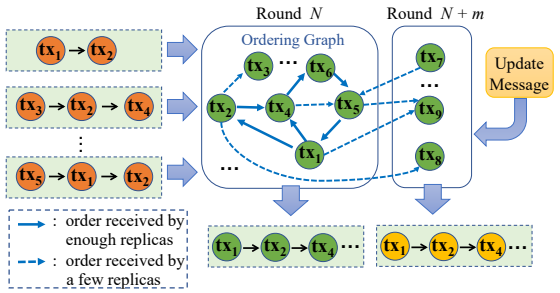
Auncel: Fair Byzantine Consensus Protocol with High Performance

Wuhui Chen[1,3], Yikai Feng[2], Jianting Zhang[4], Zhongteng Cai[5], Hong-Ning Dai*[6], and Zibin Zheng[1]
[1]School of Software Engineering, [2]School of Computer Science and Engineering, Sun Yat-Sen University, China
[3]Pengcheng Laboratory, China [4]Department of Computer Science, Purdue University, USA
[5] Department of Computer Science and Engineering, Ohio State University, USA
[6] Department of Computer Science, Hong Kong Baptist University, Hong Kong
{chenwuh, zhzibin}@mail.sysu.edu.cn, fengyk5@mail2.sysu.edu.cn, zhan4674@purdue.edu, cai.1125@osu.edu, hndai@ieee.org

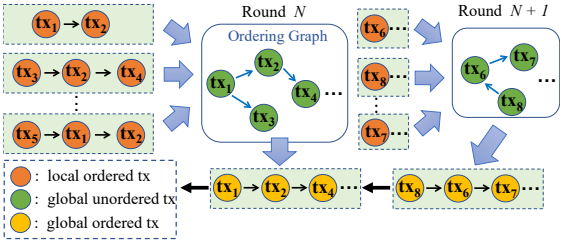
***Abstract*—Since the advent of decentralized financial applications based on blockchains, new attacks that take advantage of manipulating the order of transactions have emerged. To this end, order fairness protocols are devised to prevent such order manipulations. However, existing order fairness protocols adopt time-consuming mechanisms that bring huge computation overheads and defer the finalization of transactions to the following rounds, eventually compromising system performance. In this work, we present Auncel, a novel consensus protocol that achieves both order fairness and high performance. Auncel leverages a weight-based strategy to order transactions, enabling all transactions in a block to be committed within one consensus round, without cost computation and further delays. Furthermore, Auncel achieves censorship resistance by integrating the consensus protocol with the fair ordering strategy, ensuring all transactions can be ordered fairly. To reduce the overheads introduced by the fair ordering strategy, we also design optimization mechanisms, including dynamic transaction compression and adjustable replica proposal strategy. We implement a prototype of Auncel based on HotStuff and construct extensive experiments. Experimental results show that Auncel can increase the throughput by 6× and reduce the confirmation latency by 3× compared with state-of-the-art order fairness protocols.**

I. Introduction

Blockchain is a Byzantine Fault Tolerant (BFT) State Machine Replication (SMR), of which replicas can maintain an ever-growing consistent ledger without trusting each other. With a BFT consensus protocol, a blockchain system maintains two intrinsic properties: *safety* where all honest replicas output consistent states, and *liveness* where all transactions are eventually handled. Both of these properties, however, do not capture order relationships among transactions, making existing consensus protocols vulnerable to order manipulation risks where malicious replicas can manipulate transaction orders arbitrarily. Many recent works [1]–[3] have shown that order manipulations can extract tremendous profits from Decentralized Finance (DeFi), affecting the stability of DeFi and even threatening the security of consensus protocols. For instance, Qin et al. [3] shows that an order manipulation called sandwich attack extracted 174.34M USD in 32 months. Therefore, a new property that existing BFT protocols strive to achieve is *transaction order fairness*. Informally, the order fairness indicates that transactions are executed and committed in the same order as the majority of replicas' receiving orders.



(a) Fair transaction order processing in existing protocols



(b) Ideal fair transaction order processing

Fig. 1. Transaction processing in order fairness protocols

Motivation. To achieve order fairness and prevent order manipulations, several consensus protocols have been proposed [5]–[7], [9]–[11]. Some of them [5], [11] introduce a separate ordering stage, in which each transaction is assigned a tamper-resistant timestamp that will be used as the fair ordering indicator in the following consensus stage. However, a timestamp-based solution relies on synchronized clocks, which inevitably incurs huge costs for synchronizing clocks and is impractical in a realistic network. Moreover, separating the ordering stage from consensus is vulnerable to censorship attacks where transactions are refused to be ordered by malicious replicas, thus compromising order fairness [6]. To tackle this issue, other schemes [6]–[8] embed fairly ordering into the

* Hong-Ning Dai is the corresponding author.

The work described in this paper was supported by the National Key Research and Development Plan (2021YFB2700302), the National Natural Science Foundation of China (62172453), the National Natural Science Foundation of Guangdong province (2022A1515010154), the Major Key Project of PCL (PCL2023AS7-1), and the Pearl River Talent Recruitment Program (2019QN01X130).

TABLE I
COMPARISON OF AUNCEL WITH EXISTING CONSENSUS PROTOCOL

Protocols	Order Fairness	Throughput[1]	Latency[1]	Byzantine Threshold[2]	Censorship Resistance	No Synchronized Clocks Required
HotStuff [4]	✗	22,952 TPS	17 ms	$n \geq 3f+1$	✓	✓
Pompē [5]	✓	8,022 TPS	50 ms	$n \geq 3f+1$	✗	✗
Themis [6]	✓	2,580 TPS	155 ms	$n \geq 4f+1$	✓	✓
Aequitas [7]	✓	4,351 TPS	92 ms	$n \geq 4f+1$	✓	✓
Rashnu [8]	✓	6,593 TPS	61 ms	$n \geq 4f+1$	✓	✓
Auncel (Our Approach)	✓	20,545 TPS	19 ms	$n \geq 4f+1$	✓	✓

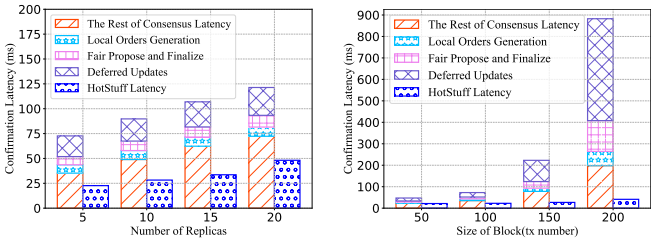
[1] Peak throughput and confirmation latency are measured through experiments conducted on two servers with 100 Mb/s bandwidth. The network size is 5 and block size is 100.
[2] n is the total number of replicas and f is the number of Byzantine replicas.

consensus with an order fairness algorithm. Its main idea is based on the ordering preferences from all replicas to generate a transaction ordering graph, where nodes represent transactions and a direct edge indicates the order of two transactions. This transaction ordering graph helps capture order fairness among transactions. In particular, if a sufficiently large number of replicas receive transaction tx_1 before transaction tx_2, there will be an edge from tx_1 to tx_2 in the ordering graph, indicating tx_1 must be ordered before tx_2 after the consensus. However, due to the byzantine behaviors (where malicious nodes claim their order preferences arbitrarily) and network delay (that leads to inconsistency of receiving orders for honest nodes), it is inevitable to generate ordering cycles as shown in Fig. 1(a). The ordering cycles will lead to global unordered transactions that will be deferred to be ordered until the ordering cycles are eliminated with update messages in the following consensus rounds. As a consequence, these protocols increase the confirmation latency of transactions and cannot achieve order fairness efficiently.

Observation. To validate our concern regarding the existing protocols, we implement the most representative scheme – Themis [6] on top of the HotStuff protocol [4] and evaluate the confirmation latency of Themis. Generally, Themis commits a transaction via several stages: local orders generation, fair proposing and finalizing order, multi-step message broadcast and validation as well as deferred updates. Fig. 2 shows the comparison of confirmation latency between Themis and HotStuff. We observe that with more replicas and larger block sizes, the complexity of generating an ordering graph increases, and more time is required to eliminate ordering cycles in the ordering graph, consequently leading to a high confirmation latency. Therefore, to enhance the performance of these order fairness protocols, a crux is to reduce the complexity of generating ordering graphs and eliminate ordering cycles efficiently.

Our solution. In this paper, we present Auncel[1], a novel consensus protocol ensuring order fairness for transactions while achieving high system performance. Fig. 1(b) presents an overview of processing transactions in Auncel. First, similar to [6]–[8], Auncel is designed to integrate an order fairness algorithm into the consensus protocol (§ IV-B). This circumvents the assumption of synchronized clocks along with their synchronized cost. With such a design, Auncel also achieves censorship resistance, indicating malicious nodes cannot intentionally prevent a transaction from being ordered and committed. The censorship resistance guarantees that transactions will be ordered fairly as long as they are received by honest replicas [6].

[1]Auncel was an ancient balance scale used for measuring fairness.



(a) Latency vs varied number of replicas (b) Latency vs varied block size

Fig. 2. Confirmation latency between HotStuff and order-fairness protocol

Then, we propose a *weight-based order fairness algorithm* to generate ordering graphs and eliminate ordering cycles efficiently (§ IV-C). Its main idea is to order and commit all transactions of a block in one consensus round. Specifically, different from previous protocols using the same weight to depict local order, our algorithm attaches a diverse weight to transactions based on their positions in a replica's local order. This enables replicas to form a linear relative order between transactions based on these weights, thereby effectively solving the ordering cycle problem.

While our order fairness algorithm can reduce the confirmation latency compared with the existing works, it also introduces some communication overheads that are required to establish a fair order for transactions, including proposing diverse transaction sets of replicas and inevitably processing local orderings from replicas. To reduce such overheads (§ IV-D), we design a dynamic transaction compression mechanism to optimize and simplify transactions and local orderings, which provides a user-friendly reference model for effectively compressing transaction sets during the consensus process. Further, we also adopt an adjustable replica proposal strategy to batch more transactions, thereby making full use of network bandwidth and improving performance. In summary, we make the following contributions in this work:

- **Novel Fairness Consensus:** We present a novel fairness

consensus protocol and integrate the consensus protocol with the fair ordering strategy to avoid censorship. Compared with other fairness consensus schemes, our fairness consensus can achieve both transaction order fairness and high performance.

- **Weight-based Ordering Strategy:** We design a weight-based ordering strategy that can commit all transactions fairly within one consensus round, significantly reducing the confirmation latency. Moreover, we present the formation process of transaction weight and two-stage ordering in detail. Furthermore, we introduce a dynamic transaction compression mechanism and an adjustable replica proposal strategy to mitigate the communication overheads from achieving order fairness.
- **Experiment Evaluation:** We implement a prototype of Auncel bootstrapped from HotStuff. Through both prototype experiments and simulations, we show that Auncel outperforms state-of-the-art baselines in terms of throughput, scalability, and confirmation latency while still ensuring order fairness.

Comparisons with existing protocols. Table I compares Auncel with several representative order-fairness protocols. Built on HotStuff, Auncel can achieve the closest performance as HotStuff, i.e., 20,000+ transactions per second (TPS) and 19 ms confirmation latency, outperforming the state-of-the-art order-fairness consensus protocols.

II. Related Work

Order manipulations. Blockchain is a state machine replication that is driven by a consensus protocol, such as Proof-of-Work [12] and Byzantine Fault Tolerance [4] protocols. However, the traditional consensus protocols adopted in both permissioned blockchains [13]–[15] and permissionless blockchains [16]–[20] have a problem that they cannot prevent the exertion of manipulative actions on transaction order by malicious replicas. Such order manipulations have been widely explored in recent years [1]–[3], [21], and have brought millions of dollars to attackers. For instance, attackers can extract 540M USD in profits from decentralized finance by injecting transactions or changing the order of transactions [3].

Censorship-resistant protocols. To address the problem of order manipulation, recent works [22]–[28] dive into diminishing the ability of block proposers (i.e., leaders) to order transactions. This leads to a so-called censorship-resistant protocol. Specifically, censorship-resistant protocols prevent malicious block proposers from only ordering those transactions from which they can extract profits. Protocols like [22], [23] rely on encrypted transactions or reputation to defend unfair censorship. However, attackers can still employ a client IP attack to censor specific transactions. Besides, there are some works that try to guard against malicious leaders. Protocols [24], [25] employ a rotating leader or periodic leader to defend against malicious leaders, while works [26]–[28] select leader randomly to avoid consecutive malicious leaders. However, the current leader still can manipulate transaction ordering without any detection although the election or rotation is normal. In summary, the above protocols lack a fair ordering pattern and a mechanism for detection regarding transactions in a block.

Order fairness protocols. To better prevent order manipulations, many recent works strive to design novel consensus mechanisms that achieve transaction order fairness. Specifically, order fairness requires transactions to be executed and committed in the order same as the majority of replicas' receiving orders. Intuitively, order fairness prevents order manipulations because transaction order is no longer decided by malicious replicas. Wendy [9] introduces an analogous notion called time-relative-fairness, but its fairness essentially provides no guarantees since it requires synchronized clocks while the honest clocks may be different locally. Pompē [5] employs median timestamps of transactions by all replicas to achieve fair order under synchronized clocks. However, a Byzantine leader is now able to censor a specific transaction from being delivered and manipulate the timestamps of some transactions. To avoid this, there are some protocols like [7], [29] construct a directed graph to provide ordering correctness of transactions, but they cannot ensure the transactions to be output finally, satisfying only weak liveness. Themis [6] leverages a deferred ordering mechanism to provide stronger liveness than Aequitas [7] but overlooks the significant performance overhead. Rashnu [8] inherits the fair ordering and deferred methodology from Themis to only determine the order of transactions with data dependence, by which Rashnu can reduce the cost of directed graph building and sorting. However, all deferred ordering mechanisms adopted by the above protocols require that transactions have to wait for updating messages to confirm a fair order in multiple instances of consensus. As shown in Fig. 2, deferring a transaction to be ordered and committed will lead to large confirmation latency.

III. System and Threat Model

A. System Model

Auncel is composed of a fixed number of replicas, one of which is appointed the leader. The actual method of leader selection is orthogonal to our work. Each replica has a unique identity. The other replicas in Auncel are responsible for proposing their own transactions and their orders, verifying the signature and transaction order from the leader, and assuring the order fairness of transactions. The leader takes the job of collecting signatures and local orders from replicas, as well as proposing transactions order and constructing a fairness block.

Auncel proceeds in rounds/instances. Each round of consensus will create a new block. Similar to previous consensus protocols [4], [6], [30], [31], we assume a partially synchronous network, where a message can be delivered to all honest nodes within an unknown time-bound after some Global Stabilization Time (GST) [32]. Moreover, Auncel assumes the existence of digital signatures and a Public Key Infrastructure (PKI) [33]. Additionally, we use a collision-resistant hash function $D(\cdot)$ to map the message m to a fixed-size digest $D(m)$.

B. Threat Model

In Auncel, there are n replicas that are divided into two types: honest and malicious replicas. Honest replicas abide

by all protocols in Auncel while malicious (i.e., byzantine) replicas may violate the protocols in arbitrary manners, such as tampering, forgery, and interception of the transaction orders or messages. We assume f out of n replicas are malicious replicas. Auncel is similar to other order fairness consensus [6]–[8], requiring $n \geq \frac{4f}{2\gamma-1} + 1$ (see § V for detailed proof) to ensure order fairness in a partially synchronous network where replica fairness parameter γ denotes the fraction of replicas receiving transactions in a particular order.

IV. System Design

A. System Overview

Auncel is a novel consensus protocol that can achieve order fairness and high performance. The key components of Auncel are briefly described as follows.

First, Auncel integrates order fairness to consensus design to provide censorship resistance, consisting of two mechanisms: (i) *Versatile View Change (VVC) Mechanism* that enables the consensus to launch normally and incorporates a novel function that allows replicas to propose transactions in advance. (ii) *Consistent Sequential Consensus (CSC) Framework* that cooperates with verification to guarantee all transactions and the global fair order reach consistency between all replicas. The details of fairness consensus are depicted in § IV-B.

Second, Auncel proposes a weight-based order fairness algorithm, which is conducive to reducing the cost brought by generating an ordering graph and eliminating ordering cycles. The workflow contains two phases: (i) *Weight Order Phase* obtains a linear weight sequence to denote transaction order, through transforming the transaction locations into specific weights. (ii) *Final Order Phase* further determines all positions of the weight sequence to obtain the final fair weight sequence of all transactions. The detailed strategy of order fairness is presented in § IV-C.

Finally, Auncel presents design refinements to reduce communication overheads brought by fairly ordering, which include two approaches. (i) We design a dynamic transaction compression mechanism to optimize and simplify transactions and local ordering, thereby reducing the communication overhead of consensus. (ii) We devise an adjustable replica proposal strategy to batch more transactions within a consensus instance, making full use of the network bandwidth. § IV-D elaborates on the design refinements in detail.

B. Fairness Consensus Design

1) Versatile View Change (VVC) Mechanism. At the beginning of each round or when the leader goes wrong, a view change is executed to select a primary replica to begin a new view. Our view-change mechanism not only inherits the features and methods of existing view-change mechanisms but also elevates itself by incorporating the functionality to articulate the intention of replicas on transactions as well as their orderings. In the phase of view change, all replicas are responsible for proposing New-View messages, each of which contains a view number and some quorum certificate (QC) according to the threshold signature scheme [34], such as the

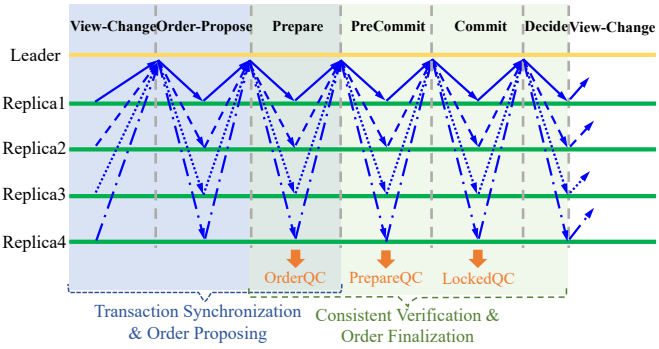


Fig. 3. Consistent and Fair Consensus Framework

highest SortQC and PrepareQC in the state tree of themselves, as described in the CSC framework (to be elaborated below). In addition, all replicas are required to construct a specified number of transaction lists based on their own transaction pools. The replicas then package these transaction lists into New-View messages and send them along with the messages to the leader. Furthermore, the novel view change mechanism can adopt a pipeline structure efficiently to confirm multiple batches of transactions with fair order in parallel. Conversely, existing representative ordering-fairness mechanisms either are not compatible with the consensus pipeline or cannot leverage the pipeline efficiently since they may be blocked for a long time at one phase if there is no sufficient information regarding the transactions and their order.

2) Consistent Sequential Consensus (CSC) Framework. Building upon the VVC mechanism described above, we then design a CSC framework as illustrated in Fig. 3, where $n = 5$ and $f = 1$. This mechanism enables the order fairness of all transactions within a round in the event of successful consensus execution, thereby ensuring consistency and liveness, without the need for additional round information or delays to supplement ordering messages. Details of these successive consensus phases are elaborated as follows.

- **View-Change:** The details of the View-Change function are given in the VVC Mechanism above.
- **Order-Propose:** The leader receives $(n - f)$ New-View messages and calculates the highest QC, referred to as HighQC, from the received OrderQCs and PrepareQCs, which can be used to build blocks and recover the consensus at the responding height conveniently like HotStuff [4] when the consensus crushes down. If successful, OrderQC and PrepareQC will have the same height. Simultaneously, the transaction lists received from replicas are verified and aggregated by the leader as the set of transactions denoted by curTxs, which need to be determined by replicas in this round. HighQC and curTxs are then encapsulated as `Tx-Sync` message and forwarded to replicas; Upon receiving the `Tx-Sync` message, replicas verify the message and check whether their transactions are lost in curTxs. If more than $(f+1)$ replicas propose a missing vote for a certain transaction, it indicates malicious behavior done by the leader. Otherwise, they build their own local orderings denoted by $\mathrm{LocalOrderings}$ along with the correct vote

as a `Sort-Vote` message, which is sent to the leader. In this case, the leader can exhibit a degree of transaction selection flexibility, such as choosing transactions that occur more than $(n-f)$ times, thereby avoiding malicious replicas from sending large transaction lists (potentially containing corrupt or fake transactions) to exhaust the bandwidth of the leader.

- **Prepare:** Upon receiving $(n-f)$ `Sort-Vote` messages, the leader collects the signatures and combines them with the signatures of the transaction set into a new signature. This merged signature is then formed as OrderQC. Furthermore, the leader consolidates all the LocalOrderings to form an OrderingMap and forwards them along with the merged signature as `Prepare` message. Replicas validate the OrderingMap, sign the `Prepare` message, and update the local OrderQC accordingly after receiving the `Prepare` message. Then, they send a `Prepare-Vote` message to the leader.
- **Precommit:** Upon receiving $(n-f)$ `Prepare-Vote` messages, the leader collects the signatures and combines them into PrepareQC, which is stored locally. Then, the leader broadcasts this Prepare QC as part of the `Precommit` message to the replicas. Upon receiving the `Precommit` message, replicas sign it and update the PrepareQC accordingly. Subsequently, replicas reply `Precommit-Vote` message to the leader.
- **Commit:** Upon receiving $(n-f)$ `Precommit-Vote` messages, the leader collects the signatures and combines them as the `PrecommitQC` just like the `Precommit` phase. Then, the leader broadcasts this `PrecommitQC` as `Commit` message to the replicas. Upon receiving the `Commit` message, replicas sign it and update the LockedQC to the `PrecommitQC`. Subsequently, `Commit-Vote` message will be sent to the leader. The system will remain locked at this stage. To speed up consensus confirmation, the leader and replicas can compute the transaction order in advance with the ordering-fairness strategy, so they can quickly commit blocks as soon as they receive enough `Commit-Vote` messages and `Decide` message respectively.
- **Decide:** Upon receiving $(n-f)$ `Commit-Vote` messages, the leader combines the signatures to CommitQC, which are then broadcast as `Decide` message to all replicas. Meanwhile, it leverages the ordering-fairness strategy to determine the order of transactions if it has not done so already. Upon receiving the `Decide` message, replicas conduct the final ordering of transactions with the ordering-fairness strategy if they have not finish the fair order. Then, a new round begins.

C. Ordering-Fairness Strategy

Definition 1 (Equal-Order-Fairness)**.** *All nodes can propose their own transaction ordering and their influence on the final transaction ordering is equal. and the transaction ordering satisfies to the following rule: given $\gamma > \frac{1}{2}$, if at least γ nodes receive transaction tx_1 before tx_2, then transaction tx_2 is placed no later than tx_1 in the final transaction ordering.*

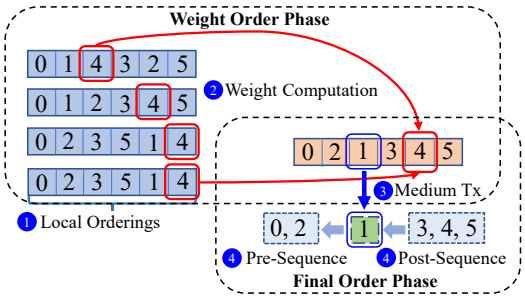


Fig. 4. Two-Phase Ordering of Fair Ordering Strategy

1) Weight Assignment. In order to satisfy the definition of fairness as well as eliminate the impact of cycles in the transaction order dependency graph shown in existing works, we avoid using graph-based sorting and adopt an ascending yet progressively diminishing weight to represent the positions of transactions within each replica's local ordering, thereby mitigating the influence of transaction position distance on sorting. In this way, the likelihood of replica's manipulation is reduced and the linear nature of the final transaction order is intuitively reflected. All transaction locations are converted into a sequence table of weight values which is used for determining fair order.

2) Two-Phase Ordering. Based on the weight assignment, leaders and replicas are responsible for employing two-phase ordering to handle local orderings and completely achieve or validate the linear transaction ordering with fairness. By doing so, they can construct and submit a block within one consensus round that determines the order of all transactions fairly. To elaborate on this mechanism, we present an example of two-phase ordering, as shown in Fig. 4.

- **Weight Order Phase:** The weight order phase aims to obtain a fairer transaction ordering sequence based on the linearization of weight values, which helps the final order phase reduce the number of compared transactions, thereby accelerating the sorting process. Firstly, a weighted formula $(W = 1-k^d)$ agreed by all replicas is determined for the LocalOrderings to assign weights to transaction ordering, where $k \in (0,1)$ represents the weight fairness parameter to reflect the degree of fairness in this stage, and $d \in \mathbb{N}$ represents the position parameter, indicating the specific position of each transaction within the local order of a replica. This results in an ascending yet progressively diminishing weight, which is advantageous in mitigating the influence of malicious nodes exploiting the parameter d, leading to a relatively fair transaction ordering. For instance, we set $k = \frac{1}{2}$ here. Secondly, leaders and replicas transform each transaction in the Local Orderings into weight values based on the weight formula, *e.g.*, we perform a Weight Computation on the tx_4 and get weight values $\{\frac{7}{8}, \frac{31}{32}, \frac{63}{64}, \frac{63}{64}\}$. Finally, leaders and replicas calculate the sum of these weight values for each transaction, *e.g.*, the weighted sum of tx_4 is $\frac{122}{32}$. This process constructs a weighted sum for each transaction relative to its position in the local ordering. If two transactions have

the same weight sum, the transactions are then sorted in ascending order based on the weight sums. Then they are further sorted in ascending order based on the transaction ID. As a result, it will yield a linearized transaction list, *e.g.*, $\{0, 2, 1, 3, 4, 5\}$, in relation to the original positions of all transactions.

- **Final Order Phase:** The final order phase sequentially examines and determines the final position of each transaction according to the fairness and achieves a deterministic fair transaction ordering, based on the linearized transaction list obtained from the weight order phase. Firstly, leaders and replicas select the Medium Tx from the linearized transaction list, *e.g.*, tx_1 in Fig. 4. According to the fairness definition, they compare the Medium Tx (tx_1) with the other transactions according to their relative positions in the local orderings. This results in two transaction sequences: the Pre-Sequence $\{0, 2\}$, consisting of transactions preceding the Medium Tx, and the Post-Sequence $\{3, 4, 5\}$, consisting of transactions succeeding the Medium Tx. Secondly, the sorting process mentioned above is recursively applied to the Pre-Sequence $\{0, 2\}$ and Post-Sequence $\{3, 4, 5\}$ until each sub-sequence contains only one transaction. Finally, all these sub-sequences are merged together, resulting in the final deterministic and fair linear transaction order.

D. Design Refinements

1) Dynamic Transaction Compression Mechanism. In the aforementioned design of our order fairness strategy, sufficient orders between transactions are conducive to eliminating ordering cycles and constructing a fair order of transactions within one instance of consensus. However, it requires substantial local orders from replicas, thereby augmenting communication overheads [35], [36]. Therefore, we devise a dynamic transaction compression mechanism to effectively compress transactions of each consensus round, thus reducing the communication overhead of consensus and simplifying the process of transaction verification.

First and foremost, in each round, the leader needs to construct a unique dynamic transaction hash and binary transaction ID mapping table based on the received transactions that require consensus. Secondly, this table is forwarded to all replicas, enabling them to easily verify the existence of transactions and convert between transaction hashes and IDs. The construction of this mapping table involves compressing each 256-bit transaction hash into incrementally increasing binary IDs with only a few bits in size. This approach greatly simplifies and compresses local orderings, particularly when there are a large number of replicas and transactions, thereby leading to a noticeable reduction in communication overhead. Finally, when the leader constructs the final block and replicas perform tasks like cleaning the transaction pool, they can utilize the dynamic mapping table for transaction mapping, ensuring a convenient and lossless restoration process.

2) Adjustable Replica Proposal Strategy. In the above design of fair consensus, all replicas are responsible for proposing their own transactions and local ordering, which helps to enhance order fairness. It raises the problem that the network bandwidth increases while there is no augmentation in the number of transaction committed within one consensus round [30], thus increasing the latency and compromising throughput. To solve this problem, we develop Auncel with an adjustable replica proposal strategy to increase the number of transaction committed in one instance of consensus.

The main idea of the strategy is to leverage the parallelism of consensus to increase the consensus commission number of transactions by enhancing the proportion of distinct transactions proposed by all replicas in each round without affecting latency, thereby achieving a relatively higher transaction throughput. Firstly, during the consensus's initial round, the transactions proposed by each replica in the View-Change phase remain identical. Subsequently, in the following round, the leader and replicas can ascertain the replicas with the highest transaction ratio in the New-View message of the current round, based on the transactions in the final consensus block. Finally, they can then adjust the transaction number proposed by each replica in the View-Change stage of the subsequent round, in accordance with their transaction ratios. For instance, nodes with greater transaction ratios will propose a higher number of transactions in the upcoming round. This approach elevates the count of distinct transactions within each round, thereby boosting the transaction throughput of the consensus.

V. ANALYSIS

In this section, we present a theoretical analysis of the order fairness, censorship resistance, and liveness of Auncel. Some arguments are inspired by Themis [6].

Lemma 1. *In a partially synchronous network, the order of transactions is possibly fair only if* $n \geq \frac{4f}{2\gamma-1} + 1$*, where* n *is the number of replicas,* f *is the number of malicious replicas, and* γ *is the fraction of replicas receiving transactions in a particular order.*

Proof. Since γ is the fraction of replicas receiving transactions in a particular order, we decide a specific transaction can rely on γn replicas. In addition, f replicas might be faulty due to the partially synchronous network, so the fair order can rely on $(\gamma n - f)$ replicas. However, f replicas may be malicious replicas, i.e., only $(\gamma n - 2f)$ replicas are honest definitely. To determine the order fairness of two transactions, a majority of replicas must reach an agreement of the same order, that is $\gamma n - 2f \geq \frac{n}{2} + 1$, namely $n \geq \frac{4f}{2\gamma-1} + 1$. □

Theorem 1. *Auncel guarantees censorship resistance and order fairness.*

Proof. Since $n \geq \frac{4f}{2\gamma-1} + 1$, $\gamma n - 2f \geq \frac{n}{2} + 1$, implying that a majority of replicas' proposals must be honest. For example, a majority of replicas in Auncel propose a transaction tx_1 and transaction tx_2 during View-Change. The leader cannot censor these transactions by delaying or dropping them since the majority of replicas will verify them. In addition, there will

be a majority of replicas propose local orderings in which transaction tx_1 is placed before transaction tx_2 through the `Tx-Sync` and `Sort-Vote`, and the order will finally reach an agreement in Auncel. Since Auncel determined these transactions' order in the Weight Order Phase satisfied the notion of majority, it will achieve order fairness. □

Theorem 2. *Auncel guarantees liveness.*

Proof. In a partially synchronous network, correct client transactions tx_1 and tx_2 will be received by all replicas. In `Tx-Sync` phase of each consensus, there are at least $(n-2f)$ replicas receiving tx_1 and tx_2. For example, $\gamma n - 2f \geq \frac{n}{2} + 1$ replicas propose tx_1 before tx_2 in their local orderings through a phase of `Tx-Sync` and `Tx-Vote` (as shown in Lemma 1). Since the order of these two transactions is agreed upon by the majority of replicas, these orders can be guaranteed in Two-Phase Ordering and can be committed with a consensus instance, with no need for further delay or considering any transaction that has not been proposed. Therefore, Auncel ensures liveness. □

VI. Evaluation

A. Implementation and Settings

We implement a prototype of Auncel bootstrapped from the HotStuff protocol [4] on top of its open-source libhotstuff codebase for performance evaluation. We compare our Auncel with the following existing schemes: 1) HotStuff without any fairness guarantee [4]; 2) the state-of-the-art fairness solutions: Themis [6] and its variant protocol Rashnu [8], both of which are implemented on top of HotStuff [4]. Details about them are elaborated on § II.

We conduct our experiments on two servers, each of which is equipped with Intel Xeon Gold 5320 2.20GHz CPUs (26 cores each) and 128GB RAM. These servers have 100 Mb/s bandwidth and connect with each other through HTTP and WebSocket. The set of transactions submitted to the system during each round is generated in advance and follows Zipfian distribution [37]. We conduct our evaluation under the SmallBank benchmark in terms of throughput and latency. We also conduct simulations leveraging Linux Netem, which can simulate the performance of implemented systems in a geo-distributed environment. We want to answer the following questions through our experiments and simulations:

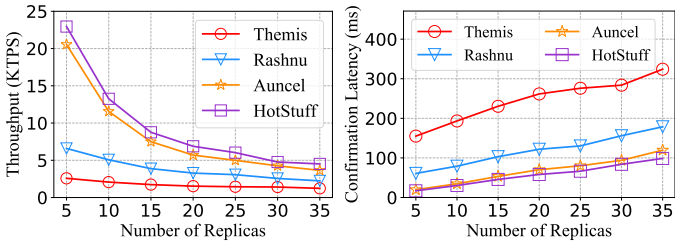
§ VI-B: Can Auncel achieve excellent performance under different network environments?

§ VI-C: Can Auncel achieve excellent performance when ordering large amounts of transactions?

§ VI-D: Can Auncel achieve the same fairness guarantee as other systems?

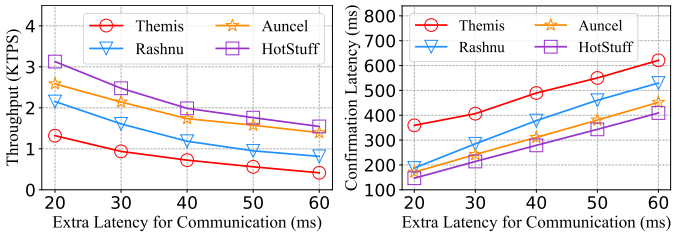
B. Performance in different network environments

Network scale. We measure the throughput and confirmation latency of the aforementioned four consensus mechanisms with different network scale. The block size is 100 transactions. The network scale increases from 5 replicas to 35 replicas. Fig. 5 shows that the throughput of all four consensus schemes decreases while their confirmation latency increases as the network scale increases. Among all the compared fairness schemes, our Auncel achieves the highest throughput and lowest latency, which is very close to the performance of HotStuff. When there are 5 replicas in the network, the throughput of Auncel is 20,545 transactions per second (TPS), which is about 3× higher than Rashnu and 8× higher than Themis. When the network scale increases, the latency of Themis and Rashnu significantly increases, further enlarging the gap with Auncel. For instance, at a replica number of 35, our latency is 119.535 ms, whereas Themis reaches 324.060 ms, almost doubling the gap observed at 5 replicas. This is because that Auncel avoids multi-step updates adopted by Themis, hence avoiding the corresponding communication overheads, which is more significant in a larger network. Although Rashnu attempts to mitigate the cost of graph building and sorting by reducing the number of transactions sorted in each round, its reliance on the graph-based sorting algorithm still incurs high computational costs associated with cycle detection and elimination, consequently resulting in considerable latency.



(a) Throughput vs network scale (b) Latency vs network scale

Fig. 5. Throughput and latency under varying network scale



(a) Throughput vs injected latency (b) Latency vs injected latency

Fig. 6. Throughput and latency with varying injected latency

Communication latency. We measure the performance of the implemented systems in a more distributed environment through injecting additional latency to the network. Fig. 6 depicts the throughput and latency with varying extra injected latency in simulations. The throughput of all protocols decreases and the latency increases when the extra latency increases from 20 ms to 60 ms. However, the change of the performance of Auncel is not as pronounced as that of Themis and Rashnu. For example, the confirmation latency of Themis and Rashnu both increase about 300 ms as the extra latency increase from 20 ms to 60 ms, while that of Auncel increases about 240 ms. This is because that the additional latency significantly prolongs the time consumed by update operations of Themis and Rashnu while Auncel has no additional updates or other communication-intensive operations.

Fairness parameter. We then evaluate the impact of replica

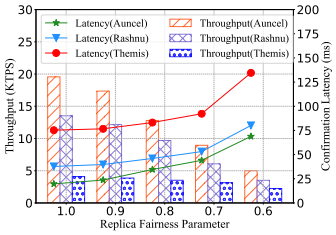


Fig. 7. Throughput & Latency vs replica fairness parameter

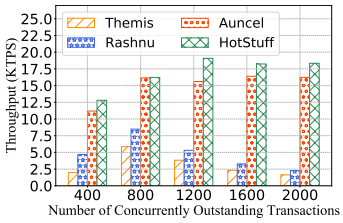
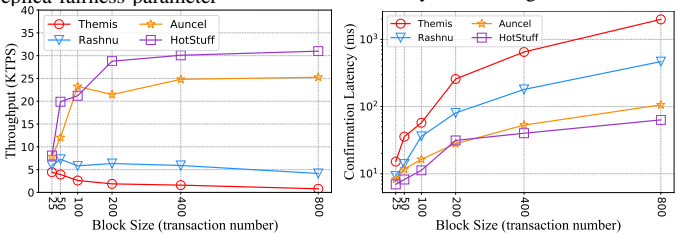


Fig. 8. Throughput vs number of concurrently outstanding transactions



(a) Throughput vs block size (b) Confirmation Latency vs block size

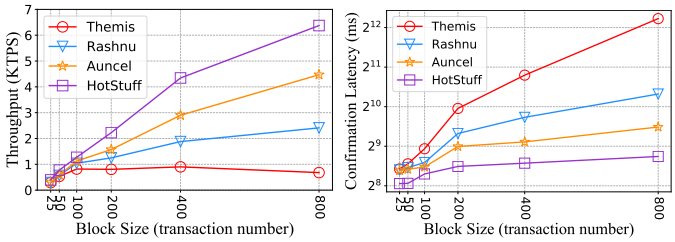
Fig. 9. Throughput and latency with varying block sizes

fairness parameter γ on the performance of the three order fairness approaches. γ influences the number of byzantine nodes and indirectly affecting the consensus process. For a fair comparison, we select the same replica fairness parameters for three systems and compare their performance under the same settings, with $\gamma \in \{1.0, 0.9, 0.8, 0.7, 0.6\}$. As we have elaborated in § V, the network size is related to γ, i.e., $n = \frac{4f}{2\gamma-1} + 1$, which means stronger fairness requirement demands larger network. As depicted in Fig. 7, reducing the fairness parameter and enlarging the network scale decreases the overall throughput and increases the latency due to the increased communication overheads between replicas, which is similar to our observation in Fig. 5. Since Rashnu and Themis require the update operations to finalize the order and hence introducing additional latency, Auncel can achieve better performance through avoiding such overheads.

Concurrently outstanding transactions. We measure the impact of the number of concurrently outstanding transactions (max-async), i.e., the number of transactions not yet committed at a time, on performance. In simulations, we set the network size to 10 and block size to 100. Max-async ranges from 400 to 2000. As depicted in Fig. 8, the throughput of all systems increase as max-async raises at first. This is because that replicas can process the increased number of transactions in parallel, thereby contributing to the increased overall throughput. However, with the increasing max-async, the throughput of Themis and Rashnu then decreases while that of Auncel remains stable, slightly lower than HotStuff. This is because that when max-asyns increases, different replicas will propose more diverse local orders, which decreases the performance of the ordering algorithm of Themis and Rashnu. Meanwhile, when the block size and network scale remain constant, the processing capability of Auncel will finally reach its maximum, thus resulting in the stable throughput.

C. Performance under different block settings

Block size. Fig. 9 depicts the performance of different systems with the increased block size. The network scale is

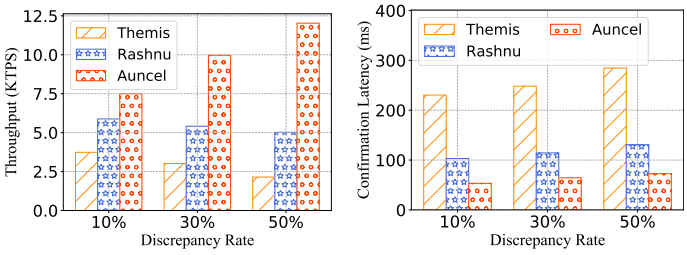


(a) Throughput vs block size (b) Confirmation Latency vs block size

Fig. 10. Throughput and latency in various block sizes in a simulated setting

5. The block size doubles each time, increasing from 25 to 800 transactions. Fig. 9(a) shows a similar trend between the throughput of Auncel and HotStuff, which increase as the block size raises. Conversely, Rashnu and Themis exhibit a declining trend in throughput, significantly lower than Auncel. When each block contains 800 transactions, Auncel achieves a throughput of 25,268 TPS, while Rashnu and Themis only obtain 4,152 and 793 TPS, respectively. This discrepancy arises because the number of transactions for fair sorting also increases when the block size increases. Our sorting algorithm boasts a low complexity when sorting transactions, without the need to detect complex circles. Meanwhile, Rashnu and Themis are fundamentally unable to avoid the expensive update operations caused by the increasing number of transactions, resulting in severe degradation of performance. It is worth mentioning that Auncel's throughput surpasses HotStuff's when the block size is 100, thanks to the efficacy of the proposed VVC mechanism (as shown in § IV-B) and Adjustable Replica Proposal Strategy (as shown in § IV-D), enabling replicas to propose and submit more transactions per round. However, in subsequent cases, Auncel's throughput diminishes even below HotStuff's due to the increased overhead in transaction sorting brought about by larger block sizes. Fig. 9(b) plots confirmation delay versus the varied block size. Our Auncel avoids frequent update operations and hence achieves the lowest latency among all fair consensus protocols, even with a large block size. The increment of the latency of Auncel is slower than that of Rashnu and Themis. When the block size is 800, the latency of Auncel is only 106.083ms, which is approximately $\frac{1}{19}$ of Themis' latency (1,987ms) in the same scenario.

We also measure the impact of varied block sizes while injecting an extra 60 ms latency to the underlying network through simulations. The results are depicted in Fig. 10. Compared with the experiments in Fig. 9 without extra latency, the system performance change differently as the block size increases. The throughput of Auncel and HotStuff improve more rapidly compared with Rashnu, while Themis's throughput first rises and then descends. It is worth mentioning that the throughput of Auncel and Rashnu continuously increase in the simulated environment while remaining stable in the experiments depicted in Fig. 9(a). This is because that when the network has higher communication latency, more proportion of the confirmation latency is caused by communication between nodes compared with that caused by sorting transactions. As a result, a larger block can increase the proportion of compu-



(a) Throughput vs discrepancy rate (b) Latency vs discrepancy rate

Fig. 11. Throughput and latency under various discrepancy rate

tation compared with communication and can lead to higher throughput. In contrast, large block size may not be beneficial to throughput since computation overheads become higher than communication overheads when the underlying latency is low, as demonstrated in Fig. 9(a). The update operations of Themis and Rashnu highly rely on communication between replicas, and Rashnu reduces the overheads by reducing transactions to be ordered. As a result, Rashnu achieves slightly better performance than Themis while both of them suffer from huge overheads. In comparison, Auncel achieves better performance. When the block size is 800, Auncel achieves $2\times$ and $7\times$ higher throughput and lower latency compared with Rashnu and Themis, respectively.

Discrepancy rate. We evaluate the impact of the degree of discrepancy between sets of transactions received by different replicas, namely *discrepancy rate*. The discrepancy rate is calculated by comparing the set of transactions proposed by each replica with that proposed by a specific replica. The network scale is 15 and each replica will propose a batch of transactions from the transaction pool with discrepancy rates of 10%, 30%, and 50%. Fig. 11 depicts the performance of these fairness consensus schemes with the increased discrepancy rate. Fig. 11(a) shows that Auncel outperforms Themis and Rashnu in terms of throughput. When the discrepancy rate increases, Auncel's throughput shows an upward trend while Themis and Rashnu's throughput declines. This is because a large discrepancy rate requires Themis and Rashunu to conduct deferred updates for fair sorting. Conversely, Auncel incorporates the Adjustable Replica Proposal Strategy (as depicted in § IV-D) that allows replicas to propose and submit more transactions per consensus, leading to the increased throughput. Fig. 11(b) shows the latency of these fairness consensus schemes increases as the discrepancy rise. The reason is that deferred updates in Themis and Rashnu prolongs the time while Auncel requires slightly more time to process more transactions.

D. Fairness comparison

Firstly, We measure the degree of order fairness of our two-phase ordering compared with Themis which can achieve order fairness. We employ the Rank Biased Overlap (RBO) methodology [38] to measure the similarity between two transaction orderings so as the degree of order fairness is quantified. RBO is achieved by calculating the intersection size of corresponding sets at varying depths to derive the similarity. A value of 1 for RBO signifies complete equality between two ranking lists,

TABLE II
CONFIRMATION LATENCY AND FAIRNESS COMPARISON WITH/WITHOUT THE SECOND PHASE OF AUNCEL'S ORDER FAIRNESS

Block Size	W/O Final Order				W Final Order
	Latency	RBO between Two-Phase			Latency
		$k=\frac{1}{2}$	$k=\frac{1}{4}$	$k=\frac{1}{8}$	
100	14.201	0.987	0.988	0.989	16.409
200	24.127	0.985	0.987	0.987	28.039
300	34.924	0.982	0.983	0.985	40.356
400	45.964	0.978	0.980	0.983	52.998

[1] Block size is the number of transaction, Latency is confirmation latency measured in milliseconds(ms), k represents weight fairness parameter.

with a higher value close to 1 indicating higher similarity. We compute and compare the RBO values under the block size varying from 100 to 400 with Themis only to assess the order of identical transaction set. We conduct each set of experiments 50 times and the results reveal that the RBO values are consistently 1 since we both share a consistent concept of fairness in transaction ordering.

Secondly, we evaluate the effect of weight fairness parameter k on the fairness between two phases of our fair ordering strategy and conduct each set of experiments 20 times. The result under various block sizes shown in Table II. We observe that the average values of RBO are close to 1 under different block sizes, indicating these two transaction sequences are highly similar in the real network. Though the sequences are not completely identical, the similarity is remarkably high, with only a few transactions undergoing position changes. The value of RBO gradually increases as k decreases. This is because the reduction in k weakens the influence of position, significantly enhancing the fairness degree of the first phase.

In summary, we conclude that the first phase of our ordering-fairness strategy can achieves a high level of fairness. Besides, we conduct experiments to evaluate the confirmation latency with and without the Final Order Phase of our consensus. The result, as shown in Table II, indicates that under these block sizes, the reduced latency accounts for approximately 13% of the total latency. This suggests that we can enhance the overall performance of fairness consensus by using only the first phase (weight order phase) if a small fraction of the fairness loss can be tolerated.

VII. CONCLUSION

This paper presents Auncel, a novel consensus protocol that can preserve order fairness and high performance. We design a weight-based sorting algorithm to allow blocks to be committed efficiently. We also integrate the ordering algorithm with the consensus protocol to avoid censorship attacks. Evaluation results show that Auncel can increase the throughput by $6\times$ and reduce the confirmation latency by $3\times$ compared with state-of-the-art protocols while preserving the similar degree of fairness. In the future work, we plan to study constant-size SNARKs on order fairness in a real world-spanning system with more large-scale nodes and further optimize communication and enhance performance.

REFERENCES

[1] P. Daian, S. Goldfeder, T. Kell, Y. Li, X. Zhao, I. Bentov, L. Breidenbach, and A. Juels, "Flash boys 2.0: Frontrunning, transaction reordering, and consensus instability in decentralized exchanges," in *IEEE SP*, 2020, pp. 910–927.

[2] L. Zhou, K. Qin, C. F. Torres, D. V. Le, and A. Gervais, "High-frequency trading on decentralized on-chain exchanges," in *IEEE SP*, 2021, pp. 428–445.

[3] K. Qin, L. Zhou, and A. Gervais, "Quantifying blockchain extractable value: How dark is the forest?" in *IEEE SP*, 2022, pp. 198–214.

[4] M. Yin, D. Malkhi, M. K. Reiter, G. G. Gueta, and I. Abraham, "Hotstuff: Bft consensus with linearity and responsiveness," in *ACM PODC*, 2019, pp. 347–356.

[5] Y. Zhang, S. Setty, Q. Chen, L. Zhou, and L. Alvisi, "Byzantine ordered consensus without byzantine oligarchy," in *OSDI*, 2020, pp. 633–649.

[6] M. Kelkar, S. Deb, S. Long, A. Juels, and S. Kannan, "Themis: Fast, strong order-fairness in byzantine consensus," in *ACM CCS*, 2023, pp. 475–489.

[7] M. Kelkar, F. Zhang, S. Goldfeder, and A. Juels, "Order-fairness for byzantine consensus," in *CRYPTO*, 2020, pp. 451–480.

[8] M. J. Amiri, H. Nagda, S. P. Singhal, and B. T. Loo, "Rashnu: Data-dependent order-fairness." [Online]. Available: https://www.seas.upenn.edu/~mjamiri/papers/rashnu.pdf

[9] K. Kursawe, "Wendy grows up: More order fairness," in *FC Workshops*, 2021, pp. 191–196.

[10] G. Wang, L. Cai, F. Gai, and J. Niu, "Phalanx: A practical byzantine ordered consensus protocol," *arXiv preprint arXiv:2209.08512*, 2022.

[11] J. Zhang, W. Chen, S. Luo, T. Gong, Z. Hong, and A. Kate, "Front-running attack in sharded blockchains and fair cross-shard consensus," *NDSS*, 2024.

[12] R. Zhang, D. Zhang, Q. Wang, S. Wu, J. Xie, and B. Preneel, "Nc-max: Breaking the security-performance tradeoff in nakamoto consensus," in *NDSS*, 2022, pp. 1–18.

[13] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. De Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich *et al.*, "Hyperledger fabric: a distributed operating system for permissioned blockchains," in *EuroSys*, 2018, pp. 1–15.

[14] Q. Wang and R. Li, "A weak consensus algorithm and its application to high-performance blockchain," in *IEEE INFOCOM*, 2021, pp. 1–10.

[15] X. Qi, "S-store: A scalable data store towards permissioned blockchain sharding," in *IEEE INFOCOM*, 2022, pp. 1978–1987.

[16] J. Zhang, Z. Hong, X. Qiu, Y. Zhan, S. Guo, and W. Chen, "Skychain: A deep reinforcement learning-empowered dynamic blockchain sharding system," in *ICPP*, 2020, pp. 1–11.

[17] Z. Hong, S. Guo, P. Li, and W. Chen, "Pyramid: A layered sharding blockchain system," in *IEEE INFOCOM*, 2021, pp. 1–10.

[18] H. Huang, X. Peng, J. Zhan, S. Zhang, Y. Lin, Z. Zheng, and S. Guo, "Brokerchain: A cross-shard blockchain protocol for account/balance-based state sharding," in *IEEE INFOCOM*, 2022, pp. 1968–1977.

[19] Z. Cai, J. Liang, W. Chen, Z. Hong, H.-N. Dai, J. Zhang, and Z. Zheng, "Benzene: Scaling blockchain with cooperation-based sharding," *IEEE TPDS*, vol. 34, no. 2, pp. 639–654, 2022.

[20] Z. Hong, S. Guo, E. Zhou, J. Zhang, W. Chen, J. Liang, J. Zhang, and A. Zomaya, "Prophet: Conflict-free sharding blockchain via byzantine-tolerant deterministic ordering," in *IEEE INFOCOM*, 2023, pp. 1–10.

[21] R. McLaughlin, C. Kruegel, and G. Vigna, "A large scale study of the ethereum arbitrage ecosystem," in *USENIX Security*, 2023.

[22] A. Asayag, G. Cohen, I. Grayevsky, M. Leshkowitz, O. Rottenstreich, R. Tamari, and D. Yakira, "A fair consensus protocol for transaction ordering," in *IEEE ICNP*, 2018, pp. 55–65.

[23] L. Heimbach, E. Schertenleib, and R. Wattenhofer, "The potential of self-regulation for front-running prevention on dexes," *arXiv preprint arXiv:2306.05756*, 2023.

[24] A. Yakovenko, "Solana: A new architecture for a high performance blockchain v0. 8.13," *Whitepaper*, 2018.

[25] M. Baudet, A. Ching, A. Chursin, G. Danezis, F. Garillot, Z. Li, D. Malkhi, O. Naor, D. Perelman, and A. Sonnino, "State machine replication in the libra blockchain," *The Libra Assn., Tech. Rep*, vol. 1, no. 1, 2019.

[26] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich, "Algorand: Scaling byzantine agreements for cryptocurrencies," in *SOSP*, 2017, pp. 51–68.

[27] K. Lev-Ari, A. Spiegelman, I. Keidar, and D. Malkhi, "Fairledger: A fair blockchain protocol for financial institutions," *arXiv preprint arXiv:1906.03819*, 2019.

[28] R. Pass and E. Shi, "Hybrid consensus: Efficient consensus in the permissionless model," in *DISC*, 2017, p. 6.

[29] C. Cachin, J. Mićić, N. Steinhauer, and L. Zanolini, "Quick order fairness," in *FC*, 2022, pp. 316–333.

[30] T. Crain, C. Natoli, and V. Gramoli, "Red belly: A secure, fair and scalable open blockchain," in *IEEE SP*, 2021, pp. 466–483.

[31] W. Chen, Z. Yang, J. Zhang, J. Liang, Q. Sun, and F. Zhou, "Enhancing blockchain performance via on-chain and off-chain collaboration," in *ICSOC*, 2023, pp. 393–408.

[32] C. Dwork, N. Lynch, and L. Stockmeyer, "Consensus in the presence of partial synchrony," *Journal of the ACM (JACM)*, vol. 35, no. 2, pp. 288–323, 1988.

[33] J. Buchmann, E. Karatsiolis, A. Wiesmaier, and E. Karatsiolis, *Introduction to public key infrastructures*. Springer, 2013, vol. 36.

[34] V. Shoup, "Practical threshold signatures," in *EUROCRYPT*, 2000, pp. 207–220.

[35] Y. Han, C. Li, P. Li, M. Wu, D. Zhou, and F. Long, "Shrec: Bandwidth-efficient transaction relay in high-throughput blockchain systems," in *ACM SOCC*, 2020, pp. 238–252.

[36] V. B. Mišić, J. Mišić, and X. Chang, "Reducing the number of transaction messages in bitcoin," *Peer-to-Peer Networking and Applications*, vol. 15, no. 1, pp. 768–782, 2022.

[37] C. Tullo and J. Hurford, "Modelling zipfian distributions in language," in *Proceedings of language evolution and computation workshop/course at ESSLLI*, 2003, pp. 62–75.

[38] A. Sarica, A. Quattrone, and A. Quattrone, "Introducing the rank-biased overlap as similarity measure for feature importance in explainable machine learning: A case study on parkinson's disease," in *International Conference on Brain Informatics*. Springer, 2022, pp. 129–139.