Proof of Neutrality

Off-Chain model

runtime verification

Contents

Abstract

The PoN network has off-chain components that affect how the on-chain code is interpreted. We present a mathematical model of the expected behavior of the off-chain side of the PoN.

1 Agents

All off-chain agents of the PoN should also be on-chain parties. Even agents without an active on-chain role, such as relayers, should be acknowledged on-chain. That is, every part of the PoN network –proposers, builders, relayers, reporters, and hosted services– should be known on-chain.

The only agents that the model considers are builders, proposers and relayers. Reporters and hosted service are not modeled. Instead, we give an account of what constitutes a penalty. Penalties should then be validated by the hosted service and reported to the PoN on-chain side.

We define $ts \in \mathbb{Q}$ as the current timestamp in decimal precision. Note that we assume that all agents and states are in sync with this variable.

1.1 Builder

Let **Addr** be the set of all ECDSA addresses and assume a fixed PoN network. We identify any ECDSA address with a builder. That is, we assume that any address can interact with the PoN network as a builder. It is the PoN's duty to only process rightful interactions. As such, we will consider any potential builder address and discharge all of the builders' states to the on-chain state.

We will refer to the set of all builders as $\mathcal{B}$. However, formally we have that $\mathcal{B} = \mathbf{Addr}$.

We will also refer to the "empty" builder as $0_{\mathcal{B}}$. That is, $0_{\mathcal{B}}$ is a placeholder builder to indicate no actual builder (identified with the zero address).

1.2 Proposer

Let **BLS** be the set of all BLS keys. We identify proposers with BLS keys registered in the Ethereum consensus layer. We discharge all proposer bookkeeping to the on-chain state. Thus, the status (unregistered, registered, active, exit_pending, exited, kicked) and the validator's stake are queried to the on-chain part of the PoN.

We will refer to the set of all proposers as $\mathcal{P}$. However, formally we have that $\mathcal{P} = \mathbf{BLS}$.

We will also refer to the "empty" proposer as $0_{\mathcal{P}}$. That is, $0_{\mathcal{P}}$ is a placeholder proposer to indicate no actual proposer.

1.3 Relayer

Relayers are central to the model. They contain functions that allow for state updates (allowing to compute if certain conditions hold) and functions that represent the state of the off-chain PoN's side.

While relayers don't perform any state-updating action, they validate state updates and store an important part of the state.

The relayer structure is defined in Subsection 2.2, once the necessary definitions have been given. We will denote the set of all relayers by $\mathcal{R}$.

2 State Structures

The following are the necessary structures and functions to describe the off-chain state.

2.1 Bid structures

Bid structures are a formalism to capture the relevant information about a bid. Before defining what is a bid, we need to define its cryptographic components. A crucial component of the PoN is the RPBS scheme, which allows relayers to check that the block offered by a builder contains the promised payment. Thus, before defining a bid, we must define its cryptographic components.

2.1.1 Cryptographic structure

The cryptographic structure $\mathbb{C}$ contains the relevant cryptographic information for the RPBS scheme. This includes:

- Block header of the offered block for the slot.
- Proof of payment from the builder.
- Signed block header by the relayer.
- Signed proof of payment by the relayer.

After the builder submits a block, the relayer must check that the proof of payment guarantees the bid offer. If the check is successful, the relayer signs both the block header and the proof of payment. However, the signed block header and proof of payment are stored in the relayer structure, not on the bid structure.

With this in mind, we define

$$\mathbb{C} = \mathbb{H} \times \mathrm{RPBS}$$

where:

- $\mathbb{H}$ is the set of hashes.
- RPBS is a placeholder set for the necessary components of the RPBS proof of payment. We don't explicitly model nor describe what the necessary components are.

We denote $0_{\mathbb{C}} = (0_{\mathbb{H}}, 0_{\mathrm{RPBS}})$ as the null element of the set $\mathbb{C}$, and analogously for each of it's components.

Given $rpbs \in \mathbb{C}$, we identify its components as follows:

$$\begin{aligned} rpbs = (&header \in \mathbb{H}, && \text{Header of the offered block} \\ &PoP \in \mathrm{RPBS}) && \text{Proof of payment provided by the builder} \end{aligned}$$

We will refer to each component of a $rpbs$ by $rpbs.component$ (e.g., $rpbs.header$ is the first component of the tuple $rpbs$).

2.1.2 Bid

We define the set of bids as

$$Bid = \mathcal{B} \times \mathcal{R} \times \mathbb{N} \times \mathbb{N} \times \mathbb{C}.$$

Given a bid $bid \in Bid$, we will identify its components as follows:

$$\begin{aligned}
bid = (&builder \in \mathcal{B}, && \text{Builder author of the bid}\\
&relayer \in \mathcal{R}, && \text{Relayer to which the bid is submitted}\\
&promise \in \mathbb{N}, && \text{Reward promised for block inclusion}\\
&slot \in \mathbb{N}, && \text{Slot to bid for}\\
&rpbs \in \mathbb{C}) && \text{Cryptographic components of the PoN}
\end{aligned}$$

We will refer to each component of a bid bid by $bid.component$ (e.g., $bid.builder$ is the first component of the tuple bid).

Also, we denote the empty bid as

$$0_{Bid} = (0_{\mathcal{B}}, 0_{\mathcal{R}}, 0_{\mathbb{N}}, 0_{\mathbb{N}}, 0_{\mathbb{C}}).$$

2.1.3 Bid sets

- Given a slot $s \in \mathbb{N}$, the *bid set of s* is

  $$Bid_s = \{b \in Bid \mid b.slot = s\}$$

  that is, the bids corresponding to the slot number.
- Similarly, the *relayer bids* for a slot s and a relayer r is defined as

  $$Bid_{s,r} = \{b \in Bid_s \mid b.relayer = r\}.$$

  That is, given a relayer r, $Bid_{s,r}$ is the set of bids that r got for slot s.

Now we can define the strategy choice function of a relayer r as:

$$\begin{aligned} c_r : \mathbb{N} &\longrightarrow Bid \\ s &\longmapsto b \in Bid_{s,r} \end{aligned}$$

Note that we're implicitly evaluating the first argument of c to r, the explicit type is

$$c : R \to \mathbb{N} \to Bid.$$

2.2 Relayer structure

We identify relayers with a fixed finite set $R \subset \mathbb{N}$. We define the relayer structure as

$$\mathcal{R} = (R, c, schedule, check\text{RPBS}, validBid, signedPoP, signedHeader, request, receive).$$

As mentioned in 1.3, relayers have two kinds of functions: computing and state functions. Computing functions are used to deterministically compute an output for any given input, whereas state functions will be updated depending on the actions of the other agents. Section 4 explains how state functions can be modified.

Computing functions:

- c choice function representing the selecting strategy implemented by the relayer (type given later).
- $schedule : \mathbb{Q} \to \{\mathrm{BID}, \mathrm{SIGN}, \mathrm{BROADCAST}\}$ schedule allocation of a given timestamp. Represents if a timestamp falls under the bid, requesting and signing, or broadcasting blocks time windows.
- $check\mathrm{RPBS} : \mathrm{RPBS} \times \mathbb{Q} \times \mathcal{B} \to \mathbb{B}$ payment commitment check. Given a proof of payment, a promised amount, and a builder, the function checks if the proof of payment for the builder paying that amount is valid.
- $validBid : Bid \to \mathbb{B}$ a bid validation function. Each relayer may have different criteria to filter which bids are valid and which are not.

State functions:

- $signedPoP : Bid \to \mathbb{S}$ records the signed proof of payment for a given bid. If $signedPoP(b) \neq 0_{\mathbb{S}}$ it means the proof of payment submitted to the relayer is valid.
- $signedHeader : Bid \to \mathbb{S}$ records the signed block header by the relayer for a given bid. If $signedHeader(b) \neq 0_{\mathbb{S}}$ it means the bid submitted to the relayer is valid.
- $request : \mathbb{N} \to \mathbb{B}$ records if the designated proposer requested a block for a given slot.
- $receive : \mathbb{N} \to \mathbb{S}$ records the signed block returned by the designated proposer for a given slot.

We refer to the "empty" relayer as $0_{\mathcal{R}}$. That is, $0_{\mathcal{R}}$ is a placeholder relayer to indicate no actual relayer. We denote the set of active relayers as $\mathcal{R}^+ = \{r \in \mathcal{R} \mid r \neq 0_{\mathcal{R}}\}$.

Notation remark: Throughout this document, we will use the following convention. Given a set **Carrier** and a structure $S = (\textbf{Carrier}, f_1, f_2, \ldots)$ with $f_i : \textbf{Carrier} \to A_i$, we will identify $s \in S$ with $s \in \textbf{Carrier}$ and $s.f_i$ with $f_i(s)$ for $s \in S$. E.g., for $r \in \mathcal{R}$, we have that $r.receive : \mathbb{N} \to \mathbb{S}$ and $receive : R \to \mathbb{S}^{\mathbb{N}}$. In the above definition, we implicitly omitted the "$R \to$" part of the type for clarity. Relayers can be thought of as indexes for the functions described above.

2.3 Auction set and function

An auction for a slot is the set of bids offered for a slot. The auction function represents an indexing by slot number of the relevant auction information for said slot.

We define the auction set $A = \left(2^{Bid}\right)^{\mathbb{N}}$.

That is, A is the set of functions from the natural numbers to sets of bids 2^{Bid}. Each function $Auction \in A$ represents a possible state of every auction of the PoN network. For each slot, $s \in \mathbb{N}$, $Auction(s)$ represents a possible auction for that slot. We will denote $Auction(s)$ by $Auction_s$.

3 State Definition

In this section, we define the functions that update the state variables. But before defining which are the state-changing functions, we must define what is the state.

3.1 On-chain state

The off-chain state should always be driven by the on-chain state. However, we consider the on-chain state as a separate state from the PoN off-chain state. We represent only the necessary components of the on-chain state that are needed from the off-chain side. We do not model how these on-chain components are updated. That is, we assume they are updated independently of the off-chain state.

The on-chain parameters are modeled as state variables or functions. We distinguish between general beacon chain parameters on the one hand, and parameters specific to the PoN on the other hand.

General beacon chain parameters:

- $slot \in \mathbb{N}$ current slot of the beacon chain.
- $proposerForSlot : \mathbb{N} \to \mathcal{P}$ chosen proposer by the consensus layer for a given slot. We model it as $\forall n \in \mathbb{N}\colon n \leq slot,\ proposerForSlot(n) \neq 0_{\mathcal{P}}$ and $\forall n \in \mathbb{N}\colon n > slot,\ proposerForSlot(n) = 0_{\mathcal{P}}$.
- $proposerStake : \mathcal{P} \to \mathbb{Q}$ effective balance of a proposer.
- $balance : \textbf{Addr} \to \mathbb{Q}$ ETH balance of an account.
- $EIP1559Fee : \mathbb{N} \to \mathbb{Q}$ EIP1559 priority fee for a given slot.
- $signedBlocks : \mathbb{N} \to \mathbb{N}$ number of signed blocks detected by the consensus layer for a given slot.

PoN-specific parameters:

- $proposerStatus : \mathcal{P} \to \{\text{UNREGISTERED}, \text{REGISTERED}, \text{ACTIVE}, \text{EXIT\_PENDING}, \text{EXITED}, \text{KICKED}\}$ status in the PoN of a given proposer. Note that $proposerStatus(0_{\mathcal{P}}) = \text{UNREGISTERED}$.
- $isBuilderOperational : \mathcal{B} \to \mathbb{B}$ returns $\texttt{BuilderRegistry} :: \texttt{isBuilderOperational}(builder)$ for $builder \in \mathcal{B}$ at ts. Note that $isBuilderOperational(0_{\mathcal{B}}) = \perp$.
- $paidInSlot : \mathbb{N} \times \mathcal{B} \to \mathbb{Q}$ amount paid by a builder for a given slot to the payout pool through the `PayoutPool::receive()` function.

We consider this its own transition system that runs in parallel with the PoN state.

3.2 Off-chain state

We define the Proof of Neutrality Network state as a tuple $(\mathcal{R}, Auction)$ where $\mathcal{R}$ is the current relayer structure and $Auction : \mathbb{N} \to 2^{Bid}$ is the current auction function.

The state represents the configuration of the relayers and how many bids were offered for a given slot. The state transition functions described in this section represent how can the state of the PoN be updated.

Note that the beacon chain state is not included in the off-chain PoN state. This is because we want to separate the off-chain workings of the PoN from the current state of the beacon chain.

3.2.1 Derived sets

From the off-chain state, we can derive the following useful sets.

Relayers with bids

We define the set of relayers who received bids offers for a slot s as

$$\mathcal{R}_s^{bids} = \{bid.relayer \in \mathcal{R} : bid \in Auction_s\}.$$

Relayers with proposers requests

We define the set of relayers who received a request by a proposer for a slot s as

$$\mathcal{R}_s^{request} = \{relayer \in \mathcal{R} : relayer.request(s) = \top\}.$$

Relayers with received signed headers

We define the set of relayers who received a signed header for a slot s as

$$\mathcal{R}_s^{receive} = \{relayer \in \mathcal{R} : relayer.receive(s) \neq 0_{\mathbb{S}}\}.$$

3.3 Initial state

The initial state of the PoN is as follows:

- $\forall r \in \mathcal{R}\ \forall s \in \mathbb{N}\ r.request(s) = \bot$ No relayers received requests.
- $\forall r \in \mathcal{R}\ \forall s \in \mathbb{N}\ r.receive(s) = 0_{\mathbb{S}}$ No relayers received signed block headers.
- $\forall r \in \mathcal{R}\ \forall b \in Bid\ r.signedPoP(b) = 0_{\mathbb{S}}$ No proof of payment is valid.
- $\forall r \in \mathcal{R}\ \forall b \in Bid\ r.signedHeader(b) = 0_{\mathbb{S}}$ No bids are valid.
- $\forall n \in \mathbb{N}\ Auction_s = \emptyset$ No bids have been offered for any slot.

The rest of the relayers' functions compute a state-independent output for any given input.

4 Agents actions

In this section, we describe the different actions that agents can do. Currently, the model contains three state-updating actions: offering bids to a relayer (builder), requesting bids from a relayer (proposer), and sending back to relayers signed bids (proposer).

When there's a state update $(\mathcal{R}, Auction) \longrightarrow (\mathcal{R}', Auction')$ by $\mathcal{R}'$ we refer to the relayer structure with the same relayers (R) but with (possibly) different functions. Thus, for any $r \in \mathcal{R}$ we have that $r \in \mathcal{R}'$. When there's room for confusion, we'll represent $r \in \mathcal{R}'$ by $r' \in \mathcal{R}'$. The key difference between r and r' is that we may have (for example) $r.receive(n) \neq r'.receive(n)$ for a slot $s \in \mathbb{N}$, but they represent the same relayer.

4.1 Builder actions

The only action a builder can take in the PoN network from the off-chain side is to submit a bid for a slot.

4.1.1 Bid offer

The function `bid` takes as input $bid \in Bid$, where $bid.builder$ is the builder submitting bid to the PoN. It updates the state as follows:

$$(\mathcal{R}, Auction) \xrightarrow{\texttt{bid}(bid)} (\mathcal{R}', Auction')$$

Preconditions:

- $bid.slot = slot$ (bid is for the correct slot)
- $proposerStatus(proposerForSlot(p)) = \text{ACTIVE}$ (slot proposer is active in the PoN)
- $isBuilderOperational(bid.builder) = \top$ (builder proposing the bid is operational)
- $bid.relayer.schedule(ts) = \text{BID}$ (PoN relayer is taking bids at the time)
- $bid.promise > 0$ (builder promises positive payment)[1]
- $|\{b \in Auction_s : b.builder = bid.builer\}| < 2$ (the builder has already bid the maximum amount of times)[2]
- $bid.relayer.validBid(b) = \top$ (the relayer to which the bid was submitted considers it valid)[3]

[1]We only require the amount to be non-zero because the state is updated with the bids offered, not with bids that would win the previous leading bid.

[2]We don't parametrize the maximum number of bids to keep the model simple, but the number 2 is not crucial to the design. Also note that in the model, a builder can call the `bid` function arbitrarily many times with the same bid, but the state doesn't change by calling it multiple times.

[3]The *validBid* function only represents the check of custom validation requirements chosen by the relayer, such as the builder not being blacklisted.

- $balance(bid.builder) \geq bid.promise$ (the builder has enough balance to fulfill the promised reward)
- $bid.relayer \neq 0_{\mathcal{R}}$ (the relayer is not the zero relayer)
- $bid.relayer.check\mathrm{RPBS}(bid.rpbs, bid.promise, bid.builder) = \top$ (the block offered contains the promised payment to the PoN by the builder)

Postconditions:

- $bid.relayer.signedPoP(bid) = \mathrm{sign}(bid.relayer, bid.rpbs.PoP)$ (the relayer signs the proof of payment)
- $bid.relayer.signedHeader(bid) = \mathrm{sign}(bid.relayer, bid.rpbs.header)$ (the relayer signs the block header)
- $\forall s \neq slot \in \mathbb{N}\ Auction'_s = Auction_s$ (all auctions for slots different than $slot$ remain the same)
- $Auction'_{slot} = Auction_{slot} \cup \{bid\} \in 2^{Bid}$ (the set of bids offered for slot is added bid)

We do not consider block-broadcasting to be a state-updating builder action since it only updates the on-chain state, and this model is only concerned with the off-chain state.

4.2 Proposer actions

Before defining the state updating functions for the proposers, note that we model bid inspection from the chosen proposer as block requesting. I.e., proposers request blocks from the relayers they consider choosing the block header from. Thus, they must only respond back to one of these relayers with a signed header.

Also, proposer bookkeeping (registration, activation, quitting, etc.) is all discharged to the on-chain status. We model the off-chain apparatus as not storing any such information and only querying the on-chain contracts to fetch it. Thus, the only interaction of a proposer with the off-chain is to request and sign block headers.

4.2.1 Request block header

The function `requestBlock` takes as inputs $p \in \mathcal{P}$ and $r \in \mathcal{R}^+$ and updates the state as follows:

$$(\mathcal{R}, Auction) \xrightarrow{\texttt{requestBlock}(p,r)} (\mathcal{R}', Auction)$$

Preconditions:

- $proposerForSlot(slot) = p$ (the proposer who requested the block is the one chosen by the consensus layer for the current slot)
- $proposerStake(p) = 32$ ETH (the effective balance of validator p is 32 ETH)

- $proposerStatus(p) = \text{ACTIVE}$ (the proposer is registered as active in the PoN)
- $r \in \mathcal{R}^{Bids}_{slot}$ (the relayer has received bids for the blockspace)
- $r.schedule(ts) = \text{SIGN}$ (the relayer is currently forwarding block headers)

Postconditions:

- $\mathcal{R} \setminus \{r\} = \mathcal{R}' \setminus \{r\}$ (all relayers except for the requested one remain the same)
- $r.request(slot) = \top$ (the requested relayer records that it has been requested for the slot)

4.2.2 Sign block header

The function `signBlock` takes as inputs $p \in \mathcal{P}$, $r \in \mathcal{R}^+$, $b \in Bid$, and $signature \in \mathbb{S}$. Then, the state is updated as follows:

$$(\mathcal{R}, Auction) \xrightarrow{\texttt{signBlock}(p,r,b,signature)} (\mathcal{R}', Auction')$$

Preconditions:

- $r.request(slot) = \top$ (the proposer requested a block from the relayer for that slot)
- $r.schedule(ts) = \text{SIGN}$ (the relayer is currently accepting signed block headers)
- $proposerForSlot(slot) = p$ (the proposer is the one chosen for *slot* by the consensus mechanism)
- $proposerStatus(p) = \text{ACTIVE}$ (the proposer is active in the PoN)
- $proposerStake(p) = 32\text{ETH}$ (the effective balance of the proposer is 32 ETH)
- $r.c(slot) = b$ (the signed bid needs to be the one chosen by the relayer for *slot*)[4]
- $r.receive(slot) = 0_{\mathbb{S}}$ (the relayer has not received a signed header before for the slot)
- $r.signedPoP(b) = \text{sign}(r, b.rpbs.PoP)$ (proof of payment must be validated by the relayer)
- $r.signedHeader = \text{sign}(r, b.rpbs.header)$ (bid must be validated by the relayer)

Note that the proposer can return an incorrect signature. That is, it may happen that $signature \neq \text{sign}(p, b.rpbs.header)$.

Postconditions:

[4]This condition ensures proposers can only submit signed bids to the relayer it was requested from.

- $\mathcal{R} \setminus \{r\} = \mathcal{R}' \setminus \{r\}$ (all relayers different form r remain unchanged)
- $r.receive(slot) = signature$ (the only change in r is updaing the *receive* function)

Note that it is not possible for a proposer to execute multiple times the `signBlock` function for the same relayer.

However, even if different block headers are signed, the builders responsible for those blocks should be exempt from any penalty since it's the proposer's wrongdoing.

There could be a race condition for reporting a builder for not having broadcasted a signed block. Namely, if another builder's block was included before the consensus layer declared it invalid because of multiple signed blocks for the same slot, a reporter could report other builders who also received a signed block. However, in practice, slots need to be finalized to be reportable. This should allow enough time for the consensus layer to invalidate the block and thus for the PoN to only allow the reporting of the proposer.

4.3 Relayer actions

In the model, there are no relayer actions. While the actual relayers do perform actions such as forwarding requested and signed block headers, we have included these actions in the actions that the builders or proposers perform. This makes sense since the relayers never initiate an action that does not follow a builder, proposer, or reporter action. Hence, the only role relayers play in this model is that of information verification and storage.

5 Agent Faults and Penalties

Both the builder and the proposer are subjected to the following rules. Non-compliance with these rules will result in a report. The spirit of the rules is: the model is partially responsible for allowing correct updates of the state. For example, not allowing unregistered builders to submit bids or proposers not selected for the current blocks to request block headers. However, responsibility for the correct update of the state is also shared by the parties that update the state (builder and proposer). Failure to do so will result in penalties. These are the rules that builders and proposers must follow when updating the state.

Note that depending on whether the payout pool was paid a positive amount or none, different rules may apply.

Also, if we have that $proposerStatus(proposerForSlot(s)) \neq$ ACTIVE, none of the rules apply. After the rules definition, we indicate in what scenarios each rule must be satisfied.

5.1 Fault rules

Given a state $(\mathcal{R}, Auction)$, if there was a demand for the blockspace of a slot $s < slot$ (i.e., if $Auction_s.B \neq \emptyset$), a proposer should have requested a block header for at least one relayer and returned it signed, and the chosen builder should have broadcasted the block.

5.1.1 Block request

The only rule regarding block requesting is that a proposer must request a block if there is demand for the slot's blockspace.

Block request rule: If there was a demand for a slot's blockspace, it must exist a relayer r from which $\mathit{proposerForSlot}(s)$ requested a block.

$$\texttt{BlockRequestRule}(s) \coloneqq \mathit{Auction}_s \neq \emptyset \rightarrow \exists r \in \mathcal{R}_s^{bid} : r.\mathit{request}(s) = \top$$

or, equivalently

$$\texttt{BlockRequestRule}(s) \coloneqq \mathcal{R}_s^{bid} \neq \emptyset \rightarrow \mathcal{R}_s^{request} \neq \emptyset$$

Failure to update the state as described will incur a penalty[5] of

$$2 \cdot \sum_{i=1}^{5} \frac{\mathit{EIP1559Fee}(s-i)}{5}.$$

5.1.2 Block signature

There are two rules regarding block signature. The first is that one block signature must be given back. The second is that the signature must be correct.

Block signature rule:[6] If there was a demand for a slot's blockspace and the designated proposer requested blocks from the relayers, it must return a signed block header to a relayer.

$$\begin{aligned}\texttt{BlockSignatureRule}(s) \coloneqq &\mathit{Auction}_s \neq \emptyset \ \wedge \ \texttt{BlockRequestRule}(s) \rightarrow \\ &\exists r \in \mathcal{R}_s^{bids} : r.\mathit{receive}(s) \neq 0_{\mathbb{S}}\end{aligned}$$

or, equivalently

$$\texttt{BlockSignatureRule}(s) \coloneqq \mathcal{R}_s^{bid} \neq \emptyset \ \wedge \ \texttt{BlockRequestRule}(s) \rightarrow \mathcal{R}_s^{receive} \neq \emptyset.$$

Failure to update the state as described will incur a penalty of

$$2 \cdot \sum_{i=1}^{5} \frac{\mathit{EIP1559Fee}(s-i)}{5}.$$

Correct signature rule: If a signature is returned, it must be a correct signature. That is, it must correspond to one of the blocks offered in the auction.[7]

$$\begin{aligned}&\texttt{CorrectSignatureRule}(s) \coloneqq \\ &\mathit{Auction}_s \neq \emptyset \ \wedge \ \texttt{BlockRequestRule}(s) \ \wedge \ \texttt{BlockSignatureRule}(s) \rightarrow \\ &\big(\forall r \in \mathcal{R}_s^{receive} \rightarrow (\mathrm{signer}(r.\mathit{receive}(s)) = \mathit{proposerForSlot}(s) \ \wedge \\ &\qquad \mathrm{signedObject}(r.\mathit{receive}(s)) = r.c(s).\mathit{rpbs}.\mathit{header})\big)\end{aligned}$$

[5] Throughout this document, we show the value that is approximated by the protocol. To see how the actual computation is done, we refer to the section "Proposer economic penalty" of the audit report.

[6] Note that a signed block header can only be forwarded to the relayer when the schedule is on SIGN. Thus, we only need to check if the block was submitted. An alternative version would be to check if the timestamp where the block was submitted was on time.

[7] The functions "signer" and "signedObject" are ad-hoc functions to compute the signer and the object being signed from a given signature.

Failure to update the state as described will incur a penalty of

$$2 \cdot \sum_{i=1}^{5} \frac{EIP1559Fee(s-i)}{5}.$$

5.1.3 Block broadcasting

The builder must broadcast the block. That is, if a block is finalized, the builder must have paid at least the promised amount to the pool. The only exception is if the consensus layer detected that the designated proposer signed multiple block headers to the same slot. In such case, no builder will be penalized, and the proposer will be kicked from the PoN.

Single signing rule: The single signing rule asserts that proposers shall only sign one block header per slot. To avoid any race condition on reporting builders when there is a double sign, as mentioned above, we assume that for every slot $s < slot$, the consensus layer will have realized any double signing penalties. If this happened, the proposer would be kicked out of the PoN, and no proposers would be penalized.

In practice, this assumption is realized by only allowing to report finalized blocks, hence giving the consensus layer enough time to detect double-signing events.

$$\begin{aligned}\texttt{singleSigned}(s) \coloneqq \quad & Auction_s.B \neq \emptyset \\ \wedge\ & \texttt{BlockRequestRule}(s) \\ \wedge\ & \texttt{BlockSignatureRule}(s) \\ \wedge\ & \texttt{CorrectSignatureRule}(s) \rightarrow signedBlocks(s) = 1\end{aligned}$$

Note that this rule only covers double signing when the proposer has successfully followed the PoN workflow, and as such, proposers could be flagged as fault for not broadcasting blocks. That is, we have that if the antecedent of **`singleSigned`**(s) and **`singleSigned`**(s) hold, then $|\mathcal{R}_s^{receive}| = 1$. However, the converse doesn't hold in general.

If a PoN proposer signs multiple blocks without interacting with the PoN, the proposer will only be kicked if it's slashed by the consensus layer.

Also, note that if a proposer has signed multiple blocks but returned them wrongly to the corresponding relayers, the **`singleSigned`** rule will be successful. This doesn't exempt the relayer from being kicked off the PoN. Since if the consensus layer is made aware of the double signing, the proposer will get slashed and thus kicked. However, we prioritize the incorrect placement of the signed headers because the builders will not be able to broadcast the blocks.

Failure to satisfy this rule will result in the kick of the proposer:

$$proposerStatus(proposerForSlot(s)) = \text{KICKED}.$$

Block payment rule: If a proposer has signed a bid, the payout pool must be paid by the builder at least the promised amount.

$$
\begin{aligned}
\texttt{blockPayment}(s) := \quad & Auction_s \neq \emptyset \\
\wedge\ & \texttt{BlockRequestRule}(s) \\
\wedge\ & \texttt{BlockSignatureRule}(s) \\
\wedge\ & \texttt{CorrectSignatureRule}(s) \\
\wedge\ & \texttt{singleSigned}(s) \\
\rightarrow\ & \forall r \in \mathcal{R}_s^{receive}\ paidInSlot(s, r.c(s).builder) \geq r.c(s).promise
\end{aligned}
$$

Note that the if the antecedent holds we have that $|\mathcal{R}_s^{receive}| = 1$.

Failure to satisfy this rule will result in a penalty of

$$2 \times r.c(s).promise.$$

Note that `blockPayment` covers two different faults. Namely, not broadcasting a block or underpaying the payout pool. However, if the designed proposer didn't satisfy any of the rules that apply to it, the `blockPayment` rule will be vacuously satisfied by builders.

5.1.4 Proposer health

The PoN only accepts healthy proposers to participate and thus earn gains. If a proposer was not of the highest quality, the PoN would kick such a proposer and never distribute the buffered rewards to it.

Healthy Proposer Rule: If a proposer is unhealthy, it can be kicked from the PoN.

$$\texttt{proposerHealth(p)} := proposerStatus(p) = \text{ACTIVE} \rightarrow proposerStake(p) < 32$$

Note that the rule does not depend on any particular slot. Thus, it can be checked, and the penalty enforced at any time.

Also note that this rule also covers the cases when a proposer is slashed or has withdrawn since, as a consequence of these situations, the effective balance will be below 32 ETH.

Failure to satisfy this rule will result in the kick of the proposer:

$$proposerStatus(p) = \text{KICKED}.$$

5.2 Rule checking

The following table summarizes under which circumstances must each rule be checked. In every case, we assume that $proposerStatus(proposerForSlot(s)) =$ ACTIVE. Otherwise, the rules are not checked.

Rule / Scenario	$paidInSlot(s) = 0$	$paidInSlot(s) > 0$
`BlockRequestRule`(s)	check	don't check
`BlockSignatureRule`(s)	check	don't check
`CorrectSignatureRule`(s)	check	don't check
`singleSigned`(s)	check	don't check
`blockPayment`(s)	check	check

In short, if the payout pool is not paid, only one rule will not be satisfied, determining the agent at fault. Otherwise, only the `blockPayment` rule applies since we must ensure that the payout pool was paid correctly if the proposer used the PoN.

Note that if the proposer requested blocks from the PoN but didn't sign any of the requested blocks, the `blockPayment` rule is satisfied since that is an allowed action by the PoN.

The following algorithm describes how to determine if a rule was broken for a given slot $s < slot$:

```
Input: Slot $s \in \mathbb{N}$
Output: Infraction[$rule] | BroadcastFailed | NoInfraction
if $proposerStatus(proposerForSlot(s))$ = ACTIVE then
    if $\exists builder \in \mathcal{B}\ paidInSlot(s, builder) > 0$ [a] then
        if blockPayment($s$) then
            return NoInfraction
        else
            return Infraction[blockPayment]
        end
    else
        if BlockRequestRule($s$) then
            if BlockSignatureRule($s$) then
                if CorrectSignatureRule($s$) then
                    if singleSigned($s$) then
                        return BroadcastFailed
                    else
                        return Infraction[singleSigned]
                    end
                else
                    return Infraction[CorrectSignatureRule]
                end
            else
                return Infraction[BlockSignatureRule]
            end
        else
            return Infraction[BlockRequestRule]
        end
    end
else
    return NoInfraction
end
```

[a]Note that is not required for the builder to be registered in the PoN network.