Hyperledger Besu CIP

Tim Beiko [*] Danno Ferrin[†] Grace Hartley [‡] Hart Montgomery [§]

1 Introduction

In this document, we present the Besu CIP proposal in our own words. We first outline some core background material that is essential for understanding the proposal. Then we delinineate the terms of the proposal.

To summarize, the Ethereum Foundation is launching an execution layer client incentive program (ELCIP) of which Hyperledger Besu is one execution client. This program will provide execution-layer client teams with locked ETH in the form of live validators to be released according to certain milestones, including post-merge performance and progress towards enabling withdrawals from the beacon chain.

The Ethereum Foundation (EF) has designed the program with the following guiding principles in mind:

- The Ethereum Foundation reserves the right to change, update or terminate the program at any point. The EF thus acts as a check and balance that Besu maintainers, their parent organizations, and the Hyperledger Foundation (HLF) continue to meet the requirements of the program.
- In summary, the program offers teams locked ETH in the form of live validators to be released according to certain milestones. Alongside the "principal" (i.e. locked Eth) portion of the program, teams' validators will also accrue staking rewards and transaction fees.
- While fees and rewards should be shared across the entire set of Besu maintainers working on mainnet, access to the locked ETH is reserved for organizations who demonstrate longstanding support for Besu's mainnet work and the Ethereum network generally.

We explain the details behind these principles throughout the course of this document.

2 Background

We start by discussing some background material. We note that this should be familiar to Ethereum experts, but those who are not familiar with Ethereum's change to proof of stake may want to consult the following material for reference.

2.1 Terminology

We begin by providing some informal definitions of terms from the Ethereum ecosystem that will be used in this document. We emphasize that some of these terms have been redefined by the community over the lifespan of Ethereum, so we encourage readers who are unfamiliar with the current state of Ethereum to take note.

[*]The Ethereum Foundation.
[†]Hedera.
[‡]Consensys.
[§]The Hyperledger Foundation.

Eth1 vs Eth2. Eth1 defines the "execution layer" of Ethereum, while Eth2 defines the "consensus layer." This was recently formalized in a blog post from the Ethereum Foundation [Eth22d]. Note that this differs from previous notation. As an example, Hyperleger Besu will live (mostly) at the Eth1 layer since its primary focus is as an Ethereum execution client. On the other hand, ConsenSys's Teku [Con22] lives at the Eth2 layer.

Grant Period. The *grant period* is the interval of time which passes between phases of the program. At the beginning/end of each grant period, the HLF will report back to the EF as described later, and categorizations of maintainers and organizations will be reassesed. After the starting period, which ends upon the activation of validator withdrawals on the Beacon Chain, initially this period will be set for six months, but the EF can determine this as they see fit, reserving the option to have totally arbitrary grant periods that change on the fly. The EF emphasizes that they would prefer to stick to a schedule, however. We note that the grant period has been referred to as the *CIP vesting period* in previous documentation. There are more precise details on the grant period later in this document.

Validator. A validator is a node that decides consensus on the proof of stake Ethereum chain. Very roughly, the validators take the role that miners have in proof of work blockchains.

Epochs. Rather than count the length of the chain in blocks, after the Bellatrix update [Eth22a], the length of the Ethereum chain will be counted in *epochs.* An *epoch* is a bundle of up to 32 blocks that validators propose and attest to over a period lasting roughly 6.4 minutes.

Splitter Contract. A splitter contract is a smart contract that takes as input some cryptocurrency (in this case, Ethereum), and splits it proportionally in a predefined formula to various entities (represented by their public keys). An example of a splitter contract is mirror splits [MX22] which has been suggested by the EF as a good splitter contract to deploy. We emphasize a splitter contract holds the cryptocurrency, but does not forward it automatically. It gives each address a claim over a share of the funds held, and keeps track of withdrawn funds, but recipients need to actively claim their funds.

2.2 Proof of Stake Consensus

A *consensus protocol* is a necessary feature of every blockchain because it protects the blockchain against potentially malicious adversaries. The Ethereum Bellatrix upgrade [Eth22a] changes the core consensus algorithm of Ethereum from proof of work to proof of stake. It would take an entire textbook to explain proof of stake consensus (we recommend [Shi20] as a general reference on modern consensus algorithms), so we do not attempt that here. We emphasize, though, that this proposal is written in a way that requires some understanding of proof of stake consensus in order to understand the rationale behind some of the design decisions. The goal of this proposal is to incentivize the development of an Ethereum mainnet client (Besu) and ensure that some validators run Besu on the Ethereum mainnet.

Informally speaking, proof of stake consensus involves participants on the network called validators "voting" on blocks, which are bundles of transactions. Note that the block structure poses some extra constraints: a transaction may be invalid at a position in a block but valid at another (e.g. a user sends 1 ETH from address A to B, which previously had 0 ETH, and then 1 ETH from B to C, the second txn would be invalid if ordered before the first in a block). Each participant's vote is weighted by the amount of stake they hold. If a participant misbehaves, they may lose their stake in a process called *slashing.* It is also worth noting that if a participant is offline, they lose a small amount of their stake, roughly equal to what they would have gained if they were online, called an inactivity penalty.

There is a huge amount of literature on proof of stake consensus [BG17, KRDO17, DPS19] and we encourage interested readers to peruse this literature for detailed explanations. We note that the Ethereum proof of stake specification is defined in a github repository [Eth22b].

2.3 Keys

In cryptocurrencies, ownership is typically represented by the ownership of signing keys. Ethereum is no exception. In our case, there are three primary sets of keys with which we need to deal.

Withdrawal Keys. The *withdrawal keys*, referred to in the specification as *withdrawal_credentials*, control the ability to withraw staked Ethereum, as well as receive validation rewards. These are best described in the aforementioned specification [Eth22b].

Currently there are two forms of withdrawal credentials supported:

- BLS signature [BLS01] key pairs over the curve BLS12-318 [BLS02, Bow17]. Note that even though they have the same acronym, BLS signatures are a cryptographic signature scheme (supporting aggregate signatures), and BLS curves are pairing-friendly elliptic curves on which BLS signatures can be built. We will avoid acronym confusion to the best of our ability.
- Eth1 addresses (in the form of 20-byte strings) denoted by *eth1_withdrawal_address*. As stated in [Eth22b], after the merge of the current Ethereum application layer into the Beacon Chain, withdrawals to *eth1_withdrawal_address* will simply be increases to the account's ETH balance that do not trigger any EVM execution.

We note that withdrawal credentials can be "cold": that is, they are not needed for frequent validator operations and only need to be used when moving staked ETH or staking rewards. The program will start by using BLS credentials for withdrawal keys (denoted $0x00$ credentials by the EF). We can update these to $0x01$ credentials (i.e. Eth1 addresses), and may have to in order to enable partial withdrawals. That said, recipients of withdrawal keys will need to secure a 0x01 credential no matter what because even if you make a withdrawal with a BLS credential, you need to specify which address to withdraw to and those must be Eth1 addresses. In both cases, recipients will need the ability to use the keys in cold storage, as the $0x00$ credential would be required to sign a withdrawal message, and the Eth1 private key is obviously required to move funds once withdrawn.

Validator Keys. The *validator keys* [Eth22b] are keys for the BLS signature scheme [BLS01] implemented with the BLS12-381 curve [Bow17]. The validator keys are used to actively sign data throughout the operation of the validator and must be "hot." These are the traditional signature keys that one associates with blockchain activity.

We emphasize that the signing key of the validator key set must be kept secure, as it cannot be changed during the operation of the vaildator (although the stake can be withdrawn and used to create a new validator).

Key Derivation. The withdrawal and validator keys are generated using key generation algorithms that the EF calls "mnemonics." There are two such algorithms, one for validator keys and one for withdrawal keys. They are similar but slightly different.

Validator Key Derivation:

- All validator active keys are derived from the key derivation algorithm defined in EIP 2333 [Bee22a] using the EIP 2334 "signing key" derivation path [Bee22b] which is given to the node operator to generate the active keystores to do live validation for all validators in the program.

Withdrawal Key Derivation:

- All validator withdrawal credentials are derived from this mnemonic using the EIP 2334 ”withdrawal key” derivation path [Bee22b].
- This seed is withheld by the EF. Upon meeting milestones in the program (described in much more detail in section 5), sets of keys from this key generation protocol will be given to those in the program for full ownership.

- As noted above, this means that two parties then have the keys (EF and recipient) which is not ideal. The EF reserves the right to change the withdrawal process so that withdrawals are enabled using a“change of key operation.” In the event that this happens, the EF will likely do a change of key to an eth1 address designated by the recipient rather than by sending raw keys.

Transaction Fee Recipient Keys. When the validators construct blocks, they can send the transactions fees from the block (including gas fees) to a recipient that is not necessarily themselves. The *feeRecipient* is a 20-byte string that designates an Eth1 address and is a part of the *ExecutionPayload* of the Beacon Chain specification. The value of this *feeRecipient* is thus a *transaction fee recipient key.*

The execution payload structure is defined in the execution engine specification [Eth22c] as well as the Bellatrix update description [Eth22a] which is the update implementing “the merge.” We note that the *feeRecipient* is equivalent to the *beneficiary* from the original Ethereum yellow paper [W+14].

3 Mainnet Maintainers and Patrons

We now define *maintainers* and *patrons* of Besu and classify them appropriately. An open-source *contributor* is anyone who has contributed code or technical artifcacts (i.e. documentation) to a project. A *maintainer* is a contributor who has made substantial contributions to a project, to the point that the community has decided to promote them to a leadership role. Maintainer responsibilities are varied across projects but generally include being one of the deciding votes on whether or not code is merged into the main branch of the codebase. The requirements to become a maintainer also vary from project to project; for Besu, all things about maintainers are defined here [Bes22].

3.1 Mainnet Maintainers

We next define mainnet maintainers. To do this, we first define what exactly constitutes mainnet work.

Mainnet Work. Below, we list some things that do and do not constitute mainnet work. We note that the EF ultimately has full control over what exactly constitutes mainnet work and deciding what is or is not valid mainnet work is up to the EF alone. However, we note that Ethereum mainnet development work includes (without limitation) contributions to features, performance, and stability required for Ethereum mainnet operation of the Besu client. This work includes supporting work necessarily deriving from this effort, such as build maintenance, code cleanup, documentation, test maintenance and development, and so forth. This also includes work on EIPs and appropriately submitted proposals intended for Ethereum Mainnet deployment, up until such a time as they are known to be excluded from Ethereum Mainnet deployment.

Some examples of work that is *not* mainnet work would include work relating to Ethereum Classic development, GoQuorum compatibility, private transactions, private transaction enclaves, permissioning, etc., and supporting work deriving exclusively from these activities.

Mainnet Maintainers. Informally, a mainnet maintainer is an active maintainer (listed in the Besu maintainers.md file [Bes22]) whose contributions are primarily focused on Ethereum mainnet development or supporting Ethereum mainnet development. It is expected that mainnet maintainers will make substantial contributions to mainnet development, including things like implementation of new features relevant to mainnet (e.g. sync, complex EIPs), significant performance improvements, and criticial security fixes.

While the above criteria is illustrative, it is informal. Formally speaking, the EF will be the sole arbiter of who is classified as a mainnet maintainer. At every grant period (see section 2.1), the following process will occur:

- Any active Besu maintainer (listed in the maintainers list [Bes22]) can apply for mainnet maintainer status. Any applicant who is not an active Besu maintainer will be immediately eliminated.

- To begin, the HLF staff will create a file *mainnet-maintainers-DATE.md* with *DATE* denoting the start of the grant period in a Besu github directory decided by the Besu maintainers. This will be done far enough in advance of a grant period to allow for a comfortable amount of time for both maintainers to apply and the EF to judge the applicants.
- To apply, an active maintainer will list their name in the above file, as well provide links to their substantial mainnet contributions.
- After a period (determined and announced by the EF), the EF will then mark each maintainer application as accepted or rejected in the .md file. In order to encourage contriubtors that haven't met the standard, the EF may provide brief explanations of rejections. The EF may also do this through private communications with the mainnet maintainers rather than in public.
- The maintainers with applications accepted in this way will be recognized as the mainnet maintainers for that particular grant period.

The EF has the right to change the mainnet maintainer recognition process however they see fit, but they should notify the Besu community reasonably in advance if this is happening.

3.2 Besu Patrons

Individual contributors and organizations that have demonstrated a long-term pattern of substantial contributions to Besu may be categorized (by the EF) as *patron organizations* or *patron maintainers*. As we explain below, the EF has the sole authority to determine patron status.

Patron Maintainer. A *patron maintainer* is someone who the EF has identified as being a long-term, substantial contributor to mainnet development on Besu. There is no formal process for becoming a patron maintainer. Anyone wishing to become a patron maintainer should contact the EF and have a discussion with them.

Patron Organization. A *patron organization* is an organization that the EF has identified as being a long-term, substantial contributor to mainnet development on Besu. Like a patron maintainer, there is no formal process for becoming a patron organization, and this category exists at the discretion of the EF.

The initial patron organizations of the Besu CIP will be ConsenSys and the HLF.

3.3 Company vs. Individual Maintainers

Some Besu maintainers contribute on behalf of a company, and do their work on company time. Others are either independent contractors or contribute in their own free time. We refer to the former as *company maintainers* and the latter as *individual maintainers.*

While the HLF and Besu treat these classes of maintainers no differently, the Besu CIP will reward these classes of maintainers differently: funds for company maintainers will be delivered to the company (which may or may not decide to give them to the maintainers), while funds for individual maintainers will go directly to the maintainer. We explain this later in more detail.

4 The Splitter Contract

A splitter contract will be used for distributing some of the rewards from the program. More precisely, the splitter contract will divide the non-withdrawal key rewards (transaction fees and validator rewards) between mainnet maintainers, node operators, and the HLF. The EF is responsible for creating and updating the splitter contract; this gives them full control over how to allocate the reward funds. We emphasize that they can change this allocation at a later date to whatever they desire.

The initial splitter contract distribution will be as follows:

20% **: Node Operator.** The node operator is the entity or entities responsible for managing and running the validator nodes. Initially, this will be ConsenSys for the Besu CIP program. We defer to section 5 for more information on the node operators. The node operators may spend these fees/rewards at their sole discretion and without limitation. No other compensation will be provided for the node operation service.

The node operator will provide access to the nodes running canary validators to Hyperledger mainnet developers and all maintainers performing mainnet work for development and debugging purposes. Access to performance nodes may be restricted to node operators, at the node operator's discretion. There is an expectation that if a contributor receives access to nodes, they will then be added to the respective monitoring channels for those nodes.

20%**: Hyperledger Foundation.** Another twenty percent goes to the Hyperledger Foundation. The Hyperledger Foundation is required to spend any money received here on things that benefit Besu mainnet work. The exact judgment of what constitutes something that benefits mainnet work will be left to the HLF. As a check against this responsibility, at every grant period, the HLF will report back to the EF the expenditures of these fees.

Some examples of items explicitly encouraged by the EF include (but are not limited to):

- Taxes associated with the reception of funds
- Services consumed by the Hyperledger Besu project, such as:
  - Continuous Integration
  - Cloud Hosting fees for development (i.e. testnet nodes)
  - Source code hosting
  - Any other such services directly related to development of mainnet capabilities in Hyperledger Besu
  - Note: These costs are separate from any incurred by any other participating organization
- Sending maintainers to speak at and promote Hyperledger Besu at Hyperledger events (such as the Global Forum and Members Summit) and Ethereum Foundation events (such as DevCon and research/development workshops)
- Posting Bounties for work on Hyperledger Besu mainnet or mainnet related work
- Spending that would directly promote interoperability with Besu Mainnet functionality
- Other spending that would directly promote and improve Hyperledger Besu's mainnet activities

60%**: Mainnet Maintainers.** The remaining 60% of the splitter contract funds are allocated to the current mainnet maintainers (i.e. those determined at the beginning of the current grant period). Initially, if there are N mainnet maintainers in a period, then $\frac{1}{N}$ of the total funds will be allocated towards each maintainer. Funds of company maintainers will be allocated to their employer, and funds of individual maintainers will be directly allocated to the maintainers themselves. Before the merge happens, the initial list of mainnet maintainers will be finalized.

The mainnet maintainers and their employers do not have any restrictions on how they can spend this money.

5 Program Structure

In this section, we define the structure of the main progrmam. We emphasize that the program aims to provide long-term support and incentives for teams towards maintaining reliable clients and a healthy network overall. We note that the description here mirrors the text of [Bei22]. Furthermore, the text in [Bei22] should be regarded as the official text with respect to what is mentioned in this section until it is deprecated.

For client teams to be eligible, they should already be contributing to the general development of Ethereum and intend to support the upcoming transition to proof of stake. Throughout the program, teams will need to maintain certain levels of performance to be eligible for the rewards. We explain which maintainers and organizations are eligible for which rewards later in this document.

The Hyperledger Besu client, unlike other Ethereum clients, is maintained by several organizations and is part of the Hyperledger Foundation. This document, and the values referenced, represent the entirety of the incentive program for all teams maintaining Hyperledger Besu. Once again, we emphasize that the EF reserves full discretion about how funds from the program are distributed across the various teams contributing to Hyperledger Besu.

The overall process of the program can be summarized as follows:

- The EF makes deposits into accounts controlled by withdrawal keys.
- The EF transfers control of the validator keys to the node operators.
- The node operators run the validators.
- The EF releases the withdrawal credentials to patron organizations and maintainers in waves.

We explain each of these steps in more detail below. Note that none of these steps explicitly require HLF involvement; however, the success of the program (and the release of the withdrawal credentials) is dependent on the Besu client meeting various requirements which implicitly involves the HLF. We note that the program will be structured so that *each release of withdrawal credentials* and *the recipients of the transaction fees for each period* will be determined by separate, individual grants. In other words, the EF will issue a new grant for each period. The grants will be identical unless it is determined that anything needs to change.

5.1 Parameter Definitions

We begin by defining some parameters that will be used in the following section. We define them here so that they may be potentially changed in updated versions of this document easily.

Name	**Value**	**Description**
$NUM\_PERFORMANCE$	128	Number of validators monitored for performance
$NUM\_CANARIES$	16	Number of canary validators
$NUM\_VALIDATORS$	$NUM\_PERFORMANCE$ $+NUM\_CANARIES$	Total number of validators
$INITIAL\_RELEASE$	32	Number of validators to release at initial major milestone
$TIMED\_RELEASES$	$6, 10, 14, 15, 22,$ $26 + NUM\_CANARIES$	Number of validators to be released each grant period after $INITIAL\_RELEASE$
$METRICS\_WINDOW$	8192	Number of epochs over which success metrics are observed
$MAX\_PROBATION\_WINDOW$	32768	Maximum number of epochs that the client can be in probation before the EF can partially or fully remove the client from the incentivization.

5.2 Step 1: The EF Makes Deposits

After Hyperledger Besu has formally agreed to join the CIP, and a node operator, responsible for running and monitoring validators, has been selected by the EF, the EF creates $NUM\_VALIDATORS$ 32-ETH deposits. For exactly how this process occurs, please see [Eth22b]. We note that ConsenSys has already been selected as the initial node operator.

Note that the total ETH at stake in the client incentivization plan is equal to $NUM\_VALIDATORS * 32$. In consultation with the node operator, a formal start date for this program will be determined where teams will gain control of validators, sometime before "The Merge" occurs (and after March 1, 2022, which has already passed by the time of this writing).

5.3 Step 2: The EF Transfers Validator Keys

After step 1, there will be $NUM\_VALIDATORS$ sets of validator keys, each of which is a (signing key, public key) tuple controlled by a single seed[1]. These keys must be securely transferred to the client team node operator.

This seed is transfered to the client via one of the following:

- Using asymmetric encryption (e.g. PGP) via a known/validated public key of the recipient client.
- Read verbally 25% at a time over 4 encrypted calls of various platforms.
- Through an alternatively negotiated, cryptographically secure means.

The client node operator then generates $NUM\_VALIDATORS$ using the seed and key generation algorithm and verifies that the key pairs are correct (more precisely, that each private key maps sequentially to the batch of validator public key deposits made in their name). The EF retains the seed in cold storage in the event that validator keys must be used to exit validators from the program.

5.4 Step 3: The Validators Run

At this point, the deposits have been made and the keys have been transferred: the node operator is in charge of the management of the associated validators until withdrawal credentials (the private keys) are released.

Specifically, the client node operator must use Hyperledger Besu as an execution-engine and is responsible for choosing and maintaining a consensus-engine throughout the incentivization period. The cliend node operator has the discretion to run, add or change which consensus layer it uses with Hyperledger Besu. The node operator will determine when and how to onboard the right consensus layer. The team will begin running with Teku, but will evaluate additional consensus layers on an as needed basis.

Performance of the client's validators can be assessed simply by viewing chain metrics, but additional node performance metrics might be requested.

Once the merge has occurred (and "real" transactions happen on the beacon chain), the node operator will send transaction fees to the splitter contract defined in section 4. In other words, the node operator will designate the $feeRecipient$ in the $ExecutionPayload$ to be the splitter contract.

5.5 Step 4: Withdrawal Credentials Released

Eventually the EF will release withdrawal credentials to patron organizations (and potentially maintainers) as it sees fit upon Hyperledger Besu meeting pre-defined milestones via the transfer of the underlying privkeys for the validator withdrawal credentials. We again emphasize that the EF reserves full discretion over which organizations are eligible to receive withdrawal credentials. The addition of organizations which are eligible to receive withdrawal credentials has no impact on the total amount of validators included in the incentive program.

[1] The EF refers to this seed as a "mnemonic."

When a wave of validators is released, this ends the obligation of both the node operator and the withdrawal credential receipient to the EF for those validators. The withdrawal credential receipients are free to choose to continue to validate, to exit, to withdraw, to spend the entire credential buying Hart beer, and so forth. The validators are required, however, to send any validator rewards earned to the “splitter” contract before exiting or withdrawing.

An exception is the HLF: they must spend their withdrawal credentials on work that benefits Besu mainnet development as defined earlier.

Note that if recipients choose to continue to validate after the withdrawal credentials are released, they are no longer required to allocate transaction fees/validator rewards to the splitter contract. In the case that withdrawal credential recipients are not the node operator (i.e. the HLF and ConsenSys), we leave it up to these entities to determine smooth processes for withdrawing credentials.

These keys will be PGP encrypted and transferred in batches.

6 Program Metrics

In this section we discuss the milestones and success metrics of the program.

6.1 Program Milestones

Due to the dynamic nature of the ever evolving Ethereum roadmap, simplicity is favored in the choice of milestones. The milestones will determine the grant periods, which again are at the sole discretion of the EF.

A wave of credentials are released when withdrawals from the beacon chain are enabled, with a minimum period of one year between the launch of the Besu CIP and the complete release of the first wave of credentials (in other words, the first grant period will be a year).

If withdrawals from the beacon chain are enabled within or before the first year of the Besu CIP, the first wave of credentials will be released monthly, in equal tranches, from the first month after withdrawals are enabled, to the one year mark of the program. For example, if withdrawals are enabled 6 months after the start of the program, then 1/6th of the first tranche will be released on months 7, 8, 9, 10, 11 and 12 (note that we are indexing on month 1 as the start month, not month 0). Otherwise, the first wave of credentials will be released when withdrawals are enabled. Subsequent waves are released over time if Hyperledger Besu continues to meet expectations. Specifically, the milestones are as follows:

- Release $INITIAL\_RELEASE$ validators at the time at which withdrawals from the beacon chain are enabled ($WITHDRAWALS\_ENABLED\_TIME$).
- for $i, NUM\_VALIDATORS \in \mathit{enumerate}\,(TIMED\_RELEASES)$, release $NUM\_VALIDATORS$ validators at time $WITHDRAWALS\_ENABLED\_TIME + (i+1) * 6_{\text{months}}$ if Hyperledger Besu continues to exhibit successful metrics.

6.2 Success Metrics

While the program exists entirely at the discretion of the EF, there are some explicit validator performance rquirements that, if not met, will automatically result in the termination of the program.

We note that the first $NUM\_PERFORMANCE$ validators of the deposited validators are tracked by the EF to assess metrics. The last $NUM\_CANARIES$ validators–the canary validators–are free to be used for testing, experimental relases, and so forth. While canary validators are not expected to constantly meet the success metrics, they are still subject to the slashing rules.

We next define some terms in a way that allows for modular changes:

Name	**Value**	**Description**
MIN_ACCEPTABLE_BALANCE	31.75 ETH	Minimum acceptable balance of client validators
MIN_ATTESTATION_PERCENT	95	Minimum acceptable percentage of attestations created by client validators
MIN_BLOCK_PERCENT	95	Minimum acceptable percentage of blocks created by client validators

Hyperledger Besu clients run by the node operators must meet the following metrics:

- Validators on average do not drop below *MIN_ACCEPTABLE_BALANCE* (over the time length of a probation window).
- Validators have at least *MIN_ATTESTATION_PERCENTAGE* percentage of expected attestations included on chain over any *METRICS_WINDOW* epoch period.
- Validators have at least *MIN_BLOCK_PERCENTAGE* percentage of expected blocks included on chain over any *METRICS_WINDOW* epoch period.

Moreover, client teams are expected to actively participate in research and development of crucial network upgrades, including but not limited to implementation of a state management solution (e.g. state expiry). The EF is solely responsible for determining whether this metric has been met.

Above all else, the EF expect client teams to actively work toward ensuring a robust and healthy network. The EF recognizes that in some scenarios these metrics are not entirely in the control of a single client (e.g. large portion of the network offline for an extended period of time due to issues with another client). In most such cases, the *METRICS_WINDOW* has been selected to be long enough to account for issues and recovery, but in such exceptional scenarios, the EF will also take into account exogenous factors outside of Hyperledger Besu's control.

In the context of this plan, validator top-ups (adding more Eth, up to 32 Eth, to a validator with less than 32 Eth) are against the rules and should generally be avoided. If in some scenario a top-up would benefit the health of the network, the EF and the Hyperledger Besu commnity can discuss and reformulate the metrics/milestones accordingly.

6.3 Probation

If the Hyperledger Besu nodes drop below the success metrics, the program's status moves into probation. During a probationary period, the Hyperledger Besu community and the node operators have *MAX_PROBATION_WINDOW* epochs to get metrics back to successful standards, and during a probationary period no withdrawal credentials will be released. The amount of time spent in probation pushes back the release of any withdrawal credentials by at least that amount of time.

If metrics remain in probationary status for more than *MAX_PROBATION_WINDOW* epochs, the EF can at their discretion partially or fully remove Hyperledger Besu from the incentivization program and partially or fully exit the program's validators.

6.4 Slashing

In the event that one or more of the designated Hyperledger Besu validators is slashed, such a validator is removed from the incentive program.

If the event is relatively isolated and quickly remedied, the EF can at its sole discretion choose to place a maximum of 16 of the slashed ETH per slashed validator back into the program to be released at the final milestone.

If the slashable event is exceedingly large, negligent, or repeated, the EF can at its discretion partially or fully remove Hyperledger Besu from the incentivization program and partially or fully exit the program's validators.

Note: Performance and canary validators are both subject to the slashing rules.

6.5 Consensus Layer Dependencies

While the Hyperledger Besu community is fully responsible for ensuring that their operation is run in a performant and non-slashable way, we recognize that there is a limit to what execution layer teams can do to mitigate issues on the consensus layer (and vice-versa). Specifically, this means we expect Hyperledger Besu and the node operator to adopt best practices with regards to running their validators, but will not penalize them in the case of a widespread consensus-layer issue. Best practices when running validators include:

- Ensuring that Hyperleger Besu can interoperate with most/all major consensus clients, at least on canary validators.
- Ensuring that Hyperleger Besu's failures are decorrelated from the rest of the network, both by relying on diverse consensus layer clients and hosting setups.
- Ideally ensuring that Hyperleger Besu's validators are split across > 1 consensus client in case of a consensus client-specific issue.
- Ensuring that Hyperleger Besu has the ability to switch from one consensus client to another in the case of a consensus client-specific issue.

7 Extra Terms

This plan is an opt-in additional incentivization plan for execution-layer clients. Participation in this plan and the amount of locked funds available in the plan will have no bearing on future client grant decisions. Prerequisites of participation in this plan are successful participation in multi-client testnets and generally demonstrating production readiness at all times.

Additionally, the EF expects the major stakeholders contributing to Hyperledger Besu to agree to participate in the program, and come to consensus on distribution schemes for incentives once received.

In general and especially in the event of exceptional and unforeseen scenarios concerning Hyperledger Besu, the teams contributing to Besu, the Hyperledger Foundation, the Ethereum roadmap, and/or the Ethereum mainnet, the EF is solely responsible for deciding how to award withdrawal credentials and/or restructure the terms of this incentive plan at any time.

Such exceptional scenarios include, but are not limited to, the following:

- Separate teams merging into one
- Teams contributing to Hyperledger Besu splitting into two
- Teams contributing to Hyperledger Besu ceasing the maintenence of a component (e.g. validator client) or the entirety of their software
- Node operators having to be replaced
- Hyperledger Besu, or a subset of it which supports the Ethereum mainnet, no longer being part of the Hyperledger Foundation
- Ethereum roadmap radically changing such that the milestones no longer reflect achievable goals
- Ethereum mainnet has extended issues with stability, finality, or otherwise proper function
- Ethereum mainnet undergoes a contentious hardfork

References

[Bee22a] Carl Beekhuizen. Eip-2333: Bls12-381 key generation, 2022. https://eips.ethereum.org/EIPS/eip-2333.

[Bee22b] Carl Beekhuizen. Eip-2334: Bls12-381 deterministic account hierarchy, 2022. https://eips.ethereum.org/EIPS/eip-2334#eth2-specific-parameters.

[Bei22] Tim Beiko. Hyperledger besu execution client incentive program, 2022. https://notes.ethereum.org/@timbeiko/besu-cip-updated.

[Bes22] Hyperledger Besu. Maintainers, 2022. https://github.com/hyperledger/besu/blob/main/MAINTAINERS.md.

[BG17] Vitalik Buterin and Virgil Griffith. Casper the friendly finality gadget. *arXiv preprint arXiv:1710.09437*, 2017.

[BLS01] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the Weil pairing. In Colin Boyd, editor, *ASIACRYPT 2001*, volume 2248 of *LNCS*, pages 514–532. Springer, Heidelberg, December 2001.

[BLS02] Paulo SLM Barreto, Ben Lynn, and Michael Scott. Constructing elliptic curves with prescribed embedding degrees. In *International conference on security in communication networks*, pages 257–267. Springer, 2002.

[Bow17] Sean Bowe. Bls12-381: New zk-snark elliptic curve construction, 2017. https://electriccoin.co/blog/new-snark-curve/.

[Con22] Consensys. Teku: The ethereum 2.0 client for institutional staking, 2022. https://consensys.net/knowledge-base/ethereum-2/teku/.

[DPS19] Phil Daian, Rafael Pass, and Elaine Shi. Snow white: Robustly reconfigurable consensus and applications to provably secure proof of stake. In *FC 2019*, LNCS, pages 23–41. Springer, Heidelberg, 2019.

[Eth22a] Ethereum. Bellatrix: The beacon chain, 2022. https://github.com/ethereum/consensus-specs/blob/dev/specs/bellatrix/beacon-chain.md.

[Eth22b] Ethereum. Ethereum consensus specification: Phase 0 – honest validator, 2022. https://github.com/ethereum/consensus-specs/blob/dev/specs/phase0/validator.md.

[Eth22c] Ethereum. Ethereum execution engine api, 2022. https://github.com/ethereum/execution-apis/blob/main/src/engine/specification.md.

[Eth22d] Ethereum. The great renaming: what happened to eth2?, 2022. https://blog.ethereum.org/2022/01/24/the-great-eth2-renaming/.

[KRDO17] Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part I*, volume 10401 of *LNCS*, pages 357–388. Springer, Heidelberg, August 2017.

[MX22] Mirror-XYZ. Mirror splits, 2022. https://github.com/mirror-xyz/splits.

[Shi20] Elaine Shi. Foundations of distributed consensus and blockchains, 2020. https://www.distributedconsensus.net.

[W$^+$14] Gavin Wood et al. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151(2014):1–32, 2014.