A Policy-Based Mitigation of Transaction Exclusion in Ethereum

[Work in Progress Paper]

Patrick Spiesberger
Karlsruhe Institute of Technology
Karlsruhe, Germany
patrick.spiesberger@kit.edu

Jan Droll
Karlsruhe Institute of Technology
Karlsruhe, Germany
jan.droll@kit.edu

Hannes Hartenstein
Karlsruhe Institute of Technology
Karlsruhe, Germany
hannes.hartenstein@kit.edu

Abstract

In the Ethereum system, the exclusion of specific transactions is currently feasible with minimal effort due to a power imbalance among entities. This censorship opportunity threatens the dependability of time-sensitive services deployable on Ethereum. In this paper, we look at this threat from an access control perspective and attribute it to a lack of accountability for censorship, a lack of policy definition and enforcement, as well as to the lack of disincentivization of policy violation. We propose an approach to enforceable policies in Ethereum. Furthermore, we demonstrate how a specific policy can address the shortcomings of existing censorship mitigation techniques, particularly *Inclusion Lists*. Under the assumption that block assemblers are unwilling to incur significant financial penalties as well as that the local view on the network messages is sufficiently consistent, the proposed approach guarantees the inclusion of a transaction in a block within 27 seconds in a non-saturated network. The empirical validation of sufficiently consistent views on outstanding transactions is currently in progress.

CCS Concepts

• **Security and privacy** → *Distributed systems security*; • **Computer systems organization** → **Reliability**.

Keywords

Policy Enforcement and Verification, Accountability in Distributed Systems, Ethereum, Blockchain, Censorship Resistance, Proposer Builder Separation, Inclusion Lists

ACM Reference Format:
Patrick Spiesberger, Jan Droll, and Hannes Hartenstein. 2025. A Policy-Based Mitigation of Transaction Exclusion in Ethereum: [Work in Progress Paper]. In *Proceedings of the 30th ACM Symposium on Access Control Models and Technologies (SACMAT '25), July 8–10, 2025, Stony Brook, NY, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3734436.3734458

1 Introduction

Ethereum [1] can be seen as a kind of replicated state machine (*RSM*) where transactions define state transitions. These transactions are propagated through Ethereum's peer-to-peer (*P2P*) network and are *eventually* executed by an entity known as the proposer. The resulting post-execution states, along with the transactions, are recorded in a block, referencing the previous block. If the correctness of the execution is validated by a committee, this block is part of the canonical blockchain. However, transaction inclusion in a block is not guaranteed: each block is constructed by a single proposer, who either selects transactions directly from their mempools — a local memory of pending transactions — or outsources this assembling process to a specialized entity, called builder. We will use the term *block assembler* to refer to the entity responsible for including transactions in a block. In both cases, the final decision of transaction inclusion follows a non-accountable inclusion policy that can vary between block assemblers and can change arbitrarily over time. This discretion allows censorship [30]. While in principle for each block a proposer is pseudo-randomly selected, block assembly is done by only a few entities nowadays. When block assemblers enforce a censoring policy, systematic transaction exclusion based on transaction-related information becomes a concern. A well-documented example is compliance with a list of sanctioned Ethereum addresses maintained by the U.S. Department of the Treasury [21, 22]. Our analysis shows that transactions involving these listed addresses take on median 2.8 times longer[1] to appear in a block than non-sanctioned transactions. Similar dynamics could emerge from other regulatory or economic reasons affecting time-sensitive applications that rely on predictable or timely execution. We therefore address the research question of whether it is possible to impose policies against systematic transaction exclusion.

In this paper, we analyze Ethereum's transaction inclusion process through the lens of *access control systems*. By mapping key access control components to Ethereum's block construction process, we reveal the lack of enforceable policies governing transaction inclusion, allowing arbitrary exclusions without penalties. To address this issue, we propose a policy-based approach that strengthens censorship resistance by enforcing accountability for transaction exclusion. The proposed policy gives transaction creators (applicants) certainty that their transaction cannot be excluded for more than 27 seconds without financial disadvantages to the block assembler. The paper is structured as follows: We first outline Ethereum's architecture (Sec. 2) and define the censorship problem (Sec. 3). We contribute two approaches for censorship mitigation (Sec. 4): Protocol-enforced individual policies as an applicant and proposer-driven power imbalance mitigation; and a corresponding design of an *Inclusion List* which is created by block assemblers in an accountable manner. Finally, we discuss important aspects and sketch the work in progress (Sec. 5) before drawing our conclusion (Sec. 6). Related work is referenced in-line.

[1]Based on observations from Block 20,651,330 to Block 21,023,471; data collected from a node located in Karlsruhe, Germany. On 21 March 2025, the considered addresses were removed from the list of sanctioned addresses. Consequently, transactions related to these addresses are no longer subject to extra delays.

2 Background on Ethereum

Ethereum is a decentralized blockchain-based network designed to execute applications, so-called smart contracts, and maintain a globally shared state about their execution. Ethereum's architecture consists of multiple logical layers. The execution layer processes transactions, sent by an applicant, as input for the stored smart contracts, determining the outcome and the desired state. The consensus layer, built on *Proof-of-Stake (PoS)* [13, 26], ensures the validity of this state and the finalization of blocks. The data layer, represented by the Ethereum blockchain, records validated transactions and the resulting state, linking all state transitions back to the genesis block (*Block* 0). However, participants may have diverging views of this layer until a block is finalized, at which point all preceding blocks become universally agreed upon. The network operates synchronized in fixed 12-second slots, grouped into epochs of 32 slots. In each slot a validator is randomly and verifiably selected as the proposer, tasked with assembling a block. Other validators form a validation committee that decides on the validity of the block and casts corresponding attestations. If a supermajority of attestations is reached, the network considers the block to be added to the chain. The network layer is implemented via *P2P* and enables the propagation of transactions, blocks and attestations through interconnected nodes. We will limit the focus of this paper to the assembling of blocks.

In the current Ethereum specification, proposers select transactions from the mempool and assemble a block. By including, excluding, or ordering specific transactions, additional profits — known as *Maximal Extractable Value (MEV)* — can be earned alongside the transaction fees paid by the applicant [9, 14, 25]. Constructing the block data — known as the payload — with high *MEV* requires sophisticated analysis and computational resources beyond the capabilities of many validators. With *Proposer-Builder Separation (PBS)* [2], the responsibility for payload construction is delegated to specialized and highly optimized builders who create financially rewarding blocks for proposers to choose from. Builders compete by submitting bids to proposers, effectively purchasing the right to construct the block. Economically acting proposers select the most profitable bid, ensuring that their block space is allocated to the highest-paying builder. Currently, the communication and handling of bids and payloads are managed by a trusted middleman, functioning as a kind of execution monitor. This monitor ensures that neither the proposer nor the builder can benefit from the other without financially rewarding them. *Enshrined Proposer-Builder Separation (ePBS)* aims to integrate this separation directly into the Ethereum protocol. For an enshrined protocol solution, it must be ensured that neither the proposer nor the builder deviates from the protocol to exploit the other. Neuder and D'Amato propose *Payload Timeliness Committees (PTCs)* [10, 20] as a mechanism to ensure compliance between both participants. The *PTC* is a committee composed of Ethereum validators whose dedicated role is to monitor the timely delivery of payloads from the builder to the network. This committee verifies that the payload submitted by the builder conforms to the required specifications and is delivered to the proposer within a specified time frame. *Enshrined PBS* is not yet part of the Ethereum specification; however, for the purposes of this paper, we assume its implementation through the *PTC*.

3 Problem Analysis and Statement

Censorship is commonly understood as the intentional restriction or suppression of information, ideas, or expressions by an authoritative entity [15]. Such restrictions can be imposed by governments, institutions, or other influential actors, either before or after dissemination of the censored subject. In the context of Ethereum's block assembling process, censorship refers to the intentional exclusion of specific valid transactions from blocks. This exclusion may be driven by regulatory requirements [22, 30], economic incentives [19], or individual preferences. In the Ethereum network, there is a limit on the total computational effort, measured in gas, that can be processed per block [6]. This gas limit sets the maximum cumulative gas expenditure allowed for all transactions in the block, so only transactions whose combined gas consumption remain within this cap can be included in a block. In case of a saturated network, an access control decision must be made by the block assembler regarding which transactions can be included (permit) and which must be excluded (deny). A transaction exclusion is not necessarily an act of censorship but may also be a consequence of the block capacity limitations. To define censorship in Ethereum, we first need to establish *Fair Ordering* and the weaker form *Fair Inclusion* as the 'ideal' rating of non-censorship. This principles are key to distinguishing between exclusion caused by block space constraints and exclusion resulting from intentional censorship.

Fair Ordering. To establish a theoretical basis of accountability in transaction inclusion, we adopt the *Fair Ordering* principle [23]. Under this principle, valid transactions are ordered primarily by descending priority fees [4] and secondarily by a verifiable ordering mechanism, such as the transaction hash. Priority fees are extra incentives paid by applicants to incentivize block assemblers to prioritize their transactions. The principle of *Fair Ordering* states that transactions must be included in the block in the given order, taking as many from the transaction sequence as possible until no additional transaction fits into the remaining block space. Transactions that would exceed the remaining available block space are skipped, and the next transaction in the sequence is considered. This ensures that as many as possible transactions with the highest inclusion incentive are included in a block. We emphasize that we want to consider *Fair Ordering* as a set of transactions rather than a sequence (i.e., an actual ordering) of them. We refer to this modified principle as *Fair Inclusion.*

Builder Censorship. We define *Builder Censorship* as follows: Let B be a block of transactions constructed by block assembler β using mempool M; let $T' \in M : T' \notin B$ be a transaction that is not included in B but with a higher priority fee than a transaction $T \in B$ in the block. We consider β censoring if *one of* the following conditions is true: (1) *Blockspace Availability Condition*: The block gas limit is not exceeded when appending T' to the block. (2) *Economic Rationality Condition*: T' requires less or equal gas than a subset of transactions $S \subseteq B$ and T' offers a higher priority fee than the sum of priority fees of S. Intuitively, the builder censors T' if this transaction could be included in addition to all other included transactions or if T' could be included instead of less profitable transactions S.

Ethereum's transaction inclusion can be understood through the lens of access control, with a specific focus on how policies are defined, evaluated, and enforced. We assume that the reader is familiar with the notions of Policy Administration, Retrieval, Information, Decision, and Enforcement Points (*PAP*; *PRP*; *PIP*; *PDP*; *PEP*) as defined in [24, 27].

To obtain a permit for inclusion, transactions must adhere to rules defined in the Ethereum specification [12, 13]. Hence, the Ethereum specification is a *global policy* which also mandates that only *valid* transactions can be considered for blocks [12]. In addition, this specification document serves as *PRP* for, e.g., validators. Each validator of a block's committee acts as *PDP* regarding this policy fulfillment while the committee as a whole forms the *PEP* that decides whether a block is valid and thus extends the canonical chain. The *PEP* enforces a decision based on the majority decision of *PDP* entities, effectively aggregating their individual assessments.

However, the *global policy* does not fully cover all aspects related to block inclusion and exclusion. Any decisions not explicitly mandated by the *global policy* fall within the discretion of the block assembler, which establishes an *individual policy.* Such *individual policies* are generally not disclosed in the current Ethereum block assembly process and may change unpredictably. As a result, the block assembler plays a partial role in the context of *PAP*, *PDP*, and *PEP* concerning *individual policies* that do not conflict with the *global policy.* Specifically, the *global policy* does not define rules for including valid transactions in a block, leaving the decision entirely to the block assembler's individual and not accountable policy. Furthermore, the block assembler can incorporate a wide range of information into the block assembly process, including network data such as the latest valid block (consensus data), currently propagated attestations (*P2P* data), and external data like financial exchange data. Therefore, the *PIP* in Ethereum is not limited to a single entity or class of information. As a result, there is no obligation for the block assembler to include a valid transaction, leading to non-accountable exclusion (deny) decisions. This lack of accountability facilitates the selective exclusion of transactions, thereby leading to *Builder Censorship.* Therefore, the problem can be stated as follows:

> *Can an anti-censoring policy be implemented in Ethereum in a protocol-enforceable way?*

While specifying a corresponding policy is straightforward, detecting and enforcing such policy violations is challenging as censorship detection in a block requires identifying violations of *Fair Inclusion.* In the absence of a global anti-censorship policy, accountability regarding exclusion decisions is desirable to be able to react on such behavior. In summary, we address an access control problem with default permit and exceptional deny and the challenge is to enforce anti-censorship policies with a decentralized execution monitor, particularly with a decentralized *PDP* and *PEP.*

4 Towards Censorship Resistance

In this section, we introduce individual policies that applicants can use to require non-censored blocks, and explain how these policies can be enforced in Ethereum (see Sec. 4.1). Subsequently, in Sec. 4.2, we give an overview over Inclusion Lists (*IL*), a community-driven concept to mitigate censorship. We present an alternative *IL* design based on *Fair Inclusion* (Sec. 3) in which a single party creates a verifiable and accountable *IL* in Sec. 4.3.

4.1 Protocol Enforced Individual Policies

Protocol Enforced Proposer Commitments (PEPC) [17] enable proposers to define explicit commitments regarding block structure and content. These commitments form two-sided *individual policy* agreements between the proposer and the builder, outlining rules such as transaction ordering or the in-/exclusion of specific transactions. Several implementation proposal are available [7, 8]. From an access control view, *PEPC* extends Ethereum's *global policy* management by granting proposers the authority to set *individual policies*, thereby acting as a *PAP*. The proposer and builder share decision-making power over the payload, depending on the strength of the commitment. The enforcement is delegated to a committee of designated entities acting as *PDP* (each entity) and *PEP* (based on majority decision). If a builder fails to comply with the specified commitments, these entities invalidate the block. One possible implementation is the *Commitment Satisfaction Committee (CSC)* [7], a committee of deterministically and verifiably selected Ethereum validators that verifies compliance with the proposer's *individual policy.* Although builders retain some autonomy over transaction inclusion or exclusion (*PDP*), they must align their block construction with the proposer's predefined policies (*PAP*), with commitments decided and enforced by the *CSC*'s validation vote.

PEPC, by separating the roles of the *PAP*, *PDP*, and *PEP* across the proposer, builder, and a designated *CSC*, enables enforceability of *individual policies.* However, a key issue arises from the financial dependence between proposers and builders, which may make it economically unfeasible for a proposer to demand strict policies. Thus, in this setup, transaction exclusion might not be effectively mitigated, as a non-altruistic proposer has no incentives to initiate such a commitment. To address this deficiency, we introduce a realization of the *PEPC* concept, termed *Protocol Enforced Requirements (PER).* The primary goal of *PER* is to extend the concept of policies beyond agreements between proposers and builders, introducing enforceable policies for all relevant parties: applicants, proposers, and builders. Unlike current *PEPC* designs, *PER* allows each entity involved in a block assembling process to define its individual policies (*PAP*), called *Requirements*, independently. These requirements are encoded within an on-chain *Policy Manager*, which serves as a repository (*PRP*) for the active requirements of all involved entities. To prevent short-term changes of the requirement, it becomes enforceable only after finalization, ensuring that timing games with equivocations are prevented. In *PER*, the *CSC* continues to fulfill the role of the *PDP* by making decisions on the validity of the block. Meanwhile, the builder retains the ability to decide which transactions are included in the block, but this decision must be made within the constraints set by the *individual policies* of applicants, proposer, and their own. Unlike *PEPC*, *PER* allows the definition of penalties, which are initiated by the *CSC* if a majority of its members determine that a policy has been violated (*PEP*). *PER* operates in three phases, visualized in Fig. 1. In the setup phase, applicants, proposers, and builders can register policies at the policy manager, a deployed smart contract acting as a register for *individual policies.* Anyone can lookup relevant requirements of other entities using the public identity information of those entities from the manager. The block construction phase aligns with the approach proposed in *EIP-7732* (*ePBS* via the *PTC* design) [10].

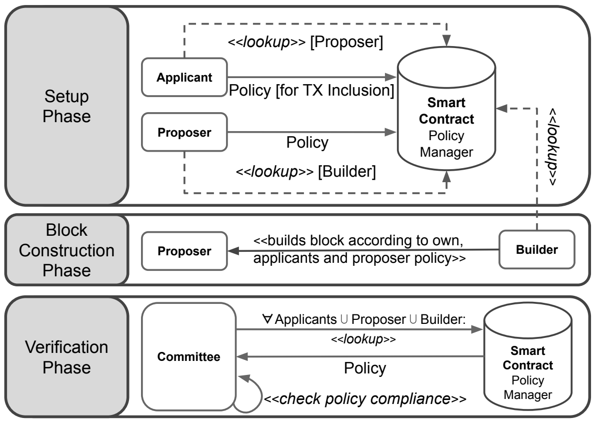
Setup Phase
Applicant
Proposer
<<lookup>> [Proposer]
Policy [for TX Inclusion]
Policy
<<lookup>> [Builder]
Smart Contract Policy Manager
<<lookup>>
Block Construction Phase
Proposer
<<builds block according to own, applicants and proposer policy>>
Builder
Verification Phase
Committee
∀ Applicants ∪ Proposer ∪ Builder:
<<lookup>>
Policy
<<check policy compliance>>
Smart Contract Policy Manager

Figure 1: The three phases of *PER*: In the setup phase, applicants, proposers, and builders register their policies. During block construction, the assembler builds the block following these policies. In the verification phase, committee members check policy compliance and enforces them accordingly.

The block assembler constructs a block proposal based on the policies defined by the proposers and the applicants whose transactions they wish to include. In the verification phase, the *CSC* verifies compliance with policies for all involved entities, including all applicants by performing a lookup in the policy manager. Alongside the standard *PTC* tasks, the committee checks whether the defined policies have been met. If a policy violation occurs, the corresponding penalty is executed. If no penalty is defined, a default "block invalid" decision is enforced. With *PER*, we introduce a mechanism to make *individual policies* accessible in the network without a single entity controlling the decision and enforcement process.

4.2 Inclusion List-Based Mitigation

In this subsection, we outline how *Fair Inclusion* can be achieved to mitigate *Builder Censorship*. To this end, we use in addition to *PER* the concept of *Inclusion Lists* [11], highlight their limitations, and propose a direction in which they can contribute to improving censorship resistance. *PER* enables two approaches: (1) applicants create this kind of financial incentive to demand *Fair Inclusion*, or (2) *Fair Inclusion* is registered as a default for all block assemblers, allowing *PER* to manage decision-making and enforcement. In the following, we focus on the latter. Regarding (1), applicants may register *individual policies* that restrict transaction inclusion to block assemblers committed to censorship-free behavior. This approach partitions the mempool into unrestricted and policy-enforced areas, impacting *MEV* extraction and creating a financial incentive, as a block assembler has more *MEV* opportunities through the potential use of the entire mempool.

Forward Inclusion Lists. Inclusion Lists (*ILs*) [11] and Forward Inclusion Lists (*FILs*) [29] are concepts designed to improve censorship resistance by shifting full control over transaction exclusion away from a single entity (see Sec. 3) to at least two entities. *ILs* allow proposers to specify certain transactions that must be included in their block by the builder. However, since proposers financially benefit from builders' *(e)PBS* bids, they have little incentive to demand strict *ILs*, making widespread adoption unlikely. *FILs* attempt to address these shortcomings by requiring the current block assembler to commit to a list of transactions — a form of an *individual policy* — that must be included by the subsequent block assembler. However, this mechanism governs only whether these transactions are included in the next block, without imposing protocol-enforced consequences on the assembler who created the *FIL*. Although this approach removes the direct financial link between proposer and builder — redistributing it to two typically independent parties (the *FIL* creator and the subsequent assembler) *FILs* remain susceptible to external pressures such as regulatory constraints, collusion between consecutive block assemblers, and bribery, where the next assembler compensates the previous one to in- or exclude specific transactions from the *FIL* [18].

Fork-Choice Enforced Inclusion Lists. FOCILs [28] strengthen censorship resistance by embedding the creation of *Forward Inclusion Lists* in the consensus mechanism. A dedicated *FOCIL* committee monitors their mempool, with each member constructing partial *FILs* from observed transactions. These lists are then broadcast and aggregated. *FOCIL* delegates the access control decision on which transaction fills the next block's space to multiple parties, who are not accountable for their partial *ILs*. Exclusion occurs only if all committee members unanimously decide to exclude a transaction. A key design challenge is balancing transaction coverage and network efficiency. Censorship resistance improves as more committee members create larger partial lists, increasing the likelihood of transaction inclusion in the aggregated *FOCIL*. However, larger lists and more committee members also increase network overhead, raising bandwidth demands. Additionally, committee members can insert arbitrary transactions, risking manipulation through spam or low-priority transactions crowding out high-priority ones. From an access control perspective, each committee member can decide on the inclusion of transactions for the next block and enforce it through consensus. However, dishonest members, bribery, or governmental restrictions could exploit this to include specific transactions, even if they contradict the principles of *Fair Inclusion*.

4.3 Fair Forward Inclusion Lists as a Policy

A key limitation of *FOCIL* is the lack of clear policies for constructing the *FIL*. To address this deficiency, we introduce *Fair Forward Inclusion Lists (FairFIL)*, which demands accountability for transaction deny decisions made by the *FairFIL* creator while upholding the principle of *Fair Inclusion* (see Fig. 2). The proposed approach mandates the block assembler to construct the *FairFIL* according to a predefined *FairFILPolicy*. This policy is then enforced by the responsible *PER* committee. The *FairFILPolicy* consists of four rules:

Rule 1 demands that every block must contain a *FairFIL* created according to the principles of *Fair Ordering*. **Rule 2** mandates that each block assembler includes all transactions from the previous *FairFIL* in their block. **Rule 3** requires the block assembler to be accountable for transaction exclusions by creating the *FairFIL*, thereby assigning the responsibility to include the censored transactions to the subsequent block assembler. This *FairFIL* consists of a sequence of transactions that were not included and, therefore, not executed but should have been included in the current block according to *Fair*

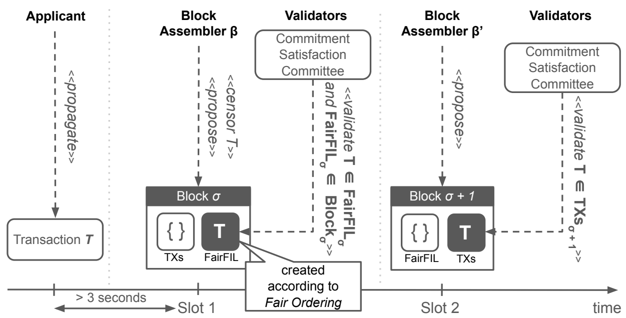
Applicant
Block Assembler β
Validators
Block Assembler β'
Validators
Commitment Satisfaction Committee
Commitment Satisfaction Committee
<<propagate>>
<<propose>>
<<censor T>>
<<validate T ∈ FairFIL$_\sigma$ and FairFIL$_\sigma$ ∈ Block$_\sigma$>>
<<propose>>
<<validate T ∈ TXs$_{\sigma+1}$>>
Transaction T
Block σ
TXs
FairFIL
Block σ + 1
FairFIL
TXs
created according to Fair Ordering
> 3 seconds
Slot 1
Slot 2
time

Figure 2: *FairFIL* concept: Excluding *T* mandates its inclusion in this slot's *FairFIL* and the next block (protocol-enforced).

Ordering. The transactions are sorted by priority fees (descending), and their number is limited by *Fair Ordering*. **Rule 4** demands that a censored transaction in the current block does not need to be included in the *FairFIL* if it cannot be included in the next block. For example, a transaction that the assembler could have included may become non-includable if the published block causes the network's minimum transaction fee to rise, making the transaction's fee fall below the required minimum [4].

With the *FairFIL* approach, a block assembler can construct a block arbitrarily, provided that (1) all transactions from the previous *FairFIL* are included, and (2) the assembler discloses any transactions censored according to *Fair Inclusion*. However, the assembler is only required to reveal transactions that would remain valid under the *global policy* if executed within the next block, following the order specified by *Fair Ordering*. *PER* committee members carry out various tasks to assess the validity of *FairFILs*, as detailed in Appendix A.

In comparison to current *FIL* designs, we see various advantages of the *FairFIL* approach: (1) By integrating the *FairFIL* directly into the corresponding block, its propagation cannot be delayed to prevent the next assembler from seeing it. (2) Block assemblers cannot equivocate about their *FIL* without equivocating about their block. Hence it is clear which *FIL* to enforce. (3) Penalties, such as block invalidation, can be enforced for violations of *Fair Ordering*. (4) *FairFILs* are smaller than traditional *FILs*, as typically only a few transactions are excluded, reducing network overhead. Our February 2025 measurements show that *FairFIL* sizes typically fall within the low two-digit range.

To be able to verify the correct handling of a FairFIL by a block assembler, i.e., for decentrally enforcing the FairFIL policy, we need a 'good enough' consistency of the views among the PER members of pending transactions. We assume that block assemblers are highly optimized regarding network connectivity, ensuring that no situation arises where 50% or more of the *PER* committee members observe a transaction at least 3 s before the assembler. Since the block assembler cannot predict which transactions the *PER* committee members have seen, they are incentivized to construct the *FairFIL* according to *Fair Ordering* to minimize the risk of block invalidation and associated opportunity costs (see Sec. 2). It is acceptable if the assembler's *FairFIL* includes correctly ordered transactions that are not expected by the *PER*. However, it constitutes a policy violation if the *PER* demands a transaction that is missing from the assembler's *FairFIL*.

With *FairFIL*, the inclusion of a transaction T is delayed at most 27 seconds: If T is propagated less than 3 s before the start of slot σ, T must be included in the *FairFIL* of block $B_{\sigma+1}$, which is 12 s after the start of slot σ, and finally included in block $B_{\sigma+2}$ after additional 12 s (see Appendix B).

5 Discussion and Work in Progress

A key challenge in designing anti-censorship measures lies in Ethereum's design principles [3]. While the proposed *FairFILPolicy* minimizes additional network bandwidth by having the *FairFIL* created by a single party (block assembler), we identify a potential issue: transactions from the *FairFIL* could be included without the applicant paying fees, violating the design goal of *no free data availability* (see Appendix A). Addressing this issue is work in progress. Another goal is to ensure that altruistic behavior is not financially penalized [3]. Both *PER* and the policy-based *FairFIL* must prevent *individual policies* from being exploited to disadvantage honest participants. In contrast to our intention, policies could also enforce censorship. While we aim to improve Ethereum's reliability by mitigating censorship, we must also consider potential attack vectors within *individual policies* and the *FairFIL* design for feasibility. To achieve accountability, we require not only clearly defined policies, but also the ability for every honest participant to monitor for their exclusion and for validators to reliably validate this. This 'decentralized monitoring for accountability' requires a sufficiently consistent view of the mempools. Our preliminary empirical validation, comparing the mempools of two locally operated independent nodes, indicates sufficient consistency. Further validation is work in progress.

6 Conclusion

We demonstrated that Ethereum's block assembling process inherently involves an access control decision over limited block space. However, the current process lacks adherence to *Fair Ordering and Inclusion*, allowing block assemblers to arbitrarily exclude valid transactions without violating any policy. To avoid granting the block assembler sole decision-making power over block payloads, we introduce *PER*, a framework that enables applicants, proposers, and builders to define individual policies governing all aspects related to their respective roles. We identified the lack of accountability for transaction exclusion in Ethereum as a key problem, as neither block assemblers are held responsible for excluding valid transactions, nor do existing *FIL* proposals adequately address this issue. To address this, we introduce a policy-based *FairFIL*, leveraging *PER*, to enhance accountability primarily via a transparency mechanism. The responsibility for policy registration remains an open question, with two possible models: a globally enforced censorship resistance policy governed by Ethereum's specification or an incentive-driven approach where applicants reward block assemblers for adhering to *Fair Inclusion*. While *FairFIL* does not eliminate *Builder Censorship* in the current slot, it prevents censorship in the next block and reduces transaction inclusion delays, with a maximum delay of 27 s in a non-saturated network.

References

[1] Vitalik Buterin. 2013. Ethereum: A Next-Generation Smart Contract and Decentralized Application Platform. *Ethereum White Paper* (2013).

[2] Vitalik Buterin. 2021. Two-Slot Proposer-Builder Separation (PBS). https://ethresear.ch/t/two-slot-proposer-builder-separation/10980.

[3] Vitalik Buterin. 2023. Design Goals for Anti-Censorship in Blockchain Protocols. Ethereum Research Notes. https://notes.ethereum.org/s3JToeApTx6CKLJt8AbhFQ [last visit: March 14, 2025].

[4] Vitalik Buterin, Eric Conner, Rick Dudley, Matthew Slipper, Ian Norden, and Abdelhamid Bakhta. 2019. EIP-1559: Fee market change for ETH 1.0 chain. https://eips.ethereum.org/EIPS/eip-1559 [last visit: November 23, 2024].

[5] Vitalik Buterin and Michael Neuder. 2023. No free lunch – a new inclusion list design. https://ethresear.ch/t/no-free-lunch-a-new-inclusion-list-design/16389. [last visit: March 17, 2025].

[6] Ben Edgington. 2023. *Upgrading Ethereum - A technical handbook on Ethereum's move to proof of stake and beyond*. https://eth2book.info/ [last visit: March 17, 2025].

[7] Diego Estevez. 2023. Commitment-Satisfaction Committees: An In-Protocol Solution to PEPC. https://ethresear.ch/t/commitment-satisfaction-committees-an-in-protocol-solution-to-pepc/17055 [last visit: March 07, 2025].

[8] Diego Estevez. 2023. PEPC-DVT: PEPC with no changes to the consensus protocol. https://ethresear.ch/t/pepc-dvt-pepc-with-no-changes-to-the-consensus-protocol/16514 [last visit: March 07, 2025].

[9] Flashbots. 2022. MEV-Boost. https://github.com/flashbots/mev-boost. [last visit: November 22, 2024].

[10] Ethereum Foundation. 2023. EIP-7732: Enshrined Proposer-Builder Separation. https://eips.ethereum.org/EIPS/eip-7732. [last visit: March 05, 2025].

[11] Ethereum Foundation. 2024. EIP-7547: Inclusion Lists. https://eips.ethereum.org/EIPS/eip-7547. [last visit: March 13, 2025].

[12] Ethereum Foundation. 2024. Ethereum Execution Client Specifications. https://github.com/ethereum/execution-specs. [last visit: March 30, 2025].

[13] Ethereum Foundation. 2024. Ethereum Proof-of-Stake Consensus Specifications. https://github.com/ethereum/consensus-specs. [last visit: March 30, 2025].

[14] Vincent Gramlich, Dennis Jelito, and Johannes Sedlmeir. 2024. Maximal extractable value: Current understanding, categorization, and open research questions. *Electronic Markets* 34 (10 2024). doi:10.1007/s12525-024-00727-x

[15] Derek Jones. 2001. *Censorship: A World Encyclopedia*. Routledge. doi:10.4324/9780203827055

[16] Georgios Konstantopoulos and Mike Neuder. 2023. Time, slots, and the ordering of events in Ethereum Proof-of-Stake. https://www.paradigm.xyz/2023/04/mev-boost-ethereum-consensus [last visit: August 01, 2024].

[17] Barnabé Monnot. 2022. Unbundling PBS: Towards Protocol-Enforced Proposer Commitments (PEPC). https://ethresear.ch/t/unbundling-pbs-towards-protocol-enforced-proposer-commitments-pepc [last visit: March 30, 2025].

[18] Barnabé Monnot. 2023. Fun and games with inclusion lists. https://ethresear.ch/t/fun-and-games-with-inclusion-lists/16557. [last visit: March 02, 2025].

[19] Bulat Nasrulin, Georgy Ishmaev, Jérémie Decouchant, and Johan Pouwelse. 2023. LO: An Accountable Mempool for MEV Resistance. In *Proceedings of the 24th International Middleware Conference* (Bologna, Italy) *(Middleware '23)*. Association for Computing Machinery, New York, NY, USA, 98–110. doi:10.1145/3590140.3629108

[20] Michael Neuder and Francesco D'Amato. 2023. Payload-timeliness committee (PTC) – an ePBS design. https://ethresear.ch/t/payload-timeliness-committee-ptc-an-epbs-design/16054 [last visit: March 13, 2025].

[21] U.S. Department of the Treasury. 2022. U.S. Treasury Sanctions Notorious Virtual Currency Mixer Tornado Cash. https://home.treasury.gov/news/press-releases/jy0916 [last visit: March 10, 2025].

[22] U.S. Department of the Treasury. 2023. Office of Foreign Assets Control - Sanctions Programs and Information. https://home.treasury.gov/policy-issues/office-of-foreign-assets-control-sanctions-programs-and-information [last visit: March 10, 2025].

[23] Ariel Orda and Ori Rottenstreich. 2021. Enforcing fairness in blockchain transaction ordering. *Peer-to-peer Networking and Applications* 14, 6 (2021), 3660–3673.

[24] Dimitrios Pendarakis, Raj Yavatkar, and Roch Guerin. 2000. A Framework for Policy-based Admission Control. RFC 2753. doi:10.17487/RFC2753

[25] Rated Network. 2022. The Merge series (part 3): MEV landscape. https://blog.rated.network/blog/merge-mev. [last visit: March 07, 2025].

[26] Caspar Schwarz-Schilling, Joachim Neu, Barnabé Monnot, Aditya Asgaonkar, Ertem Nusret Tas, and David Tse. 2021. Three Attacks on Proof-of-Stake Ethereum. *CoRR* abs/2110.10086 (2021). arXiv:2110.10086

[27] David Spence, George Gross, Cees de Laat, Stephen Farrell, Leon HM Gommans, Pat R. Calhoun, Matt Holdrege, Betty W. de Bruijn, and John Vollbrecht. 2000. AAA Authorization Framework. RFC 2904. doi:10.17487/RFC2904

[28] Thomas Thiery, Francesco D'Amato, Julian Ma, Barnabé Monnot, Terence Tsao, Jacob Kaufmann, and Jihoon Song. 2024. EIP-7805: Fork-choice enforced Inclusion Lists (FOCIL). https://eips.ethereum.org/EIPS/eip-7805 [last visit: March 29, 2025].

[29] Terence Tsao. 2023. Spec'ing out Forward Inclusion-List w/ Dedicated Gas Limits. https://ethresear.ch/t/specing-out-forward-inclusion-list-w-dedicated-gas-limits/17115 [last visit: November 26, 2024].

[30] Anton Wahrstätter, Jens Ernstberger, Aviv Yaish, Liyi Zhou, Kaihua Qin, Taro Tsuchiya, Sebastian Steinhorst, Davor Svetinovic, Nicolas Christin, Mikolaj Barczentewicz, and Arthur Gervais. 2023. Blockchain Censorship. *CoRR* abs/2305.18545 (2023). doi:10.48550/ARXIV.2305.18545 arXiv:2305.18545

A Enforcement of the FairFILPolicy

Each selected *PER* member is tasked with verifying the fulfillment of the *FairFILPolicy*. To this end, they perform the following checks for slot σ and enforce the penalty if any of the checks evaluate as false.

Each *PER* member takes a snapshot of the mempool at time $\sigma - 3$ s. From this snapshot, all transactions that cannot be included in block $B_{\sigma+1}$ are filtered out. The remaining transactions are sorted by priority fees (descending).

Are all transactions from $\mathit{FairFIL}_{\sigma-1}$ included in B_σ? (Check 1)

Does the $\mathit{FairFIL}_\sigma$ exist in B_σ? (Check 2)

Is $\mathit{FairFIL}_\sigma$ sorted by priority fees (descending)? (Check 3)

Would $B_{\sigma+1}$ be valid if the entire $\mathit{FairFIL}_\sigma$ is included? (Check 4)

Are all transactions from the filtered and sorted snapshot that fall under *Builder Censorship* included in $\mathit{FairFIL}_\sigma$? If not, would each transaction exceed $B_{\sigma+1}$'s gas limit if added correctly to $\mathit{FairFIL}_\sigma$? (Check 5)

If one of the checks is not fulfilled, *then* invalidate B_σ. (Penalty)

The requirements of *Check 2* contradict the *no free data availability* (FreeDA) design goal [3], which states that all published data — including *FILs* — must incur a cost to prevent the network from being exploited as a free storage or distribution layer. To address this problem, we consider the "No Free Lunch" approach [5] a promising complement, which requires only minimal modifications to the presented *FairFIL* design.

B Censoring Opportunities and Timeliness

Let σ (time $t_\sigma = 0$ s) and $\sigma + 1$ ($t_{\sigma+1} = 12$ s) be consecutive slots, where each proposer-builder pair $P\beta_\sigma$ consists of a proposer and a builder β_σ. Policy-compliant pairs are denoted as $\vec{P\beta}_\sigma$. A $FairFIL_\sigma$ records censored transactions at slot σ. A censored transaction TX_c is added to $FairFIL_\sigma$ if it was observed at $t_{\text{propagation},TX_c} < -3\ s$ but not included in B_σ. A compliant builder at $\sigma+1$ must include all transactions from $FairFIL_\sigma$. In the worst case, TX_c is first observed at $t = -3\ s$, remains excluded for two slots, and is included no later than 27 s after detection, plus up to 12 s for becoming canonical [16]. Without any compliance, no upper bound exists. However, under intermittent policy adherence (i.e., if there is at least one policy-compliant assembler $\vec{P\beta}_x$ in the network), the inclusion time satisfies $t_{\text{inclusion},TX_c} \leq (n+2) \times 12\text{ s} + 3\ s, \quad \text{for } n \in \mathbb{N}_0$, where n denotes the number of consecutive non-compliant proposer-builder pairs encountered before the first compliant pair appears.