Inclusion List Variation

: Inclusion List summary of slot n

is committed to Beacon Block of slot n

This document details a version of the inclusion list (EIP-7547) where the current slot inclusion list summary is committed within a block

. The inclusion list transaction is broadcasted out of the band as a side car. The document is divided into multiple sections, each supplemented by diagrams. These sections cover the processes of receiving and verifying the block and the inclusion list and proposing the block and inclusion list. A special note highlights the splitting scenario where manual requests for the inclusion list as a disadvantage of this version.

Thank you [@Potuz](#) for identifying some initial errors and providing feedback.

Components

We first examine the various components required to make the inclusion list work.

[

Screenshot 2024-02-26 at 3.22.48 PM

1020×463 20.7 KB

](https://ethresear.ch/uploads/default/original/2X/7/7ca069ae4af04cb5cf176530f1850c0dec63c3c2.png)

- Consensus layer (CL) clients

: Lodestar, Nimbus, Teku, Lighthouse, and Prysm

- Validator client

: The same team implements both CL and validator clients. A validator client could interop with a different pair of CL clients.

- Execution layer (EL) client

: Geth, Nethermind, Besu, Erigon, and Reth

- Relayer

: Ultrasound, Bloxroute, Flashbots, Agnostic Gnosis and more * MEV-boost is omitted. For now, assume a direct connection from the CL client to the relayer

- MEV-boost is omitted. For now, assume a direct connection from the CL client to the relayer

- Consensus P2P

: A P2P gossip mesh where consensus objects, such as beacon blocks, attestations, and exits, are gossiped. For this post, a node can get block and IL(s) at different time.

- Execution API

: An API that defines the interaction between the CL client and the EL client.

- Validator API

: An API that defines the interaction between the CL client and the validator client.

- Builder API

: An API that defines the interaction between the CL client and the relayers.

- Execution Mempool

: The source of the inclusion list transactions. How they are selected in implementation detail.

Validating a Consensus Layer Inclusion List at Slot n

First, we look at what happens when a CL client receives an inclusion list over the network.

[

Screenshot 2024-02-26 at 3.30.27 PM

939×351 41.3 KB

](https://ethresear.ch/uploads/default/original/2X/6/6f32c146962823d6a8c1e7081faaec8ce80cb2b9.png)

1. Signed inclusion list(s) are received over the wire from the inclusion list gossip network. A proposer may send multiple inclusion lists.

2. After an inclusion list passes gossip validations (slot, proposer index, and signature), it is re-gossiped to peers. Clients should only gossip one inclusion list max. Clients should only import one inclusion list max. The proposer can distribute different inclusion lists to various peers, deliberately obscuring which one you receive, knowing that peers will only gossip back and import at most one list.

3. The CL client forwards the inclusion list to the EL client for validation using the Execution-API's newInclusionList

.

1. The EL client verifies that the inclusion list is valid for slot n

and has sufficient gas limit for validation in slot n+1

.

Validating a Consensus Layer Block at Slot n

Second, we look at what happens when a CL client receives a beacon block over the network.

[

Screenshot 2024-02-26 at 3.30.59 PM

981×358 48.1 KB

](https://ethresear.ch/uploads/default/original/2X/3/3e94c59b5ead63189af9029336a6a5295e8d3221.png)

1. Signed block is received over the wire from beacon block gossip net.

2. Once the block passes gossip validations, it is re-gossiped to its peers.

3. For consensus validation, the block payload's previous IL summary should match the one from the parent beacon state, ensuring alignment across blocks.

4. The block's current IL summary root is cached in the beacon state for slot n+1

consensus validation.

1. The CL client forwards the payload to the EL client for validation via Execution-API's newPayload

.

1. For execution validation, assuming the EL client has stored previous IL transactions and historical execution blocks, it verifies that the current payload's previous IL summary aligns with previous IL transactions.

2. The EL client checks that current payload transactions satisfy all IL transactions (7a), ensuring transactions not satisfied in the current slot were satisfied in slot n-1

(7b).

Determining a Head for a Block at Slot n

Third, how does a node determine the canonical head that is different than today?

[

Screenshot 2024-02-26 at 3.32.06 PM

702×249 13.7 KB

](https://ethresear.ch/uploads/default/original/2X/9/960df0ec65d047ead2b55299a68591f039c68923.png)

- Today, the client verifies consensus and execution in parallel for beacon blocks.

- Post Dencun: the client will not import the beacon block into the fork choice store until the blob is available.

- We extend the previous assumption hold until the inclusion list is available. This implies the block can not be the head until the corresponding inclusion list is available and verified.

- The CL client can verify the inclusion list and block in parallel. One of the inclusion lists must match the one specified by the summary in the block. The EL client must verify the inclusion list conforms to the summary for the slot

- The CL client can also verify consensus and execution in parallel for the inclusion list.

Proposing an Inclusion List at Slot n

As a proposer, let's first look at how to construct and broadcast an inclusion list for an assigned slot n

.

[

Screenshot 2024-02-26 at 3.32.39 PM

777×383 27.4 KB

](https://ethresear.ch/uploads/default/original/2X/d/d0a796ee2c98f837e5049308d90814b90e6d1a25.png)

1. At the start of slot n

, the CL client requests a new inclusion list from the EL client via the Execution-API's newInclusionList

.

1. This inclusion list is then passed to the validator client for signing.

2. The CL client records the inclusion list, which will be used in block proposal, and then broadcasts the signed inclusion list to the inclusion list gossip network.

Proposing a Beacon Block at Slot n

using mev-boost

Second, let's look at what changed with block construction using builder API and relayer network for an assigned slot n

.

[

Screenshot 2024-02-26 at 3.31.29 PM

870×545 46.7 KB

](https://ethresear.ch/uploads/default/original/2X/f/fdb0dbce8b5b9126b29da73a61228765cb76e9f5.png)

1. At the start of slot n

, the CL client requests a new local payload from the EL client via the Execution-API's getPayload

.

1. The CL client gets builder header from relayers, mev-boost selects the best header for the client.

2. The CL client then chooses between the local payload and builder header to pass to the validator client for signing. The CL client also inserts the previously cached inclusion list summary to the block.

3. If the local payload was chosen, the CL client can directly broadcast the block to the block gossip network. If the builder header is selected, the CL client needs to return the signed blind block along with the inclusion list to the relayer.

4. The relayer verifies that the inclusion list matches the summary in the block and then gossips both the block and the inclusion list to its P2P gossip networks.

5. Without mev-boost, this interaction is more straightforward, skip steps 2, 4a, and 5.

Scenario: block with different inclusion list summary

If a block misses its inclusion list, the CL client must manually request it via a P2P RPC request using the summary's root as the RPC handler. Evaluating the risk of a split-view attack is crucial. Including the IL summary in the block ensures commitment at the time of the block proposal. Otherwise, proposers could exploit MEV opportunities during their slot and sell an IL to the next block builder.

Verifying IL availability becomes complex as every node and current slot attesters must confirm a valid IL that matches the summary committed in the block. A concerning scenario arises when validators receive an IL, retain it, and then receive a block with a differing summary. This discrepancy forces validators to manually request missing IL from peers, potentially leading to delays in block imports due to the absence of a valid IL. This scenario was feared in EIP4844 and triggered a late change to include block inclusion in the blob sidecar. Alternatively, not committing to a summary allows for the gossiping of the summary and IL as a sidecar, enabling builders to select any summary for their blocks. This method simplifies verification and offers market benefits by preventing proposers from selling access to their IL but this introduced a greater risk of proposers negotiating off-band with future builders.

If the split view attack is deemed more dangerous, solutions could include broadcasting the IL with the block or refraining from committing the summary in the block, each with its own challenges and considerations.