBS — Julian Ma