Special thanks to @potuz for the discussions.

To achieve strong network censorship resistance, this post outlines the essential modifications required for implementing the forward inclusion list with its dedicated gas limit, focusing on the client's viewpoint. The implementation is guided by specific assumptions detailed in the consideration section. For a deeper understanding, it's advisable to review the research document by Mike and Vitalik.

The state transition and fork choice adjustments draw heavily from Potuz's ePBS work, while the p2p and validator alterations are primarily influenced by my ePBS work. The execution API changes take inspiration from NC (EPF)'s work. Credit is duly attributed to all the esteemed authors who contributed to these concepts and developments.

Before delving into the specification modifications, let's rewind and provide a high-level summary of the forward inclusion list:

High Level Summary

Proposing by Proposer for Slot N:

- Proposers for slot N submit a signed block.
- In parallel, they broadcast pairs of summaries and transactions for slot N+1.
- Transactions are lists of transactions they want to be included at the start of slot N+1.
- Summaries include addresses sending those transactions and their gas limits.
- Summaries are signed, but transactions are not.

Validation by Validators for Slot N:

- Validators only consider the block for validation and head if they have seen at least one pair (summary, transactions).
- They consider the block invalid if the inclusion list transactions are not executable at the start of slot N and if they don't have at least 12.5% higher than the current slot's maxFeePerGas.

Payload Validation for Slot N+1:

- The proposer for slot N+1 builds its block along with a signed summary of the proposer of slot N utilized.
- The payload includes a list of transaction indices from block of slot N that satisfy some entry in the signed inclusion list summary.
- The payload is considered valid if: 1) execution conditions are met, including satisfying the inclusion list summary and being executable from the execution layer perspective. 2) consensus conditions are met, there exists a proposer signature of previous block.

[

1980×646 120 KB

](https://ethresear.ch/uploads/default/original/2X/b/baa81969f0c13b7e4d1fc75a49b255b6889f25b2.png)

Key Changes Spec Components:

Beacon Chain State Transition Spec:

- New inclusion_list

object:

Introduce a new inclusion_list

for proposer to submit and nodes to process.

- Modified execution_payload

and execution_header

objects:

Update these objects to align with the inclusion list requirements.

- Modified beacon_block_body

:

Modify the beacon block body to cache the inclusion list summary.

- Modified process_execution_payload

function:

Update this process to include checks for inclusion list summary alignment, proposer index, and signature for the previous slot.

Beacon Chain Fork Choice Spec:

- New is_inclusion_list_available

check:

Introduce a new check to determine if the inclusion list is available within the visibility window.

- New notification action:

Implement a new call to notify the Execution Layer (EL) client about a new inclusion list. The corresponding block is considered invalid if the EL client deems the inclusion list invalid.

Beacon Chain P2P Spec:

- New Gossipnet and validation rules for inclusion_list

:

Define new rules for handling inclusion list in the gossip network and validation.

- New RPC request and respond network for inclusion_list

:

Establish a new network for requesting and responding to inclusion list.

Validator Spec:

- New Duty for inclusion_list

:

Proposer to prepare and sign the inclusion list.

- Modified Duty for beacon_block_body

:

Update the duty to prepare the beacon block body to include the inclusion_list_summary

.

Builder and API Spec:

- Modified payload attribute SSE endpoint:

Adjust the payload attribute SSE endpoint to include payload summaries.

Execution-API Spec:

- New get_inclusion_list

:

Introduce a new function for proposers to retrieve inclusion

- New new_inclusion_list

:

Define a new function for nodes to validate the execution side of the inclusion list for N+1

inclusion

- Modified forkchoice_updated

:

Update the forkchoice_updated

function with a payload_attribute

to include the inclusion list summary as part of the attribute.

- Modified new_payload

:

Update the new_payload

function for EL clients to verify that payload_transactions

satisfy payload.inclusion_list_summary

and payload.inclusion_list_exclusions

.

[

1742×880 163 KB

](https://ethresear.ch/uploads/default/original/2X/6/6c43ba8730ad0e999f51b6a91f41a5da0d64ebc8.png)

(execution API usages around inclusion list)

Execution Spec:

- New Validation Rules:

Implement new validation rules based on the changes introduced in the Execution-API spec.

New Reference Objects

MAX_TRANSACTIONS_PER_INCLUSION_LIST

= 16

MAX_GAS_PER_INCLUSION_LIST

= 2**21

MIN_SLOTS_FOR_INCLUSION_LIST_REQUEST

= 1

class InclusionListSummaryEntry(Container): address: ExecutionAddress gas_limit: uint64

class InclusionListSummary(Container) slot: Slot proposer_index: ValidatorIndex summary: List[InclusionListSummaryEntry, MAX_TRANSACTIONS_PER_INCLUSION_LIST]

class SignedInclusionListSummary(Container): message: InclusionListSummary signature: BLSSignature

class InclusionList(Container) summary: SignedInclusionListSummary transactions: List[Transaction, MAX_TRANSACTIONS_PER_INCLUSION_LIST]

class ExecutionPayload(Container): // Omit existing fields inclusion_list_summary: List[InclusionListSummaryEntry, MAX_TRANSACTIONS_PER_INCLUSION_LIST] inclusion_list_exclusions: List[uint64, MAX_TRANSACTIONS_PER_INCLUSION_LIST]

class ExecutionPayloadHeader(Container): // Omit existing fields inclusion_list_summary_root: Root inclusion_list_exclusions_root: Root

class BeaconBlockBody(Container): // Omit existing fields inclusion_list_summary: SignedInclusionListSummary

End to End Workflow

Here's the end-to-end flow of the forward inclusion list implementation from the perspective of the proposer at slot N and nodes at slots N-1 and N:

Node at Slot N-1:

1. At Second 0:

The proposer at slot N-1 broadcasts one or more InclusionList

alongside the beacon block for slot N. Multiple inclusion lists can be broadcasted.

1. Between Second 0 to Second 4:

Validators at slot N-1 begin to receive the beacon block alongside the inclusion list. To verify the beacon block, they must also validate at least one of the inclusion lists as valid. A block without a valid inclusion list will not be considered the head block, and it will be outside the node's canonical view.

1. P2P Validation:

Nodes perform P2P-level validation on the inclusion list. They:

- Ensure that the inclusion list is not from a future slot (with a MAXIMUM_GOSSIP_CLOCK_DISPARITY

allowance).

- Check that the inclusion list is not older than MIN_SLOTS_FOR_INCLUSION_LIST_REQUEST

.

- Verify that the inclusion list's transactions do not exceed MAX_TRANSACTIONS_PER_INCLUSION_LIST

.

- Confirm that the inclusion list's summary has the same length as the transactions.
- Validate that the proposer index in the inclusion list summary matches the expected proposer for the slot.
- Verify the inclusion list summary's signature to ensure it's valid for the proposer.
- Ensure that the inclusion list is not from a future slot (with a MAXIMUM_GOSSIP_CLOCK_DISPARITY

allowance).

1. Check that the inclusion list is not older than MIN_SLOTS_FOR_INCLUSION_LIST_REQUEST

.

1. Verify that the inclusion list's transactions do not exceed MAX_TRANSACTIONS_PER_INCLUSION_LIST

.

1. Confirm that the inclusion list's summary has the same length as the transactions.
2. Validate that the proposer index in the inclusion list summary matches the expected proposer for the slot.
3. Verify the inclusion list summary's signature to ensure it's valid for the proposer.
4. On Block Validation:

Before running the state transition, nodes send the inclusion list to the EL client for validation:

- Nodes check if there is an inclusion list associated with the block (similar to blob_sidecar in EIP4844). If no inclusion list is available, the node will not proceed.
- Using the inclusion list, nodes make an execution-API call new_inclusion_list

, providing parameters like parent_block_hash

, inclusion list summary, and inclusion list transactions.

- The EL client performs various checks:
- Ensures alignment between inclusion list summary and transactions.
- Validates that inclusion list transactions satisfy the gas limit.
- Checks the validity of inclusion list summary entries.
- Validates the inclusion list transaction's validity.
- Ensures alignment between inclusion list summary and transactions.
- Validates that inclusion list transactions satisfy the gas limit.
- Checks the validity of inclusion list summary entries.
- Validates the inclusion list transaction's validity.
- Nodes check if there is an inclusion list associated with the block (similar to blob_sidecar in EIP4844). If no inclusion list is available, the node will not proceed.
- Using the inclusion list, nodes make an execution-API call new_inclusion_list

, providing parameters like parent_block_hash

, inclusion list summary, and inclusion list transactions.

1. The EL client performs various checks:
2. Ensures alignment between inclusion list summary and transactions.
3. Validates that inclusion list transactions satisfy the gas limit.
4. Checks the validity of inclusion list summary entries.
5. Validates the inclusion list transaction’s validity.
6. Ensures alignment between inclusion list summary and transactions.
7. Validates that inclusion list transactions satisfy the gas limit.
8. Checks the validity of inclusion list summary entries.
9. Validates the inclusion list transaction’s validity.
10. After Passing Validations:

If the block passes both P2P and state transition-level validations, it is considered a head block.

1. Beacon Block Preparation:

If there exists a next slot proposer, the node calls FCU and attributes in the execution API to signal its intent to prepare and build a block. The node communicates its intention to prepare a block with an inclusion list summary.

1. Local Payload Building:

The node’s local payload building process includes a new inclusion list summary field in the PayloadAttribute

. This field informs the EL client to prepare the payload with the inclusion list summary. The inclusion list will be included in the payload’s transactions field.

1. Optional: Builder Payload Building:

The builder’s payload building process is updated with a new field, inclusion list summary, in the payload attribute SSE endpoint. This signals to builders that they must build the block using the specified inclusion list summary as a constraint.At slot N, the proposer and nodes at slot N operate as follows:

Proposer at Slot N, at second 0

1. Inclusion List Retrieval:

The proposer at slot N calls the execution API’s get_inclusion_list

function using the parent block’s hash. This call results in the generation of an inclusion list that satisfies specific conditions based on the parent block’s hash.

1. Inclusion List Summary:

The retrieved inclusion list includes both transactions and an array of summary entries. These summary entries will eventually be signed and transformed into a SignedInclusionListSummary

.

1. Broadcasting Inclusion List:

The proposer broadcasts the InclusionList

which consists of SignedInclusionListSummary

and Transactions

alongside the SignedBeaconBlock

. This inclusion list is intended for the proposer at slot N+1 and is assigned to its dedicated gossip subnet.

1. Building the block body:

The proposer builds the beacon block body by including the signed inclusion list summary of N-1.

1. Broadcasting the Block:

The proposer broadcasts the block for N

and the inclusion list for slot N+1.

Node at Slot N:

1. Block Received:

Nodes at slot N receive the gossiped block from the proposer at slot N.

1. Consensus Check:

To verify the block, nodes perform a consensus check. They verify the alignment between the block body and the payload. The block body should contain the signed inclusion list summary from the previous block (N-1). Nodes check block body's inclusion list summary's proposer index and signature are correct. They also ensure that the summaries in the block align with the summaries in the payload.

1. Execution Check:

Nodes conduct an execution check by passing the payload to the EL client for verification. The EL client verifies:

- That the payload's transactions are valid and satisfy the conditions.
- That the exclusion summaries in the payload align with the exclusion list received from the EL client earlier in slot N.
- That the payload's transactions are valid and satisfy the conditions.
- That the exclusion summaries in the payload align with the exclusion list received from the EL client earlier in slot N.

The validations ensured alignment between previous slot's inclusion list summary, and execution_payload.inclusion_list_summary

and execution_payload.transactions

.

Design Considerations

Here are the considerations outlined in the document:

Gas Limit:

- One consideration is whether the inclusion list should have its own gas limit or use the block's gas limit.
- Having a separate gas limit simplifies complexity but opens up the possibility for validators to outsource their inclusion lists for side payments.
- Alternatively, inclusion lists could be part of the block gas limit and only satisfied if the block gas limit is not full.

However, this could lead to situations where the next block proposer intentionally fills up the block to ignore the inclusion list, albeit at potential expense.

- The decision on whether the inclusion list has its own gas limit or not is considered beyond the scope of the document.

Inclusion List Ordering:

- The document assumes that the inclusion list is processed at the top of the block. This simplifies reasoning and eliminates concerns about front-running (proposers front-running the previous proposer). Transactions in the inclusion list can be considered belonging to the end of the previous block (N-1) but are built for the current slot (N).

Inclusion List Transaction Exclusion:

- Inclusion list transactions proposed at slot N can get satisfied at slot N. This is a side effect of validators using MEV-boost because they don't control which block gets built.
- Due to this, there exists an exclusion field, a node looks at each transaction in the payload's inclusion_list_exclusion

field and makes sure it matches with a transaction in the current inclusion list. When there's a match, we remove that transaction from the inclusion list summary.

MEV-Boost Compatibility:

- There are no significant changes to MEV-boost. Like any other hard fork, MEV-boost, relayers, and builders must adjust for structure changes.
- Builders must know that execution payloads that don't satisfy the inclusion list summary will be invalid.
- Relayers may have additional duties to verify such constraints before passing them to validators for signing.
- When receiving the header, validators can check that the inclusion_list_summary_root matches their local version and skip building a block if there's a mismatch, using the local block instead.

Syncing using by range or by root:

- To consider a block valid, a node syncing to the latest head must also have an inclusion list.
- A block without an inclusion list cannot be processed during syncing.
- To address this, there is a parameter called MIN_SLOTS_FOR_INCLUSION_LIST_REQUEST

. A node can skip inclusion list checks if the block's slot plus this parameter is lower than the current slot.

- This is similar to EIP-4844, where a node skips blob sidecar data availability checks if it's outside the retention window.

Finally, I'd like to share some food for thought

. Please note that this is purely my perspective. Ideally, I envision a world where relayers become completely removed. While the forced inclusion list feature is an improvement over the current situation, it does make the path to eliminating relayers more challenging.

One argument is that to achieve full enshrined PBS, we may need to take incremental steps. However, taking these steps incrementally could potentially legitimize the current status quo, including the presence of relayers.

Therefore, we should carefully consider whether we want this feature to stand alone or if it's better to bundle everything together and move towards ePBS in one concerted effort.