Age-aware Fairness in Blockchain Transaction Ordering for Reducing Tail Latency

Yaakov Sokolik, Mohammad Nassar and Ori Rottenstriech

***Abstract*—In blockchain networks, transaction latency is crucial for determining the quality of service (QoS). The latency of a transaction is measured as the time between its issuance and its inclusion in a block in the chain. A block proposer often prioritizes transactions with higher fees or transactions from accounts it is associated with, to minimize their latencies. To maintain fairness among transactions, a block proposer is expected to select the included transactions randomly. The random selection might cause some transactions to experience high latency following the variance in the time a transaction waits until it is selected. We suggest an alternative, age-aware approach towards fairness so that transaction priority is increased upon observing a large waiting time. We explain that a challenge with this approach is that the age of a transaction is not absolute due to transaction propagation. Moreover, a node might present its transactions as older to obtain priority. We describe a new technique to enforce a fair block selection while prioritizing transactions that observed high latency. The technique is based on various declaration schemes in which a node declares its pending transactions, providing the ability to validate transaction age. By evaluating the solutions on Ethereum data and synthetic data of various scenarios, we demonstrate the advantages of the approach under realistic conditions and understand its potential impact to maintain fairness and reduce tail latency.**

I. Introduction

In a blockchain system, it is critical to reach an agreement on block content between all network nodes. This is typically achieved by applying a consensus mechanism such that a single node proposes a new block that goes through some approval process among other nodes. Upon achieving consensus on the proposed block, it is added to the blockchain.

A proposer can be selected using different techniques. The most common one is Proof-of-Work (PoW) [2], which is implemented in the Bitcoin [3] and Ethereum [4] 1.0 networks, among others. In a PoW scheme, a node needs to solve a cryptographic puzzle for gaining the right to propose a new block. The puzzle can be finding some nonce such that the hash value of the block and the nonce is below a defined threshold. Another widely used technique is Proof-of-Stake (PoS) [5], [6], by which the proposer is selected randomly according to some token distribution over the network nodes.

Each block is constructed from issued transactions (*txs*), pending to be written to the blockchain. The amount of transactions contained in a block is limited. In case the proposer has a pending transaction pool that is larger than the maximal block size, the proposer needs to select which transactions will be included in the proposed block.

Yaakov Sokolik, Mohammad Nassar and Ori Rottenstreich are with the Technion - Israel Institute of Technology, Haifa, Israel (emails: yaakovsoki@gmail.com, mohammadnassar22@gmail.com, or@technion.ac.il).

There are blockchain networks (e.g., Bitcoin and Ethereum), by which a transaction is associated with a fee that its issuer agrees to pay. The proposer gains the fees of the transactions included in the block upon approval of a proposed block. Therefore, the proposer typically selects those transactions which maximize its profit from the proposed block. The mechanism of fees implies that some transactions have higher chances to be included in the block than others.

The autonomy granted to block builders in determining transaction order gives rise to the concept of *miner extractable value (MEV)* or *blockchain extractable value (BEV)*. MEV, as described in the research by Daian et al. [7], refers to the potential value obtained by manipulating transaction sequencing during block creation. Some decentralized exchanges employ a dynamic trading ratio that adjusts with each transaction. In decentralized finance (DeFi) systems, malicious actors such as miners or traders can exploit the system's transparency to profit from front-running and back-running transactions, extracting value from these transactions for personal gain. Numerous studies, including those mentioned in [8], [9], [10] have examined this phenomenon.

We study fairness in decentralized blockchain networks. Such a network consists of nodes that serve multiple clients (e.g. using cryptographic wallets). Transactions of all such users are recorded on the same blockchain. Often, for security reasons, each client is connected to multiple nodes. A key factor in a client's quality of service (QoS) is its transaction latency — the time between a transaction's issuance and its inclusion in the blockchain. Transactions are not associated with a particular fee.

A node is motivated to minimize the transaction latency of some clients it knows and so it inserts as many of its own transactions to a proposed block. Fairness in transaction selection can be interpreted as providing each pending transaction with identical chances to be selected in a new block. The concept of transaction ordering fairness was introduced in Helix [11], where it was proposed to achieve fairness in block selection by selecting block transactions *randomly* among the proposer pending pool. This ensures that each pending transaction had the same chances of being selected. Other types of fairness in blockchain have been suggested [12]. For instance, fairness among nodes [13] such that in every round each node proposes a similar number of transactions and order-fairness among transactions [14] which assigns a pair of transactions an order based on the identity of the transaction received earlier by majority of the nodes.

Beyond the mean value of the latency, a recent work by Shang, Ilk and Fan [15] indicated the high importance in reducing the latency higher quantiles (i.e., tail of the distri-

bution). It is claimed that long waiting times often cause low service reuse and customer satisfaction [16], [17]. Moreover, firms communicate waiting time guarantees to customers as "your wait is 20 min" which are perceived by customers as "I (most likely) need to wait no more than 20 min", rather than "my average waiting time is 20 min" [18]. Beyond the context of fairness several other recent works suggested methods to reduce tail latency [19], [20], [21].

In fact, the randomness in transaction selection, even when followed honestly by all nodes, can cause some transactions to wait a long time before their inclusion in the blockchain. Our perspective towards bounding the transaction latency is that an older transaction would have a higher priority to be selected in a block over a recently issued transaction. Since such priority is determined by the elapsed time regardless of the identity of the particular transaction, fairness is still maintained.

A preliminary version of this article appeared in IEEE/ACM International Symposium on Quality of Service (IWQoS) '20, June 2020 [1].

Contributions: We make the following contributions:

- We present the notion of *age-aware fairness* in block selection for reducing the tail latency of transactions.
- We study approaches for deriving age-aware fairness in various characteristics of transaction propagation.
- We detail declarations schemes on the content of transaction pools that allow demonstrating high age of transactions.
- We conduct experiments based on Ethereum and synthetic data to evaluate the performance of the approach.

II. Age-aware Fairness in Block Selection

A. Motivation and Definition for Age-aware Fairness

We refer to nodes that play two main key roles. A primary node is responsible for selecting a block from a pool of pending transactions. Once a primary node selects a block, a committee of members is responsible for validating it. The committee is composed of a subset of nodes chosen from the network. Each member independently evaluates the block, and the validations are aggregated to determine the block's overall evaluation. The notion of block validation by committee members for enforcing fairness is not new. As fairness refers to the selection of transactions by a node among its pool of transactions, it implies an inherent challenge as the pools of pending transactions at different nodes are not identical due to network latency. Accordingly, several previous works suggest various methods for block validation by such committees [11], [22], [23]. Often, the number of committee members is small with respect to the number of network nodes.

Average transaction latency was analyzed as a key metric to measure the scalability of the blockchain [24]. However, the skewness of the latency distribution significantly affects the user experience when providing guarantees on confirmation time. In this work, we define latency as the duration between the issuance of a transaction and its inclusion in the blockchain. We address the issue of reducing tail latency (e.g., q-99) which significantly affects blockchain QoS. The q-k latency is defined as the latency time that at least k percent of the transactions have lower latency. In particular, we are interested in reducing high values of k as q-99 or q-99.9, meaning that 99% or 99.9% of transactions have lower latency. We propose a method that reduces the tail latency by prioritizing senior transactions (those already observing high latency), in a way that keeps fairness.

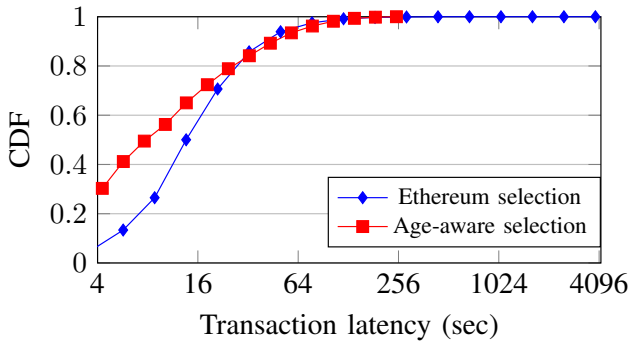


Fig. 1. Reducing tail latency of Ethereum transactions. The blue line shows the cumulative distribution function (CDF) of measured latency in Ethereum. The red line shows a CDF that was produced by selecting transactions in an age-aware manner. For instance, the 99$^{\text{th}}$ percentile reduces from 380 seconds to 203 seconds and the 99.9$^{\text{th}}$ percentile from 2197 seconds to 236 seconds.

Besides the inherent priority to senior transactions, the method maintains fairness among all other transactions that have not yet observed high latency. This is done through providing them identical chances to be selected randomly for being included in a block.

We collected data from an Ethereum node about its pending transactions over one hour on December 17, 2019. Fig. 1 shows the latency we measured for transactions that were in the pending transactions pool and were included in some block during our recording (Ethereum blocks 9120140 to 9120366). We simulated different blocks in which transactions are selected in an age-based method of choosing 20% of the block transactions by their age while the rest of the block was selected randomly. Transactions that were not seen by our node during the recording were not affected and were left inside the block. We observed that the q-99 latency was reduced from 380 seconds to 203 seconds. For a higher percentile, the improvement is more significant as the q-99.9 latency decreases from 2197 seconds to 236 seconds. It is important to note the inherent differences with regards to Ethereum as it does not apply a random selection policy. In Ethereum, each transaction has a gas price which sets the reward that the proposer gets for including this transaction. The proposer tries to maximize its reward from the block and does not care about the latency of transactions. This sometimes causes transactions with low gas prices to observe high latency.

B. Challenges in Maintaining Age-aware Fairness

Age-aware Fairness can be challenging due to several key points:

(i) **Balancing fairness and latency**: Prioritizing senior transactions breaks the randomness in transaction selection which is crucial to achieving fairness. This granted priority should maintain fairness among non-senior transactions.

(ii) **Transaction age differs between nodes**: Following inherent heterogeneity in the propagation of transactions, the pending transaction pools of every pair of nodes might be different. They differ not only in the set of transactions that they are aware of but also in the transaction age. Each node becomes aware of a transaction at a different time so nodes can see the same transaction as having different ages.

(iii) **Avoid forging transaction age**: We need to make sure a node cannot forge its transactions' age. Namely, a committee member should be able to verify the transaction age.

C. Model and Terminology

Our blockchain network consists of a decentralized peer-to-peer (P2P) architecture with multiple interconnected nodes. Transactions can be issued by any user or node within the network and are disseminated through a gossip-based communication mechanism. To append new blocks to the blockchain, a block selection round takes place in which nodes within the network construct a new block. During this process, certain nodes are selected as consensus nodes, either randomly or based on specific criteria. Among these consensus nodes, a single node is designated as the block proposer responsible for suggesting a new block from its pool of pending transactions. The proposed block is then disseminated throughout the network, and other nodes, known as committee members, assess the fairness aspects of the proposed block and determine whether to approve it or not [6].

Table I summarizes commonly used notations. We make the following model assumptions.

Node identities: The identity of each node participating in the protocol is known to all other nodes.

Selection of the block proposer and committee members: Every block, a hash value is computed for each node. Besides the node id, the input to the function also includes the hash of the last block so the order is hard to be predicted in advance. Nodes with minimal hash values are selected as the single leader and N_C committee members for block validation, where N_C is the fixed size of the committee.

Node's behavior: There are two distinct roles a node can play: an honest node or a malicious node. An honest node adheres strictly to the protocol by proposing blocks and validating the proposed blocks in accordance with the established rules. On the other hand, a malicious node exhibits different behavior. It may prioritize certain transactions in its proposed blocks, deviating from the protocol's guidelines. Additionally, a malicious node may validate proposed blocks in a manner that diverges from the prescribed protocol specifications.

Proposer' behavior: The probability that a proposer acts dishonestly is larger than the probability that a block proposed by an honest proposer does not pass the validation check.

Transaction propagation: Transactions are propagated within the network through gossip-based communication. The transaction propagation is not manipulated by nodes.

Gas price: Each transaction incurs the same gas price, gas price is the per unit of work done.

Gas limit: Gas limit is proportional to the amount of work estimated for a validator to do on a particular transaction.

Time: Time is simply measured in the granularity of the blocks such that transactions are viewed based on the number of blocks added to the blockchain since the declaration message. Accordingly, there is no need for a global highly-accurate clock among network nodes.

D. Implementations Concepts

We refer for simplicity to a block of a bounded size in term of its transaction number b. A discussion of simple generalization of the model to block with bounded amount of expected computation can be found in Section IX.

Transactions that wait a long time (higher than a threshold) in the pending pool of the proposer are referred to as *senior transactions*. The definition is based on the *local* age of a transaction. Thus, a transaction can be defined as *senior* by one node but not by some other node.

The proposed block consists of two parts:

1) The first section of the block, lower transactions, is an age-based section that includes *senior transactions* of size s, where s represents the size of the senior transactions within the block. The proposer selects the s oldest *senior transactions* based on their age.
2) In the second part, transactions with a size of $b - s$ are randomly selected from the pool. Here, b represents the total size of the block. In each round, a hash function is adjusted using a unique seed and applied to all the pending transactions. Subsequently, the transactions are sorted based on their hash values, and the ones with the lowest values and a size of $b - s$ are chosen to be included in the block.

Age-aware Block Validation: Each committee member runs the validation process of the proposed block which is composed of validating the seniority of transactions declared as *senior transactions* and validating the selection at random of the non-senior transactions. If the proposed block passes both parts, the committee member accepts it and votes accordingly.

Our method enforces the seniority of transactions included in the age-based part, implying that the proposer should include only *senior transactions*. Otherwise, the proposed block should fail the validation. To maintain fairness among transaction selection, the parameter s should be some small fraction of b. The way nodes are selected to be the proposer of a block implies identical chances for each node to be selected. Such a node can include senior transactions in the block in addition to the transactions selected randomly.

For non-senior transactions, a validator compares the proposed transactions with its transaction pool. To pass the validation, the local block, which is a set of transactions constructed locally from the validator's transaction pool, and the random part must demonstrate a significant level of similarity. The specific similarity threshold is determined based on the similarity between transaction pools. Similar approaches have been utilized in prior studies such as [11], [23].

The age-aware protocol distinguishes itself from Ethereum 1.0 in several ways. Firstly, to generate new blocks it employs a leader and validation routine by committee members. Roles are assigned randomly based on hash values of previous

b	$\triangleq$	Block size.
s	$\triangleq$	Maximum size of *senior transactions* included in the block because of their age.
T_{old}	$\triangleq$	Threshold on transaction age.
$T_{propagation}$	$\triangleq$	Maximal transaction propagation time if it exists.
$T_{declaration}$	$\triangleq$	Time interval between two declaration phases.
T^x_{issue}	$\triangleq$	Random variable representing issuance time of transaction x.
$T^{i,x}_{prop}$	$\triangleq$	Random variable representing time since issuance of transaction x until node i learned about it.
$t^{i,x}_{learn}$	$\triangleq$	Time that node i learned about transaction x.
$t^{i,x}_{prop}$	$\triangleq$	Time since transaction issuance until node i learned about transaction x.
G^i_x	$\triangleq$	Random variable representing age of transaction x as seen by node i.
g^i_x	$\triangleq$	Age of transaction x as seen by node i.

TABLE I
TABLE OF NOTATION

blocks. On the other hand, Ethereum 1.0 relies on PoW (Proof of Work) for the selection of the leader who can select any transactions for the block, while often aiming to maximize its gain such that the particular transaction selection is not validated. Secondly, Ethereum utilizes a distinct fee mechanism where fees can impact the block creation process. In contrast, our protocol prioritizes senior transactions, emphasizing their significance during the block creation process.

III. Block Proposal and Validation

Since fairness refers to block selection among pending transactions, enforcing fairness depends on the similarity of the pending transaction pools among the proposer and the committee members. When the different pending transaction pools observe higher similarity or when one node has a better knowledge of other nodes' pool, it is easier to test the validity of block selection. In the following, we explain how this can be done for different characteristics of transaction propagation.

A. Bounded-Time Transaction Propagation Case

For some networks, their characteristics imply an upper bound on transaction propagation time among all network nodes, namely from the time a transaction is issued by a node until it is received by any other node. Denote such a maximal transaction propagation time as $T_{propagation}$. For a transaction x in pending transactions pool of some node i with age g^i_x, the existence of such a bound implies that for any node j that also has x in its pending transactions pool its age is $g^j_x \in [g^i_x - T_{propagation}, g^i_x + T_{propagation}]$.

Transactions with a *local* age larger than T_{old} are defined as *senior transactions*. In order that *senior transactions* will appear in all nodes pending pools T_{old} should be greater than $T_{propagation}$.

In the *senior transactions* validation, the committee member validates that all the *senior transactions* in the proposed block are also in its pending transactions pool and their *local* age is at least $T_{old} - T_{propagation}$. If the proposer included those transactions honestly (namely their age as seen by the proposer indeed satisfies the condition), these transactions should also be in the committee member's pending pool due to the bound on the transaction propagation. The committee member m also finds the *senior transaction* that is included in the proposed block with the smallest *local* age. Let this transaction be denoted as x. The committee member checks in its pending transactions pool whether some transaction y that is not included in the proposed block satisfies $g^m_y - T_{propagation} > g^m_x + T_{propagation}$, which means that the proposer has for sure learned about y before x and dishonestly omitted y, and then the block is denied. The pseudocode of the approach appears in Algorithm 1. The $local\_age(x)$ function which appears in Algorithm 1 returns the *local* age of transaction x or 0 if it has not learned about it.

This assumption on the propagation time restricts the ability of a node to manipulate the age of a transaction. Assuming a dishonest proposer forges its transaction age, it can add to each transaction age at most $T_{propagation}$. If this transaction is selected in the random part, the change of the age would not impact its latency. Alternatively, if the transaction was included in the age-based part its real age must be larger than $T_{old} - T_{propagation}$. In case $T_{propagation}$ is much smaller than T_{old}, it does not have a large impact on the latency.

B. Transactions Propagation Distribution is Known

Even in less restricted networks that do not have the upper bound on transaction propagation time among all network nodes, we can make use of transaction propagation distribution. To illustrate this we assume that the time that each node learns about some transaction is distributed exponentially [25].

Given a transaction, its propagation distribution depends on its issuance time. To calculate the propagation probability of a transaction, we need its issuance time. Let denote transaction issuance time as t_{issue}. Since the propagation distribution of a transaction is related to its issuance, let denote the propagation cumulative distribution function (CDF) as $CDF(t_{issue})$.

A transaction whose age is larger than T_{old} is defined as *senior transaction* for the age-based part of the block. The proposer p includes for each *senior transaction* x in the age-based part, the time that it claims it learned about the transaction $t^{p,x}_{learn}$. The claimed learning time enables committee members to calculate the probability that x is indeed *senior*. A potential generalization we do not elaborate on here is calculating this probability without proposer's claims on *senior transactions* learning times. Such calculation can be based on T_{old} and not on a claimed age.

The committee member m validates the age-based part by computing for each *senior transaction* x the probability that its age as seen by the proposer p is larger or equal than the age the proposer claims. The proposed block passes the age-based part validation if the product of the probabilities for all *senior transaction* equals at least some threshold v.

$$\prod_{\text{senior transaction } l} \Pr(G^p_l \geq g^p_l \mid t^{m,l}_{learn}) \geq v.$$

Fig. 2 illustrates a timeline of the transaction propagation since its issuance among the proposer (above the arrow) and the committee member (below the arrow).

$\Pr\left(G^p_x \geq g^p_x \mid t^{m,x}_{learn}\right)$ is the probability that for transaction x its age as seen by the proposer p is larger or equal than the

Algorithm 1 Senior validation in bounded transaction propagation

```
1: procedure VALIDATEBOUNDED(pool, block)
2:     senior ← senior transactions of block (as of proposer)
3:     if senior ⊈ pool then
4:         return false
5:     oldPool ← max{local_age(x)|x ∈ pool}
6:     youngBlock ← min{local_age(x)|x ∈ senior}
7:     if youngBlock < T_old − T_propagation then
8:         return false
9:     if oldPool − T_propagation > youngBlock + T_propagation then
10:        return false
11:    return true
```

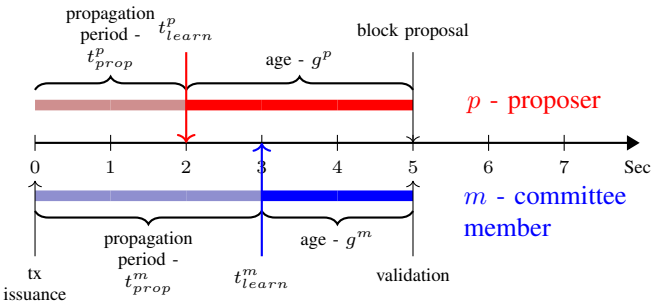


Fig. 2. Illustration of transaction propagation timeline since its issuance.

age the proposer claims, given the time the committee member m learned about x. The committee member needs to compute this probability for each *senior transaction* x which depends on the issuance time of x. In order to estimate it from the time the committee member learned about the transaction, we compute:

$$\Pr(G^p_x \geq g^p_x \mid t^{m,x}_{learn}) =$$
$$\int_{t_{issue}} \Big(\Pr(G^p_x \geq g^p_x \mid t^{m,x}_{learn}, t_{issue}) \cdot f_{T^x_{issue}|t^{m,x}_{learn}}(t_{issue}) \Big) dt_{issue} =$$
$$\int_{t_{issue}} \Big(\Pr(T^{p,x}_{prop} \leq t^{p,x}_{learn} - t_{issue} \mid t_{issue}) \cdot f_{t^{m,x}_{learn} - T^{m,x}_{prop}|t^{m,x}_{learn}}(t_{issue}) \Big) dt_{issue}$$

The integral runs over values of t_{issue}. Thus, all the parameters of the propagation distribution $T^{p,x}_{prop}$ and $T^{m,x}_{prop}$ are known. The first component inside the last integral $\Pr(T^{p,x}_{prop} \leq t^{p,x}_{learn} - t_{issue} \mid t_{issue})$ is the probability that propagation period of transaction x to the proposer is lower or equal than the difference between the time the proposer claimed of learning about x and integration variable t_{issue}. It is calculated directly from $CDF(t_{issue})$. The second component inside the integral $f_{t^{m,x}_{learn} - T^{m,x}_{prop}|t^{m,x}_{learn}}(t_{issue})$ is the PDF (probability density function) value of transformed $T^{m,x}_{prop}$ (multiplied by -1 and adding a constant) which is calculated by the derivation of the transformed $CDF(t_{issue})$.

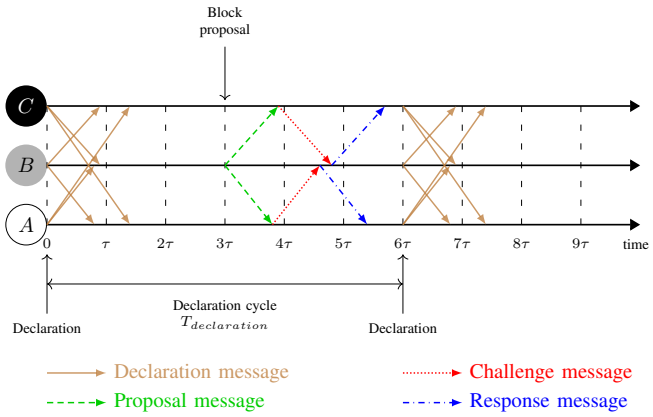


Fig. 3. Illustration of a declaration cycle in a blockchain network of 3 nodes. Each node broadcasts a declaration about its pending transactions to all other nodes which learn about the declaration at different times. Time unit is denoted by τ. At the time 3τ node B proposes a block. Nodes A, C serve as committee members and they validate the proposed block. Validation of *senior transactions* age is based on past declarations of the proposer. In Merkle tree and Accumulator schemes, the committee members send a challenge to the proposer and validate the response.

C. General Propagation Characteristics

We would like the committee member to be able to validate the transaction's age also in a more general case where the transaction propagation distribution is not well known. We suggest using a declarations mechanism to enable this validation. We make a distinction between transactions to declarations for which we still have an assumption on the bounded delay.

In this mechanism, a declaration phase occurs periodically every time of $T_{declaration}$. At a declaration phase, each node sends a declaration of its pending transactions to all other nodes as described later in section IV. Once a transaction was included in a declaration, there is no need to include it again in a later declaration. While a node receives such a declaration message, it stores the declaration along with the reception time (denoted as $timestamp(declaration)$) and the sending node. $T_{declaration}$ should be a time by which necessarily all nodes got a declaration message from the declarer node. The age threshold for defining a transaction as a *senior transaction* is T_{old}. The block proposer includes in the age-based part of the block only transactions that can be validated as *seniors*. Since the block proposal can be immediately after the declaration phase, it implies that only transactions that passed at least T_{old} since their declaration can be selected as the age-based part of the block.

Fig. 3 illustrates a declaration phase in which each node sends a declaration on its pending transactions to all other nodes. In the example, time slot is denoted by τ and $T_{declaration}$ is 6τ. In section IV, we detail several declaration schemes and provide pseudocode for the validation process of each.

IV. Declaration Schemes

In this section, we refer to a general propagation delay. We explain how declaration on pending transactions pool can serve

as a basis for age-aware block validation and in particular the *senior transactions* in the block. We discuss three schemes based on complete lists, Merkle trees, and accumulators. We explain that the last two of them are communication efficient but on the other hand require a challenge-response process where a committee member questions the block proposer. Intuitively, when a block proposer proposes a block, it includes senior transactions in the block and can be challenged by a committee member with regard to the seniority of such transactions. The block proposer then serves as a prover (and the committee member as a verifier) and can refer to a past declaration it has at least T_{old} time earlier in which the examined transaction served as part of the input to the computation of the declaration.

A. Complete List Scheme

In this scheme, a declaration contains all undeclared pending transactions as a complete list. Those declarations might be very large messages causing a communication overhead, yet it can serve as a baseline.

In the age-based part validation, the committee member validates that the *senior transactions* in the proposed block were included in some declaration. The validation is done by checking that every transaction in the age-based part that its *local* age is smaller than T_{old} was included in the list of transactions that the proposer sent in declarations where their age is at least T_{old}. A pseudocode is shown in Algorithm 2.

B. Merkle Tree Scheme

In this scheme, a declaration is the Merkle tree [26] root of the pending transactions. The Merkle tree is a full binary tree that stores the data as leaves and every inner node is the hash of its children concatenation. The node which sends the declaration constructs this tree and saves it in memory.

The pseudocode of the age-based part validation is shown in Algorithm 3. In case some transaction is included as a *senior transaction* and the committee member does not have it in its pending transactions pool or has it with an age smaller than T_{old}, the committee member challenges the proposer to prove the transaction age. The proof is a Merkle proof to some previous declaration with an age of at least T_{old}. The Merkle proof of a transaction is all siblings of the nodes on the path from the transaction leaf to the Merkle root as illustrated in Fig. 4. For instance, the Merkle proof of tx_2 is $\{h_3, h_{0-1}, h_{4-7}\}$. If the proposer managed to provide appropriate proofs, the block passes the age-based part validation.

The declarer can delete from its memory a Merkle tree after all the transactions its stores were included in the blockchain. Since the proposer selected the oldest *senior transactions* to the age-based part, the committee members can delete all declarations of the proposer that are older than the declaration used to prove a transaction age.

C. Accumulator Scheme

Another bandwidth and memory efficient scheme is declaring the accumulated [27] value of all pending transactions.

Algorithm 2 Senior validation with complete list declaration

```
1: procedure VALIDATELIST(block, proposer)
2:     senior ← senior transactions of block (as of proposer)
3:     declared ← transactions included in declarations
                  of proposer with timestamp ≥ T_old
4:     for each tx ∈ senior and local_age(tx) < T_old do
5:         if tx ∉ declared then
6:             return false
7:     return true
```

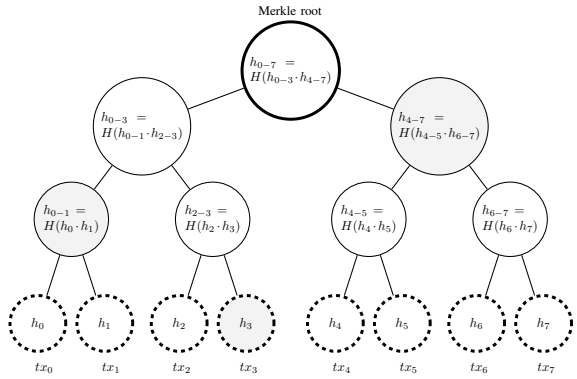


Fig. 4. Illustration of a Merkle tree of 8 txs. The Merkle proof of tx_2 is nodes h_3, h_{0-1} and h_{4-7} filled with grey.

Algorithm 3 Senior validation with Merkle tree declaration

```
1: procedure VALIDATEMERKLETREE(block, proposer)
2:     senior ← senior transactions of block (as of proposer)
3:     merkleRoots ← all Merkle roots sent by proposer
4:     to_prove ← []
5:     for each tx ∈ senior and local_age(tx) < T_old do
6:         to_prove ← to_prove + tx
7:     if to_prove == [] then
8:         return true
9:     proofs ← CHALLENGEMERKLEPROOF(to_prove)
10:    for each tx ∈ to_prove do
11:        tx_proof ← proof for tx in proofs
12:        mr' ← CALCMERKLEROOT(tx_proof, tx)
13:        if mr' ∉ merkleRoots then
14:            return false
15:        for each mr ∈ merkleRoots do
16:            if mr' == mr and timestamp(mr) < T_old
   then
17:                return false
18:    return true
```

The accumulator data structure can support set membership proving. It is based on a *quasi-commutative* function

$$f : X \times Y \to X$$

which $\forall x \in X$ and $\forall y_1, y_2 \in Y$,

$$f(f(x, y_1), y_2) = f(f(x, y_2), y_1)$$

This property enables accumulating a hash of a set of elements $\{y_1, y_2, \ldots, y_m\}$:

$$z = f(f(f(...f(f(f(x, y_1), y_2), y_3, ..., y_{m-2}), y_{m-1}), y_m)$$

Algorithm 4 Hashing transactions to prime numbers [28]

```
procedure HASHTOPRIME(x)
    y ← H(x)
    while y is even or PRIMALITYTEST (y) == False do
        y ← H(y)
    return y
```

Algorithm 5 Senior validation with accumulator declaration

```
procedure VALIDATEACCUMULATOR(block, proposer)
    senior ← senior transactions of block (as of proposer)
    accumulators ← all accumulators sent by proposer
    to_prove ← []
    for each tx ∈ senior and local_age(tx) < T_old do
        to_prove ← to_prove + tx
    if to_prove == [] then
        return true
    proofs ← CHALLENGEACCPROOF(to_prove)
    for each tx ∈ to_prove do
        tx_proof ← proof for tx in proofs
        z' ← h(tx_proof, HASHTOPRIME(tx))
        if z' ∉ accumulators then
            return false
        for each z ∈ accumulators do
            if z' == z and timestamp(z) < T_old then
                return false
    return true
```

such that the accumulated hash of all permutations of y_i gives the same value z. The partial accumulated value except y_j is

$$z_j = f(...f(f(...f(x, y_1), ...y_{j-1}), y_{j+1})..., y_m)$$

and since $f(z_j, y_j) = z$ the value z_j is the proof of existence of y_j.

A common accumulator is the collision free RSA based [29], which accumulates prime numbers using the quasi-commutative modular exponentiation function $e_n(x, y) = x^y \mod n$. The modulus value n is chosen as a product of two suitable large primes as in the RSA [30], and the initial value x is some random number smaller than n.

In order to accumulate transactions, we use the transaction hash value (*txHash*) which identifies the transaction. The *txHash* is mapped to a prime number with the same number of bits (which is typically 256 bits) by the HASHTOPRIME function [28] that Algorithm 4 shows its pseudocode. Since the domain of HASHTOPRIME is larger than its codomain, there might be collisions meaning two different *txHash*s mapped to the same prime. However, in our scheme it does not cause a problem since assuming the mapping is random and the number of primes smaller than 2^λ is $O(\frac{2^\lambda}{\log 2^\lambda}) = O(\frac{2^\lambda}{\lambda})$. λ is the number of bits in the prime number which is typically very big (256 bits), hence, the probability that two such transactions are in the pending pool is negligible.

The HASHTOPRIME algorithm makes use of a collision resistant hash function H. The primality test of y can be done by the Baillie-PSW primality test [31]. A number that passes this test is very probably prime. The expected number of iterations of calling H assuming HASHTOPRIME uses a random hash function H is $O(\lambda)$.

While constructing the declaration, the proposer saves the proof for each transaction. Every declaration makes use of a different RSA modulus n in order to obtain security. The declaration includes the accumulator value and the modulus n.

Validation of the age-based part is done similarly to the Merkle tree scheme and its pseudocode is shown in Algorithm 5. The committee member challenges the proposer in case some transaction is included as *senior* and its pending pool does not contain it or its local age is smaller than T_{old}. The proposer responds with the membership proof of the *senior transaction* in some previous accumulator (*tx_proof*). The committee member calculates $h(tx\_proof, \text{HASHTOPRIME}(txHash))$ and verifies that it is equal to some previous declaration that is older than T_{old}.

Transaction Hash (256 bit)
0xf5b4c966692b6c7ba4d5585140c4e83d5d9a54ae94a126329f277a3abca2e889
0x3bce38326633a8fc1de1f75add1a0b38d21c6d320ce36281dbe7e49a77f3da39
0x250688d7397cb0ed9547de1e1bb7ec14f20a035cd9505f319285829777d677ea
0xbe19f11c1a9a65dc84812d3e027baa31ca03b4e2e47d8ffbcd7acdc880af246c
0xeda481c1dfd0f69488d007b6daedd28f6000b142a4c74e0d3f34bd4f1004ca22
0x1fd7e61ec4587cd8a8385c14d5e568df2450f1b8a851ba70a5e4c9efe19e542f
0x5335ecbf65280ee652ebabd4111f2d9c9fbcfbfcc58b7411172d9c995b48c535
0xf9506b280cad8213c5082550aff486e7a7d34152f33554e30d99620019a8deb4

TABLE II
BLOCK EXAMPLE: TRANSACTIONS HASH OF ETHEREUM BLOCK #871477 (OF 8 TXS)

V. EXAMPLE AND SCHEME COMPARISON

A. Illustrative Example for Declarations

We illustrate the declaration schemes through a simple real Ethereum block (#871477 of January 19, 2016), containing only 8 transactions. E.g., we assume that the pending transactions pool and produced declaration contain only those 8 transactions. We illustrate the various declarations over such a pool. The number of transactions in the declaration is denoted by n_d. Each transaction is represented by its hash value of 256 bit (w denotes this length). The values of transaction hash values (*txhash*) are listed in Table II. We chose this block for our example because of its unique number of transactions.

Complete list scheme - The complete list declaration is very simple, and there is no need for further proof. The declaration is the whole transaction table. The declaration size in our example is of $n_d \cdot w = 8 \cdot 256 = 2048$ bits.

Merkle tree scheme - The Merkle tree declaration is the Merkle root. In the example, we built the Merkle tree of the transactions as in Fig. 5 where we used SHA256 as the hash function. The Merkle root value (also of $w = 256$ bit) is $MR =$0xc6a9d81cd4fc1fab04ebae22388d74e38c614a37bf701f1bfa64e1047bf1e813. The declaration is simply the Merkle root, implying declaration size of 256 bit. The proof of transaction inclusion e.g tx_2 is the values of

tx_3=0xbe19f11c1a9a65dc84812d3e027baa31ca03b4e2e47d8ffbcd7acdc880af246c

h_{0-1}=0x762bbcb0389a1793ccb39eefb11e8966913da8fd24b7dfad078f9861b55eaf9a

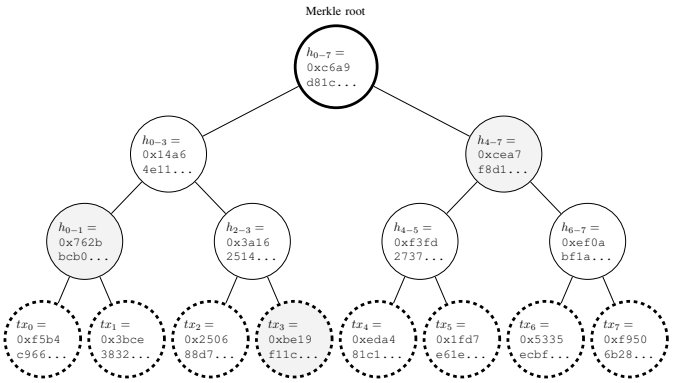


Fig. 5. Merkle root computation for the block of 8 txs from Table II. Filled with grey are hash values used for showing the inclusion of tx_2 in the block.

HASHTOPRIME(txHash) (256 bit)
0x2b3b5312255e85590929506a702691954bee4fa2c0ccdd2abc3df8b2a83393c9
0x64c15009c6dc4430ae62b7b8d6abc162efceb03f3bffa5a51074b415e8d1d561
0x422134d3b572a67cd550895f347269c8415fe192771b9a0a7f39163904484b9b
0x35f6eaf93696fb2f5089c78f4c7831323335e079580ed54eb41a0a282580642f
0xaf5d065b8948dce72d5b538d3775ae24cc58038b4a53bd62fa956a8310921f77
0xbfaa8c4e5231c4bdf2b98564360947290c7f1fba3b3ea386cd71988eaee4a5b5
0x484d9321170d064fde2d5576bae3b7a3b954051c88e41e15676d2a00f8cfed7f
0xdfe80ed79ed2a104ac0fafa455f41059316229c2723d9ddd50015b3865b05d65

TABLE III
PRIME NUMBERS REPRESENTING THE TRANSACTIONS OF THE BLOCK OF 8 TXS FROM TABLE II THROUGH CALLING HASHTOPRIME (ALGORITHM 4)

h_{4-7}=0xcea7f8d17ad5a6a0cb67242f5a34dcad6060645cdd0b992ecdd1f6c4146e921d . Validating the inclusion of tx_2 is done by comparing MR to

$$\begin{aligned}&\mathrm{H}(\mathrm{H}(h_{0-1}\cdot\mathrm{H}(tx_2\cdot tx_3))\cdot h_{4-7}) =\\ &\mathrm{H}\Big(\mathrm{H}\big(\texttt{0x762bbcb0...}\\ &\quad\cdot\mathrm{H}(\texttt{0x250688d7...}\cdot\texttt{0xbe19f11c...})\big)\\ &\quad\cdot\texttt{0xcea7f8d1...}\Big).\end{aligned}$$

The proof size is $O(\log(\#\text{ of txs}))$ and is of size $(\log_2 n_d)\cdot w = 3\cdot 256 = 768$ bit.

Accumulator scheme - For the above block example, we used SHA256 as the hash function H in HASHTOPRIME (Algorithm 4) as in Table III.

The accumulator declaration is the accumulated value. For security reasons, the RSA modulus field should be a very large number (e.g., 3072 bit), but in our example, in order to be able to show the number, we use only 64 bit for the modulus, $n =$ 0x4aad342c8d171f3 which is the product of 2 randomly primes. The initial value is $x =$ 0x42c79f75c2f308b.

The declaration is the accumulator value $z = \left(\left(\left(x^{tx_0}\right)^{tx_1}\right)^{\cdots}\right)^{tx_7} \mod n =$ 0x356e2acb83eaa7f. Its size is as the RSA modulus, in our example is 64 bit, but usually 3072 bit. The proof of each transaction in our example is in Table IV. The proof size is also as the RSA modulus.

Validating the inclusion of tx_2 is done as shown in Fig. 6 by comparing the declaration z to the computed value $tx_2\_proof^{\text{HASHTOPRIME}(tx2)} \mod n$.

tx_proof (64 bit)
0x995c9ed6269f5c
0xf50ba5dd021f8e
0x3bd0a206390314a
0x222eb08cf39f3eb
0x1a5ac9525df262c
0x18328f09ae4f5be
0x19a28bff37e4bd
0x4598171f94b1614

TABLE IV
ACCUMULATOR SCHEME: PROOF OF INCLUSION FOR EACH TRANSACTION IN THE BLOCK OF 8 TXS FROM TABLE II

Scheme	Declaration size (bits)	Proof size (bits)	Validation method	Declaration difficult tasks
Complete list	$n_d \cdot t$	-	Local	-
Merkle tree	t	$\log(n_d)\cdot t$	Challenge	$O(n_d)$ hash functions
Accumulator	r	r	Challenge	n_d HASHTOPRIME+ n_d modular exponentiations
Parameters:	n_d - # of transactions in declaration,			t - hash value in bits, r - RSA modulus in bits

TABLE V
FUNDAMENTAL PROPERTIES OF AGE PROVING BY VARIOUS DECLARATION SCHEMES

B. Scheme Comparison

We compare the schemes based on their complexity. The number of transactions in a declaration is denoted by n_d, the number of bits of the transaction hash by t, the number of bits in the RSA modulus by r and let the number of bits of the hash function output be assumed the same as transaction hash t. Table V summarizes properties of the different schemes for the transaction declaration that we have proposed.

The simplest *complete list* scheme does not require challenging the proposer during the validation process. Its drawback is the declaration size that is sent to every node in the network. The *Merkle tree* scheme declaration is small but the declarer needs to compute the Merkle tree and store it in its memory to answer challenges. The size of a Merkle proof is a log factor of the number of transactions in the declaration.

The *accumulator* scheme declaration is with a fixed size as the Merkle tree declaration, but practically would be larger since the RSA modulus would be a large number. The proof of a transaction is also a fixed size of the RSA modulus. The computational effort in producing the accumulator declaration is high since it requires finding a prime representation for each transaction (by Algorithm 4) and calculating the modular exponentiation of all prime numbers. The validation of the proof also requires finding a representation of the transaction and calculating one modular exponentiation. The expected number of hash computations for each transaction in running HASHTOPRIME is linear in the number of bits t, representing the hashed value.

VI. EXPERIMENTAL RESULTS

In Fig. 1 (presented in Section II) we demonstrated the potential impact of the age-aware block selection on transaction latency in the Ethereum network.

We have conducted two simulations. First, we built a toy blockchain simulation based on synthetic data of transaction issuance rate and block sizes. Second, we collected data from

$$tx_2\_proof^{\text{HASHTOPRIME}(tx2)} \mod n = \texttt{0x3bd0a206390314a}^{\texttt{0x422134d3b572a67cd550895f347269c8415fe192771b9a0a7f39163904484b9b}} \mod \texttt{0x4aad342c8d171f3} = \texttt{0x356e2acb83eaa7f} \stackrel{?}{=} z$$

Fig. 6. Validation of the seniority of a transaction tx_2 based on a proof tx_2_proof and a declaration $z = \texttt{0x356e2acb83eaa7f}$.

the Ethereum Ropsten testnet [32] on the issuance rate and block sizes and evaluated our blockchain simulation according to the collected data.

Before presenting detailed results, we highlight the main observations from the experimental study.

- Age-aware block selection reduces the q-99 percentile by 8%-40%.
- A strong correlation exists between the number of senior transactions and the total pending pool size.
- In networks with longer transaction propagation delays, the proposer must prove knowledge of more transactions—transactions seen as senior by the proposer but not necessarily by other nodes.
- Computing a Merkle tree declaration is three orders of magnitude faster than computing the accumulator declaration.

A. Synthetic Simulation

Setup - The network is constructed from four nodes and every round a random node issues new transactions. In the simulation, we considered different block sizes that characterize high and low network load. In every round, a random node is elected to propose a new block and new transactions are issued by some random node. The amount of new transactions being issued, k, is uniformly distributed over $L = \{1, 3, 5, 7, 9, 11, 13, 15, 17\}$. Sampling the amount of new transactions is a memory-based event, such that k at time t - denoted as k_t satisfies $|k_t - k_{t-1}| \leq 2$. The memory-based sampling aims to simulate different network loads over the simulation with periods of a high rate of new transactions and other periods with a low rate. The propagation delay time of an issued transaction to reach some peer node is uniformly distributed over $\{0, 2, 4\}$ rounds. The propagation delay of a transaction is independent for every peer in the network. We considered two block sizes of $b = 10$ or $b = 7$ transactions per block such that the larger block size allows more transactions to be included in each block and results in typically lower latency values. In the age-aware block selection, we set $s = 2$ for both block sizes, meaning that s among the b transactions in a block were selected due to their age.

Reducing Tail Latency by Age-aware Selection - Fig. 7 shows the CDF of the observed latency in units of blocks. The higher the load the network observes, the more significant is the improvement of the age-aware selection method in reducing the tail latency. In the experiment, the q-95 percentile of the latency is reduced from 40 to 37 blocks for low network load and in the high network load, it drops from 111 to 90, a reduction of 19%. The improvement of the higher q-99 percentile is even more significant. It is reduced from 60 blocks to 45 blocks in low load and from 173 to 104 blocks in high load, an improvement of roughly 40%.

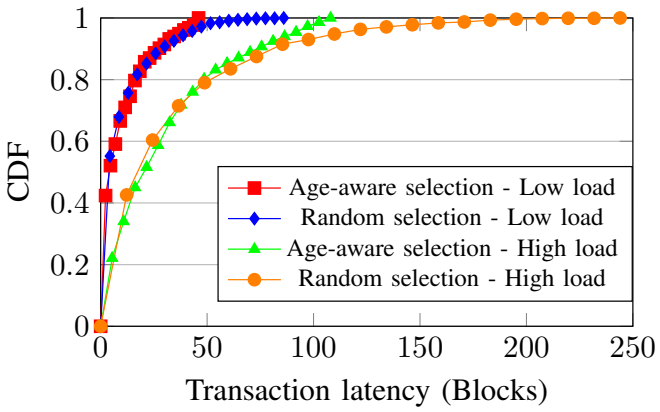


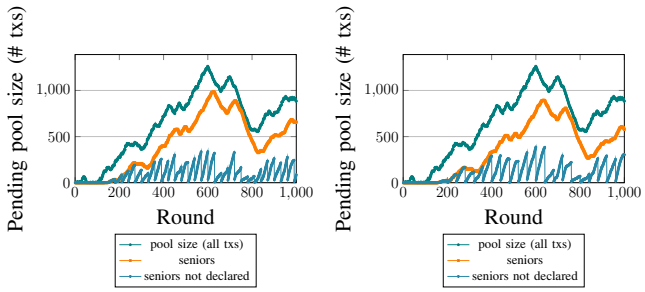
Fig. 7. CDF of transaction latency in random selection and in age-aware selection. Latency is measured as elapsed number of blocks between transaction issuance to inclusion. This shows the impact of age-aware selection in different network loads.

Pending Transactions Pool Characteristic - We inspected the pending transactions pool in order to see patterns in the amount of *senior transactions* and the amount of undeclared *senior transactions* with various $T_{declaration}$ values. We used $T_{old} = T_{declaration} \in \{30, 40\}$ and in each round we counted all pending transaction, the *senior transactions*, and *senior transactions* which were not declared yet. As can be seen in Fig. 8 the amount of *senior transactions* has the same trend as the total pending pool size since when the pool size is larger it causes transactions to wait longer time meaning more transactions will be *senior*. The amount of *senior transactions* is smaller in (b) than in (a) since there are fewer transactions which their age is greater than $T_{old} = 40$. The amount of undeclared *senior transactions* is reduced when a declaration phase occurs every $T_{declaration}$ rounds.

B. Simulation based on the Ropsten Testnet

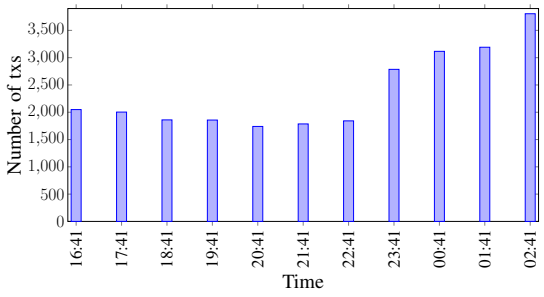
In order to get real data to simulate transactions arrival over time, we have used the Ropsten testnet of Ethereum network [32]. We have listened to a node and collected all transactions and blocks that were propagated from its peers. Our recording was performed for 12 hours beginning from September 5, 2020, 4:41pm (UTC). In Fig. 9 we see the distribution of the number of recorded new transactions over time in resolution of 1 hour (in (a)), 5 minutes (in (b)) or 1 minute (in (c)).

We used the collected data to simulate the arrival rate and block sizes by aggregating the number of arrived transactions

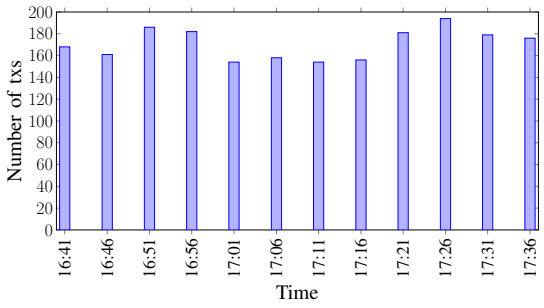


(a) $T_{old} = T_{declaration} = 30$ (b) $T_{old} = T_{declaration} = 40$

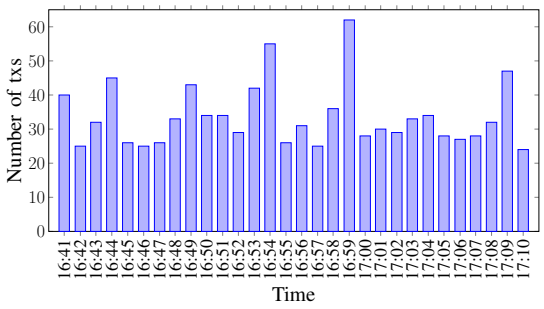
Fig. 8. Changes in pending pool size of a node through time. For different $T_{declaration}$ we measured in each round the pending pool size of some node, the number of *senior transactions* and the number of *senior transactions* which were not declared yet.



(a) Arrival rate in time units 1 hour. Each bar represents the number of new transactions in the hour that starts from the time in the label below it.



(b) Arrival rate in time units 5 minutes. It shows the first hour of recording.



(c) Arrival rate in time units 1 minutes. It shows the first 30 minutes of recording.

Fig. 9. Transactions arrival rate, recorded from Ethereum Ropsten testnet. It is shown in different time units resoultions.

in every minute and summing the number of transactions that are included in blocks propagated in every minute. In each round in our simulation, the number of issued transactions is

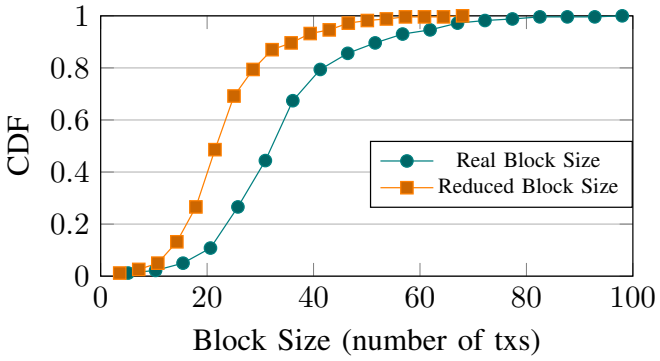


Fig. 10. CDF of block size used in the simulation. Measured block size is the number of transactions that were included in all blocks in some minute. Reduced block size is 70% of the measured block size, and is used for simulating high load.

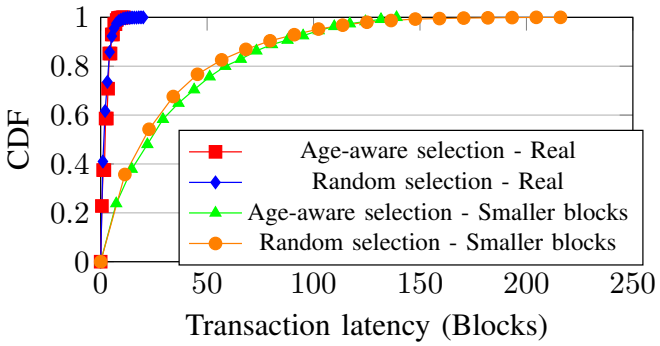


Fig. 11. Ropsten Testnet - CDF of random selection and of age-aware selection using real issue transaction rate and real vs. lower block size values. Lower block size corresponds to higher load of transaction.

the number of arrived transactions in the corresponding minute and the block size that the proposer creates is the number of transactions included in blocks in the same minute.

We used the measured data and reevaluated the simulation from Fig. 7 based on the real data. The number of senior transactions selected in the age-aware is proportional to the block size as $s = \lfloor 0.2b \rfloor$. In this simulation, the block size is not fixed. Let denote the number of transactions included in all blocks which were proposed in i^{th} minute from starting the recording as b_i. High latency is when the block size in each round i is set to $\lfloor 0.7b_i \rfloor$ and in low latency it is set as b_i. Fig. 10 depicts the CDFs of the measured block size and reduced block size. In the measured block size distribution 70% of the blocks include 20-42 transactions and 94% of the blocks have size of at most 60 transactions. This indicates that most blocks are within a factor of 3x in size from each other.

Fig. 11 depicts the CDFs of the latency. In the experiment, the q-99 percentile of the latency is reduced from 9 blocks to 8 blocks in low load and from 142 to 131 blocks in high load, an improvement of roughly 8%.

Fig. 12 shows the pending pool sizes in this simulation. The maximum of the pool size is smaller compared to the synthetic simulation in Fig. 8 since there are smaller periods when there are more issued transactions than the proposed block can include. Because of the relatively small pool size

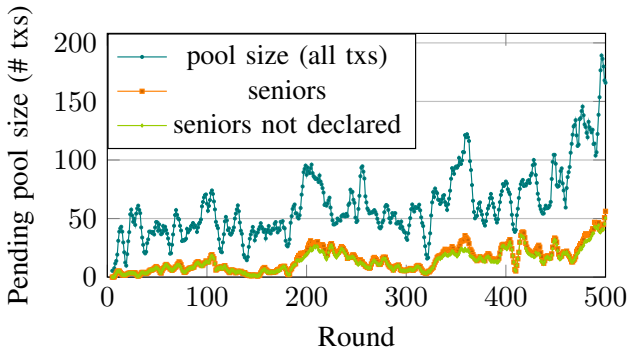


Fig. 12. Changes in pending pool size of a node through time using the testnet rates data and $T_{declaration} = 5$. Each point is computed as the average of 5 rounds.

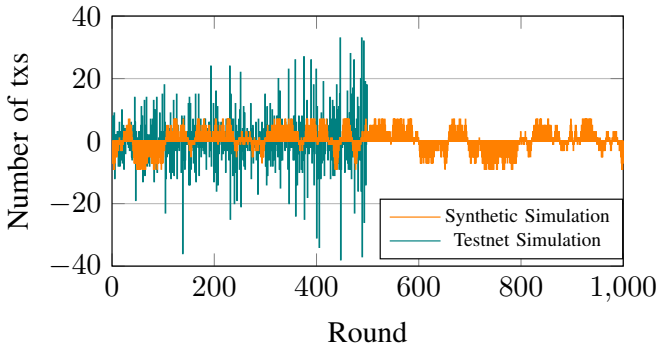


Fig. 13. Difference (either positive or negative) between number of issued transactions and block size in each round. The Ropsten testnet is evaluated for a shorter time period (500 vs. 1000 rounds) and observes higher variance.

we used $T_{old} = T_{declaration} = 5$.

The difference between the number of issued transactions number and the proposed block size is depicted in Fig. 13. One can see the consecutive positive values in the synthetic simulation that result in growing numbers in the pending transactions in the pool. The high variance between two sequential rounds in the testnet simulation difference causes pending pool size in Fig. 12 to be noisier than that of the synthetic simulation in Fig. 8.

C. Size of Declarations

We evaluated the usage of different schemes in terms of the declarations, challenge, and response messages size under various transaction delays scenarios. In the experiment, we used the blockchain described above except that the propagation delay of transactions is uniformly distributed over $[0, ..., d]$ rounds where d denotes the maximal delay. T_{old} and $T_{declaration}$ were set as 30 rounds. Transaction identifier ($txHash$) is of 256 bit. For every maximal delay, we measured for each scheme the mean of its declaration size among all declarations made in 1000 rounds.

Fig. 14a shows the mean size of declarations in each scheme. As higher the propagation delay is, there are fewer transactions in the pending pool since more transactions get included in some proposed block before their arrival to other nodes. Thus, when the maximal delay is higher, there are fewer transactions to include in declarations. In Merkle tree and accumulator schemes, the declaration size is fixed regardless of the number of included transactions as summarized in Table V.

When the proposer has included in the age-based part of the block transactions which are not defined as *seniors* by a committee member, there is a challenge and response epoch between the proposer and the committee member. In the experiment, we measured in every round the number of transactions included in the age-based part and were not defined as *seniors* by some other node. The sum of such transactions across all nodes gives the total number of challenge messages in any round. The proposer sends a response to a challenge, providing the inclusion proof for each transaction in some previous declaration. The mean sizes of challenge and response messages across all rounds appear in Fig. 14b and 14c, respectively. In the complete list scheme, there is no need for challenging the proposer. Therefore, the challenge and response message's size is shown as 0.

The larger the maximal delay is, the higher variance between the age of transaction seen by different nodes. The high variance of the age causes more transactions to be defined as *senior* by the proposer but not by other nodes. As seen in Fig. 14c the proposer has to prove the seniority of more transactions upon a higher maximal delay.

D. Declaration Computation Measurement

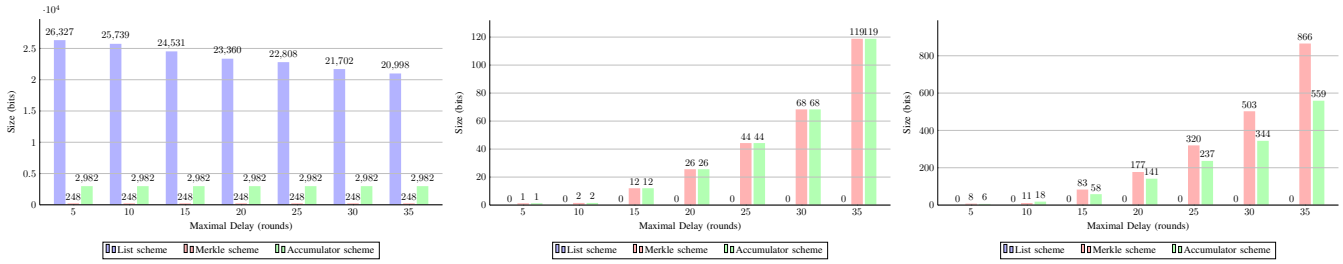
We evaluated the time of computing a declaration using a Python implementation of the Merkle tree and RSA accumulator. We have chosen different sizes of transaction pools and calculated the time to create the declaration by the mean of 10 runs. Table VI shows the comparison between the Merkle tree and accumulator schemes production times. The Merkle tree production time is 3 orders of magnitude faster than the accumulator since calculating the modular exponentiations in the accumulator is a very heavy task. Fig. 15 depicts the production times. The production time is linear with the number of transactions the declaration contains as analyzed in Table V. Times were measured on Macbook Pro with hardware that includes Processor: Intel(R) Core(TM) i7-7567U CPU @ 3.50GHz, Memory: 16 GB 2133 MHz LPDDR3.

VII. Analysis

In this section, we present a simple model for transaction latency and likewise refer to the freshness of transaction declarations through the notion of age of information.

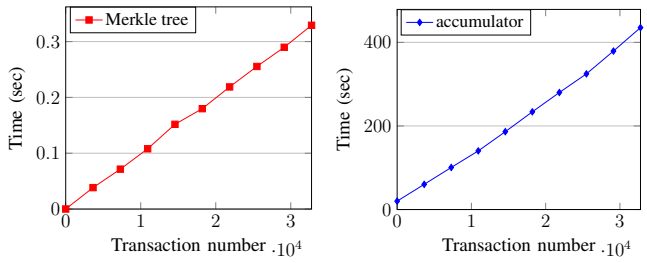
A. Delay Distribution for Age-aware Selection

Assume a constant rate of transaction issuance as well as a constant block size. These imply that the pending pool size is fixed. Let denote the pool size as n_p and block size as b. For random selection, the probability of a transaction to be chosen in a block is b/n_p. Latency of a transaction denoted by k where $k \geq 0$ is defined as the number of rounds that the transaction was in the pending pool and was not selected to a



(a) Declaration Size (mean) (b) Challenge Message Size (mean) (c) Response Message Size (mean)

Fig. 14. Measurement of declaration size, challenge and response messages under different declaration schemes.



(a) Merkle tree calculation time (b) Accumulator calculation time

Fig. 15. Time length of computing a declaration - time is linear in number of included transactions.

pool size	Merkle tree mean (sec)	Accumulator mean (sec)
2	6.493e-5	20.09
3642	0.03838	60.27
7283	0.07148	100.6
10924	0.1079	140.3
14564	0.1518	186.2
18205	0.1798	233.8
21846	0.2188	280.0
25486	0.2555	324.6
29127	0.2898	379.2
32768	0.3292	435.3

TABLE VI
DECLARATION COMPUTATION TIME FOR DIFFERENT POOL SIZE. VALUES ARE BASED ON THE MEAN FOR 10 RUNS.

proposed block. The latency in random selection follows the geometric distribution such that

$$\Pr\left(latency = k\right) = \left(1 - \frac{b}{n_p}\right)^k \left(\frac{b}{n_p}\right),$$

and

$$CDF = \Pr\left(latency \leq k\right) = 1 - \left(1 - \frac{b}{n_p}\right)^{k+1}.$$

For the age-aware selection, the number of *senior transactions* in each block is denoted as s. The number of senior transactions in the proposer pool is a random variable N_s. This number of transactions depends on the ratio between the issuance transactions rate and the block size and of course on the parameter of T_{old} which defines the minimal age of a *senior transaction*. To simplify the analysis further assume that N_s is also fixed and let denote it as n_s. Then, in age-aware selection:

$$\Pr\left(latency = k\right) = \begin{cases} k < T_{old} & \left(1 - \frac{b-s}{n_p}\right)^k \cdot \left(\frac{b-s}{n_p}\right) \\ k \geq T_{old} & \left(1 - \frac{b-s}{n_p}\right)^{T_{old}} \cdot \left(1 - \frac{s}{n_s} - \frac{b-s}{n_p}\right)^{k-T_{old}} \\ & \cdot \left(\frac{s}{n_s} + \frac{b-s}{n_p}\right) \end{cases}$$

The probability when $k \geq T_{old}$ is composed of three factors. The first factor means the probability of not being chosen for T_{old} rounds. Then the transaction becomes senior, and the second factor is the probability of not being selected for additional $k - T_{old}$ rounds. The third factor is of being chosen in a round following the earlier k rounds either in the random selection or as a senior.

For high percentiles where the latency is greater than T_{old}, the latency CDF holds

$$\Pr\left(latency \leq k\right) = 1 - \left(1 - \tfrac{b-s}{n_p}\right)^{T_{old}} \left(1 - \tfrac{s}{n_s} - \tfrac{b-s}{n_p}\right)^{k-T_{old}+1}.$$

B. Age of Information

Age of Information (AoI) [33] is a metric to measure freshness of status updates about some system. In our model, the information covers a set of pending transactions of some other node, expressed according to some declaration method. The pending transaction pool size of a node grows with the transaction issuance rate and decreases with the throughput of the blockchain (number of transactions in each block multiplied by the number of new blocks in time unit).

Declarations D_k^i of node i are sent in time points $\alpha_k^{(i)}, k \in \{1, 2, 3, ...\}$ with measurement intervals $\Gamma_k(i) = \alpha_k^{(i)} - \alpha_{k-1}^{(i)} = T_{declaration}$. Arrival time of declaration D_k^i to some other node j is denoted by $\beta_k^{(ij)}$ and follows an exponentially distributed delay [25], such as $t_{comm} = \beta_k^{(ij)} - \alpha_k^{(i)} \sim Exp(\eta)$. The *Peak Age of Information* (PAoI) is the value of age achieved immediately before receiving the declaration

$$\begin{aligned} A_k(D_k^i, j) = T_{declaration} + t_{comm} &= \Gamma_k(i) + t_{comm} \\ &= \beta_k^{(ij)} - \alpha_{k-1}^{(i)}. \end{aligned}$$

The average *Peak Age of Information* of a declaration is

$$\begin{aligned} A = E\left[\Gamma_k(i) + t_{comm}\right] &= T_{declaration} + E\left[t_{comm}\right] \\ &= T_{declaration} + 1/\eta. \end{aligned}$$

Intuitively, the age of information represents the elapsed time from the sending time of the recent declaration the node has received. After a declaration is sent, as the time ticks there might be more transactions whose local age is greater than T_{old}. Those transactions cannot be included in the age-based part of the block since their age cannot be verified. The smaller the average PAoI of the proposer's declaration, the more pending transactions are declared and can be selected as part of the age-based part of the block. Therefore, from the AoI aspect, the smaller $T_{declaration}$ the more transactions can be included in the age-based part.

VIII. Related Work

Fairness in Blockchain Systems and Beyond: There are other works that ensure fairness among transactions by forcing the proposer to randomly select the transactions [11] while the proposer is elected randomly and by a non-leader consensus protocol, meaning that every node proposes a random subset of transactions [34]. Declarations on the transactions known to a node have been studied in [22] to improve fairness. Another study by Nassar et al. [23], introduced a protocol that utilizes zone structures to improve fairness by capitalizing on the inherent structure of the network. This approach aims to further enhance fairness within the system. The notion of fairness among transactions is defined differently in [13] as each node gets a fair share of the ledger. Namely, each block contains the same number of transactions from each node assuming that they have infinite streams of transactions.

Another line of research [35], [36], [37], [38], [14] explores the concept of "order fairness," which involves selecting transactions based on their arrival order. This enforces transaction selection such that if many nodes learned about a transaction before some other transactions such an order should be reflected in the ledger. The protocol proposed in [14] focuses on achieving order fairness in permissioned environments, while [37] suggests approaches for achieving order fairness in permissionless contexts. Notably, [36] and [35] contribute techniques to streamline communication complexity, and [36] additionally achieves a standard liveness property, distinguishing it from other works that exhibit weaker liveness guarantees. Weaker potential definitions refer to the unfairness of the order of two transactions only if they were received sufficiently apart in time. Wendy [39] discusses *relative order fairness* and claims for fairness requirements for subsets of transactions, e.g., those belong to several existing markets.

Beyond blockchain, aspects of fairness have been studied in other networking and computer system settings where a restricted resource is shared among multiple entities. Examples include a queue with a bounded service rate or a link with limited capacity [40]. A well-known notion is the *max-min fairness*, which suggests how to determine a resource partition based on the demands of multiple users, summing up to more than the resource availability. Intuitively, such a fair allocation tries to maximize the share of users of small demands. On the other hand, users asking for a share larger than their relative part, can be negatively affected by other users. Generalizations of the definition have also been suggested [41], [42].

Transactions Propagation: Once a user sends a transaction to some node, the node is responsible to propagate this transaction through the network. In Bitcoin [3] the propagation is done in a gossip-based flooding method, namely a node that receives a transaction potentially relays it to all its peers which propagate it onwards. In Ethereum [4], there are different client protocols as Geth which sends the transactions to all peers, and Parity which sends transactions only to a square root of its peers [43].

Transaction Selection: In our work, we discuss how a block proposer should select which transactions to include inside the proposed block. In commonly used blockchains based on PoW consensus as Bitcoin [3] and Ethereum [4], the selection process is not part of the protocol and can be decided by the proposer. Since transactions might have different fees, the block proposer typically selects those of maximal fee it is aware of in order to maximize its profit [44].

Reducing Tail Latency: We focus on reducing tail latency of transactions. The problem of tail latency is known to highly affect QoS also in other environments such as storage systems or cloud services and various approaches have been suggested to mitigate it. This includes reducing the tail latency of read and write operations in SSD [45] while relying on garbage collection, the response time of interactive services [46] with parallelism, and request completion time of cloud servers [47] through the use of data duplication.

IX. Discussion and Generalizations

We highlight potential generalizations and points to consider this work opens up.

Block Computation Capacity: Rather than being limited to include a fixed number of b transactions, in some blockchain networks, a block can be restricted based on the expected amount of computation. In Ethereum, a transaction is associated with a bound on its implied computation described as gas limit. The gas is small for simple payment transactions and higher for advanced smart contract transactions that involve computational-intensive operations and allocate extra memory. The age-aware fairness model can also support such a computation-based model.

Denote by C_b the block computation capacity. Within such a block, up to C_s of its capacity can be dedicated to senior transactions. Similar to the description in Section II-D, in each round, the transactions are sorted based on their hash values. Based on the order, the maximal number of first transactions that their sum of expected computation is below $C_b - C_s$ are selected to be included in the block.

The definition of fairness among transactions can be extended to avoid starvation of transactions with low computation by a small number of computation-intensive transactions. One method to do that is to scale the random to a range based on the expected gas limit of a transaction so that light transactions have higher chances to be included in a block.

Ordering within Block: While there are many definitions of ordering fairness, the definition in this paper refers to the inclusion of transactions in a block. Beyond that definition, in several blockchain systems, the internal order of transactions

within a block can also be of importance such as with regards to miner extractable value (MEV, see for instance the Introduction for more details). The age-aware fairness can also enhance fairness with regard to internal block ordering by restricting the miner's flexibility. Two simple ways to do so can be to include all senior transactions first and then regular (namely, typically non-senior) transactions such that within its subset transactions must be ordered according to the hash values computed for every transactions. Similarly, we can mix the senior and the regular transactions such that all transactions must be ordered due to the hash values. For both restrictions, validation of the internal block order is simple through computing the hash values and can be done easily also be committee members that do not have some of the block transactions in their pools.

Verifiable Delay Function (VDF) declaration scheme: VDF [48], [49] is a recent cryptographic tool used for adding delay in decentralized applications. The VDF is a function $f : \mathcal{X} \rightarrow \mathcal{Y}$ which its computation time is prescribed, even on a parallel computer. Once the function was computed on $x \in \mathcal{X}$ its output $y \in \mathcal{Y}$ is unique and can be verified quickly. The VDF is composed of three algorithms:

- $setup(\lambda, T) \rightarrow pp$ which initializes the VDF environment. It takes a security parameter λ and a time bound T, and outputs public parameters pp.
- $evaluation(pp, x) \rightarrow (y, \pi)$ which takes an input $x \in \mathcal{X}$ and outputs a $y \in \mathcal{Y}$ together with its a proof π.
- $verification(pp, x, y, \pi) \rightarrow \{accept, reject\}$ outputs $accept$ if y is indeed the right evaluation of the VDF and otherwise $reject$.

In this scheme, committee members of the first block run the $setup$ which initializes the public parameters pp. The parameters pp are saved in each node and when a new node joins the network it asks for the parameters pp from its peer. Each node either periodically or whenever the amount of new pending transactions is large arranges all transactions in its pending pool as a Merkle tree and computes the VDF function on the Merkle root MR. Consider a block verification by a committee member such that some transactions are included in the age-based part as *senior*. To validate such a selection the committee member challenges the proposer to send the transaction Merkle proof with the root MR together with the VDF proof π of MR.

There are two common implementations of VDFs (By Pietrzak and by Wesolowski [50]). We describe the VDF algorithms so they can be implemented in either construction.

- The *setup* generates an RSA group shared by all network nodes by using a decentralized algorithm for multi-party [51]. It produces an RSA modulus N of $\lambda = 2048$ bits. The SHA256 is used as the hashing function and the desired time-bound is set as T_{old}.
- The *evaluation* uses as input the Merkle root of the pending pool $x = MR$, and then computes $y = MR^{2^{T_{old}}} \mod N$. The proof π which is part of the output is defined in each VDF construction.
- The $verification$ algorithm differs between each VDF construction. While Pietrzak's performance of the verification step is better than Wesolowski's it trades off the network overhead of the VDF proof which is higher.

The computation produces a proof that enables everyone to efficiently verify it. One might consider applying VDFs as another scheme for proving the age of a transaction. Basically, the availability of the output of a VDF computed over a transaction can demonstrate that its age is at least the time it takes to compute the function. The computational overhead of such a potential scheme can be high in practice. A node should periodically produce a Merkle tree from its pending transactions pool and calculate a VDF on it and store the Merkle tree together with the VDF proofs.

Age-aware fairness overhead: Using our age-aware block selection schemes compared to other block selection methods necessitates overhead. The validation process of a new block is computationally more complex. In the schemes where the committee members challenge the proposer, it also causes communication overhead and delay. The declaration schemes increase the communication overhead and computing the Merkle tree or accumulator declarations requires computational resources. Since the validation process speed affects the rate of new blocks added to the blockchain, achieving age-aware fairness might degrade the network transactions throughput to some extent.

Senior priority level: The scheme allows including s among the b block transactions based on their seniority. One can wonder how to select the value of $s \in [1, b]$. On the one hand, selecting s such that a large portion of the block contains senior transactions reduces fairness since the senior transactions are not selected randomly and this increases the latency of lower percentiles. On the other hand, selecting a low value of s reduces tail latency improvement. In time the network has low load, the values of s does not have high impact as often the number of transactions becoming senior is low and if a transaction indeed becoming so, it often can be included within the next block. However, consider for instance a burst of transactions that are issued in a short period of time. If not included in a block within some time, such transactions can become senior at similar times. For a high value of s, those transactions can be included in the next block while for a low s value, such transactions would have to wait longer and appear in one of the several next blocks. This highly affects the probability of a transaction observing a delay much longer than T_{old}, the time it takes to be senior. On the other hand, high value of s also reduces the number of randomly selected transactions in the block, implying lower chances to observe very low latency. Overall, higher s values make the latency distribution more dense.

Weighted random transaction selection: The proposed age-aware block selection is based on the ability of the block proposer to include s *senior transactions* among a block of b transactions while other $b - s$ transactions are randomly selected. An alternative scheme to prioritize senior transactions is to select all b transactions of the block randomly among the pool of pending transactions but with associating a probability to each transaction to be selected based on its age. In such a model, the notion of seniority is continuous rather than the discrete definition we used for seniority (an age beyond a

	Age-aware selection	Weighted random selection
Notion of seniority	discrete	continuous
Block validation	validation of each *senior transaction*	validation based on similarity between proposed block to local pool content

TABLE VII
COMPARISON OF WEIGHTED RANDOM SELECTION TO AGE-AWARE SELECTION

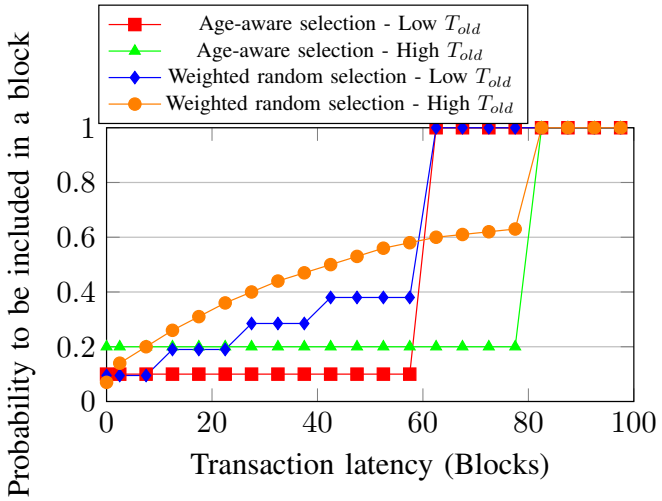


Fig. 16. Illustration of Weighted random transaction selection.

threshold). A block can be validated based on the similarity between its content to a local computation by a committee member. Although the age of a transaction can differ among nodes, it is expected to be similar when the variance of transaction propagation delay is relatively small. Thus, blocks are selected from similar distributions and some level of similarity between the blocks is expected. This scheme of transaction selection has the advantage that the number of *senior transactions* in a block does not need to be predefined by some parameter and it is controlled by the weighted selection. Table VII shows different attributes of weighted random selection and age-aware selection. Fig. 16 illustrates the weighted random selection by showing the probability of a transaction to be included in a block based on its latency. We assume here for simplicity that the number of senior transactions is small so they can be included in the next block. Two curves refer to the traditional age-aware selection with two values of the threshold T_{old}: low $T_{old} = 60$ (in red) and high $T_{old} = 80$ (in green). Transactions of lower latency observe identical chances to be selected. The next two curves refer to weighted random selection. In both, even below the threshold T_{old} the probability increases for higher latency values. First, in blue in the form of a step function. Second, in orange, as a strictly monotonic function. These are simple examples for weighted random selection and the particular computation of the probability as a function of the latency can vary. Moreover, in these examples the probability jumps to 1 upon a latency of T_{old} which is not mandatory.

Finality of a committee decision: Once the committee votes are collected and the proposed block is approved, the new block is propagated to all network nodes and is not further changed. This typically requires a predetermined minimum number of supporting votes among the committee but once achieved implies finality. This is unlike for instance the typical proof-of-work block selection in Bitcoin when finality of a block is not achieved immediately after block approval.

Combining declarations in proposed blocks: Consider a scenario in which a node is selected as a block proposer every fixed interval (e.g., in a round-robin selection) and the interval is relatively small to a high percentile of transaction latency. Then, it is possible to reduce communication overhead by including the proposer declaration as part of the proposed block. Since the communication overhead in the Merkle tree and accumulator schemes is relatively small, we do not expect it to reduce the transactions throughput.

Flooding attacks prevention: In the fairness model where transactions are served equally, we need to prevent a potential attack of a node that floods the blockchain with transactions to achieve denial of service (DoS). Simple mitigation could be applying a fixed fee to each transaction such that this attack cost will be high. Another mitigation can be to monitor nodes that spam the network and ignore their transactions.

X. CONCLUSION

Transaction confirmation time is of importance and highly affects the quality of service of blockchain systems. In existing fair block selections that time can observe a large variance. This paper proposes mechanisms to reduce the skewness of that time through prioritizing in the block selection transactions with earlier issuance time. We explain how this can be achieved while keeping the fairness property. The solutions are designed for multiple distributions of transaction propagation and make use of concise declarations on pending pools. Experiments demonstrate a reduction in high percentiles of the confirmation time.

REFERENCES

[1] Y. Sokolik and O. Rottenstreich, "Age-aware fairness in blockchain transaction ordering," in *IEEE/ACM International Symposium on Quality of Service (IWQoS)*, 2020.

[2] C. Dwork and M. Naor, "Pricing via Processing or Combatting Junk Mail," in *CRYPTO*, 1992.

[3] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System," 2008. [Online]. Available: http://bitcoin.org/bitcoin.pdf

[4] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," 2014. [Online]. Available: https://gavwood.com/paper.pdf

[5] A. Kiayias, A. Russell, B. David, and R. Oliynykov, "Ouroboros: A Provably Secure Proof-of-Stake Blockchain Protocol," in *CRYPTO*, 2017.

[6] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich, "Algorand: Scaling Byzantine Agreements for Cryptocurrencies," in *Symposium on Operating Systems Principles (SOSP)*, 2017.

[7] P. Daian, S. Goldfeder, T. Kell, Y. Li, X. Zhao, I. Bentov, L. Breidenbach, and A. Juels, "Flash boys 2.0: Frontrunning in decentralized exchanges, miner extractable value, and consensus instability," in *IEEE Symposium on Security and Privacy (SP)*, 2020.

[8] S. Eskandari, S. Moosavi, and J. Clark, "SoK: Transparent dishonesty: Front-running attacks on blockchain," in *Financial Cryptography and Data Security (FC) International Workshops*, 2019.

[9] V. Manahov, "Front-running scalping strategies and market manipulation: why does high-frequency trading need stricter regulation?" *Financial Review*, vol. 51, no. 3, pp. 363–402, 2016.

[10] L. Heimbach and R. Wattenhofer, "Sok: Preventing transaction reordering manipulations in decentralized finance," *CoRR*, 2022.

[11] D. Yakira, A. Asayag, G. Cohen, I. Grayevsky, M. Leshkowitz, O. Rottenstreich, and R. Tamari, "Helix: A fair blockchain consensus protocol resistant to ordering manipulation," *IEEE Trans. Netw. Serv. Manag.*, vol. 18, no. 2, pp. 1584–1597, 2021.

[12] S. Shyamsukha, P. Bhattacharya, F. Patel, S. Tanwar, R. Gupta, and E. Pricop, "PoRF: Proof-of-reputation-based consensus scheme for fair transaction ordering," in *International conference on electronics, computers and artificial intelligence (ECAI)*, 2021.

[13] K. Lev-Ari, A. Spiegelman, I. Keidar, and D. Malkhi, "FairLedger: A Fair Blockchain Protocol for Financial Institutions," in *International Conference on Principles of Distributed Systems (OPODIS)*, 2019.

[14] M. Kelkar, F. Zhang, S. Goldfeder, and A. Juels, "Order-Fairness for byzantine consensus," in *Annual International Cryptology Conference (CRYPTO)*, 2020.

[15] G. Shang, N. Ilk, and S. Fan, "Need for speed, but how much does it cost? unpacking the fee-speed relationship in bitcoin transactions," *Journal of Operations Management*, 2022.

[16] J. De Vries, D. Roy, and R. De Koster, "Worth the wait? How restaurant waiting time influences customer behavior and revenue," *Journal of operations Management*, vol. 63, pp. 59–78, 2018.

[17] P. B. Goes, N. Ilk, M. Lin, and J. L. Zhao, "When more is less: Field evidence on unintended consequences of multitasking," *Management Science*, vol. 64, no. 7, pp. 3033–3054, 2018.

[18] P. Kumar, M. U. Kalwani, and M. Dada, "The impact of waiting time guarantees on customers' waiting experiences," *Marketing science*, vol. 16, no. 4, pp. 295–314, 1997.

[19] K. Wang and H. S. Kim, "Fastchain: Scaling blockchain system with informed neighbor selection," in *IEEE International Conference on Blockchain (Blockchain)*, 2019.

[20] Y. Zhu, C. Hua, D. Zhong, and W. Xu, "Design of low-latency overlay protocol for blockchain delivery networks," in *IEEE Wireless Communications and Networking Conference (WCNC)*, 2022.

[21] R. Saltini, "Bigfoot: A robust optimal-latency bft blockchain consensus protocol with dynamic validator membership," *Computer Networks*, vol. 204, p. 108632, 2022.

[22] A. Orda and O. Rottenstreich, "Enforcing Fairness in Blockchain Transaction Ordering," in *IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, 2019.

[23] M. Nassar, O. Rottenstreich, and A. Orda, "CFTO: communication-aware fairness in blockchain transaction ordering," *IEEE Trans. Netw. Serv. Manag.*, vol. 21, no. 1, pp. 490–506, 2024.

[24] K. Croman, C. Decker, I. Eyal, A. E. Gencer, A. Juels, A. Kosba, A. Miller, P. Saxena, E. Shi, E. Gün Sirer, D. Song, and R. Wattenhofer, "On Scaling Decentralized Blockchains," in *Financial Cryptography and Data Security*, 2016.

[25] C. Decker and R. Wattenhofer, "Information propagation in the Bitcoin network," in *IEEE International Conference on Peer-to-Peer Computing (P2P)*, 2013.

[26] R. C. Merkle, "Secrecy, Authentication, and Public Key Systems," PhD Thesis, Department of Electrical Engineering, Stanford University, 1979.

[27] J. Benaloh and M. de Mare, "One-Way Accumulators: A Decentralized Alternative to Digital Signatures," in *EUROCRYPT*, 1993.

[28] D. Boneh, B. Bünz, and B. Fisch, "Batching Techniques for Accumulators with Applications to IOPs and Stateless Blockchains," in *CRYPTO*, 2019.

[29] N. Barić and B. Pfitzmann, "Collision-Free Accumulators and Fail-Stop Signature Schemes Without Trees," in *EUROCRYPT*, 1997.

[30] R. L. Rivest, A. Shamir, and L. Adleman, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems," *Communications of the ACM*, vol. 21, no. 2, pp. 120–126, 1978.

[31] C. Pomerance, J. L. Selfridge, and S. S. Wagstaff, "The Pseudoprimes to $25{\cdot}10^9$," *Mathematics of Computation*, vol. 35, no. 151, pp. 1003–1026, 1980.

[32] "Ropsten Ethereum Testnet Explorer." [Online]. Available: http://ropsten.etherscan.io/

[33] A. Kosta, N. Pappas, and V. Angelakis, "Age of Information: A New Concept, Metric, and Tool," *Foundations and Trends® in Networking*, vol. 12, no. 3, pp. 162–259, 2017.

[34] A. Miller, Y. Xia, K. Croman, E. Shi, and D. Song, "The Honey Badger of BFT Protocols," in *ACM CCS*, 2016.

[35] Y. Zhang, S. T. V. Setty, Q. Chen, L. Zhou, and L. Alvisi, "Byzantine ordered consensus without byzantine oligarchy," in *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2020.

[36] M. Kelkar, S. Deb, S. Long, A. Juels, and S. Kannan, "Themis: Fast, strong order-fairness in byzantine consensus," *IACR Cryptol. ePrint Arch.*, p. 1465, 2021.

[37] M. Kelkar, S. Deb, and S. Kannan, "Order-fair consensus in the permissionless setting," in *ACM ASIA Public-Key Cryptography Workshop*, 2022.

[38] C. Cachin, J. Micic, N. Steinhauer, and L. Zanolini, "Quick order fairness," in *Financial Cryptography and Data Security (FC)*, 2022.

[39] K. Kursawe, "Wendy, the good little fairness widget," *arXiv preprint arXiv:2007.08303*, 2020.

[40] R. Jain, D.-M. Chiu, and W. R. Hawe, "A quantitative measure of fairness and discrimination," *Eastern Research Laboratory, Digital Equipment Corporation, Hudson, MA*, 1984.

[41] E. Danna, A. Hassidim, H. Kaplan, A. Kumar, Y. Mansour, D. Raz, and M. Segalov, "Upward Max-Min Fairness," *Journal of the ACM (JACM)*, vol. 64, no. 1, pp. 2:1–2:24, 2017.

[42] P. Nilsson, "Fairness in communication and computer network design," PhD Thesis, Department of communication systems, Lund University, 2006.

[43] S. K. Kim, Z. Ma, S. Murali, J. Mason, A. Miller, and M. Bailey, "Measuring Ethereum Network Peers," in *ACM Internet Measurement Conference (IMC)*, 2018.

[44] W. Wang, D. T. Hoang, P. Hu, Z. Xiong, D. Niyato, P. Wang, Y. Wen, and D. I. Kim, "A Survey on Consensus Mechanisms and Mining Strategy Management in Blockchain Networks," *IEEE Access*, vol. 7, pp. 22 328–22 370, 2019.

[45] W. Kang and S. Yoo, "Dynamic management of key states for reinforcement learning-assisted garbage collection to reduce long tail latency in SSD," in *Annual Design Automation Conference (DAC)*, 2018.

[46] M. E. Haque, Y. h. Eom, Y. He, S. Elnikety, R. Bianchini, and K. S. McKinley, "Few-to-Many: Incremental Parallelism for Reducing Tail Latency in Interactive Services," in *ACM ASPLOS*, 2015.

[47] H. M. Bashir, A. B. Faisal, M. A. Jamshed, P. Vondras, A. M. Iftikhar, I. A. Qazi, and F. R. Dogar, "Reducing Tail Latency using Duplication: A Multi-Layered Approach," in *ACM CoNEXT*, 2019.

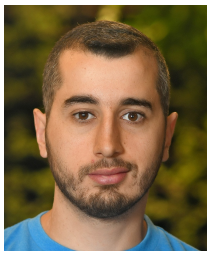
[48] D. Boneh, J. Bonneau, B. Bünz, and B. Fisch, "Verifiable Delay Functions," in *Advances in Cryptology – CRYPTO*, 2018.

[49] K. Pietrzak, "Simple Verifiable Delay Functions," in *Innovations in Theoretical Computer Science (ITCS)*, 2019.

[50] D. Boneh, B. Bünz, and B. Fisch, "A Survey of Two Verifiable Delay Functions," *IACR Cryptology ePrint Archive*, vol. 2018, p. 712, 2018.

[51] V. Attias, L. Vigneri, and V. Dimitrov, "On the Decentralized Generation of the RSA Moduli in Multi-Party Settings," *arXiv:1912.11401*, 2019.

Yaakov Sokolik received the B.Sc. and M.Sc. degrees in Computer Science from the Technion, Haifa, in 2017 and 2020, respectively. He is currently a Software Engineer at Pinecone, where he works on building a scalable vector database.



Mohammad Nassar received the B.Sc. and degree in Computer Engineering from the Technion, Haifa, Israel. In 2023 he completed the M.Sc. degree at the Technion's Viterbi Department of Electrical and Computer Engineering.

Ori Rottenstreich is an associate professor at the department of Computer Science and the department of Electrical and Computer Engineering of the Technion, Haifa, Israel. Previously, he was a Postdoctoral Research Fellow at Princeton university. Ori received his B.Sc. degree in Computer Engineering and Ph.D. degree in Electrical Engineering from Technion.