LO
An Accountable Mempool for MEV Resistance

Nasrulin, Bulat; Ishmaev, Georgy; Decouchant, Jérémie; Pouwelse, Johan

DOI
10.1145/3590140.3629108

Publication date
2023

Document Version
Final published version

Published in
Middleware 2023 - Proceedings of the 24th ACM/IFIP International Middleware Conference

LØ: An Accountable Mempool for MEV Resistance

Bulat Nasrulin
b.nasrulin@tudelft.nl
Delft University of Technology
Delft, Netherlands

Georgy Ishmaev
g.ishmaev@tudelft.nl
Delft University of Technology
Delft, Netherlands

Jérémie Decouchant
j.decouchant@tudelft.nl
Delft University of Technology
Delft, Netherlands

Johan Pouwelse
j.a.pouwelse@tudelft.nl
Delft University of Technology
Delft, Netherlands

ABSTRACT

Manipulation of user transactions by miners in permissionless blockchain systems is a growing concern. This problem is a pervasive and systemic issue that incurs high costs for users of decentralised applications and is known as Miner Extractable Value (MEV). Furthermore, transaction manipulations create other issues such as congestion, higher fees, and system instability. Detecting transaction manipulations is difficult, even though it is known that they originate from the pre-consensus phase of transaction selection for building blocks, at the *base layer* of blockchain protocols. In this paper, we summarize known transaction manipulation attacks. We present LØ, an accountable base layer protocol designed to detect and mitigate transaction manipulations. LØ is built around the accurate detection of transaction manipulations and assignment of blame at the granularity of a single mining node. LØ forces miners to log all the transactions they receive into a secure mempool data structure and to process them in a verifiable manner. Overall, LØ quickly and efficiently detects censorship, injection or re-ordering attempts. Our performance evaluation shows that LØ is also practical and only introduces a marginal performance overhead.

CCS CONCEPTS

• **Security and privacy → Distributed systems security**; • **Computing methodologies → Distributed algorithms**.

KEYWORDS

Blockchain, Mempool, Accountability, Transaction reordering.

ACM Reference Format:
Bulat Nasrulin, Georgy Ishmaev, Jérémie Decouchant, and Johan Pouwelse. 2023. LØ: An Accountable Mempool for MEV Resistance. In *24th International Middleware Conference (Middleware '23), December 11–15, 2023, Bologna, Italy*. ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3590140.3629108

1 INTRODUCTION

Enabled by blockchain technologies, Decentralised Finance (DeFi) tools and mechanisms have generated a lot of interest as building blocks for novel digital markets, both in terms of practical applications amounting to over 80 billion USD in total value locked at the moment of writing, and in terms of significant research interest [43]. Furthermore, these tools enable monetization mechanisms for the new paradigm of Web3 development, providing alternatives to monopolistic centralised digital platforms. Decentralised exchanges, lending markets, derivatives, and other products built on permissionless blockchains are just some examples of these novel financial applications. However, these developments are undermined by unresolved issues of transaction manipulations, such as *censorship*, *injection*, and *re-ordering* of transactions, at the expense of application users at underlying layers of blockchain protocols

This problem led to the notion of Miner Extractable Value (MEV)[1], which refers to the maximum revenue a miner can obtain from benign or manipulative transaction selection for block production [13, 34]. It is a pervasive and systemic issue at a large scale as exemplified by the Ethereum blockchain, where MEV transaction manipulations have generated over 320 USD million of revenue for bots and miners [40]. Furthermore, over 90% of the blocks produced on Ethereum contain MEV transaction manipulations [34]. Such manipulations not only undermine users' trust but also induce systemic issues like congestion, inflated fees, and system instability [27].

We argue that the root cause of MEV is a lack of accountability at the base layer of permissionless blockchain protocols, sometimes referred to as 'dark forest' [34]. By base layer, we refer to the processing steps that happen before consensus has to be reached on a block, such as sharing pending transactions (recorded in the mempool) with other miners and assembling them into blocks. In contrast to what happens at the consensus layer, at the base layer, miners are expected to act as trusted parties. As such, a miner that creates a new block can arbitrarily select the transactions from its mempool. In practice, miners can therefore arbitrarily censor, inject or reorder transactions [18].

The issue of transaction manipulations, while addressed by some MEV mitigation tools, remains without comprehensive solutions [32]. Many current methods primarily target the application and consensus layers [45]. Rather than preventing MEV attacks, they often just aim to mitigate their effects. Notably, the Proposer Builder Separation (PBS) approach, deployed in Ethereum's Flashbots middleware, doesn't eliminate MEV but rather redistributes its revenues [3]. Some theoretical models, like fair ordering consensus protocols [22], do prevent manipulations. Yet, they are tailored for permissioned environments, smaller network sizes, and entail significant changes to the blockchain consensus layer.

As transaction manipulations arise from the lack of accountability at the base layer of blockchain protocols, we argue that

Middleware '23, December 11–15, 2023, Bologna, Italy
© 2023 Association for Computing Machinery.
ACM ISBN 979-8-4007-0177-1/23/12.
https://doi.org/10.1145/3590140.3629108

[1] Sometimes also referred to as Blockchain Extractable Value, or Maximum Extractable Value.

comprehensive mitigation of MEV requires addressing trust assumptions at this particular layer. To address them, we design LØ, an *accountable mempool* protocol.

In LØ miners become accountable for the process of transaction selection and ordering. As new transactions are propagated among miners, they exchange and record commitments on the content of their mempools with each other. New transactions are shared in bundles, and commitment is recorded on a whole transaction bundle. This provides a local partial ordering of transactions. Our system is based on *pairwise commitments* that are exchanged during a mempool reconciliation phase, which is executed before the consensus protocol. This allows miners to witness each others' transaction selection and commit to a particular order and set of transactions that they will use for block generation. Therefore, LØ ensures that any transaction manipulation, such as transaction censorship, injection and reordering, can be detected and proven by a correct node.

Our system is agnostic to the specific type of consensus protocol used in a permissionless blockchain system. It can be seamlessly integrated with existing blockchain solutions, as a relatively simple modification of Peer-to-Peer (P2P) protocol that propagate transactions and blocks. It does not require any additional cryptographic setups, and it does not impose a significant performance overhead. We leverage the Minisketch data structure for the reconciliation of mempools to implement bandwidth-efficient commitments [29].

Overall, this paper makes the following contributions:

- We categorize transaction manipulation primitives behind any potential MEV attack, correlating them with the transaction processing stages susceptible to miner manipulations (Sec. 2).
- We outline our system model in (Sec. 3) and introduce LØ, a protocol engineered to mitigate transaction manipulations by detecting and attributing them to individual mining nodes. Sec. 4 delves into its unique policies that enable the detection.
- We detail how LØ detects transaction manipulation attacks and potential mechanisms for the enforcement of these policies (Sec. 5), and possible attacks against accountability in LØ.
- We present our performance evaluation, which demonstrates that LØ is practical. It is both bandwidth and memory-efficient. For example, it only requires up to 10 MB of additional storage for a network of 10,000 nodes and a workload of 20 transactions per second. At the same time, it is at least four times more efficient than the classical flooding-based mempool exchanges (Sec. 6).

2 TRANSACTION MANIPULATIONS AT THE BASE LAYER

This section explores the layers and stages of blockchain transactions, with a focus on manipulation primitives, illustrated in Fig. 1. The figure demonstrates the functional modules of a permissionless blockchain, separated into two essential layers, base and consensus, and further broken down into four specific transaction processing stages.

2.1 The Base Layer versus the Consensus Layer

We distinguish the *base layer* of a blockchain system from its consensus layer. In the complete life-cycle of a transaction, the base layer corresponds to the steps that precede the block consensus phase as illustrated in Fig. 1. These steps include the creation of the transaction and its initial sharing, its inclusion in the mempools, the reconciliation of the mempools between miners, and the inclusion of the transaction in a candidate block.

We emphasize that the block-building stage, where a miner selects transactions that it includes in a candidate block is a preconsensus phase. Indeed, while sometimes block building is described as part of blockchain protocols, it is strictly speaking not a part of the consensus mechanism as blocks can be produced offline, as illustrated by PBS in Ethereum and selfish mining in Bitcoin [17]. We further distinguish the base layer from the network layer of blockchain protocols, as the latter is required in all transaction processing stages, including during consensus.

The base layer typically provides much lower guarantees against misbehaving nodes than the consensus layer. In the base layer, miners only conduct checks on the validity and priority of transactions (which is related to miner's fee) and add them to a local pool of unconfirmed transactions referred to as the 'mempool' [41]. However, miners are considered to be trusted parties with regard to the selection, withholding, and ordering of transactions [18]. Therefore, all stages of the transaction life-cycle that precede consensus allow transaction manipulations.

2.2 Transaction Manipulation Primitives

We consider practical attacks that include reordering of transactions by miners. These attacks have been observed in practical settings and described in academic works that relate to MEV [45]. In practice, these attacks combine different types of transaction ordering manipulations. A common taxonomy of MEV attacks is application-specific and depends on the source of attack revenue. Well-known attack types include *sandwich-attacks, front running*, and *back running*, which are associated with decentralized exchanges, *sniping*, which is associated with Non Fungible Token auctions, and *liquidations*, which are associated with collateralized loan protocols. This taxonomy evolves as new MEV attacks rapidly emerge with new applications.

In this paper, we consider a different taxonomy focusing on specific attack primitives on the base layer. These primitives allow a broad range of MEV techniques, either on their own or in combination, are *censorship, injection*, and *re-ordering* of transactions.

Censorship. Censorship is the ability of a miner to delay or ignore new transactions. Censorship can enable different financially motivated MEV attacks, such as *sniping*, executed alone or in combination with other primitives. For example, when receiving transactions for a bid in Non Fungible Token auctions, a faulty miner can censor competing transactions to become the auction winner. This censorship mechanism can take place during the direct communication with the client (Stage I), during the mempool exchange (Stage II) or during the block building stage (Stage III).

Mempool Censorship. Faulty miners can ignore transactions received from some other nodes, and exclude their valid transactions from their mempool. We assume that a faulty miner either provides a fake transaction reception acknowledgement, or does not acknowledge it at all. This type of attack enables censorship at the

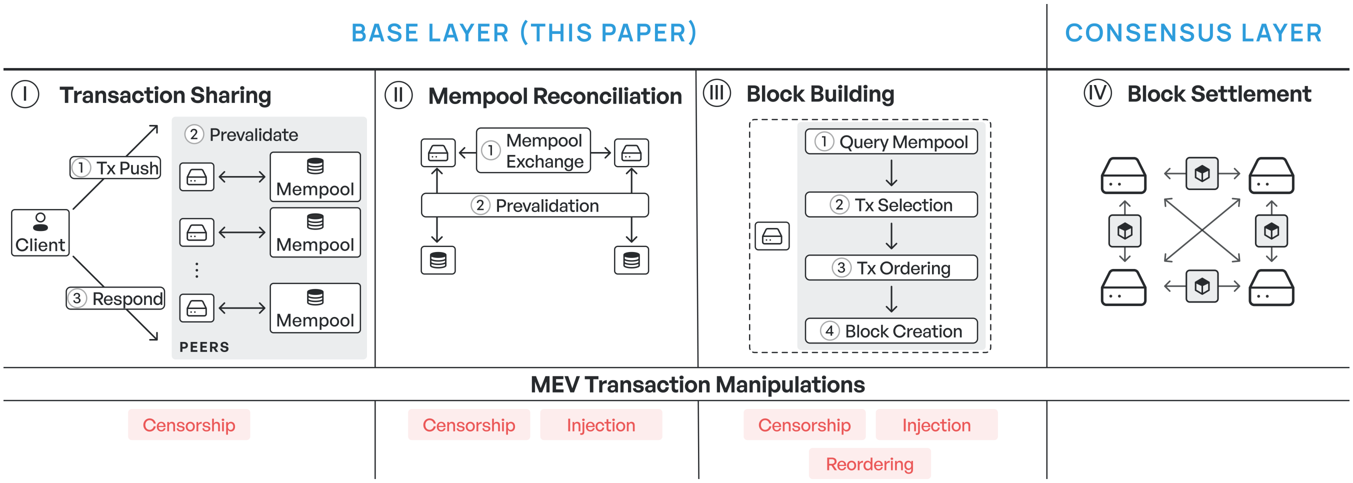

Figure 1: Three types of transaction manipulation primitives that occur accross the functional modules of a permissionless blockchain divided into two layers (base and consensus) and four stages.

level of a mempool [45], and facilitates transaction manipulation based on front-running.

Blockspace censorship. Faulty miners can exclude valid transactions from blocks, even after acknowledging their reception and including them in their mempool.

Injection. We assume that honest miners include transactions received from other nodes in new blocks following a deterministic order. Honest miners can also add their own new transactions, under the assumption that updated mempool commitment is shared with other nodes and acknowledged. Faulty miners inject new transactions in blocks in an arbitrary manner, without prior sharing of the updated mempool and without acknowledgements. This type of attack can result in certain types of transaction manipulations such as *front-running, sandwich,* and *back-running* [13].

Re-ordering. Upon successfully mining a block, faulty miners can attempt to populate it with transactions in an order that deviates from a protocol specification and that can be detrimental to other nodes. In a reordering attack a faulty miner processes transactions in an order that differs from the one they were received with. Contrary to injection attacks, a faulty node that executes a re-ordering attack does not add new transactions.

2.3 Transaction Processing Stages

Attacks can happen at different stages of the transaction life-cycle. We model the processing of a transaction in a generic blockchain system in Fig. 1. This processing happens in four stages: (I) initial transaction sharing, (II) mempool reconciliation, (III) block building and (IV) block settlement. In the following, we describe each stage and discuss the corresponding attacks that enable transaction manipulations such as MEV.

Stage I. Initial transaction sharing. A transaction is first created on the client side. The client signs the transaction with its private key. The transaction contains all the required context to be processed by miners, such as signature, wallet address, execution commands, transaction fee, etc. The client shares the transaction with a subset of peers that it personally knows or whose identity is publicly known (step ①). The peers receive the transaction and attempt to prevalidate it (step ②). Our system is agnostic with respect to specific the requirements for transaction prevalidation. For example, successful prevalidation of a transaction may require: a valid signature from a client, sufficient amount of funds in a client account, and the inclusion of a sufficient transaction processing fee.

Miners that successfully prevalidate a transaction insert it in their local mempool storage. Optionally, miners might respond to the client with the transaction status, to acknowledge inclusion of a transaction in a mempool (step ③). Also optionally, a client can query a miner to get an acknowledging of transaction inclusion in a mempool. A malicious peer can censor the transaction at the point of prevalidation, without adding it to the mempool. For example, a peer can exclude a client based on its id, e.g. all transactions originating from a specific address. At the same time, the client and peer can collude to include an invalid transaction into a mempool.

Stage II. Mempool reconciliation. At this stage, peers share their transaction mempools (step ①). Typically, a mempool exchange is implemented to first share the transaction IDs, and only later selectively share the transaction content of the corresponding IDs. Once a miner receives the transaction content, it prevalidates the transaction (step ②), similarly to stage I. In theory, this stage allows the miner to converge to the same transaction set for any peer-to-peer network. Unfortunately, in practice, there is no guarantee that miners will converge. A client can be partially or completely excluded from learning particular transactions when communicating with malicious peers. Moreover, miners can inconsistently exchange their mempools. Finally, without a requirement for the mempool reconciliation, a malicious miner can exclude or include any transaction without being detected by other miners. Different types of *injection* and *censorship* can be performed by faulty miners at that stage. For example, a malicious miner receiving a high-fee transaction can withhold it from sharing with other nodes in order to include its in own block later.

Stage III. Block building. Upon creating a block, a miner populates it based on information stored in its local mempool data (step ①). For each block, the miner selects a subset of transactions to fill up the blockspace (step ②). The selected transactions are included in the block in a specific order chosen by the miner (step ③). A final block contains additional metadata, like signature, nonce, or timestamp (step ④). Most of the reported MEV is happening at the stage of block building. Indeed, miners can freely inject, exclude, or order transactions to maximize their profit, performing *injection*, *reordering* and *blockspace censorship*.

Stage IV. Block settlement. LØ is agnostic to the specific consensus process to finalize the blocks. We model miner selection as a random process, where a selected miner builds its block and sends it to other miners. The attacks on this stage are extensively discussed in previous works. The most discussed manipulations include block withholding, block reordering and equivocation attacks. We consider the accountability on this stage out of scope. Our solution can be combined with other solutions addressing the manipulations at this stage, such as Polygraph [11].

3 SYSTEM MODEL

This section describes our system model, which is the common one for permissonless blockchain protocols, such as Bitcoin, Ethereum, etc.

The mining nodes (miners) belong in a set $\Pi = \{p_1, p_2, \dots\}$ and communicate with each other by exchanging messages over the network. We assume that each miner is equipped with a cryptographic key pair, and is uniquely identified by its public key. Nodes have access to a cryptographic signature scheme and messages are authenticated.

Communication Overlay. Nodes are connected in an undirected communication graph. Nodes are free to unilaterally add or drop local connections, and they can leave and later rejoin the network at any time. Nodes exchange messages with their overlay neighbors through a direct connection. We use the notation N_i to refer to the neighbors of a node p_i, i.e., the nodes that are currently directly connected with it.

Bootstrap and Peer Discovery. We assume that nodes that join the system are able to contact bootstrap nodes that facilitate node discovery. When (re)joining the network, each correct node requests a set of known active nodes from the bootstrap nodes. The bootstrap nodes are correct, i.e., they don't bias the peer introduction process. As a result, the nodes operate in a single connected network.

Continuous Sampling: Correct nodes persistently sample the network using a node discovery procedure. LØ employs a Byzantine-resilient uniform sampling algorithm, such as those detailed in [4, 7]. It presumes that the peer sampling algorithm ensures interaction between any correct node within a finite time frame, a premise vital to LØ's detection assurances. Malicious nodes can delay the node discovery procedure, however, it is guaranteed that correct nodes will eventually be able to communicate with each other.

Types of Nodes. In different consensus protocols nodes participating in block creation can be called validators, proposers, builders, etc. Here we only consider the role of block creator and refer to the nodes that create blocks as miners. For the sake of simplicity, we do not consider light clients, which our model can nonetheless cover without modifications. Miners can create new transactions, and they can also propose new blocks with ordered transactions to be included in the blockchain. All nodes maintain a list of unconfirmed transactions (mempool) and exchange it with other nodes in the network through messages.

3.1 Attacker Model

In our network, each node is either correct or faulty. Correct nodes adhere to the reference protocol without data tampering and generate valid messages. Faulty nodes, on the other hand, can deviate arbitrarily from the reference protocol.

We assume that a faulty miner can execute any of the transaction manipulations we previously described: censoring transactions, injecting new transactions out-of-order, or deviating from the canonical transaction order [51]. These attacks can be carried out by a faulty miner in a naive way by sending the same message (e.g., a reordered set of transactions) to all neighbouring nodes, or they can attempt to evade detection of manipulations by *equivocating*, i.e., sending conflicting messages to different nodes. We elaborate further on the security model in Sec. 5. We also assume that faulty nodes might avoid interacting with some other nodes.

3.2 Accountability

We consider the standard accountability property for distributed systems and protocols [20]. We define accountability as the ability for honest nodes to detect transaction manipulations and assign blame at the granularity of a single mining node. Here, a *blame* refers to a message that captures the identity of the miner and its operational state, but also includes evidence that justifies or supports that stated status. This operational state, which we term "status," provides insights into whether the miner is functioning correctly, or exhibits malicious behavior.

Effective accountability relies on the premise of nodes' interaction, typically involving *requests* sent to and responses received from other nodes. In asynchronous networks, an adversary can try to evade detection as it is challenging to distinguish between a misbehaving node that deliberately ignores requests and a slow node. To circumvent this difficulty, we divide blames into two types: *suspicions* and *exposures*. An exposure is a verifiable proof of misbehavior, while a suspicion is a lack of response to a request.

We consider two desirable properties of accountability:

Accuracy: (1) *Temporal.* No correct node is perpetually suspected by a correct node, and (2) *No false positives.* No correct node is exposed as misbehaving by any other nodes.

Completeness: (1) *Suspicion completeness.* Every misbehaving node that ignores requests is perpetually suspected by all correct nodes. (2) *Exposure completeness.* When a node is exposed as misbehaving by a node, it will eventually be exposed by every correct node.

4 LØ: ACCOUNTABLE BASE LAYER

In this section, we present LØ which achieves accountability at the base layer. Specifically, LØ is implemented as a modification of mempool reconciliation and block building stages as defined in Sec. 2.3.

Addressed Manipulation	Current implicit policies	New explicit policies
Censorship	Unreliable Transaction Gossip	Inclusion of All Transactions
Injection	Out-Of-Order Transaction Selection	Transaction Selection in Received Order
Reordering	Arbitrary Order in a Block	Verifiable Canonical Order in a Block

Table 1: Implicit policies in the base layer of typical permissonless blockchain and the new explicit policies we replace them with to detect transaction manipulations.

Our solution at a glance: We address transaction manipulation at the mempool layer. Through pairwise interactions, we design a protocol that makes it infeasible for users to misrepresent the transactions they have observed and the precise times at which they observed them. Using a deterministic block-building protocol, we can reorder transactions. Our accountability protocol enables many ways for effective anti-abuse enforcement.

4.1 New Explicit Policies at the Base Layer

This section introduces LØ, our accountable base layer protocol for permissionless blockchains. LØ improves over the 'vanilla' mempool reconciliation and block-building protocols of permissionless blockchains (stages II and III of Fig. 1).

To enable accountability we require to modify some currently implicit or ill-defined policies at the base layer. Our observation is that current implementations of blockchain systems use implicit policies that significantly complicate the detection of transaction manipulations. First, transaction censorship is not possible to attribute to a miner given an unreliable transaction relay. Every miner has its own relaying policy and even perfectly correctly behaving nodes may choose not to relay anything at all. Second, miners can build a block with any transactions from the mempool, or even inject new transactions during the block creation. Third, there is no 'canonical order' inside a block, allowing for any type of reordering [2].

Instead of these ill-defined policies, we propose three alternative explicit policies to enable the detection of any transaction manipulations, as presented in Table 1. In a nutshell, LØ introduces three new explicit policies: *Inclusion of All Transactions*, *Transaction Selection in Received Order*, and *Verifiable Canonical Order in a Block*. Transaction manipulations are detected as violations of our explicit policies during the mempool reconciliation, or when inspecting the content of a block.

Inclusion of All Transactions. Each miner includes all valid transactions it encountered during the system run in its locally maintained append-only transactions set. Once two nodes are connected they directly exchange their known transactions. The transaction exchange is implemented as a sequence of set reconciliations. The miners exchange multiple transactions in one transaction bundle. This allows two nodes to efficiently obtain the transactions they are missing and as a result, end up with the same transaction sets.

The key ability of LØ is that after a successful round of reconciliation both correct nodes are ensured to have a common set of observed transactions. To ensure that none of the transactions is censored and all processed in the same way miners keep all valid transactions they encounter. Miners commit to be able to reveal all transactions they know about, if necessary.

Transaction Selection in Received Order. During the reconciliation process, each miner commits to the order it received a transaction bundle from another miner. To mitigate any out-of-order injections, the miners are required to process the transactions following their insertion order in their mempool. As miners learn and commit on their mempool transactions, the transactions are then naturally ordered according to the order in which they were received.

Verifiable Canonical Order in a Block. Transactions that are inserted into a newly created block are selected according to a deterministic process. In more detail, committed transaction bundles are first assembled following a sequential order. The order inside a bundle is then pseudo-random: transactions are shuffled using a known shuffling algorithm and an *order seed* value. The order seed value is based on the hash of the last created block.

4.2 Mempool Reconciliation

The mempool reconciliation process (cf. Sec. 2.3) forces miners to correctly share the transactions they accepted into their mempool. In practice, LØ's mempool reconciliation uses two techniques: (i) anti-entropy gossip reconciliations [16, 39]; and (ii) signed commitments [15, 29].

Nodes maintain a mempool of all pending transactions and keep a record of all transactions they have ever received. Nodes reconcile their mempools to disseminate transactions throughout the system and generate commitments that are exchanged during mempool reconciliations. These commitments cover not only the transactions in the current mempool, but all valid transactions ever received by a node at the time of reconciliation.

Mempool reconciliation serves two purposes: (1) it allows miners to learn about new transactions from their neighbours; and (2) it ensures that miners commit to a specific transaction partial order during reconciliation. This partial order must be maintained during block creation. Miners mutually commit to the order by first exchanging a commitment. Miners are inherently motivated to receive transactions from other miners. However, they only disclose the transactions after their counterpart has committed to a specific order of transactions.

Transaction Flow. Alg. 1 provides LØ's pseudocode for a node $p_i \in \Pi$.

A transaction is generated at a node p_i (lines 6-9). Once created, this transaction is stored locally at p_i. The transaction's id is then incorporated into the commitment C_i of node p_i. We describe C_i as a commitment to the sequence of transactions introduced by miner p_i. Concurrently, this commitment acts as a cryptographic verification of the incorporated mempool transactions. The transaction's id is determined using a hash algorithm, represented as $H(tx)$.

At regular intervals, miners prompt their neighbors to commit to fresh transactions by dispatching a new commitment request (lines 11-18). The reconciliation of our mempool between nodes p_i and

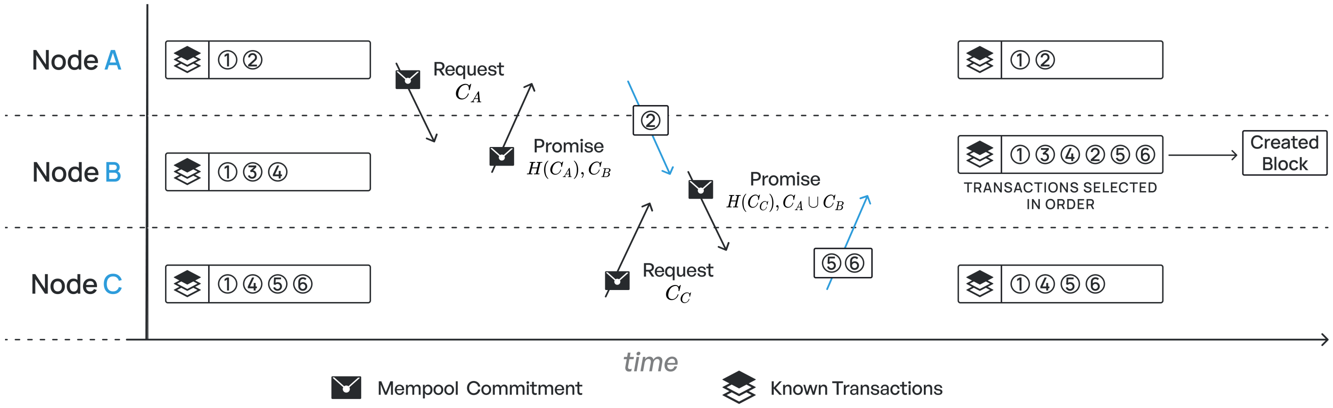

Figure 2: Example of a mempool reconciliation in LØ between node B and nodes A, C. Node B preserves the transaction order for the block it eventually creates.

Algorithm 1 LØ on node p_i.

```
1:  $\widehat{C_1}, \ldots, \widehat{C_N} \leftarrow \emptyset, \ldots, \emptyset$        ▷ Last observed commitments
2:  $\mathcal{E} \leftarrow \emptyset$                                   ▷ Set of exposed miners
3:  $\mathcal{S} \leftarrow \emptyset$                                 ▷ Set of suspected miners
4:  $txs \leftarrow \emptyset$
5:
6:  procedure TransactionCreate($tx$)
7:      $txs \leftarrow txs \cup \{tx\}$
8:      $txid \leftarrow H(tx)$                                    ▷ Hash of $tx$
9:      $C_i \leftarrow C_i \cup \{txid\}$
10:
11: procedure NeighborsSync
12:     for $p_j \in Neighbors(i)$ do
13:         if $C_i \setminus \widehat{C_j} \neq \emptyset$ then                   ▷ Peer j is outdated
14:             $\mathcal{S} \leftarrow \mathcal{S} \cup \{p_j\}$
15:             $\Delta C_{ij} \leftarrow C_i \setminus \widehat{C_j}$
16:             send request $\Delta C_{ij}$ from $p_j$
17:         else
18:             $\mathcal{S} \leftarrow \mathcal{S} \setminus \{p_j\}$
19: on Receive request $\Delta C_{ji}$
20:     $Req \leftarrow \emptyset$
21:     for $txid \in \Delta C_{ji} | txid \notin C_i$ do
22:         $C_i \leftarrow C_i \cup \{txid\}$                       ▷ Commit for tx id
23:         $Req \leftarrow Req \cup \{txid\}$
24:     send $C_i, Req$ to $p_j$
25:
26: on Receive $C_j, Req$
27:     if $\widehat{C_j} \subset C_j$ then
28:         $\widehat{C_j} \leftarrow C_j$
29:         send $txs$ with ids $\in Req$ to $p_j$
30:     if $(C_j \setminus \widehat{C_j} \neq \emptyset \ \& \ \widehat{C_j} \setminus C_j \neq \emptyset)$ then
31:         $\mathcal{E} \leftarrow \mathcal{E} \cup \{p_j\}$                               ▷ Expose
32:         Broadcast $C_j, \widehat{C_j}$
33:
```

p_j transpires in two stages. Initially, node p_i directs a commitment request to node p_j, signifying the new set of transactions possessed by p_i (lines 15-16). As long as the request is outstanding, node p_j remains under suspicion (line 14).

Upon receiving a request from p_j (lines 19-24), node p_i reciprocates with its updated C_i encompassing all the new transaction ids previously requested by node p_i (line 24). In other words, it's an assurance to process them immediately following all known local transactions. Along with this commitment, node p_i also sends out a request for any unacquainted new transactions.

After receiving the commitments of their neighbours, nodes calculate their transaction set differences with them (lines 27 and 30). Since the commitment is signed, it can later be used as a proof of inclusion of transactions—any receiver can use the commitment C_j as verifiable evidence that node p_j should have included transactions in its mempool.

All miners store at least the last received commitments from their overlay neighbours (line 28). On receiving a checksum C it is first validated against previously received set $\widehat{C}$ (lines 27-32). The set $\widehat{C}$ is grow-only and keeps all the transactions committed by the node. If C is inconsistent with the previously reported messages $\widehat{C}$, the evidence of the faulty behaviour is shared with other nodes (line 32). This inconsistency could happen for example when a faulty node is trying to hide a previously reported message or does not report a message received from other nodes.

Example. Fig. 2 illustrates a possible mempool reconciliation. Nodes A, B, and C first exchange transaction commitments. Note that commitments can also be received indirectly, but this scenario is not included in Fig. 2 for simplicity. Node A sends a request, along with the mempool commitment C_A, to node B. Node B reconciles commitment C_A with its own C_B and promises to include node A's missing transactions immediately after all transactions C_B. Node A promptly sends the missing transaction 2 to node B. Shortly afterwards, node C reconciles with node B in a similar manner. However, this time, node B promises to include transactions of node C only after the transactions 1,3,4,2. Let's assume that later, node B creates a new block, possibly because it is elected as a consensus leader. Node B must then select all transactions in the order of the commitment it made, which is 1,3,4,2,5,6.

Implementation Details. LØ utilizes *Minisketch* and *Bloom Clocks* to efficiently orchestrate the mempool reconciliation protocol. In this framework, a commitment comprises both the miner's Bloom Clock and Minisketch. These data structures have dual core functionalities: (1) they detect discrepancies in the digests shared during prior rounds, and (2) they aid set reconciliation to pinpoint a miner's unknown transactions. Collectively, these structures underpin the commitments $\mathcal{C}$ employed in Alg. 1. In particular, Minisketch executes the set reconciliation algorithm used in lines 13 and 30. To expedite inconsistency identification, Bloom Clocks facilitate the consistency verification at line 30.

A **Minisketch** [29] implements the PinSketch [15] algorithm for creating and decoding set sketches based on linear error-correcting codes. Sets of items are portrayed as polynomials with roots equating to the 32-bit integer representation of transaction hashes. The algorithm computes a "sketch" by evaluating this polynomial at specific points, with polynomial coefficients forming the sketch. If you have two sketches (from two different sets), they can be combined by performing a bitwise XOR operation on the coefficients. The result is a new polynomial whose roots represent the symmetric difference between the two original sets. A critical parameter when creating a sketch is the "capacity," which signifies the estimated maximum difference between the two sets. If the actual difference between the sets exceeds this capacity, reconciliation using the sketch fails. When the reconciliation fails we divide the data into two subsets and attempt the reconciliation process on each subset.

The **Bloom Clock** is a space-efficient probabilistic data structure used to order events in distributed systems [35]. It is implemented as a counting Bloom filter, where each item signifies a mempool transaction. Items are hashed and placed into one of the m cells, each containing an integer counter of number of values mapped. This structure quickly highlights discrepancies between nodes based on cell values. We integrate the Bloom Clock with Minisketch to bolster reconciliation efficiency. The process starts with a bloom filter comparison, detecting inconsistencies between sets. Later, nodes construct a Minisketch specific to their transaction set, corresponding to the cells that the bloom filter has flagged as inconsistent. The Bloom Clock's strength is in its preliminary estimation of differences between two sets, significantly reducing Minisketch reconciliation failures.

Summary. *The pairwise commitment scheme ensures that miners are committed to all transactions they discover according to the order in which they are received.*

4.3 Block Building

To avoid manipulations during the block-building stage, we slightly modify the 'vanilla' block-building process with our new policies. The modified block-building process is shown in Fig. 3.

Transaction Selection. Peers select all transactions they encounter during the mempool reconciliation phase and that are included in the mempool (step 1). Miners must verify these transactions. The transactions that are not valid are not included in the block. The transactions that have fees lower than some threshold are not included in the block, and are rejected (step 2).

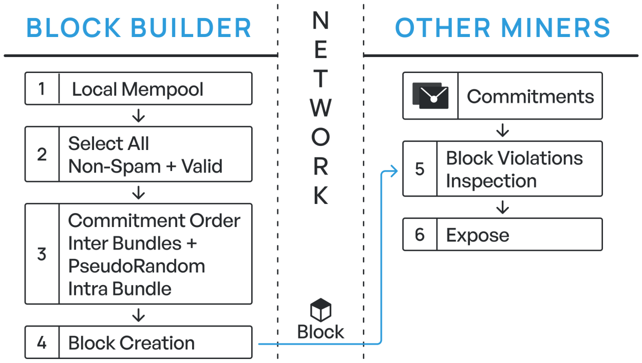
BLOCK BUILDER
NETWORK
OTHER MINERS
1 Local Mempool
2 Select All Non-Spam + Valid
3 Commitment Order Inter Bundles + PseudoRandom Intra Bundle
4 Block Creation
Block
Commitments
5 Block Violations Inspection
6 Expose

Figure 3: Block building and inspection in LØ.

Transaction Ordering. The selected transactions are ordered in a verifiable canonical way (step 3). Recall from the mempool reconciliation process that transactions are partially ordered with the commitments order as the commitments define the order between transaction bundles. We also define a deterministic pseudo-random order function inside each of the bundle. We use a hash of previous block as a seed for the intra-bundle order function.

Block Inspection. Next, a block is created (step 4) and shared with the network. Given the mempool commitments, any node can verify the produced block by inspecting its content with respect to the LØ reference protocol (step 5). Note that block inspection is a separate process from block validation, and does not affect the block inclusion into the chain. Any violation exposes the block creator (step 6), by comparing the block content with the known commitments. This is achieved by comparing the transaction order in the block to the declared mempool commitments. Each commitment and block has an incremental counter for appropriate comparison. This means that a correct node can expose a malicious node upon detecting any inconsistent commitment.

Summary. *During the block-building process, miners select and order transactions deterministically.*

5 DEALING WITH ATTACKS

This section delves into potential attacks and the countermeasures facilitated by detection mechanisms and enforcement tools. LØ focuses on reliably detecting MEV attacks on transaction ordering, addressing manipulations such as *censorship*, *injection*, and *re-ordering* in Sec. 5.2. We also discuss threats to our *detection mechanism* in Sec. 5.3. While LØ emphasizes detection and is consensus-agnostic, we briefly touch upon enforcement mechanisms enabled by L0 in Sec. 5.4.

5.1 Assumptions and Scope of Attacks

Given our assumption on peer discovery, LØ relies on an overlay that facilitates frequent neighbor shuffling. Specifically, LØ's underlying overlay must be resilient to Sybils, which is realistic since previous work showed robust guarantees even in presence of 99% of Sybils [38]. Each peer periodically rotates its neighbors, and the peer discovery process continues until it is provided with a sufficient number of non-suspected and non-exposed peers.

We employ a peer sampler that periodically provides each node with a set of peers that are chosen uniformly at random [4, 7, 28]. We impose two requirements on the sampling algorithm: (i) all honest peers in the network must eventually form a connected subgraph; and (ii) the algorithm should achieve an unbiased uniform sampling of network members.

5.2 Detecting Transaction Manipulation

We consider the transaction manipulation attacks described in Sec. 2. Every node utilizes a block inspection module to detect violations. Nodes are required to disclose all their known transactions and they must consistently disclose each commitment or they run the risk of being identified as faulty. Manipulation is detected either by comparing two commitments or by contrasting the block content with known commitments signed by the block creator.

Manipulation detection. The ability to detect manipulation hinges on the availability of commitments signed by the block creator. Initially, overlay neighbors gather these commitments when connecting with each other. Nodes are mandated to timely respond to these commitment requests. A failure will lead to suspicion of fault by every honest miner in the network. Furthermore, nodes inherently have an incentive to acquire new transactions from other nodes, an action which only ensues after responding with a commitment.

Second, to ensure that other nodes can also detect and reach an agreement on whether transaction manipulation has occurred, nodes periodically share their most recent commitments with their neighbors. This sharing is triggered when establishing a new overlay connection during a shuffle, or when disseminating a blame message. All commitments are retained, so when a node comes online, it disseminates its latest known commitments triggering a 'NeighboursSync' as shown in Alg. 1.

Countering Attacks during Block Building. The order function ensures that order manipulation attacks can be detected, as any block where the transaction order deviates from the canonical one will be detected. Similarly, a block-space censorship attack is detected as a deviation from the selection function rules. While a block creator can add their own transactions to a newly created block without first providing a commitment, they must first include all previously committed transactions in their block. The new transaction can only be appended after all committed transaction bundles.

Suspicion and Misbehavior Sharing The *Accountability Mechanism* incorporates liveness checks and requires a timely response. If a node does not respond to transaction requests before a timeout, it is suspected by the requester. The requester may resend the request multiple times before suspecting the node. Correct nodes retain all pending requests. If a node is suspected, the requester broadcasts the suspected requestee's identity to other nodes, along with information on pending requests and the requestee's last known commitments. A node may retrieve pending requests after a partition or a crash. Once it publicly responds to all pending requests, no correct node will suspect it.

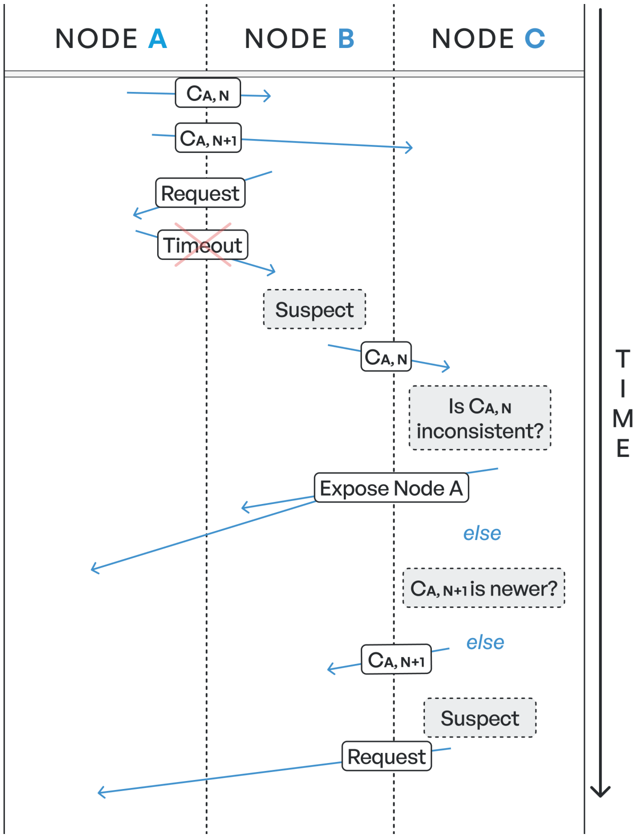

Figure 4: Consistency check and suspicion mechanisms.

In Fig. 4, node B has an earlier commitment $(C_{A,n})$ from node A. Node C has the latest commitment $(C_{A,n+1})$ from node A. Node B sends a request for a commitment on a particular transaction τ from node A, but does not receive a response. After a timeout, node B suspects node A and broadcasts a suspect status along with the latest commitment $(C_{A,n})$ from A that is available to B to its neighbors, in this case, to node C. The suspicion mechanisms is used for every *request* message used in Alg. 1, such as the request of the new commitment (line 16), or the request of transactions (line 24).

Equivocation Detection. A consistency check occurs when a node is suspected. Commitments are append-only sets and thus follow chronological order. When a node has two commitments, it can easily detect any inconsistency between the previous commitment n and the latest commitment $n + 1$ by reconciling two Minisketches. Without loss of generality, consider an example of suspicion and consistency check in. Node C receives two commitments originating from node A, i.e., commitment $(C_A, n + 1)$ from node B, and $(C_A, n + 1)$ from node A. Node B has tried to get a commitment on transaction τ from A and suspects A because of the high response delay. Node C will check whether (C_A, n) and $(C_A, n + 1)$ are consistent with each other.

- If these commitments are inconsistent, node C exposes A as a misbehaving node.
- If (C_A, n) and $(C_A, n{+}1)$ are consistent and $(C_A, n{+}1)$ already includes a commitment on a transaction τ, then node C will share the latest commitment $(C_A, n + 1)$ with B.

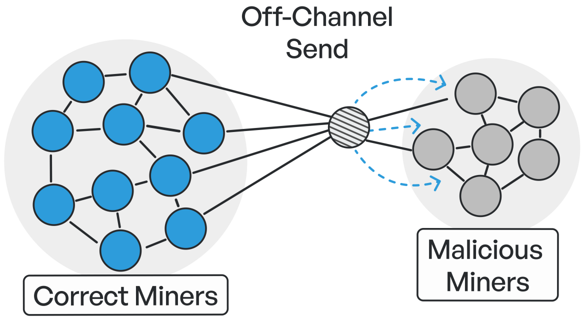

Figure 5: Colluding malicious miners communicating off-channel to evade detection.

- If (C_A, n) and $(C_A, n+1)$ are consistent but $(C_A, n+1)$ does not include a commitment on τ, then C will send a request for commitment on τ to C and suspect C.

5.3 Attacks on LØ

In our model, we assume that miners are incentivized to learn about more transactions. This assumption aligns with empirical observations, as miner profitability correlates with their ability to discover new transactions [32].

A malicious miner might opt not to share commitments with other peers. Yet, this behavior confines the miner's communication strictly to other colluding miners. Suppose the originator of transaction t, denoted as node A, disseminates the transaction t after gathering commitments from its neighbors N_A. Now, if a malicious miner C, who is not part of N_A, wishes to include the transaction t out of order in its block, they first need to learn about transaction t from another peer B. This peer B must have directly or indirectly interacted with node A. The malicious miner C has two strategies when communicating: (1) register the interaction and commitment with a colluding node B, or (2) exchange transaction t off-channel without making any commitments. This attack strategy is illustrated in Fig.5.

Detection of collusion hinges on tracking the commitment chain from the transaction's original creator, node A, to the block creator C. Post-block creation, node A can obtain commitments related to transaction t from node C, revealing its source. If node B reordered transaction t, it would be implicated; otherwise, node C would be exposed. If the transaction t lacks recording, node C faces blame for introducing a transaction without node A's commitment.

Additionally, colluding miners or Sybil miners must have a high probability of becoming the consensus leader to include a specific transaction. To increase this success rate, a substantial set of colluding miners or Sybils is required, which is costly considering the initial investment and the absence of profits from honest protocol participation.

5.4 Possible Enforcement Policies

Reliable detection and blame assignment allow for MEV mitigation through the enforcement of policies. The choice of specific enforcement mechanisms depends on the consensus protocol. Given that LØ is agnostic to the particular consensus algorithm used, a detailed analysis of specific enforcement mechanisms is beyond the scope of this paper.

However, in *Proof-of-Stake* (PoS) consensus algorithms, various slashing strategies can be applied to misbehaving nodes [9]. Since validating nodes in PoS must invest a certain amount of funds to become validators, slashing of stake incurs a financial loss. For consensus algorithms based on the reputation of validating nodes, slashing of reputation can equivalently serve as a penalization mechanism [46]. Misbehaving nodes can also be penalized at the network layer level, such as temporary disconnection from the network [18]. In addition to penalizing misbehaving miners, detection allows the implementation of mechanisms for the rejection of blocks that deviate from the canonical transaction order [36].

6 EVALUATION

In this evaluation section, we assess LØ's performance and resilience, particularly when confronted with malicious nodes within the network. Our examination focuses on several key metrics, including the system's ability to timely detect transaction manipulations the time required for transaction processing, and the overall system overhead.

6.1 Experimental setup

Unless otherwise stated, the parameters in our reported experiment are set as follows: There are 10, 000 nodes, generating a workload of 20 transactions per second, with each transaction being 250 bytes in size. The transactions were injected into our system based on a realistic dataset of Ethereum transactions [31]. Each experiment was repeated 10 times, and the average result of these runs is reported.

We constructed a connected topology where each node had eight outgoing connections and up to 125 incoming connections, in line with the default Bitcoin parameters. Every node attempted to reconcile with three random neighbors every second. The request timeout was set to 1 second. If a request was not fulfilled within this time, it was resent three times, after which the node was suspected of being faulty. The Minisketch size was set to 1,000 bytes, sufficient to reconcile a set difference of up to 100 transactions, allowing the Minisketch to fit into a single UDP packet. If reconciliation failed, all transactions were divided into two subsets, and the process was repeated with two sketches. The size of Bloom-Clocks was fixed at 32 cells (i.e., 68 bytes in total).

We evaluate LØ experimentally on a national research cluster [5]. Each server in the cluster is equipped with an Intel Xeon E5-2630 CPU with 24 physical cores operating at 2.4 GHz, hyper-threading enabled, and 128 GiB of main memory. The servers are interconnected via a Gigabit Ethernet network. LØ is implemented in Python, and our prototype is publicly available online [12]. We emulate realistic network latencies using netem [30] and incorporated ping statistics from 32 cities worldwide from the WonderNetwork dataset [42]. Every miner is assigned to a city in a round-robin manner.

6.2 Resilience To Malicious Miners

We evaluate the influence of colluding miners engaged in censorship activities within the network. Specifically, we focus on how

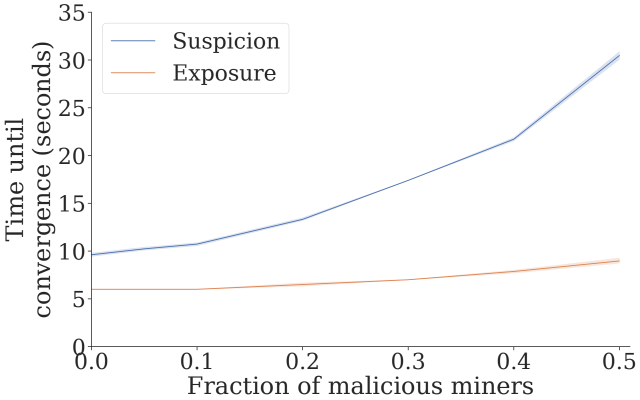

Figure 6: Time necessary in LØ to suspect or expose a malicious miner depending on the fraction of colluding malicious miners in the system censoring the transactions.

they affect the convergence across correct miners. These malicious miners aim to hinder correct nodes from receiving information about transactions, commitments, exposure, and suspicion messages, effectively executing attack on LØ detection mechanism, as described in Sec. 5.3 in parallel to attempting to execute MEV attacks as described in Sec. 2.2. All malicious miners are assumed to be interconnected. In these experiments, we ensure that all correct nodes maintain connectivity, meaning that for every pair of correct nodes, there exists at least one path between them consisting solely of correct nodes within the network. We achieve it by first running an unbiased sampling algorithm [4, 7].

Fig. 6 'Exposure' depicts the duration needed for all correct nodes to achieve convergence, based on the count of faulty nodes present in the network. Notably, the introduction of faulty nodes slightly prolongs the time for all correct nodes to become aware of the exposure message, pushing it to a range of 6-7 seconds post the initial detection and message creation by the first miner.

Furthermore, we show LØ capability to identify faulty nodes that ignore requests. The duration until each correct node suspects all faulty nodes is presented in Fig. 6 under 'Suspicion'. Predictably, the span until all faulty nodes are suspected surpasses the time taken for nodes to recognize an exposure message. This is attributed to the nodes having to initiate a request and subsequently await its timeout.

6.3 Transaction Latency

In this experiment, we document two specific types of transaction latencies: first, the duration required for a transaction to be included into a mempool, and second, the time necessary for its inclusion in a block. We detail the period it takes for miners to detect a transaction and assimilate it into their mempool, with the latency density distribution illustrated in Fig. 7. The data reveals that convergence on the transaction among nodes is achieved after an interaction with 5 to 6 nodes. On average, a transaction is discovered by a node in 1.14 seconds.

To demonstrate the effects of our new policies on block building, i.e., selecting transactions in order, we simulate a block creation process at randomly selected miners with an average block time of 12 s, which is the block time in Ethereum. We report the average time it takes for a transaction to be included in a block in Fig.8. We compare the policy for block creation described in Sec. 4.3 ('FIFO' ordering) with the policy that is currently widely used in public blockchains, i.e., creating a block with the highest-fee transactions of the mempool (referred to as Highest Fee').

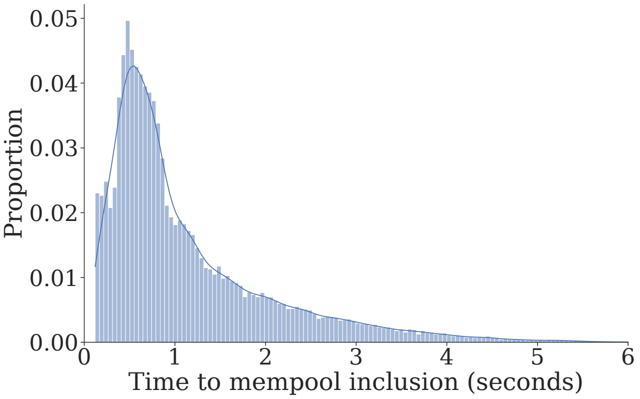

Figure 7: Density distribution of the time required in LØ for a miner to include a transaction into its mempool.

The average transaction latency for the 'FIFO' ordering is 3 seconds, while it is around 7-8 seconds for the 'Highest Fee' strategy. Furthermore, we observe that the 'Highest Fee' strategy exhibits a much larger variation, with many low-fee transactions experiencing very high latency. LØ's orders transactions according to the order with which they have been received by miners, which leads to transactions being processed sequentially.

6.4 Comparative Analysis of Bandwidth Efficiency

We compare LØ to three baselines: 'Flood', PeerReview [20], and Narwhal [14]. 'Flood' is the standard mempool exchange method where miners relay a 'Mempool' message listing their current transaction hashes. Receivers subsequently request any transactions they don't recognize. PeerReview is a universal accountability protocol, where each miner keeps a message log, where each miner maintains a message log, with eight random witnesses assigned per miner. These witnesses periodically retrieve and review miners' logs for any indications of malicious activity, whether it be injection (commission) or censorship (omission). Narwhal is a DAG-based mempool protocol designed to more efficiently deliver transactions.

We implemented the Flood, PeerReview and Narwhal protocols in Python and tested them in the same environment as LØ. Specifcailly, Specifically, in Narwhal, each node creates batches of recent transactions every 0.5 seconds and reliably broadcasts them. A batch, upon receiving acknowledgments from over two-thirds of the network, is then incorporated into a header. The header is broadcast to the network. Peers who are missing any batch from the header have the option to directly request it from the originator of that header. In our comparison involving 200 nodes, our results demonstrate that Narwhal outperforms LØ in terms of latency by

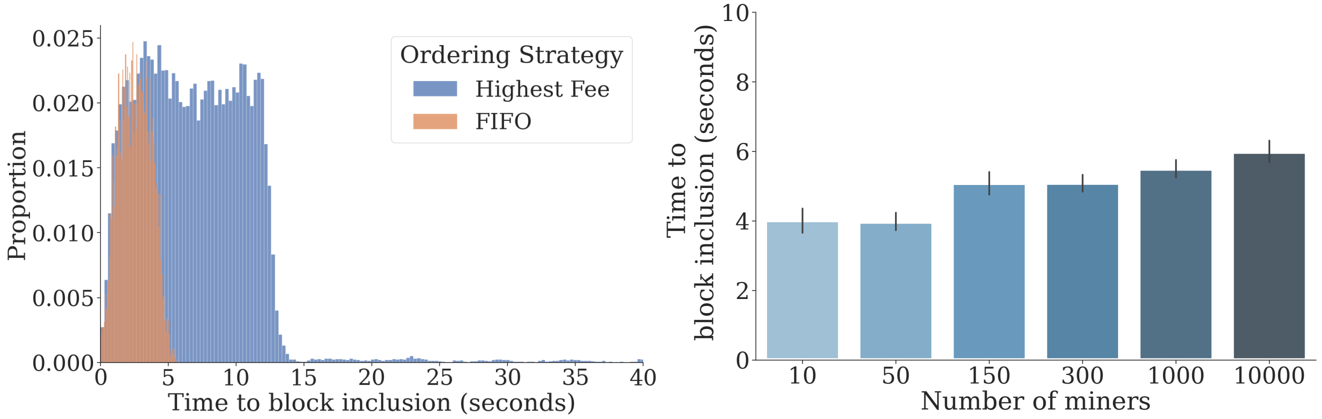

Figure 8: (Time until transaction is included in a block with two policies 'Highest Fee' vs. 'FIFO' ordering (Left). Time until transaction is included in a block in LØ as a function of the system's size (Right).

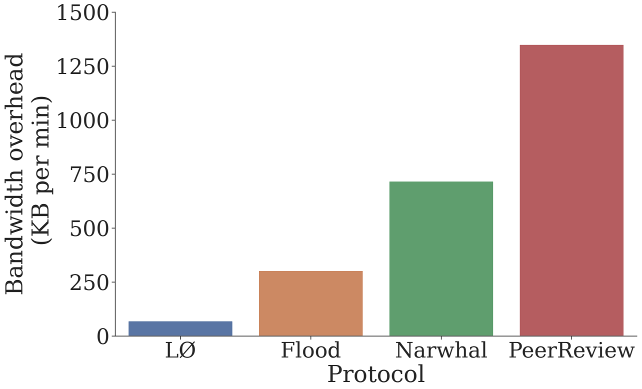

Figure 9: Bandwidth overhead measured for different protocols.

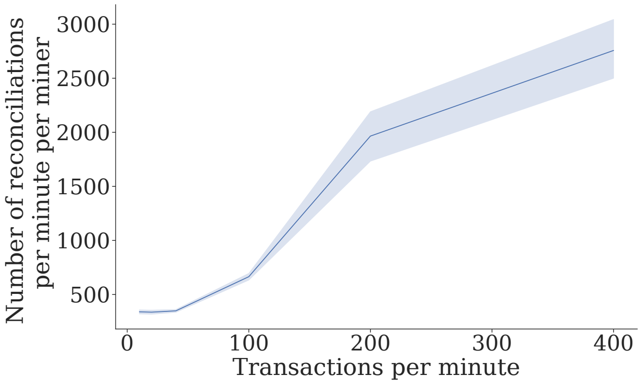

Figure 10: The average number of reconciliations in LØ per minute depending on the workload.

1-2 seconds. However, this performance comes at a cost: Narwhal incurs a substantial increase in bandwidth overhead, leading to usage rates that are 7 to 10 times greater than those associated with LØ. This increased overhead is primarily due to Narwhal's need for the swift gathering of batch approvals from at least two-thirds of the network. Therefore, despite Narwhal's potential latency advantages, its scalability in permissionless environments is hindered by its substantial bandwidth requirements.

The comparison of the protocols' bandwidth overhead is depicted in Fig. 9. Note that we omit the bandwidth overhead for sharing transactions, as it is the same for all three protocols. Our protocol proves to be the most bandwidth-efficient among the three, incurring bandwidth overhead that is 20 times less than that of PeerReview.

6.5 Memory and CPU Overhead

The process of encoding and decoding with Minisketch involves an overhead that increases proportionally with the magnitude of the set difference, as documented in [19]. For instance, calculating a set difference comprising 1, 000 items takes approximately 10 seconds using Minisketch. To circumvent this limitation, we propose an optimization strategy for our system, LØ.

Our approach primarily involves hash-partitioning the mempool space into multiple subsets. This method is based on the techniques described in [19]. Specifically, when reconciliation fails, instead of attempting to reconcile the entire mempool, the node divides it into two partitions and generates an additional Minisketch for each segment. This strategy significantly reduces the time required for encoding and decoding. For a set difference of 1, 000 items, our method completes all necessary sketches in under 100 ms, a substantial improvement over the original 10-second duration.

The effectiveness of this optimization is further illustrated in Fig. 10, which depicts the average number of sketch reconciliations per minute per node under varying workloads.

Regarding memory overhead, LØ necessitates minimal extra memory to store the commitments for each neighboring node. The actual size of these commitments depends on the system's workload. Under a workload of 120 transactions per minute, the commitment size is approximately 1.17 KB. This size increases with the workload, reaching around 9.36 KB under a workload of 24,000 transactions per minute. Notably, even under extreme conditions where a miner may need to store the commitments of all 10,000 nodes in the network, the total memory required would only amount to roughly 87 MB, which is relatively insignificant considering the total processing resources.

7 RELATED WORK

The problem of MEV has attracted a considerable amount of research [34, 45, 51]. Different MEV mitigation mechanisms can be categorized according to implementation at different layers: *application, consensus, base.*

7.1 MEV Mitigation at the Application Layer

Decentralized Exchange (DEX) such as Cowswap implement mechanism known as *Batch Auctions.* In this system, instead of immediate execution, orders are accumulated off-chain over a period, then aggregated and executed collectively in batches [1]. However, the utility of this approach is confined to its original context, rendering it effective against a narrow spectrum of MEV attacks, specifically front-running and sandwich attacks.

A^{2}MM is a DEX design that atomically performs optimal routing and arbitrage among the considered integrated Automated Market Makers (AMMs) , minimizing subsequent arbitrage transactions [50].

7.2 MEV Mitigation at the Consensus Layer

Proposer-builder separation (PBS) is a proposal aiming at MEV minimization [8]. The latest iteration of this mechanism, MEV-Boost, is implemented as a middleware. It enables private communication channels between clients creating new transactions and validating nodes. However, this approach has significant trust assumptions, such as relays not reordering or censoring transactions, which empirically do not hold [45].

Pre-ordering solutions aim to separate transaction ordering from execution to ensure 'fair' ordering. The Helix consensus protocol [44] guarantees the random selection and ordering of transactions in blocks by relying on a randomness beacon within the consensus protocol. Aequitas [23] provides guarantees on transaction ordering within a block but assumes a permissioned environment and introduces significant communication overhead. Pompe [49] is a Byzantine ordered consensus (BOC) protocol that clearly separates ordering from consensus, thus enforcing guarantees on the total order of transactions. Wendy [24, 25] describes ordering protocols for permissioned systems. Enforcing relative order requires building a dependency graph to prevent transactions from being included in a block before their dependencies [10, 22, 23]. Enforcing fair ordering is more resource-intensive than enforcing our accountability properties and is not practical in a permissionless setting.

Heimbach and Wattenhoffer suggest encrypting the content of transactions, then ordering them, revealing the content only after this process [21]. This method is employed by Fino, which melds MEV protection with a BFT protocol under a partial synchrony model, utilizing a DAG transport protocol [26]. Similarly, Lyra [47], another Byzantine ordered consensus protocol, adopts a commit-reveal scheme, dependent on Verifiable Secret Sharing (VSS). However, the encrypt-commit-reveal strategy demands more resources compared to our accountability-centric method and necessitates extra trust assumptions to guarantee the eventual revelation of encrypted transactions.

7.3 MEV Mitigation at the Base Layer

Secret Mempools conceal transaction content to prevent censorship or reordering. F3B represents a universal method for real-time transaction encryption, utilizing a commit-and-reveal structure [48], while Ferveo offers a protocol dedicated to Mempool Privacy within BFT consensus blockchains [6]. Both approaches, however, operate under the assumption of permissioned environments. Conversely, Shutter, a system designed to safeguard Ethereum smart contracts from frontrunning, employs a threshold cryptography-based distributed key generation (DKG) protocol [37]. Despite its efficacy, Shutter incurs additional latency and hinges on substantial trust assumptions, given that the key generation process is entrusted to a select committee operating on a private, Tendermint-based blockchain.

ZeroMEV is a existing MEV mitigation solution designed for the Ethereum network, functioning as a base-layer implementation through a Geth software fork used as a validator execution client [33]. It prioritizes transactions using a timestamp-based system, adhering to a local FIFO (First In, First Out) order. However, ZeroMEV lacks accountability measures and hinges on a substantial trust assumption, necessitating reliance on the validators' altruism.

8 CONCLUSION

We introduced LØ, an accountable base layer for permissionless blockchains, which provides detection guarantees against MEV attacks. To do so, LØ mandates that miners log all the transactions they receive into a secure mempool data structure, and exchange commitments on their mempool content. Any inconsistency, such as transaction withholding or equivocation, is exposed during a mempool reconciliation process with a correct miner, which happens with high probability thanks to an underlying Byzantine-resilient peer sampling protocol. More precisely, to ensure the exposure of faulty miners, LØ simply requires correct miners to be eventually interconnected with each other.

We outlined the transaction manipulation attacks associated with MEV that miners might execute and mapped different attack types to the relevant stages of a transaction's lifecycle within the protocol. Our performance evaluation demonstrates the practicality of LØ. It is bandwidth and memory efficient, using only 10 MB with 10,000 miners and a workload of 20 transactions per second. Moreover, it is at least four times more bandwidth efficient than classical flooding-based mempool exchanges.

ACKNOWLEDGMENTS

This work was funded by NWO/TKI grant BLOCK.2019.004

REFERENCES

[1] [n. d.]. Cowswap. https://cowswap.exchange.

[2] [n. d.]. Ethereum Documentation. https://ethereum.org/nl/developers/docs/mev/.

[3] [n. d.]. Flashbots Blockspace Auction. https://docs.flashbots.net/flashbots-auction/overview.

[4] Alex Auvolat, Yérom-David Bromberg, Davide Frey, and François Taïani. 2021. BASALT: A rock-solid foundation for epidemic consensus algorithms in very large, very open networks. *arXiv preprint arXiv:2102.04063* (2021).

[5] Henri Bal, Dick Epema, Cees de Laat, Rob van Nieuwpoort, John Romein, Frank Seinstra, Cees Snoek, and Harry Wijshoff. 2016. A medium-scale distributed system for computer science research: Infrastructure for the long term. *Computer* 49, 5 (2016), 54–63.

[6] Joseph Bebel and Dev Ojha. 2022. Ferveo: Threshold Decryption for Mempool Privacy in BFT networks. Cryptology ePrint Archive, Paper 2022/898. https://eprint.iacr.org/2022/898 https://eprint.iacr.org/2022/898.

[7] Edward Bortnikov, Maxim Gurevich, Idit Keidar, Gabriel Kliot, and Alexander Shraer. 2009. Brahms: Byzantine resilient random membership sampling. *Computer Networks* 53, 13 (2009), 2340–2359.

[8] Vitalik Buterin. 2021. *State of research: increasing censorship resistance of transactions under proposer/builder separation (PBS).*

[9] Vitalik Buterin and Virgil Griffith. 2017. Casper the friendly finality gadget. *arXiv:1710.09437* (2017).

[10] Christian Cachin, Jovana Mićić, Nathalie Steinhauer, and Luca Zanolini. 2022. Quick order fairness. In *Financial Cryptography and Data Security: 26th International Conference, FC 2022, Grenada, May 2–6, 2022, Revised Selected Papers*. Springer, 316–333.

[11] Pierre Civit, Seth Gilbert, and Vincent Gramoli. 2021. Polygraph: Accountable byzantine agreement. In *ICDCS*. IEEE, 403–413.

[12] Code for the system [n. d.]. https://github.com/tribler/bami.

[13] Philip Daian, Steven Goldfeder, Tyler Kell, Yunqi Li, Xueyuan Zhao, Iddo Bentov, Lorenz Breidenbach, and Ari Juels. 2019. Flash Boys 2.0: Frontrunning, Transaction Reordering, and Consensus Instability in Decentralized Exchanges. *arXiv preprint arXiv:1904.05234* (2019).

[14] George Danezis, Lefteris Kokoris-Kogias, Alberto Sonnino, and Alexander Spiegelman. 2022. Narwhal and Tusk: a DAG-based mempool and efficient BFT consensus. In *EuroSys*. ACM, Rennes France, 34–50. https://doi.org/10.1145/3492321.3519594

[15] Yevgeniy Dodis, Rafail Ostrovsky, Leonid Reyzin, and Adam Smith. 2008. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. *SIAM journal on computing* 38, 1 (2008), 97–139.

[16] David Eppstein, Michael T Goodrich, Frank Uyeda, and George Varghese. 2011. What's the difference? Efficient set reconciliation without prior context. *ACM SIGCOMM CCR* 41, 4 (2011), 218–229.

[17] Ittay Eyal and Emin Gün Sirer. 2018. Majority is not enough: Bitcoin mining is vulnerable. *Commun. ACM* 61, 7 (2018), 95–102.

[18] Arthur Gervais, Hubert Ritzdorf, Ghassan O. Karame, and Srdjan Capkun. 2015. Tampering with the Delivery of Blocks and Transactions in Bitcoin. In *CCS*. ACM, Denver Colorado USA, 692–705. https://doi.org/10.1145/2810103.2813655

[19] Long Gong, Ziheng Liu, Liang Liu, Jun Xu, Mitsunori Ogihara, and Tong Yang. 2020. Space-and computationally-efficient set reconciliation via parity bitmap sketch (PBS). *Proceedings of the VLDB Endowment* 14, 4 (2020), 458–470.

[20] Andreas Haeberlen et al. 2007. PeerReview: Practical accountability for distributed systems. *ACM SIGOPS operating systems review* 41, 6 (2007), 175–188.

[21] Lioba Heimbach and Roger Wattenhofer. 2022. SoK: Preventing Transaction Reordering Manipulations in Decentralized Finance. In *Proceedings of the 4th ACM Conference on Advances in Financial Technologies, AFT 2022, Cambridge, MA, USA, September 19-21, 2022*, Maurice Herlihy and Neha Narula (Eds.). ACM, 47–60. https://doi.org/10.1145/3558535.3559784

[22] Mahimna Kelkar, Soubhik Deb, Sishan Long, Ari Juels, and Sreeram Kannan. 2021. Themis: Fast, strong order-fairness in byzantine consensus. *Cryptology ePrint Archive* (2021).

[23] Mahimna Kelkar, Fan Zhang, Steven Goldfeder, and Ari Juels. 2020. Order-Fairness for Byzantine Consensus. In *CRYPTO*. Vol. 12172. Springer International Publishing, Cham. https://doi.org/10.1007/978-3-030-56877-1_16

[24] Klaus Kursawe. 2020. Wendy, the good little fairness widget: Achieving order fairness for blockchains. In *Proceedings of the 2nd ACM Conference on Advances in Financial Technologies*. 25–36.

[25] Klaus Kursawe. 2021. Wendy grows up: More order fairness. In *Financial Cryptography and Data Security. FC 2021 International Workshops: CoDecFin, DeFi, VOTING, and WTSC, Virtual Event, March 5, 2021, Revised Selected Papers 25*. Springer, 191–196.

[26] Dahlia Malkhi and Pawel Szalachowski. 2022. Maximal Extractable Value (MEV) Protection on a DAG. *arXiv:2208.00940* (2022).

[27] Bruno Mazorra, Michael Reynolds, and Vanesa Daza. 2022. Price of MEV: Towards a Game Theoretical Approach to MEV. In *ACM DeFi*. ACM, Los Angeles CA USA, 15–22. https://doi.org/10.1145/3560832.3563433

[28] Bulat Nasrulin, Rowdy Chotkan, and Johan Pouwelse. 2023. Sustainable Cooperation in Peer-To-Peer Networks. In *2023 IEEE 48th Conference on Local Computer Networks (LCN)*. IEEE, 1–9.

[29] Gleb Naumenko, Gregory Maxwell, Pieter Wuille, Alexandra Fedorova, and Ivan Beschastnikh. 2019. Bandwidth-efficient transaction relay for bitcoin. *arXiv:1905.10518* (2019).

[30] Netem documentation [n. d.]. https://www.linux.org/docs/man8/tc-netem.html.

[31] Giuseppe Antonio Pierro and Henrique Rocha. 2019. The influence factors on ethereum transaction fees. In *2019 IEEE/ACM 2nd International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB)*. IEEE, 24–31.

[32] Julien Piet, Jaiden Fairoze, and Nicholas Weaver. 2022. Extracting godl [sic] from the salt mines: Ethereum miners extracting value. *arXiv preprint arXiv:2203.15930* (2022).

[33] pmcgoohan. [n. d.]. zeromev-geth. https://github.com/zeromev/zeromev-geth/blob/master/README.md. Accessed: 2023-01-12.

[34] Kaihua Qin, Liyi Zhou, and Arthur Gervais. 2022. Quantifying Blockchain Extractable Value: How dark is the forest?. In *SP*. IEEE, San Francisco, CA, USA, 198–214. https://doi.org/10.1109/SP46214.2022.9833734

[35] Lum Ramabaja. 2019. The bloom clock. *arXiv preprint arXiv:1905.13064* (2019).

[36] Peiyao Sheng, Gerui Wang, Kartik Nayak, Sreeram Kannan, and Pramod Viswanath. 2021. BFT protocol forensics. In *CCS*. 1722–1743.

[37] ShutterNetwork. [n. d.]. Global Ping Statistics. https://github.com/shutter-network/shutter. Accessed: 2023-01-12.

[38] Quinten Stokkink, Can Umut Ileri, Dick Epema, and Johan Pouwelse. 2023. Web3 Sybil avoidance using network latency. *Computer Networks* 227 (2023), 109701.

[39] Robbert Van Renesse, Dan Dumitriu, Valient Gough, and Chris Thomas. 2008. Efficient reconciliation and flow control for anti-entropy protocols. In *LADIS*. 1–7.

[40] Anton Wahrstätter, Liyi Zhou, Kaihua Qin, Davor Svetinovic, and Arthur Gervais. 2023. Time to Bribe: Measuring Block Construction Market. arXiv:2305.16468 [cs.NI]

[41] Taotao Wang, Chonghe Zhao, Qing Yang, Shengli Zhang, and Soung Chang Liew. 2021. Ethna: Analyzing the underlying peer-to-peer network of ethereum blockchain. *IEEE Transactions on Network Science and Engineering* 8, 3 (2021), 2131–2146.

[42] WonderNetwork. [n. d.]. Global Ping Statistics. https://wondernetwork.com/pings. Accessed: 2023-01-12.

[43] Jiahua Xu, Krzysztof Paruch, Simon Cousaert, and Yebo Feng. 2023. Sok: Decentralized exchanges (dex) with automated market maker (amm) protocols. *Comput. Surveys* 55, 11 (2023), 1–50.

[44] David Yakira et al. 2021. Helix: A Fair Blockchain Consensus Protocol Resistant to Ordering Manipulation. *IEEE TNSM* 18, 2 (2021), 1584–1597. https://doi.org/10.1109/TNSM.2021.3052038

[45] Sen Yang, Fan Zhang, Ken Huang, Xi Chen, Youwei Yang, and Feng Zhu. 2022. SoK: MEV Countermeasures: Theory and Practice. *arXiv:2212.05111* (2022).

[46] Jiangshan Yu, David Kozhaya, Jeremie Decouchant, and Paulo Esteves-Verissimo. 2019. Repucoin: Your reputation is your power. *IEEE Trans. Comput.* 68, 8 (2019), 1225–1237.

[47] Pouriya Zarbafian and Vincent Gramoli. 2023. Lyra: Fast and Scalable Resilience to Reordering Attacks in Blockchains. (2023).

[48] Haoqian Zhang, Louis-Henri Merino, Vero Estrada-Galinanes, and Bryan Ford. 2022. Flash freezing flash boys: Countering blockchain front-running. In *2022 IEEE 42nd International Conference on Distributed Computing Systems Workshops (ICDCSW)*. IEEE, 90–95.

[49] Yunhao Zhang, Srinath Setty, Qi Chen, Lidong Zhou, and Lorenzo Alvisi. 2020. Byzantine ordered consensus without byzantine oligarchy. In *Proceedings of the 14th USENIX Conference on Operating Systems Design and Implementation*. 633–649.

[50] Liyi Zhou, Kaihua Qin, and Arthur Gervais. 2021. A2mm: Mitigating frontrunning, transaction reordering and consensus instability in decentralized exchanges. *arXiv preprint arXiv:2106.07371v2* (2021).

[51] Liyi Zhou, Xihan Xiong, Jens Ernstberger, Stefanos Chaliasos, Zhipeng Wang, Ye Wang, Kaihua Qin, Roger Wattenhofer, Dawn Song, and Arthur Gervais. 2023. Sok: Decentralized finance (defi) attacks. In *2023 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2444–2461.