Themis: Fast, Strong Order-Fairness in Byzantine Consensus

Mahimna Kelkar[1,2] Soubhik Deb[†3] Sishan Long[†1,2] Ari Juels[1,2] Sreeram Kannan[3]

[1]Cornell Tech
[2]Cornell University
[3]University of Washington, Seattle

Abstract

We introduce Themis, a scheme for introducing *fair ordering* of transactions into (permissioned) Byzantine consensus protocols with at most f faulty nodes among $n \geq 4f + 1$. Themis is the first such scheme to achieve (optimistic) linear communication complexity. At the same time, it enforces the strongest notion of fair ordering proposed to date. Themis also achieves standard liveness, rather than the weaker notion of previous work.

We show experimentally that Themis can be integrated into state-of-the-art consensus protocols with minimal modification or performance overhead. Additionally, we introduce a suite of experiments of general interest for evaluating the practical strength of various notions of fair ordering and the resilience of fair-ordering protocols to adversarial manipulation. We use this suite of experiments to show that the notion of fair ordering enforced by Themis is significantly stronger in theory and for realistic workloads than those of competing systems.

We believe Themis offers strong practical protection against many types of transaction-ordering attacks—such as front-running and back-running—that are currently impacting commonly used smart contract systems.

[†]Equal second-author contribution.

Contents

1 Introduction

Decentralized Finance (DeFi), meaning the deployment of financial instruments on blockchains, has attracted substantial interest in recent years, with over 85 billion USD locked in Ethereum DeFi as of August 2021 [11]. Unfortunately, while DeFi continues to gain popularity, a long line of work [10, 13, 20, 31, 41] has shown the rise of adversaries extracting profit by manipulating the ordering and inclusion of transactions in DeFi applications. In decentralized exchanges and lending contracts, for example, where transaction execution order is critically important, such *order manipulation* results in attackers profiting at the expense of ordinary users.

Order manipulation is possible in existing protocols largely because the formal properties required of state machine replication (SMR) or consensus—the primitive that underpins blockchains—place *no restriction on how transactions are ordered*. Neither consistency nor liveness, the two pillars of consensus security, enforces any relationship between the order in which transactions arrive in the network and their final ordering in the log. Indeed, in both permissioned consensus protocols, e.g., PBFT [8] and Hotstuff [39], and permissionless ones, e.g., Ethereum, the current "leader" fully controls the inclusion and ordering of transactions within a block that it creates.

To address this gap in traditional consensus research, a recent line of work [18, 19, 21, 22, 40] proposes consensus protocols with so-called *fair ordering* properties—properties that prevent adversarial manipulation of transaction ordering. These works propose several definitions of *fairness*[1] along with protocols that realize them. These notions are different and in many cases stronger than past ordering properties such as causal ordering [6, 32] which only prevents reordering of transactions based purely on their content and fails to account for a range of attacks, e.g., those based on transaction metadata leakage or prioritizing adversarial transactions over others (e.g., to get the best purchase price for an asset [28]). The recent line of work on fair ordering attempts to tackle transaction ordering at a more fundamental level; notably, [18, 19, 40] all found exciting connections of the fair ordering problem to social choice theory.

Existing proposed fair-ordering protocols, however, have serious limitations. The protocol in [19] has impractically high $\mathcal{O}(n^3)$ communication complexity, while for [22, 40] it is $\mathcal{O}(n^2)$ which is suboptimal. Moreover, as we show in this work, there are subtle censorship issues in [40] and the notion of fairness in [22, 40] is potentially weaker in both theory and practice than that of [19]. Further, [22, 40] also depend on the existence of synchronized clocks among the protocol nodes, which can introduce significant additional overhead. We include a few key comparison points in Table 1.

1.1 Themis: An efficient fair-ordering protocol

We introduce a new fair-ordering protocol called Themis that overcomes these limitations of previous work. Themis operates with a committee of n nodes of which at most f may be arbitrarily adversarial where $n \geq 4f + 1$; it is designed to run in the partially synchronous network model. In contrast to previous order-fair protocols, which require $\mathcal{O}(n^2)$ communication [22, 40] or $\mathcal{O}(n^3)$ communication [19] even in the optimistic case, Themis is the first order-fair protocol that achieves *linear communication*, i.e., $\mathcal{O}(n)$, in the optimistic case. This is optimal and equivalent to state-of-the-art consensus protocols that lack any fair ordering guarantees, e.g., Hotstuff [39].

[1] Throughout this paper, we use "fairness" to mean *fairness of transaction ordering* or *fair ordering*, although the term fairness has been used in the past for unrelated notions (e.g., fair PoW mining [30]).

Protocol	Corruption	Synchronized Clocks?	Comm. Complexity		Transaction Ordering	Liveness	Censorship Resistance
			Optimistic	Worst			
Aequitas [19]	$n \geq 4f+1$	**No**	$\mathcal{O}(n^3)$	$\mathcal{O}(n^3)$	**Batch-order-fairness** (Definition 3.1)	Weak	**Yes**
Wendy [22]	$\boldsymbol{n \geq 3f+1}$	Yes(1)	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$	Timed-Relative-Fairness(2) (Section 6.1)	**Standard**	**Yes**
Pompē [40]	$\boldsymbol{n \geq 3f+1}$	Yes	$\mathcal{O}(n^2)$(3)	$\mathcal{O}(n^2)$(3)	Ordering Linearizability(2) (Section 6.1)	**Standard**	No
Themis (This Work)	$n \geq 4f+1$ (4)	**No**	$\boldsymbol{\mathcal{O}(n)}$	$\boldsymbol{\mathcal{O}(nf)}$	**Batch-order-fairness**(5)	**Standard**	**Yes**

Table 1: Comparison of Themis to existing fair ordering protocols. An entry in **green** indicates that it is the best property among the protocols that we compare to. (1) While the core Wendy protocol does not require synchronized clocks, if honest local clocks are far apart (e.g., each an hour apart), then their fairness definition will never apply, and therefore the system essentially provide no guarantees; (2) We define *fair separability* in Definition 6.1 which corresponds to timed-relative-fairness used in [22]. The ordering linearizability definition from Pompē is exactly the same as fair separability but only pertains to transaction pairs that are output by the protocol; (3) The design of Pompē can be modified to take advantage of general-purpose SNARKs; this results in a communication complexity of $\mathcal{O}(n \log n)$ in the optimistic case and $\mathcal{O}(n \log n + nf)$ in the worst case. (4) Themis can also work when $n \geq 3f+1$ if only crash-faults are present or if censorship is allowed; (5) In Themis, transactions in a "batch" are output contiguously rather than at the same time.

As our $\mathcal{O}(n)$ Themis protocol is somewhat computationally intensive, we also introduce a $\mathcal{O}(n^2)$ protocol that may be potentially more scalable in practice, just as Hotstuff itself is implemented as a practical $\mathcal{O}(n^2)$ protocol the original codebase [25] as well as in LibraBFT [2]. We emphasize that even our $\mathcal{O}(n^2)$ protocol provides significant advantages over existing protocols; Themis satisfies a much stronger fairness property compared to existing $\mathcal{O}(n^2)$ fair-ordering protocols and also removes the reliance on synchronized clocks.

We show that integration of this variant of Themis into Hotstuff results in acceptable performance overhead in practice; our implementation was still able to achieve a latency of roughly 13ms and a peak throughput of 43,622 transactions per second, which should suffice for most applications.

As we now explain, Themis satisfies the strong fairness property of [19] and achieves standard liveness, while [19] only achieves a notion called weak-liveness.

Themis: Fair-ordering property. Themis achieves the *batch-order-fairness* property proposed by Kelkar et al. [19]. Informally, batch-order-fairness[2] with parameter $\frac{1}{2} < \gamma \leq 1$ dictates that if γ fraction of nodes receive a transaction tx before tx$'$ from the client, then tx should be ordered no later than tx$'$ in the final output. (While [19] also introduced a stronger receive-order-fairness property for which tx must be ordered strictly before tx$'$, the authors show that it is impossible to satisfy in the general case without strong synchrony assumptions.)

Batches arise as a result of non-transitive Condorcet cycles in message receipt times across

[2]Here, *batch* is unrelated to the standard SMR optimization of increasing throughput (at the cost of latency) by amortizing consensus over many transactions.

nodes [9] (see Section 3.1 for more details). As shown by [19], the "no later than" condition of batch-order-fairness is required only when Condorcet cycles arise and in that sense represents a minimal relaxation of receive-order-fairness. To bolster this point, we show in this work that in practice, Condorcet cycles arise infrequently and tend to be quite small in reasonable network settings. Consequently, the final transaction ordering output by Themis should be very close to that guaranteed by the strong receive-order-fairness notion. This is because Themis guarantees that for any two subsequent transactions tx_j and tx_{j+1} in the final output, it holds that tx_j was received before tx_{j+1} by at least $n(1-\gamma)+1$ honest nodes.

We find that the batch-order-fairness property supported in Themis is stronger in practice than other ordering notions proposed in Zhang et al. [40] and Kursawe [22]. One of our contributions is an extensive set of experiments comparing these different notions and demonstrating the practical security advantage achieved by batch-order-fairness in adversarial order-manipulation attacks.

Themis design. Themis can be bootstrapped from any leader-based consensus protocol. For concreteness, we choose to build on Hotstuff [39], adding to it to the batch-order-fairness property. Themis maintains the basic structure of Hotstuff, with three modifications. First, before constructing a block, all replicas send information about the order in which they received client transactions to the current leader. Second, we specify an algorithm for an honest leader to construct a fair proposal from the replica orderings. Finally, we provide a way for the replicas to verify the fairness of a leader's proposal as well as extract out the final ordering.

For the fair proposal algorithm, we extract out key techniques from [19] for ordering transactions (much as [18] does). Unfortunately, applying these techniques naïvely results in a loss of liveness, similar to the original Aequitas protocol from [19] which achieves only a weaker notion of liveness due to the possible "chaining" of Condorcet cycles (see Section 3.1 for details). Concretely, in Themis, loss of liveness would mean empty blocks being produced by several honest leaders until the "chaining" of the current Condorcet cycle is completed.

Our solution is a new technique that we call *deferred ordering*. With deferred ordering, blocks produced by a leader contain some transactions that are fully ordered, while other transactions are only partially ordered. Partially ordered transactions await total ordering by a subsequent honest leader. Notably, the finalization of these partially ordered transactions *happens within the network delay* and does not have to wait indefinitely for the ordering of future transactions, e.g., the presence of Condorcet cycles. This feature of Themis allows us to circumvent the liveness problem of Aequitas. Thanks to deferred ordering, Themis achieves the standard liveness property. Deferred ordering is a technique of general interest: It can, e.g., be retrofitted to [19] to achieve standard liveness there too.

Suite of fair-ordering experiments. We study the performance of Themis experimentally. There exists no prior work on practical measurement or empirical comparison of the strength of fair ordering protocols. A key contribution of our work, therefore, is a general, systematically conceived suite of experiments to quantify the practical impact of both fairness definitions and protocols and understand their design tradeoffs. We use this suite here to study batch-order-fairness and Themis and compare them with alternatives, and believe it will be useful for any future work on fair ordering protocols.

In particular, we propose two kinds of experiments: The first attempts to understand the strength and robustness of a given fair ordering property in an ideal honest setting (with no adversarial nodes).

The second experiment considers the resilience of fair ordering techniques in adversarial settings. For this, we consider three broad class of adversarial attacks, which are commonly seen in practice and evaluate how different protocols fare against them. Specifically, this includes the following: (1) Network layer insertions — attempting to insert malicious transactions at the network layer (i.e., even before the consensus begins) through e.g., frontrunning; (2) Censorship — attempting to censor specific user transactions; and (3) Reordering — attempting to reorder transactions compared to the "honest" execution.

Experimental results. We run our suite of experiments on the fair-ordering definitions proposed by Kelkar et al. [19], Zhang et al. [40], Kursawe [22]. As a highlight, we find that the fairness notions from [19] (even for the strictest parameter choice of $\gamma = 1$) perform better in practice compared to the notions from [22, 40] both in ideal and adversarial settings. Additionally, we discuss potential censorship threats in existing protocols.

1.2 Contributions

Our contributions in this work are as follows:

- Themis*:* We introduce Themis, the first partially synchronous fair-ordering protocol with optimistic *linear* communication complexity. Themis realizes the strong fairness property of [19], but unlike [19] achieves practical performance.
- *Liveness:* We introduce the technique of *deferred ordering*, which enables us to achieve standard liveness in Themis (as opposed to the weak liveness of [19]) and is of general interest. Additionally, we show subtle censorship concerns existing fair ordering protocols.
- *Experimental suite:* We introduce an experimental suite of general interest for the study of fair-ordering protocols. Using this suite, we run experiments showing that the fairness property of Themis is stronger in practice than alternative notions and the occurrence of Condorcet cycles is likely to be relatively rare.

2 Preliminaries and Model

Model. Our setup is a permissioned system with a set $\mathcal{N}$ of n known protocol *nodes* or *replicas*, of which at most f are controlled by an adversary (denoted by $\mathcal{A}$) and can deviate arbitrarily from the protocol description. Transactions to be sequenced are sent by system clients to all replicas. For our fair ordering protocols, we will consider the times at which a transaction was received by the replicas to decide on its overall ordering in the final ledger. For communication between replicas, we assume the presence of a PKI, and the security of digital signatures. The network itself is partially synchronous [12]; specifically, there exists a network delay Δ that bounds the message delivery time between replicas, but is not known to the replicas. $\mathcal{A}$ controls all message delivery, and can delay and reorder messages up to the bound Δ.

Arguments of knowledge. We make use of generic arguments of knowledge—specifically, SNARKs [3, 16] for NP languages—in a black-box way to verify computation. For a language L in NP with witness relation $\mathcal{R}_L$, a SNARK for $\mathcal{R}_L$ is a tuple of efficient algorithms $(\mathsf{Gen}, \mathsf{Prove}, \mathsf{Verify})$ where Gen generates the trusted parameters pp (e.g., the CRS) given 1^λ where λ is the security parameter

for the argument system, Prove is the prover algorithm, and Verify is the verifier algorithm. Given pp and any $(x, w) \in \mathcal{R}_L$, $\mathsf{Prove}(pp, x, w)$ produces a proof π attesting that $x \in L$. The proof can be checked using $\mathsf{Verify}(pp, x, \pi)$. We require the standard completeness, soundness, and knowledge properties for SNARKs. For (asymptotic) efficiency of our protocol, the SNARKs need to have constant-sized proofs excluding a poly-log factor in the security parameter (e.g., [16]). We do not require any zero-knowledge property.

Graph terminology and algorithms. We make use of common graph algorithms to reason about the ordering dependencies between transactions. We therefore start by recalling standard graph terminology. $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ denotes a graph with vertex set $\mathcal{V}$ and edge set $\mathcal{E}$. Unless specified, all graphs will be directed and unweighted. We often use vertices and transactions interchangeably when referring to graphs where vertices represent transactions. We call $\mathcal{G}$ a *tournament* graph if there is exactly one edge between each pair of vertices.

For $V \in \mathcal{V}$, $\mathsf{SCC}_{\mathcal{G}}(V)$ denotes the strongly connected component that contains V. The subscript $\mathcal{G}$ can be dropped when the context is clear. Recall that a strongly connected component is a maximal subgraph such that there is a path in each direction between each pair of vertices in the component. $\mathcal{G}^*$ denotes the condensation of $\mathcal{G}$ (i.e., the transformed graph where vertices in the same SCC are collapsed into a single vertex). Note that $\mathcal{G}^*$ is guaranteed to be acyclic.

Within a graph, a Hamiltonian path is a path (i.e., a sequence of vertices) that visits each vertex exactly once. A Hamiltonian cycle is a Hamiltonian path that forms a cycle, i.e., there is also an edge from the last vertex to the first vertex in the path. For an acyclic graph $\mathcal{G}$, a topological sorting is a linear ordering of vertices such that for any vertices U and V, U is ordered before V if $(U, V) \in \mathcal{G}.\mathcal{E}$.

Given a graph $\mathcal{G}$, its condensation can be easily computed in time $\mathcal{O}(|\mathcal{V}| + |\mathcal{E}|)$ using depth-first search techniques [36]. Moreover, if $\mathcal{G}$ is acyclic, then it can also be topologically sorted with the same asymptotic complexity. While the problem of detecting Hamiltonian paths in generic graphs is NP-complete, for acyclic graphs in particular, it is solvable in time $\mathcal{O}(|\mathcal{V}| + |\mathcal{E}|)$ (through topological sorting). This is also true for tournament graphs [27].

Fair-ordering consensus protocols. Standard SMR or consensus protocols are required to satisfy two properties: consistency and liveness. Informally, consistency ensures that all replicas eventually agree on the same transaction ledger, while liveness guarantees that new transactions proposed by clients are incorporated into the ledger quickly.

Motivated by practical use cases where transaction ordering is of critical importance, recent works [19, 22, 40] have proposed adding a third property that captures a notion of fairness in transaction ordering. While a variety of properties have been proposed, at a high level, this style of fairness seeks to guarantee a specific ordering in the finalized ledger based on how transactions arrive into the network.

In this work, we will primarily focus on the fairness desiderata proposed by Kelkar et al. [19]. We justify our choice in Section 6 by illustrating through comprehensive network simulations why this fairness notion is better suited to handle natural order-manipulation attacks than other related properties. Through these experiments, we provide a detailed comparison to the the protocols proposed in [19, 22, 40].

We describe the techniques of Kelkar et al. [19] in Section 3.1.

3 Building Blocks

3.1 Aequitas Technique

Our starting point is the work of Kelkar et al. [19], which explored several variants of fair transaction ordering along with the assumptions necessary to realize them. All definitions were parameterized by an order-fairness parameter γ bounded by $\frac{1}{2} < \gamma \leq 1$, which intuitively represents the fraction of replicas in the system that ensures a definitive transaction ordering. Specifically, the system as a whole is guaranteed to decide on a particular global ordering of transactions tx and tx$'$ if some γ-fraction of replicas all received the transactions in a specific order. The primary notion considered, and subsequently realized by their leaderless Aequitas protocol, was batch-order-fairness. In this paper, we will be mainly concerned with this notion, which is as follows:

Definition 3.1 (γ-batch-order-fairness)**.** Suppose that transactions tx and tx$'$ are received by all protocol nodes. If γn nodes received tx before tx$'$ locally, then all honest nodes deliver tx no later than tx$'$.

A stronger notion, receive-order-fairness (exactly as Definition 3.1, but now, tx must be delivered before tx$'$), was also considered in [19]. An impossibility result, however, rules out realization of that notion except in specific synchronous settings. The impossibility result arises from the Condorcet paradox [9] in voting theory. Abstractly, this allows for non-transitive global preferences even if each party's local preferences are transitive.[3] Consequently, the global "fair" transaction ordering (according to receive-order-fairness) could contain cycles, which we refer to as Condorcet cycles. Batch-order-fairness sidesteps this by enabling transaction delivery in batches (transactions can still be totally ordered within a batch), and subsequently ignoring unfairness resulting from any cyclic orderings in the same batch.

Aequitas protocol. The Aequitas protocol from [19] consists of three stages that each transaction goes through before being delivered to the ledger. First, in the *gossip* stage, all nodes broadcast transactions in the order they were locally received from clients. The FiFo broadcast primitive [17] is used to ensure that the broadcast order of an honest sender is maintained. Next, in the *agreement* stage, a variant of Byzantine Agreement [23] is used to agree on whose local orderings to use to order a particular transaction. Lastly, the *finalization* stage is used to non-interactively determine the final transaction ordering. Given the local orderings of other nodes, the finalization algorithm builds a dependency graph of transactions as they arrive. Here, edges represent ordering dependencies between transactions (e.g., an edge from tx to tx$'$ signifies that a large number of nodes have received tx earlier). Disregarding the complexity introduced by the graph being built at different rates by different nodes, the core algorithm removes transactions from the graph and *delivers* them when, informally, they no longer have any dependencies.

Aequitas realizes batch-order-fairness and circumvents the impossibility result by delivering transactions in the same Condorcet cycle at the same time (i.e., in the same "batch"). Further, Aequitas guarantees that batch-relaxation is minimal in the sense that if no Condorcet cycles are present, each batch includes only a single transaction (i.e., the stronger receive-order-fairness notion will be met). In this paper, we will show the practical significance of this feature by showing that

[3] As a simple example, suppose that three nodes receive transactions in the order $[a, b, c]$, $[b, c, a]$ and $[c, a, b]$. Here, each of "a before b", "b before c", and "c before a" holds for a majority of nodes, resulting in a non-transitive global preference.

under standard network models, Condorcet cycles tend to arise infrequently and are typically of small size.

Condorcet cycles and weak-liveness. A crucial problem with the Aequitas protocol, however, is that it achieves only a weak notion of liveness. Once again, the existence of Condorcet cycles proves problematic here. Kelkar et al. [19] note the possibility for Condorcet cycles to "chain" together, and extend for an arbitrarily long time and even include transactions input later (this happens because of specific input transaction orderings and not necessarily the adversary). In other words, specific input orderings could prevent the delivery of transactions for arbitrarily long. To provide intuition for, and confirm this perhaps surprising observation from [19], we show an explicit construction of such an input ordering in Appendix A. The Aequitas protocol, therefore, can only guarantee liveness after the current chain completes, which may take arbitrarily long in the worst case.

3.2 Evading the Weak-Liveness Problem

Potential Condorcet cycles. Since some local orderings can be adversarial, we need to account for potential Condorcet cycles, which are also implicitly accounted for in Kelkar et al. [19]. We formally define them below as they lead to a key piece of our protocol design.

Definition 3.2 ((Potential) Condorcet Cycle)**.** A list $[\mathrm{tx}_1, \ldots, \mathrm{tx}_l]$ is a potential Condorcet cycle of length l if the following holds: For all $i \in \{1, \ldots, l\}$, where $\mathrm{tx}_1 = \mathrm{tx}_{l+1}$, at least $n(1-\gamma)+1$ *honest* nodes have received tx_{i+1} before tx_i.

Further, for each i, if at least $\gamma n - f$ nodes "claim" to have received tx_{i+1} before tx_i (honest nodes claim the order in which they receive transactions while adversarial nodes can claim arbitrary orderings), then $[\mathrm{tx}_1, \ldots, \mathrm{tx}_l]$ will be form a Condorcet cycle.

For potential cycles, note that when tx' is received before tx by at least $n(1-\gamma)+1$ honest nodes, the order-fairness definition does not guarantee that tx is ordered before tx', which means that the two transactions could potentially end up in the same Condorcet cycle. We emphasize however that not all potential cycles will end up being actual cycles. For the actual cycle to be formed, for each edge in the cycle, at least $\gamma n - f$ nodes should claim to have received that ordering.

Now, we observe that while transactions in the same Condorcet cycle are delivered at the same time by Aequitas, in many practical applications, some total ordering will need to be imposed for transaction execution. Here, Aequitas encourages choosing a deterministic (e.g., alphabetical) ordering for transactions in the same cycle. However, since cycles can chain, it may take an unknown amount of time to identify all the transactions that should be part of the current cycle.

To avoid this problem, we make a crucial observation: in most scenarios, it should be reasonable to output transactions in the same cycle *contiguously*, even potentially in successive blocks rather than together. This means that a part of the current cycle can be output later as long as no transaction from a later cycle comes before it.

This creates a surprisingly powerful tool: in essence, it will allow us to "continue" the same cycle across consecutive consensus leaders without compromising on fairness. While identifying the end of the current cycle brings us back to the same weak-liveness problem, doing so is no longer necessary when cycles can be output contiguously. In this spirit, we introduce the technique of *deferred ordering*, which will propose a partial ordering for some transactions and defer their total

ordering to the next leader. Note that the total ordering for deferred transactions will not depend on the chaining of Condorcet cycles, and therefore, we are able to achieve standard liveness.

Notably, we can still guarantee two important fairness properties: (1) No transaction is censored; (2) The total ordering can be linearly partitioned into cycles, i.e., a transaction in a different cycle cannot be ordered between two transactions in the same cycle. Similar to earlier, if there are no (potential) Condorcet cycles, the stronger receive-order-fairness notion will still be satisfied.

3.3 Protocol Design

We describe the core design of Themis in this section. Themis can be bootstrapped from *any* leader-based consensus protocol and will endow it with our fair ordering property.

Recall that standard protocols provide the leader complete authority when creating its proposal—only the validity of transactions and not their ordering is checked by the replicas. The way Themis will achieve fairness is by providing a mechanism for replicas to also check the ordering in the proposal. To enable the leader to construct a *fair* ordering, all replicas will first submit their local transaction orderings to the leader. By *local ordering*, we mean transactions received at a given node ordered by their receive times. Without loss of generality, we will assume that the local orderings are totally ordered.

Specifically, we start by providing an algorithm used by the leader to construct a fair-ordered proposal from a set of replica local orderings. Abstractly, this algorithm will guarantee that most transactions are proposed (which is important for liveness) but at the same time, fairness is not violated even when a new leader needs to continue the ordering proposed by an earlier leader. When sending the proposal to replicas, the leader will also include a SNARK proof that the proposal algorithm was executed correctly. This will be used to guarantee the fair ordering of transactions in the proposal. We point out though, that our use of SNARKs is primarily to reduce the protocol communication complexity. If this is a non-goal, correctness can also checked by forwarding all replica local orderings to all other replicas. This alternative strategy will incur an extra $\mathcal{O}(n)$ communication factor. Section 4 contains a full description of the fairness components of our Themis design.

Our changes to the overall structure of the underlying consensus protocol are quite minimal. Notably, our design will only need to modify how the leader proposal is constructed and how its fairness is verified by the replicas. We require only a single extra communication step (to send the local orderings from the replicas to the leader) on top of the underlying protocol. Other standard consensus mechanisms (e.g., detecting leader equivocation, leader rotation etc.) will not be affected and can be plugged in as before.

While we can bootstrap from any leader-based consensus protocol, for concreteness, we will use the partially synchronous Hotstuff protocol from Yin et al. [39]. For a single (honest) leader, Hotstuff requires only $\mathcal{O}(n)$ communication; this will also be the communication complexity of our protocol. In the worst case with f cascading leader failures, the communication complexity of both protocols will be $\mathcal{O}(nf)$.

4 Fair Leader Protocol

In this section, we describe the algorithms used by an honest leader to propose a fair ordering among the transactions it receives from the replicas, as well as how fairness can be verified by the replicas.

We extract out the underlying consensus protocol and only specify the fair ordering components. Any leader-based consensus protocol can utilize these components to achieve fair ordering.

Basic structure. Recall that the technique by which we circumvent the weak-liveness problem is to allow Condorcet cycles to continue across multiple leader proposals. Specifically, we will allow a leader to propose a partial ordering, which will later be completed by a future leader without compromising on fairness. For this, we provide two algorithms, `FairPropose` and `FairUpdate`, which detail how an honest leader should construct a proposal, and how it should update a previous leader's proposal, respectively. We also describe an algorithm `FairFinalize`, that allows all nodes to extract a fair transaction ordering from *fully specified* proposals.

Local replica orderings. Before the leader constructs a proposal, each replica sends its local transaction ordering to the leader. Note that since the network is partially synchronous, the leader will need to work with only $n-f$ orderings. To enable the leader to construct a proposal, an honest replica i sends the following: (1) List_i containing the transactions received by i that are not part of any previous proposal, in the order that they were received (by i); (2) $\sigma_i(\mathsf{List}_i)$ which is i's signature on List_i. Further, to allow the leader to update previous proposals, i sends the following: (1) Update_i containing transactions from previous proposals that are not fully specified (we will elaborate on this later) in the order that they were received (by i); (2) $\sigma_i(\mathsf{Update}_i)$ which is i's signature on Update_i. Notably, our protocol only requires *orderings* from the replicas, and not individual timestamps for transactions. This means that Themis can work even without synchronized clocks.

Local order notation. For a set $\mathcal{L}$ of $n-f$ orderings, we use $\mathrm{tx} \in_k \mathcal{L}$ (resp. $\mathrm{tx} \notin_k \mathcal{L}$) to denote that tx is present in at least (resp. less than) k orderings in $\mathcal{L}$. We use $\mathrm{tx} \prec_{(\mathcal{L},k)} \mathrm{tx}'$ to denote that tx appears before tx' in at least k orderings in $\mathcal{L}$. We call tx a *solid* transaction (w.r.t. $\mathcal{L}$) if $\mathrm{tx} \in_{n-2f} \mathcal{L}$, a *shaded* transaction if $\mathrm{tx} \in_{n(1-\gamma)+f+1} \mathcal{L}$ but $\mathrm{tx} \notin_{n-2f} \mathcal{L}$, and a *blank* transaction if $\mathrm{tx} \notin_{n(1-\gamma)+f+1} \mathcal{L}$. tx is a non-blank transaction if it is either solid or shaded. We use $\mathsf{Weight}_{\mathcal{L}}(\mathrm{tx}, \mathrm{tx}')$ to denote the maximum value k such that $\mathrm{tx} \prec_{(\mathcal{L},k)} \mathrm{tx}'$.

Deferred ordering. Looking ahead, solid transactions are the ones that have been received by enough replicas to make an ordering decision. Note that both shaded and blank transactions are not present in enough replicas yet to be finalize an ordering. However, in some cases, there can be an edge from a shaded transaction tx to a solid transaction tx'. Here, to retain fair ordering, tx will be included in the leader proposal but may only be partially ordered until the time it becomes a solid transaction for a later leader's proposal (which only depends on the actual network delay). Furthermore, this also defers the ordering for tx' until then.

4.1 Overview of Leader Algorithms

We now describe the `FairPropose` and `FairUpdate` algorithms used by an honest leader.

Proposal algorithm. The algorithm `FairPropose` is used by an honest leader to construct a block proposal upon receiving a set $\mathcal{L}$ of $n-f$ orderings. Abstractly, the goal of this algorithm is to propose as many transactions as possible while making sure that the exclusion of any transaction from the proposal does not violate fairness.

The algorithm proceeds as follows: First, it constructs the dependency graph $\mathcal{G}$ by adding a vertex for each non-blank transaction. An edge $(\mathrm{tx}, \mathrm{tx}')$ is added to the graph whenever $\mathrm{tx} \prec_{(\mathcal{L},n(1-\gamma)+f+1)} \mathrm{tx}'$; only one of $(\mathrm{tx}, \mathrm{tx}')$ or $(\mathrm{tx}', \mathrm{tx})$ is added however, i.e., if both conditions are satisfied, the one

Algorithm `FairPropose`($\mathcal{L}$)

`// Propose a fair ordering for new transactions.`
On input a set $\mathcal{L}$ containing the local transaction orderings of $n - f$ nodes:

1. **Build Dependency Graph**
   - Create an empty graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$.
   - For each non-blank tx, add a vertex tx to $\mathcal{V}$.
   - For each $\text{tx}, \text{tx}' \in \mathcal{V}$, let $k = \mathsf{Weight}_{\mathcal{L}}(\text{tx}, \text{tx}')$ and $k' = \mathsf{Weight}_{\mathcal{L}}(\text{tx}', \text{tx})$. If $k \geq n(1-\gamma) + f + 1$ and $k \geq k'$ and $(\text{tx}', \text{tx}) \notin \mathcal{E}$, then add the edge (tx, tx') to $\mathcal{E}$.
   - Compute the condensation graph $\mathcal{G}^*$ of $\mathcal{G}$ and its topological sorting S.
2. **Output Fair Ordering**
   - Let V be the last vertex of S that contains a solid transaction.
   - Remove those transactions from $\mathcal{G}$, that are part of vertices after V in S.
   - Output $\mathcal{G}$.

Figure 1: Leader Proposal Algorithm

with the larger value will be added. Specifically, let $k = \mathsf{Weight}_{\mathcal{L}}(\text{tx}, \text{tx}')$ and $k' = \mathsf{Weight}_{\mathcal{L}}(\text{tx}', \text{tx})$. If both $k, k' \geq n(1-\gamma) + f + 1$, then add the edge (tx, tx') if $k > k'$ and the edge (tx', tx) if $k' > k$; If $k = k'$, then one of the two edges can be added deterministically. Note that no edges are added when both $k, k' < n(1-\gamma) + f + 1$. When this happens, an edge will be added by a later proposal.

Next, the condensation of the graph is computed and topologically sorted. Let V be the last vertex in the topological sorting that contains a solid transaction. Then, the `FairPropose` outputs the graph structure obtained by removing all transactions tx from $\mathcal{G}$ such that $\mathsf{SCC}(\text{tx})$ occurs after V in the topological sorting. The details are given in Figure 1.

Proposal properties. A proposal created using `FairPropose`($\mathcal{L}$) is guaranteed to contain all solid transactions in $\mathcal{L}$. Further, it contains no blank transactions, and exactly those shaded transactions that contain an outgoing path into a solid transaction within the dependency graph. We show in Lemma 4.2 that the exclusion of any transaction that is not proposed does not violate fairness; specifically, if tx is the last solid transaction proposed, and tx' is a transaction that is not proposed, then at least $n(1-\gamma) + 1$ honest nodes that have received tx before tx'.

Further, within the graph $\mathcal{G}$, there is at exactly one edge between any two vertices, except possibly when both vertices correspond to shaded transactions. Future leaders cannot modify existing edges but will be able to add missing edges between shaded transactions. This proposal of a "partial graph" is crucial as it allows us to avoid the weak-liveness problem. The main observation we use here is that any transaction that is not proposed will either be part of the same or later potential Condorcet cycle than any transaction that is proposed; in other words, fairness is not lost by including shaded transactions that have missing edges within the proposal. We prove this formally in Lemma 4.3.

Update algorithm. The update algorithm allows a leader to add missing edges to a previous leader's proposal graph. For this, each replica i also sends the ordering Update_i that orders *shaded* transactions from previous proposals as seen by i. The update algorithm is quite simple: upon

Algorithm `FairUpdate`($\mathcal{L}$)

`// Update the ordering for previous proposals`
On receiving a set $\mathcal{L}_{\text{updates}}$ containing the local transaction orderings of $n - f$ nodes for previously proposed shaded transactions:

1. **Output Dependencies**
   - Let $\mathcal{E}_{\text{updates}} \leftarrow \varnothing$.
   - For all tx and tx$'$ that are part of the same leader proposal and between whom, no edge has been proposed yet, let $k = \mathsf{Weight}_{\mathcal{L}_{\text{updates}}}(\text{tx}, \text{tx}')$ and $k' = \mathsf{Weight}_{\mathcal{L}_{\text{updates}}}(\text{tx}', \text{tx})$. If $k \geq n(1-\gamma) + f + 1$ and $k \geq k'$ and $(\text{tx}', \text{tx}) \notin \mathcal{E}_{\text{updates}}$, then add the edge (tx, tx') to $\mathcal{E}_{\text{updates}}$.
   - Output $\mathcal{E}_{\text{updates}}$.

Figure 2: Update Algorithm

receiving $n - f$ of such orderings from replicas, for all transactions tx and tx$'$ that were part of the same previous leader's proposal and do not currently have an edge between them, the edge (tx, tx') is added if $\text{tx} \prec_{(\mathcal{L}, n(1-\gamma)+f+1)} \text{tx}'$. Once again, only one of (tx, tx') or (tx', tx) is added however exactly as in the `FairPropose` algorithm. The new edges are specified in an update list $\mathcal{E}_{\text{updates}}$. Intuitively, this allows all nodes to use these update edges to fill in the missing details within earlier proposals in order to compute the final fair ordering. We provide details in Figure 2.

4.2 Constructing A Fair Ordering

Replica verification. We use generic arguments of knowledge to enable replicas to verify the correctness of a leader's proposal. Specifically, the leader will prove knowledge of transaction orderings from $n - f$ nodes such that when they are run through the `FairPropose` algorithm, the graph output is the one that's proposed. The knowledge of these transaction orderings can be verified through digital signatures. We do this so as to prevent the leader from needing to forward all transaction orderings and signatures to all nodes, which incurs an $\mathcal{O}(n^2)$ cost.

More formally, we can consider the language L in NP such that (x, w) is in the witness relation R_L if the following holds: x is a string representation of directed graph $\mathcal{G}$, w is parsed into $\mathsf{List}_{i_1} \parallel \cdots \parallel \mathsf{List}_{i_{n-f}} \parallel \sigma_{i_1}(\mathsf{List}_{i_1}) \parallel \cdots \parallel \sigma_{i_{n-f}}(\mathsf{List}_{i_{n-f}})$ where each $(\mathsf{List}_j, \sigma_j)$ represents a transaction ordering from a distinct node along with a signature from the node, and `FairPropose`($\mathcal{L}$) outputs $\mathcal{G}$ where $\mathcal{L} = \{\mathsf{List}_{i_1}, \ldots, \mathsf{List}_{i_{n-f}}\}$. The string representation of a graph can be decided upon deterministically, for example, by choosing vertices in alphabetical order. A well studied line of work ([16, 26] among numerous others) has shown how to construct constant size SNARKs that enable such proofs for any language in NP. We use these proofs to provide witness proofs for our language L. The same technique can be used to prove correctness of the update list $\mathcal{E}_{\text{updates}}$.

Finally, the overall proposal block B by the leader will be the tuple $(\mathcal{G}, \mathcal{E}_{\text{updates}}, \pi_1, \pi_2)$ where $\mathcal{G}$ is the graph for newly proposed transactions, $\mathcal{E}_{\text{updates}}$ is the set of edges to add to previous proposals, π_1 is the SNARK for $\mathcal{G}$, and π_2 is the SNARK for $\mathcal{E}_{\text{updates}}$. The upshot is that now, the size of B depends no longer depends on the number of nodes, enabling the underlying consensus algorithm to achieve communication complexity $\mathcal{O}(n)$.

Note that the use of SNARKs enables linear asymptotic communication complexity and is not necessary if $\mathcal{O}(n^2)$ complexity is acceptable; the leader can simply include all orderings and

Algorithm `FairFinalize()`

Given a sequence of proposals $[B_1, B_2, \ldots, B_l]$:

1. **Update Graphs**
   i. For all B_i and transactions tx, tx$'$ in B_i that do not have an edge between them, if (tx, tx') is in some $B_j.\mathcal{G}$, then add that edge to $B_i.\mathcal{G}$.
   ii. Let k be the last index such that the graph $B_k.\mathcal{G}$ is a tournament.
   iii. Compute the condensation graphs of $B_1.\mathcal{G}, \ldots, B_k.\mathcal{G}$ and their topological sortings $S_1, \ldots, S_k$.
2. **Retrieve Fair Ordering**
   i. For each $S_i = [v_{i1}, \ldots, v_{il_i}]$ where $i \in \{1, \ldots, k\}$, let H_{ij} be a Hamiltonian cycle of v_{ij} (pick one deterministically if there are multiple). Note that this exists since each v_{ij} is a strongly connected tournament.
   ii. Let tx_{il_i} be a solid transaction in each v_{il_i} (pick one deterministically if there are multiple), and let H'_{il_i} be the cyclic rotation of H_{il_i} so that the last transaction is tx_{il_i}.
   iii. The final transaction ordering for the block B_i is now given by $H_{i1}, \ldots, H_{i(l_i-1)}, H'_{il_i}$.

Figure 3: FairFinalize Algorithm

signatures within the proposal, which can in turn be checked by the replicas. In fact, SNARKs could be less practical than the $\mathcal{O}(n^2)$ variant of our protocol due to, to the best of our knowledge, the current lack of any efficient SNARKs geared towards verifying graph algorithms. A practical method to achieve $\mathcal{O}(n)$ complexity is to let the leader propose a block without a SNARK while maintaining the replica orderings locally. This can facilitate later auditing leading to potential reversal of the proposed order and/or slashing of the leader stake.

Choosing a final fair ordering. As the final step, we now describe an algorithm `FairFinalize` used by replicas to arrive at the fair ordering. This step is carried out locally by all nodes after proposal blocks have been confirmed by the underlying consensus protocol. `FairFinalize` will decide on the ordering for transactions within *fully specified* graph proposals; $\mathcal{G}$ is *fully specified* when it is a tournament graph, i.e, there is exactly one edge between any pair of vertices. An important point to highlight here is that the confirmation of transactions will depend only on the network delay and does not suffer from the weak-liveness problem present in the Aequitas protocol. This is because missing edges in a graph $\mathcal{G}$ will be added (making it fully specified) as soon as the transactions are received by the network and an honest leader is elected. As an optimization, transactions whose SCC as well as all SCCs that precede it contains only solid transactions can be finalized immediately even before the proposal graph is fully specified.

Now, given a sequence of blocks $B_1, \ldots, B_l$, the `FairFinalize` algorithm first uses the $B_j.\mathcal{E}_{\text{updates}}$ to add missing edges to earlier graphs. Let k be the largest index such that all of $B_1.\mathcal{G}$ to $B_k.\mathcal{G}$ are fully specified. Intuitively, `FairFinalize` will order all transactions within these blocks.

Consider a graph $B_j.\mathcal{G}$ where $j \leq k$. Note that each SCC in $B_j.\mathcal{G}$ is also a tournament. An old result by Camion [7] shows that all strongly connected tournaments contain a Hamiltonian cycle (a cycle that visits every vertex exactly once). Intuitively, we will use this Hamiltonian cycle to determine a total ordering within an SCC in such a way that it connects to the ordering from other SCCs. Note that although the problem of finding Hamiltonian cycles is NP-complete for general

graphs, for tournaments specifically, they can be found in time linear in the number of edges [27].

More concretely, given a graph $B.\mathcal{G}$, for each SCC C in the graph, we first find a Hamiltonian cycle H_C containing all the transactions within C. If multiple cycles exist, one of them can be chosen deterministically to be used for the total ordering. Now, let $[V_1, V_2, \ldots, V_c]$ denote the topological sorting of the condensation graph $B.\mathcal{G}^*$. Note that each of V_1 to V_c represent an SCC within $B.\mathcal{G}$. The ordering for transactions within V_1 to V_{c-1} is simply the sequence of the Hamiltonian cycles H_{V_1} to $H_{V_{c-1}}$. For the final SCC V_c, we will cyclically rotate the corresponding Hamiltonian cycle. Recall that our `FairPropose` algorithm guarantees that this contains at least one solid transaction, say tx. Now, to obtain the transaction ordering for V_c, we rotate H_{V_c} so that tx is now the final transaction among those output. This is done to ensure that the any tx' output in a later block is such there are at least $n(1-\gamma)+1$ honest nodes that have received tx before tx'.

A note on our use of Hamiltonian cycles. While we make use of Hamiltonian cycles, we note that other techniques, such as using median timestamps, or alphabetical ordering, can certainly be used to decide on the final ordering for transactions within a Condorcet cycle.

Our technique has some distinct advantages however. First, it does not rely on synchronized clocks between protocol nodes to ensure correct timestamps, since only the the local orderings (in fact, the entire Themis protocol does not rely on synchronized clocks).

Second, and perhaps more interestingly, we can use Hamiltonian cycles to cleanly "connect" the boundaries of subsequent Condorcet cycles. More specifically, if $[\text{tx}_1, \ldots, \text{tx}_l]$ is the final output ordering, then for any j, tx_j is received before tx_{j+1} by at least $n(1-\gamma)+1$ honest nodes, regardless of whether tx_j and tx_{j+1} are in the same Condorcet cycle. Notably, this is not achieved if transactions in a cycle are ordered alphabetically as suggested by Aequitas [19].

4.3 Themis Properties

We will now formally define and prove the properties satisfied by Themis in a partially-synchronous network when $n > \frac{4f}{2\gamma-1}$ where n is the number of nodes, f is the maximum number of adversarial nodes, and $\frac{1}{2} < \gamma \leq 1$ is the order-fairness parameter.

Lemma 4.1 (Dependency Graph Properties)**.** *The following properties hold for the output of the algorithm* `FairPropose(`$\mathcal{L}$`)`*: (1) It contains all solid transactions; (2) For all solid* tx *and non-blank* tx' *contained in the output, exactly one of the edges* (tx, tx') *and* (tx', tx) *is present in the output graph.*

Proof. Let $\mathcal{G}$ be the dependency graph constructed by `FairPropose(`$\mathcal{L}$`)`. It is easy to see why (1) holds since the final graph output only removes those SCCs from $\mathcal{G}$ that do not contain a single transaction that is not solid.

To prove (2), first notice that since tx is solid and tx' is non-blank, at least $n-2f > 2n(1-\gamma)+2f$ orderings in $\mathcal{L}$ contain at least one of tx or tx', i.e., order the two transactions. Therefore, at least one of $\text{tx} \prec_{\mathcal{L},n(1-\gamma)+f+1} \text{tx}'$ or $\text{tx} \prec_{\mathcal{L},n(1-\gamma)+f+1} \text{tx}'$ holds. Since only one edge is added even when both holds, we conclude that exactly one of the edges (tx, tx') and (tx', tx) is contained in the final graph output. □

Lemma 4.2. *If* `FairPropose(`$\mathcal{L}$`)` *outputs a solid transaction* tx *and does not output* tx' *(and* tx' *was not present in a previous proposal), then there are at least* $n(1-\gamma)+1$ *honest nodes that have received* tx *before* tx'*.*

Proof. We break this into two cases. First, if tx' is blank, then since tx' is present in at most $n(1-\gamma)+f$ orderings we have $\mathrm{tx} \prec_k \mathrm{tx}'$ where $k = n-2f-n(1-\gamma)-f = \gamma n - 3f > n(1-\gamma)+f$ since $n > \frac{4f}{2\gamma-1}$. Equivalently, since at most f of those are adversarial, it holds that more than $n(1-\gamma)+1$ honest nodes have received tx before tx'.

On the other hand, if tx' is shaded, since it was not output, this means that there is no path from tx' to any solid transaction in the graph. In particular, there is also no edge from tx' to tx. This implies that either $\mathsf{Weight}_{\mathcal{L}}(\mathrm{tx}', \mathrm{tx}) \leq n(1-\gamma)+f$ or $n(1-\gamma)+f+1 \leq \mathsf{Weight}_{\mathcal{L}}(\mathrm{tx}', \mathrm{tx}) \leq \mathsf{Weight}_{\mathcal{L}}(\mathrm{tx}, \mathrm{tx}')$. The second case directly implies that tx is received before tx' by at least $n(1-\gamma)+1$ *honest* nodes. For the first case, notice that since tx is solid, we have $\mathsf{Weight}_{\mathcal{L}}(\mathrm{tx}, \mathrm{tx}') \geq (n-2f) - (n(1-\gamma)+f) \geq \gamma n - 3f > n(1-\gamma)+f$ since $n > \frac{4f}{2\gamma-1}$, which completes the proof. □

Lemma 4.3 (Contiguous Cycles)**.** *If* `FairPropose(`$\mathcal{L}$`)` *outputs a transaction* tx *and does not output* tx' *(and* tx' *was not present in a previous proposal) and if* tx' *is received before* tx *by* γn *nodes, then* tx *and* tx' *are in the same potential Condorcet cycle.*

Proof. We first note that tx cannot be solid within the leader proposal since this would imply that there are at least $n(1-\gamma)+1$ honest nodes that have received tx before tx' (from Lemma 4.2), which contradicts the given condition. Therefore tx must be shaded and consequently, it will be included in the proposal only when there is a sequence of transactions $[\mathrm{tx} = \mathrm{tx}_0, \mathrm{tx}_1, \ldots, \mathrm{tx}_l]$ such that the dependency graph contains each of the edges $(\mathrm{tx}_i, \mathrm{tx}_{i+1})$ and tx_l is a solid transaction. This also means that tx_l was received before tx' by at least $n(1-\gamma)+1$ honest nodes.

Now, since tx' is received before tx by γn nodes, this directly implies that $[\mathrm{tx}_0, \mathrm{tx}_1, \ldots, \mathrm{tx}_l, \mathrm{tx}']$ is a (potential) Condorcet cycle. □

Theorem 4.4 (Themis satisfies batch-order-fairness)**.** *At any time, let* $[\mathrm{tx}_1, \ldots, \mathrm{tx}_l]$ *be the final transaction ordering output by* `FairFinalize`*. Then, there are indices* $1 = i_1, \ldots, i_k = l$ *such that for all* $j \in \{1, k-1\}$ $C_j = [\mathrm{tx}_{i_j}, \ldots, \mathrm{tx}_{i_{j+1}}]$ *is a potential Condorcet cycle and the ordering* $[C_1, \ldots, C_{l-1}]$ *satisfies batch-order-fairness.*

Further, for each $i \in \{1, l-1\}$*, it holds that* tx_i *was received before* tx_{i+1} *by at least* $n(1-\gamma)+1$ *honest nodes.*

Proof. To show batch-order-fairness, consider tx and tx' such that tx was received before tx' by at least γn nodes. Now, we follow a few cases:

- (Case 1) At some time, the current correct leader proposal includes tx and tx' is not included in this or any earlier proposal. Now, by Lemma 4.3, we know that tx' must be in the same or later Condorcet cycle as tx.
- (Case 2) At some time, the current correct leader proposal includes tx' and tx is not included in this or any earlier proposal. Since we are also given that tx was received before tx' by γn nodes, by Lemma 4.3, we know that tx' must be in the same potential Condorcet cycle as tx.
- (Case 3) At some time the current correct leader proposal includes both tx and tx'. Since tx was received before tx' by γn nodes, we know that there is an edge from tx to tx' within the dependency graph. Therefore, once again, either tx' will be output in the same potential cycle as tx or a later one.

The above cases show that the final transaction ordering can be split into contiguous potential Condorcet cycles, and consequently, it will hold that batch-order-fairness is satisfied.

Now, in the final output ordering, consider transactions tx_j and tx_{j+1} where tx_{j+1} is output immediately after tx_j. We consider the following cases:

- (Case 1) Both transactions were part of the same initial leader proposal. Now, if they were part of the same SCC, then since the output order is a Hamiltonian cycle, $(\mathrm{tx}_j, \mathrm{tx}_{j+1})$ is an edge in the dependency graph; in other words, tx_j was received before tx_{j+1} by $n(1-\gamma)+1$ honest nodes. On the other hand, if tx_j was in an earlier SCC, then $(\mathrm{tx}_j, \mathrm{tx}_{j+1})$ is already an edge. Note that since tx_j was output earlier, it cannot be in a later SCC.
- (Case 2) tx_{j+1} was proposed in the block after tx_j was proposed. This means that, in some valid leader proposal, tx_j was present but tx_{j+1} was not, and further tx_j was the last transaction in the previous proposal, which implies that tx_j was a solid transaction. Therefore, we can directly conclude the result by Lemma 4.2.
- (Case 3) Note that it not possible for tx_j to be proposed in a block after tx_{j+1} since the final ordering contains tx_j first.

□

Corollary 4.4.1. Themis *satisfies receive-order-fairness when there are no potential Condorcet cycles.*

Theorem 4.5. Themis *satisfies consistency and (standard) liveness.*

Proof. Consistency is a direct consequence of the consistency of the underlying consensus algorithm. For liveness, consider a transaction tx that has been received by all nodes, i.e., it will be present in at least $n-2f$ local replica orderings sent to the leader. In other words, tx is solid, and will be included in the leader's proposal. Note that the final ordering of tx only depends on earlier (shaded) transactions that have been proposed by the current or an earlier proposal (since this can cause edges to be missing in the graph proposal). Note that the final ordering of tx no longer depends on any transactions that are not yet proposed. This means that as soon as the edges between the previously shaded transactions are added (this will happen when the shaded transactions are received by enough nodes which is only dependent on the network delay). Consequently, Themis achieves standard liveness. □

4.4 Other Properties

Audit friendliness. The fairness properties of Themis are also quite friendly for auditing. In particular, an optimistic fast path can be used where the fairness of the leader's proposal is not checked by the replicas for ordering, but can be validated at a later time, even by an external auditor. This is of course possible only when the auditor can force a revert of the system to an earlier state and/or impose a penalty upon detection of a malicious leader proposal.

This is straightforward to achieve for Themis. The leader constructs the proposal as normal but does not need to send an proof of correctness to the replicas; it can hold on to this for any subsequent audit check.

Crash-fault tolerant protocol for $n \geq 3f+1$. Themis can also be modified in a straightforward way to create an $3f+1$ version when $\gamma = 1$, and in settings with only crash faults (instead of explicit adversarial behavior).

For this, we start by redefining solid transactions to be the ones present in all of the received $n-f$ orderings by the leader. This can be done since any replica that has not crashed will eventually include all transactions in the ordering it submits to the leader. The algorithms `FairPropose` and `FairFinalize` work exactly as before. Still, in all fairness proofs, since solid transactions are present in $n - f$ lists, only a bound of $n \geq 3f + 1$ is required. Note that the underlying consensus mechanism for agreeing on the proposals already supports $n \geq 3f + 1$.

Interestingly, the same protocol works for Byzantine faults but at the cost of allowing the leader node to censor transactions.

5 Implementation and Experiments

We ran an extensive set of experiments for Themis that we detail in this section. In addition to the standard performance benchmarks (Section 5.1) for throughput and latency, we also design a suite of fairness experiments in Section 6 which are useful in quantifying the extent of fair transaction ordering. Through these experiments, we analyze not only our protocol and fairness definitions, but also those of several related works [19, 22, 40]. We begin with an overview of our implementation below.

Implementation details. Our implementation for Themis is bootstrapped from the Hotstuff protocol [39]: a state-of-the-art leader-based protocol for the partially synchronous setting. For this, we started from the author's open-source libhotstuff codebase [25] (which implements the $\mathcal{O}(n^2)$ version of the Hotstuff protocol). Our primary code changes were having the leader generate fair transaction sequences and having the replicas validate them. We use this implementation to benchmark the performance of Themis.

5.1 Performance and Benchmarks

We start with the standard performance experiments.

Experimental setup. For our performance experiments, we consider a system with $n = 5$ nodes within the same AWS region. Each node is run on an AWS EC2 C5.4xlarge instance with 16vCPUs and 32GiB memory. In addition, we utilize separate "client" nodes to generate and transmit transactions. We consider two settings: (1) Same Region: All nodes are in the same AWS region (us-east-2); (2) Geo-distributed: Nodes are distributed across across 5 regions (us-west-1, us-east-1, ap-northeast-1, ap-northeast-2, eu-central-1) with 1 node in each region.

Latency and throughput. We report on the latency and throughput of Themis and compare it to the performance of standard Hotstuff run with our experimental setup as a baseline. The results are shown in Table 2 (single datacenter setting) and Table 3 (geo-distributed setting).

For the single datacenter setting, we use the default blocksize $= 400$ parameter set in libhotstuff [25]. We find that the latency of Themis is quite comparable to the Hotstuff. While the throughput is smaller, it is still more than 43000 tx/sec, and we expect this to be sufficient for most applications that require the fairness properties of Themis. For the geo-distributed setting, the performance of Themis is competitive with Hotstuff.

Protocol	Median Latency (ms)	Peak Throughput (txs/sec)
Hotstuff (Baseline)	12.595	214,332
Themis (Our Work)	13.212	43,622

Table 2: Performance results for Themis compared to Hotstuff as a baseline with 5 nodes in the same datacenter.

Protocol	Median Latency (ms)	Peak Throughput (txs/sec)
Hotstuff (Baseline)	3,373.345	1,400
Themis (Our Work)	3,433.579	1,382

Table 3: Performance results for Themis compared to Hotstuff as a baseline with 5 geo-distributed nodes.

We also believe that an optimized implementation of the graph algorithms that Themis requires could further push the throughput. Furthermore, the throughput can also be increases by choosing a higher blocksize parameter albeit at the cost of slightly higher latency.

6 Suite of Fair Ordering Experiments

To better analyze and compare different fair ordering definitions and protocols on an equal footing, we created a suite of experiments with accompanying simulation environments that allow fine-tuning several parameters that may influence how different fairness notions fare. In our experiments, we compare the fairness definitions from [22, 40] as well as receive-order-fairness, and batch-order-fairness, with γ being taken from $\{1, 0.8, 0.6, 0.51\}$ for the latter two definitions. We also compare Themis to existing fair ordering protocols.

The experiments we propose are as follows: First, we compare the strength of different fairness definitions in an ideal setting in Section 6.2. Next, in Section 6.3, we analyze the susceptibility to network level transaction insertions, e.g., through frontrunning attacks. Third, in Section 6.4, we compare fair ordering protocols on censorship resistance. And finally, in Section 6.5, we analyze to what extent an adversary can influence the final transaction ordering.

6.1 Preliminaries and Setup

We start by introducing the fairness notions from [22, 40].

Fair ordering definitions from [22, 40]. Zhang et al. [40] and Kursawe [22] took a different approach to defining fairness than Kelkar et al. [19]. Both works use similar definitions of fairness and therefore, we consolidate this into a notion we call *fair separability*, given in Definition 6.1.

Definition 6.1 (Fair Separability)**.** Suppose that two transactions tx and tx′ are received by all protocol nodes. If *all* honest nodes receive tx before *any* honest node received tx′, then all honest nodes deliver tx before tx′.

[22] calls this *timed-relative-fairness*. We compare to the Wendy protocol from [22] which satisfies this definition. An identical property was also stated as a desideratum in [14] although no protocols achieving this property were given.

The definition from [40] further only applies when tx and tx′ are both be delivered in the log. [40] calls this *ordering-linearizability*. Note that ordering-linearizability considers it acceptable if only tx′ if delivered and tx is not, even when the antecedent of the above definition is true, i.e., all honest nodes received tx before any honest node received tx′. Consequently, [40] satisfies Definition 6.1 only for transaction pairs that are output by the protocol. By doing so, the protocol is able to ensure that the delivery of tx′ is not held back by tx even when tx is stuck in a slow network. However, by making this tradeoff, another problem is introduced: a network adversary or even a Byzantine leader node is now able to *censor* a specific transaction from being delivered; this is potentially far more problematic, especially for our primary motivation of DeFi. We discuss this further in Section 6.4.

Simulation setup. We start with a "sending" process that generates transactions with the delays between consecutive transactions sampled from a distribution GenerationDist. It is most fruitful to analyze the ordering for transactions sent around the same time. The scenario where tx′ is sent after tx has already been ordered, for instance, is not particularly interesting. Therefore we take the approach of studying sets of temporally clustered transactions, rather than workloads of extended duration. We use sets of 1000 such transactions in our experiments.

Now, for each transaction, we simulate when the transaction would reach different consensus nodes by randomly sampling the network latency from a distribution NetworkDist. The simulation outputs the receive timestamps of the generated transactions at each node. Let $\mathsf{Send}(\mathrm{tx})$ denote the time that tx was generated and $\mathsf{Recv}(\mathit{node}, \mathrm{tx})$ denote the time that tx is received by *node*.

We instantiate each distribution using the exponential distribution, which is standard in networking literature. The rate of generation or arrival of messages is usually modeled as a Poisson process, which is equivalent to the distribution for intervals between messages (i.e., GenerationDist) being modeled as an exponential distribution. Separately, the network delay is also usually modeled as an exponential distribution.

For our experiments, we will evaluate how the relative rate of transaction generation to the network latency affects fairness. Specifically, for this, we set GenerationDist to the exponential distribution $\mathsf{Exp}(1/\mu)$ with mean $\mu = 1$, and chose NetworkDist to be *independently distributed* with mean $r\mu = r$ for a *network ratio* parameter r.[4]

The network ratio r serves in our experiments as a proxy for how far apart the consensus nodes are from one another; a small r ($\ll 1$) captures a setting where all nodes are in the same local network, while a larger r (10 to 100) is typically more reflective of a geo-distributed setting.

On the network ratio r and the fairness granularity. We highlight that even when the network delay is very large, transaction latencies with smaller r can be approximated by abstractly, setting a coarser granularity for fairness. We define granularity below.

Definition 6.2 (Fairness granularity)**.** For granularity g, we consider that timestamps are bucketed into slots of interval g time each (e.g., $[0, g)$, $[g, 2g)$ and so on). Events within a time bucket are assumed to happen at the same time.

[4]The choice of $\mu = 1$ is without loss of generality since exponential distributions satisfy the scaling property — If $X \sim \mathsf{Exp}(1/\mu)$, then $kX \sim \mathsf{Exp}(1/k\mu)$.

In particular, a granularity of g is equivalent to only quantifying fair ordering across these width g intervals, and ignoring unfairness of transactions within the same interval. Note the time intervals can be w.r.t. the local time at a given node, and synchronized clocks are not required. In other words, the granularity can be changed without affecting any protocol assumptions.

In all our experiments, we assume the finest possible granularity. (i.e., $g = \epsilon \approx 0$) since this is the most challenging setting, but we note that in some cases, a courser granularity (i.e., larger g) may be acceptable.

6.2 Comparison of Fairness Definitions

Our first set of experiments attempts to quantify to what extent a given fair ordering definition is useful *in an ideal, non-adversarial setting*. For this, we ask a simple question: How "close" is the ordering *guaranteed by* a given fair ordering definition to the ideal ordering? Note that given our simulation setup, there is a canonical "ideal" global transaction ordering, namely the order in which transactions are generated.

Once transaction receive times have been simulated, we count the number of transaction pairs $(\mathrm{tx}, \mathrm{tx}')$, $\mathsf{Send}(\mathrm{tx}) < \mathsf{Send}(\mathrm{tx}')$ where the given fair ordering definition requires that tx be ordered before tx' in the output. For example, fair separability (Definition 6.1) will order tx before tx' if all of the timestamps for tx are smaller than any of the timestamps for tx', but otherwise does not stipulate an ordering of the two transactions. For batch-order-fairness, although it only guarantees that tx is ordered no later than tx', note that tx and tx' will be placed in the same batch only when they are part of the same Condorcet cycle. Therefore, to quantify its usefulness conservatively, we will only count the transaction pairs that are *strictly* ordered in the output (e.g., end up in different Condorcet cycles).

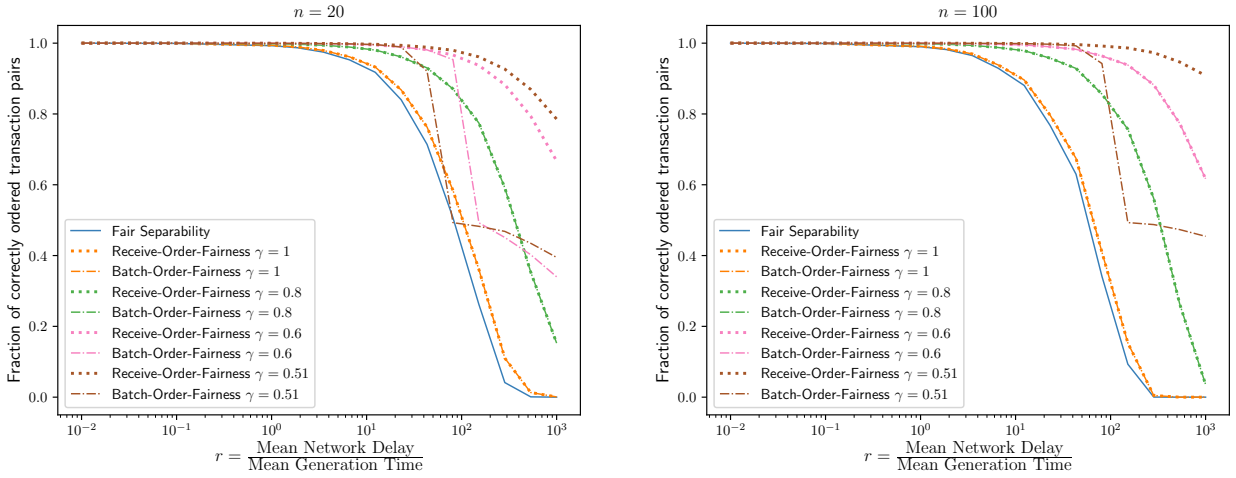
Recall that the mean generation time μ and the mean network delay is $r\mu$. Of course, as r grows (i.e., the network delay gets larger), the order of transactions received at the nodes will be vastly different than the send ordering, i.e., the number of transaction pairs ordered correctly will drop to zero. Still, the interest in this experiment is understanding how quickly this drop takes place for different definitions.

Results. Figure 4a and Figure 4b show the results for $n = 20$ and $n = 100$ respectively. For each fairness definition we consider, we compute the fraction of transaction pairs that it would order the same as the ideal ordering. We vary the network ratio r from 10^{-2} to 10^{3}.

We start with some simple observations: both graphs show that as r gets larger, the gap between send ordering and final transaction ordering grows exponentially. However, the order-fairness definitions (receive- and batch-order-fairness) ensure closer to ideal ordering than fair separability everywhere. Further, having a larger n makes each definition fare better.

Comparing receive and batch order-fairness. For $n = 100$, note that the plots for receive-order and batch-order are identical except for when $\gamma = 0.51$. Even for $\gamma = 0.51$, they are identical for a small r. When $n = 20$, the two definitions also deviate for $\gamma = 0.6$. This is because Condorcet cycles are much more common when r is large and when γn gets close to $n/2$ — when n is large, this effect is not seen until γ gets close to 0.5.

Abstractly, the deviation between the definitions can be seen as the result of the frequency with which Condorcet cycles arise in practice. Note that the presence of cycles also means that receive-order-fairness cannot be achieved in these settings. Therefore, a natural question to ask now is how commonly do cycles arise, and when they do, how large do they get?



(a) $n = 20$ (b) $n = 100$

Figure 4: Comparison of the fraction of transaction pairs correctly ordered by different fairness notions as a function of r.

Cycle size. We find the number of transactions that are present in Condorcet cycles of each size in the ideal setting. Note that the length of *potential* cycles may be much larger. In fact, even when the actual cycles are small, the potential cycles may extend arbitrarily long. In such a scenario, Aequitas will still only satisfy weak-liveness, while our deferred ordering technique allows Themis to satisfy standard liveness.

Figure 5a and Figure 5b contain results for $n \in \{20, 100\}$ and for reasonable network ratios $r \in \{1, 10\}$. For example, for a given length-k cycle, we count k transactions in the bucket for k. Note that $k = 1$ means that there is no cycle.

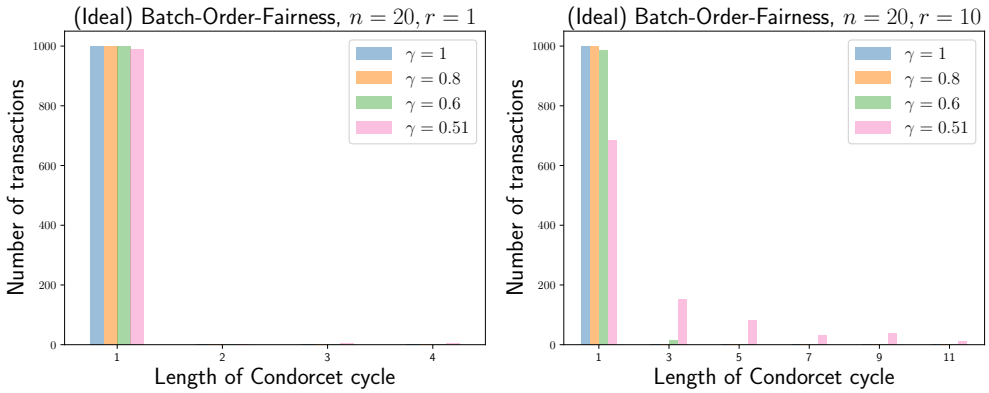
We found that for $r = 1$, there were no cycles found for $\gamma \in \{1, 0.8, 0.6\}$. For $\gamma = 0.51$, for $n = 20$, there was one 3-length cycle and one 4-length cycle, while there was just one 3-length cycle for $n = 100$.

When $r = 10$, we did not find cycles for $\gamma \in \{1, 0.8\}$. There were a few length-3 cycles for $\gamma = 0.6$ for $n = 20$ but none for $n = 100$. For $\gamma = 0.51$, cycles are more common but still a reasonable network ratio of $r = 10$, the max cycle length was 11 for $n = 20$ and 5 for $n = 100$. We also notice that it is less common to find cycles (for the same γ) as n increases due to a smaller variance.

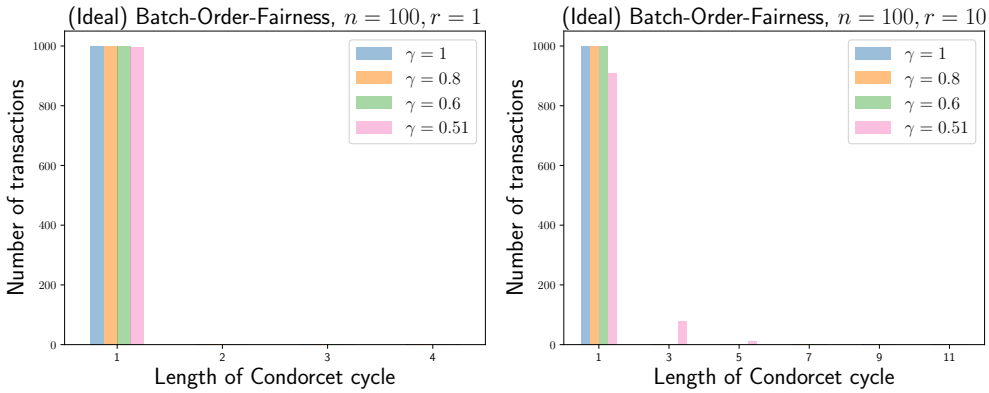
6.3 Network Level Frontrunning

In this set of experiments, we analyze how robust a fairness definition is to network-level *frontrunning*, which is arguably the core ordering issue in today's networks. We define network level frontrunning as a node Y being able to perform the following attack: Upon receiving a transaction tx from another node X, it attempts to create a new transaction tx$'$ and get it sent to other nodes in an attempt to get tx$'$ ordered before tx. Note that this frontrunning takes place before transactions are received by all honest nodes, i.e., even before the the consensus protocol begins.

While secure causal ordering or privacy techniques can help hide transaction data before ordering, thus thwarting targeted frontrunning (i.e., based on the honest user transaction content),



(a) $n = 20$



(b) $n = 100$

Figure 5: Comparison of the number of transactions in a Condorcet cycle of specific lengths

they do not help against problems such as metadata leakage, or non-targeted frontrunning (i.e., based only on the transaction existence rather than its content). Therefore, quantifying network frontrunning remains important. Notably, our experiment measures frontrunning protection even in cases when transaction privacy is not sufficient. Nevertheless, we note that all fair ordering protocols can incorporate secure causal ordering techniques as a complementary protection — e.g., as a backstop against fair-ordering failures in extreme settings such as full adversarial network control. We also emphasize that our experiment can be easily adapted to other network-level attacks like backrunning, sandwiching, and priority attacks (e.g., first in line for an ICO [28]).

Basic setup. We performed this experiment using a real network rather than a simulation to understand the possibility of frontrunning in practice. We start with a simple example geo-distributed network with one node each in five different and popular AWS regions: us-west-1 (California), us-east-2 (Ohio), ap-northeast-1 (Tokyo), ap-northeast-2 (Seoul), and eu-central-1 (Frankfurt) and measured ping timings between each pair of servers (in both directions) as a proxy for the communication latency between them. Table 4 contains ping times averaged over 1000 samples. Our goal now, is to find the number of *frontrunnable pairs* — i.e., (A, B) such that B is able to frontrun transactions originating from A. While we experimentally find the number of such pairs, we emphasize that the success of actual frontrunning attacks in practice also depends on many other factors

— for instance, the specific protocol, the expected profit of a particular frontrunning strategy, the variance in network latency, the presence of competing entities etc.

Frontrunning for fair separability. For fair separability, frontrunning is possible if there exists nodes A, B, C, D (where A, B, and C are distinct, D is different from A and B but potentially the same as C) such that $\mathsf{ping}(A, B) + \mathsf{ping}(B, C) < \mathsf{ping}(A, D)$. When this happens, notice that B can receive a transaction tx from A (working as a client) and forward its own adversarial transaction tx' to C before tx was received by all honest nodes. Here, fair separability will not apply since the antecedent is violated — there is an honest node (C) that received tx' before some other honest node (D) received tx.

For our toy AWS setting, out of the 180 possible tuples (A, B, C, D), the above property is violated by 16 of them. As a concrete example based on Table 4, an adversary in Tokyo could frontrun a user transaction from Seoul by sending its adversarial transaction to California before the transaction from Seoul reached Frankfurt. Overall, there are 10 frontrunnable pairs (A, B) such that B can frontrun transactions originating from A (i.e., there exists some C and D that satisfy the aforementioned property).

Frontrunning for receive/batch order-fairness. On the other hand, for (receive/batch)-order-fairness, a node B can possibly frontrun transactions from A if there are at least $n(1-\gamma)+f+1$ nodes C such that $\mathsf{ping}(A, B) + \mathsf{ping}(B, C) < \mathsf{ping}(A, C)$. Note that this is because, now, the transaction tx from A is no longer received before the transaction tx' by at least $\gamma n - f$ nodes. Frontrunning success is guaranteed if there are more than $\gamma n - f$ such nodes C.

Observe that each satisfying (A, B, C) corresponds to breaking triangle inequality. Therefore, the possibility of frontrunning here for (A, B) is synonymous to triangle inequality being violated in the network for a *non-trivial* (at least $f + 1$) number of nodes. We find that is quite unlikely to hold for real-world networks. In fact, within Table 4, there are no tuples (A, B, C) for which the triangle inequality is violated, i.e., there are no frontrunnable pairs in the example AWS setting. For order-fairness, we will report on both the $\gamma = 1$ case (for which triangle inequality needs to be violated $f + 1$ times for the pair to be frontrunnable), and the optimal case (for which it needs to be violated $> n/2$ times)

Analysis over a large network. While our example experiment is helpful for a basic understanding of how susceptible fairness notions are to front running, we also carried out a much larger scale analysis with data from 250 servers. For this, we used the publicly available WonderProxy dataset [38] which contains ping times (in both directions) between each pair of 250 servers averaged over 30 pings per hour over the two day period of July 19-20 2020.

Now for both $n = 20$ and $n = 100$, we randomly sample an n-sized committee from the 250 servers and once again compute the number of frontrunnable pairs. We average the results over 100 randomly chosen committees. Table 5 reports on our findings for the number of frontrunnable pairs for different fairness definitions.

For fair separability, 82% and 94% pairs on average were found frontrunable for $n = 20$ and $n = 100$ sized committees respectively. On the other hand, for order-fairness, the corresponding numbers were 4.5% and 2.8% for $\gamma = 1$ and 0.26% and 0.16% in the optimal case. Notably, frontrunning gets easier on larger networks for fair separability while it gets harder for order-fairness.

Origin \ End	us-west-1 (California)	us-east-2 (Ohio)	ap-northeast-1 (Tokyo)	ap-northeast-2 (Seoul)	eu-central-1 (Frankfurt)
us-west-1	-	51.407	105.733	133.771	147.308
us-east-2	51.166	-	131.609	159.931	98.827
ap-northeast-1	106.684	131.359	-	31.723	228.356
ap-northeast-2	134.018	159.561	32.397	-	222.829
eu-central-1	145.962	97.972	228.862	234.641	-

Table 4: Average ping times (in ms) between different AWS servers

Setting	Total Pairs	Number of frontrunnable pairs		
		Fair Separability	Receive/Batch Order-Fairness ($\gamma = 1$)	Receive/Batch Order-Fairness (Optimal)
5 AWS nodes	21	10	0	0
Random $n = 20$	380	313	14	1
Random $n = 100$	9900	9326	262	16

Table 5: Number of frontrunnable pairs for fair separability and order-fairness for several settings. The first line denotes to our example setting with 5 geo-distributed AWS nodes. For the other rows, we randomly sample n-sized committees from the Wonderproxy ping dataset [38]. We chose $n = 20, f = 4$ and $n = 100, f = 24$ for the second and third row respectively. The average number (rounded to the nearest integer) of frontrunnable pairs is given over 100 random committee samplings.

6.4 Censorship Resistance and Attacks

Censorship resistance is defined as the property that if an honest client sends a transaction to the consensus nodes, it will eventually be output by the protocol. For the fair ordering setting, since transactions are sent to all nodes (instead of just the leader), an honest transaction is defined as one that has been received by all nodes. It is desirable to guarantee the delivery of all honest transactions. However, as we see, there is an inherent tradeoff between liveness and censorship resistance for some existing fair ordering protocol designs.

Censorship attack on [40]. Pompē [40] uses a pre-protocol (specifically an ordering phase prior to consensus) to compute the median timestamp for transactions, with the fair ordering being taken as the transactions in ascending order of the median timestamp. For $n \geq 3f + 1$, as long as $(\mathrm{tx}, \mathrm{tx}')$ satisfies the antecedent of fair separability (Definition 6.1), the median of tx will be smaller than that of tx'. In other words, as long as both transactions *complete the ordering phase*, tx will be ordered earlier by the protocol; this condition is explicitly stated in the *ordering linearizability* definition used in Pompē. Unfortunately, this results in censorship resistance being guaranteed only for transactions that pass the pre-protocol phase.

For Pompē, in a partially synchronous network, an adversary can delay sending the median time-stamp for tx from the replicas long enough till tx' is delivered. If this happens, since the rest of the output is fair, tx can never be output by the protocol. Concretely, Pompē assumes that there is a known Δ_2 which is the upper bound on the time for an honest node to complete the ordering phase; they use this bound as the maximum time a transaction in the consensus phase

waits for any potential transaction from earlier to finish the ordering phase. When this bound is not satisfied, while safety is not broken, earlier transactions are now essentially censored since the consensus phase has moved on to a later timestamp even though they were timestamped during the ordering phase. In order words, to achieve censorship resistance, Pompē needs to assume a known *synchronous* bound on the time for an honest node to complete the ordering phase. Note that such a bound cannot be assumed to be known in a partially-synchronous network.

While this was partially acknowledged in [40] as a possible way a Byzantine node could selectively chose which of its transactions to submit to the consensus phase, the censorship repercussions were not identified, and in the context of DeFi, both selective disclosure and censorship are far more severe. This also surfaces an inherent tradeoff between liveness and censorship resistance within such protocols. Since the client (or the protocol node in charge of the client's transaction) is in charge of accumulating timestamps from the nodes in the ordering phase before proceeding to the consensus phase, as [40] notes, it is not possible to distinguish between whether the transaction was sent to the consensus phase in time or not. Consequently, either there will be a liveness failure (by potentially waiting forever for such a transaction), or censorship resistance will not be possible to achieve.

Our analysis is that the primary reason for this is the separation between the pre-protocol that computes the median timestamp and the consensus phase, along with the requirement that a single node is in charge of moving a specific transaction from the former to the latter phase. In such a protocol design, due to the presence of Byzantine nodes, it is impossible to ensure that any transaction that has been timestamped at the protocol nodes also makes its way into the consensus phase.

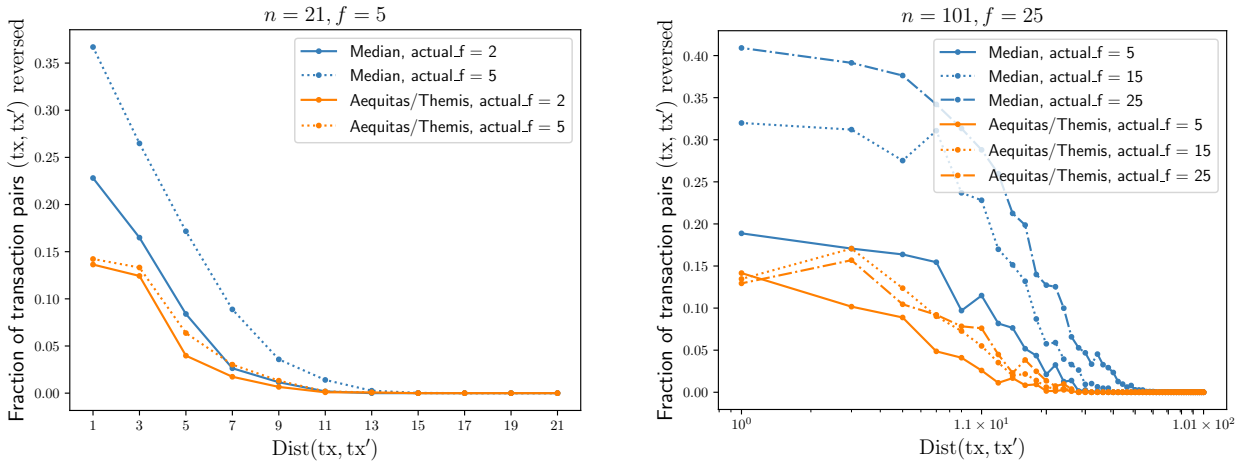
On the other hand, Wendy as well as both Themis and Aequitas [19] are not susceptible to these form of censorship; if a transaction is honest, and therefore received by all honest replicas, it will not be censored, and moreover be sequenced in a fair order.

6.5 Adversarial Reordering

While all notions we considered in our experiment suite guarantee "fair" ordering even in the presence of an adversary for transaction pairs that satisfy a given property (e.g., order-fairness applies when tx is received before tx′ by at least γn nodes), it is also beneficial to understand for various ordering protocols how an adversary can impact the ordering for transaction pairs where the fairness notion *does not apply*.

Zhang et al. [40] show that it is impossible to make the final ordering completely independent of the adversary (since intuitively, adversarial nodes are indistinguishable from honest nodes). Still, ideally, a good ordering protocol restricts adversarial influence only to transactions that are already "very close" in the honest ordering — close in the sense that even their honest ordering could have been inverted by small network delays. Effectively, this would imply that the impact of the adversary is similar to that of small network fluctuations. We analyze this property in our final experiment.

To quantify "closeness" between tx and tx′, we use as a proxy metricthe absolute value of the difference between the number of times tx is received before tx′ and the number of times tx′ is



(a) $n = 21, f = 5$ (b) $n = 101, f = 25$

Figure 6: Adversarial influence on transaction ordering

received before tx, i.e.:

$$\mathsf{Dist}(\mathrm{tx}, \mathrm{tx}') = \\ \left|\#\left[\mathsf{Recv}(\mathrm{tx}) < \mathsf{Recv}(\mathrm{tx}')\right] - \#\left[\mathsf{Recv}(\mathrm{tx}') < \mathsf{Recv}(\mathrm{tx})\right]\right|.$$

Dist varies from $(n \bmod 2)$ to n in increments of 2. A small $\mathsf{Dist}(\mathrm{tx}, \mathrm{tx}')$ implies "fragile" transaction tx, where the number of nodes that received tx earlier is roughly the same as the number that received tx' earlier, while a large Dist value implies that one of the two transactions was received earlier by a large number of nodes.

Adversarial strategy. Since the space of possible adversarial strategies is practically unbounded, we consider just one specific attack in which the adversary will attempt to flip the ordering of transactions. This adversarial strategy is quite simple: an adversarial node simply flips the ordering of all transactions it received as input. For instance, if it received transactions in the order $[\mathrm{tx}_1, \ldots, \mathrm{tx}_l]$, it claims to have received them in the order $[\mathrm{tx}_l, \ldots, \mathrm{tx}_1]$. Despite the simplicity, this should be a reasonable strategy for attempting to reverse the final ordering for many transaction pairs. Still, we acknowledge that better adversarial strategies likely exist and fair ordering protocols should also be tested against them. We leave this for future work. For instance, another strategy could be attempting to invert the final ordering for a specific targeted transaction pair.

Measurement. Given our adversarial strategy of transaction flipping, we now find how frequently it enables an adversary to reverse the ordering (w.r.t. the honest ordering) of a transaction pair with a given distance. We consider two settings: $n = 21, f = 5$ and $n = 101, f = 25$, and vary the *actual* number of corruptions. In each instance, we find the fraction of transaction orderings the adversary successfully inverted. The results are shown in Figure 6.

We did not implement the specific protocols from [22, 40], but we compare Themis to a simple, nearly equivalent protocol that computes the median timestamp for each transaction and then sorts them in ascending order.

From the graphs, we first notice that for large $\mathsf{Dist}(\mathrm{tx}, \mathrm{tx}')$ (≥ 13 for $n = 21$, and ≥ 71 for $n = 101$), there was no successful reordering against either protocol, even with the maximum

number of corruptions (for Themis, this was true for all distances ≥ 11 for $n = 21$ and ≥ 29 for $n = 101$). Note that the X-axis in Figure 6b is in log-scale to better highlight reordering for transactions pairs with small distance (since reordering was absent for large values of Dist).

We also note that in all cases, the fraction of reversed transaction pairs drops sharply as Dist gets larger, which shows both fair ordering protocols working as expected. For small $\mathsf{Dist}(\mathrm{tx}, \mathrm{tx}')$, the transactions are so close that even non-adversarial random network delays can reverse their ordering, and thus it is unreasonable to expect resilience to adversarial strategies.

Overall, we found that for a given actual number of corruptions, it is easier to reverse the transaction ordering for a median timestamp protocol than for Themis. In fact, in our experiment, it was easier to reverse the ordering against the median timestamp protocol even with a small number of corruptions than it was against Themis. Further, we also note that for this specific adversarial strategy, the additional gain against Themis through extra corruptions is small.

7 Related Work

We compare our protocols to related work on fair ordered consensus in this section. We focus our comparison primarily on the recent works of Kelkar et al. [19], Kursawe [22], and Zhang et al. [40] since these recent works all consider some notion of fair transaction ordering. Our experimental suite from Section 6 contains several comparisons but we use this section to highlight a few more nuances.

7.1 Comparison to Other Techniques

We start by comparing the line of work on fair ordering to alternative techniques.

Censorship resistance. Many protocols provide censorship resistance [29] — the property that honest transactions are eventually ordered by the system — or utilize a reputation-based system to detect censorship. Note however that this does not guarantee that the *ordering* of transactions is "fair." Interestingly, as we saw in Section 6.4, protocols can fairly order transactions *that are output by a pre-protocol* yet may still be susceptible to censorship in the pre-protocol itself. We emphasize that the Themis protocol achieves both fair ordering and censorship resistance since it realizes batch-order-fairness for all transactions (and not just those output by a pre-protocol).

Causal ordering and privacy techniques. Prior work in classical distributed systems [6, 32] had considered a limited notion of preventing reordering based on transaction content, but as mentioned earlier (and also identified by [18, 19, 22]), this is not robust against metadata leakage, collusion with protocol nodes, and order-manipulation attacks that do not rely on the content of honest transactions.

Informally, causal ordering uses threshold encryption to hide transaction data before its position is finalized in the total ordering. In a similar spirit, other recent protocols have proposed to use alternative cryptographic confidentiality mechanisms to accomplish the same goal. This includes protocols that utilize, for instance, commit-and-reveal schemes [4], time-lock encryption, verifiable delay functions (VDFs) [35], or even multi-party-computation (MPC) among the protocol nodes [24]. We note that these protocols have the same drawbacks as causal ordering when compared to the line of work on fair ordering protocols. In addition, these protocols are somewhat ad hoc, to the best of our knowledge lacking formal analysis on their properties.

Additionally, many of these protocols involve choosing a random ordering among transactions within a specific epoch. As noted in, e.g., [18], such techniques are susceptible to flooding attacks (e.g., [28]) where now with high probability, at least one adversarial transaction will be be unfairly ordered ahead of the honest user transaction.

We note, however, that confidentiality-based techniques can nicely complement fair ordering protocols; confidentiality can protect transactions against ordering attacks at the network layer, while fair ordering does so at the consensus layer.

7.2 Comparison to Fair-Order Consensus Protocols

Comparison to Aequitas [19]. Kelkar et al. [19] initiated the recent line of work on fair ordering, and they detail several protocols that achieve batch-order-fairness in different settings. For our comparisons, we use their leaderless Aequitas protocol that works in partially synchronous and asynchronous networks. While a leader-based variant is also provided, it uses the same structure as the leaderless protocol, requires more communication, and does not provide any additional benefits. Although Themis and Aequitas satisfy the same notion of batch-order-fairness, Themis has a three-fold advantage over Aequitas: first, our Themis protocol requires only $\mathcal{O}(n)$ communication in the optimistic case while Aequitas always requires $\mathcal{O}(n^3)$. Second, we evade the liveness problem of Aequitas, allowing Themis to satisfy standard liveness. And finally, while Aequitas provides a general compiler to make any standard consensus protocol fair by using it to instantiate FiFo broadcast and (single-shot) Byzantine agreement protocols, Themis can utilize any leader-based protocol as is and add to it the same notion of fairness.

The structure of the dependency graph constructed by Themis is also somewhat different than the one from Aequitas. For some inputs, both Aequitas and Themis build the dependency graph in the same way. In other cases, Themis seems to approximate the graph structure for Aequitas with a smaller γ (since Themis will always attempt to add an edge between any transaction pair to construct, while some edges may be missing from the Aequitas graph). However, to extract a total ordering, Aequitas still needs to compare transactions without an edge, for which it uses the number of descendants in the graph. In Themis, the comparison between such transactions is done explicitly by adding the edge corresponding to the larger weight.Further, Themis's use of Hamiltonian cycles allows for connecting different Condorcet cycles, which appears to provide a stronger notion of "fairness within a batch."

This simple analysis reveals that although both protocols satisfy batch-order-fairness, there are other subtleties within the graph structure that are not captured by the definition. We leave the problem of uncovering these to future work.

Comparison to Zhang et al. [40]. Concurrent to [19], work by Zhang et al. [40] also found fundamental connections of transaction ordering to social choice theory, including the works of Arrow [1] and Gibbard-Satterthwaite [15, 33]. Since our work is geared towards efficient fair ordering protocol, our comparison is based on their specific definition of transaction ordering, and their proposed protocol Pompē.

We point out that the design of Pompē can be modified to achieve optimistic linear complexity through the use of SNARKs for verifying the computation of the median timestamp. Specifically, after the ordering phase, in Pompē, the median timestamp for a given transaction is computed and transmitted during the consensus phase. This is done by the client but Pompē assumes that the

all clients are protocol nodes. In this case, the median computation can be proved (by the client) using constant-size SNARKs which enables linear communication in the optimistic case.

However, we note that for standard usecases, clients and protocol nodes can be separate entities and not all clients need to be present during the system initialization. In such a case, SNARKs with trusted setup might not provide an appropriate trust model for clients. Current constant-size SNARKs, however, all require trusted setup. Further, even if a trusted setup can be used, the public parameters (e.g., the CRS) need to be communicated to the clients. Since the CRS size is dependent on n for known constant-size SNARKs, this means that in the usual setting where clients and nodes can be distinct, the optimistic communication complexity of Pompē will not linear in n.

While SNARKs without trusted setup can be used instead, current state-of-the-art constructions [5, 34, 37] all have at least logarithmic proof sizes. This means that, even with the use of SNARKs, at best, $\mathcal{O}(n \log n)$ optimistic communication complexity can be achieved.

8 Conclusion

We introduced Themis, the first consensus protocol that satisfies a strong fair transaction-ordering property, requires only linear communication in the optimistic case, and has efficiency comparable even to state-of-the-art consensus protocols that lack fair-ordering properties. We also introduced a new systematically designed suite of experiments related to fairness to evaluate and compare Themis to the recent exciting line of work on fair-ordering protocols. These experiments show that the fairness property satisfied by Themis is stronger in practice than alternative notions.

Acknowledgments

We thank Haobin Ni, Bowen Xue, Maofan (Ted) Yin, and Yunhao Zhang for helpful discussions on Hotstuff. This work also benefited from several interesting discussions on BFT ordering with Yunhao Zhang. Mahimna Kelkar, Sishan Long, and Ari Juels were funded by NSF grants CNS-1564102, CNS-1704615, and CNS-1933655 as well as generous support from IC3 industry partners. Soubhik Deb and Sreeram Kannan were funded by NSF grants CCF-1908003 and CCF-1651236.

References

[1] Kenneth J. Arrow. *Social Choice and Individual Values*. Wiley, 1951.

[2] Mathieu Baudet et al. *State machine replication in the Libra blockchain*. 2019.

[3] Nir Bitansky et al. "From Extractable Collision Resistance to Succinct Non-Interactive Arguments of Knowledge, and Back Again". In: *ITCS*. 2012, 326–349.

[4] Lorenz Breidenbach et al. *To Sink Frontrunners, Send in the Submarines*. https://hackingdistributed.com/2017/08/28/submarine-sends/. 2017.

[5] Benedikt Bünz and Ben Fisch. "Transparent SNARKs from DARK Compilers". In: *EUROCRYPT*. 2020, pp. 677–706.

[6] Christian Cachin et al. "Secure and Efficient Asynchronous Broadcast Protocols". In: *CRYPTO*. 2001, pp. 524–541.

[7] Paul Camion. “Chemins et circuits hamiltoniens des graphes complets”. In: *Comptes Rendus de l’Académie des Sciences de Paris* 249 (1959), pp. 2151–2152.

[8] Miguel Castro and Barbara Liskov. “Practical Byzantine Fault Tolerance”. In: *OSDI*. 1999, pp. 173–186.

[9] *Condorcet Paradox*. https://wikipedia.org/wiki/Condorcet_paradox. Accessed Aug. 2021.

[10] Philip Daian et al. “Flash Boys 2.0: Frontrunning in Decentralized Exchanges, Miner Extractable Value, and Consensus Instability”. In: *IEEE S&P*. 2020, pp. 585–602.

[11] *DeFi Pulse*. defipulse.com. Accessed Aug. 2021.

[12] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. “Consensus in the presence of partial synchrony”. In: *J. ACM* 35.2 (1988), pp. 288–323.

[13] Shayan Eskandari, Seyedehmahsa Moosavi, and Jeremy Clark. “SoK: Trans- parent Dishonesty: Front-Running Attacks on Blockchain”. In: *FC*. 2019, pp. 170–189.

[14] Juan Garay and Aggelos Kiayias. “SoK: A Consensus Taxonomy in the Blockchain Era”. In: *CT-RSA*. 2020, pp. 284–318.

[15] Allan Gibbard. “Manipulation of Voting Schemes: A General Result”. In: *Econometrica* 41.4 (1973), pp. 587–601.

[16] Jens Groth. “On the Size of Pairing-based Non-interactive Arguments”. In: *EUROCRYPT*. 2016, pp. 305–326.

[17] Chi Ho, Danny Dolev, and Robbert van Renesse. “Making distributed systems robust”. In: *OPODIS*. 2007, pp. 232–246.

[18] Mahimna Kelkar, Soubhik Deb, and Sreeram Kannan. “Order-Fair Consensus in the Permissionless Setting”. In: https://eprint.iacr.org/2021/139.

[19] Mahimna Kelkar et al. “Order-Fairness for Byzantine Consensus”. In: *CRYPTO*. 2020, pp. 451–480.

[20] Ariah Klages-Mundt and Andreea Minca. *(In)Stability for the Blockchain: Deleveraging Spirals and Stablecoin Attacks*. 2020. arXiv: 1906.02152.

[21] Klaus Kursawe. *Wendy grows up*. 2021.

[22] Klaus Kursawe. “Wendy, the Good Little Fairness Widget: Achieving Order Fairness for Blockchains”. In: *ACM AFT*. 2020, pp. 25–36.

[23] Leslie Lamport, Robert Shostak, and Marshall Pease. “The Byzantine Generals Problem”. In: *TOPLAS* 4.3 (1982), pp. 382–401.

[24] Yunqi Li et al. *HoneyBadgerSwap: Making MPC as a Sidechain*. https://medium.com/initc3org/honeybadgerswap-making-mpc-as-a-sidechain-364bebdb10a5. 2021.

[25] *libhotstuff: A general-purpose BFT state machine replication library with modularity and simplicity*. https://github.com/hot-stuff/libhotstuff. 2018.

[26] Mary Maller et al. “Sonic: Zero-Knowledge SNARKs from Linear-Size Universal and Updatable Structured Reference Strings”. In: *CCS*. 2019, 2111–2128.

[27] Y. Manoussakis. “A Linear-Time Algorithm for Finding Hamiltonian Cycles in Tournaments”. In: *Discrete Appl. Math.* 36.2 (1992), 199–201.

[28] Alex Manuskin. *The fastest draw on the Blockchain: Ethereum Backrunning*. https://medium.com/@amanusk/the-fastest-draw-on-the-blockchain-bzrx-example-6bd19fabdbe1. 2020.

[29] Andrew Miller et al. "The Honey Badger of BFT Protocols". In: *ACM CCS*. 2016, pp. 31–42.

[30] Rafael Pass and Elaine Shi. "FruitChains: A Fair Blockchain". In: *PODC*. 2017, pp. 315–324.

[31] Kaihua Qin, Liyi Zhou, and Arthur Gervais. *Quantifying Blockchain Extractable Value: How dark is the forest?* 2021. arXiv: 2101.05511.

[32] Michael K. Reiter and Kenneth P. Birman. "How to Securely Replicate Services". In: *ACM Trans. Program. Lang. Syst.* 16.3 (1994), 986–1009.

[33] Mark Allen Satterthwaite. "Strategy-proofness and Arrow's conditions: Existence and correspondence theorems for voting procedures and social welfare functions". In: *Journal of Economic Theory* 10.2 (1975), pp. 187–217.

[34] Srinath Setty. "Spartan: Efficient and General-Purpose zkSNARKs Without Trusted Setup". In: *CRYPTO*. 2020, pp. 704–737.

[35] StarkWare. *Presenting: VeeDo a STARK-based VDF Service*. https://medium.com/starkware/presenting-veedo-e4bbff77c7ae.

[36] Robert Tarjan. "Depth-first search and linear graph algorithms". In: *SIAM J. Comput.* 1.2 (1972), pp. 146–160.

[37] Riad S. Wahby et al. "Doubly-Efficient zkSNARKs Without Trusted Setup". In: *IEEE S&P*. 2018, pp. 926–943.

[38] Wonderproxy. *A day in the life of the Internet*. https://wonderproxy.com/blog/a-day-in-the-life-of-the-internet/. 2020.

[39] Maofan Yin et al. "HotStuff: BFT Consensus with Linearity and Responsiveness". In: *PODC*. 2019, pp. 347–356.

[40] Yunhao Zhang et al. "Byzantine Ordered Consensus without Byzantine Oligarchy". In: *OSDI*. 2020, pp. 633–649.

[41] Liyi Zhou et al. "High-Frequency Trading on Decentralized On-Chain Exchanges". In: *IEEE S&P*. 2021.

A Explicit Condorcet Chaining Example

We provide an explicit algorithm (Algorithm 1) for generating an arbitrarily large Condorcet cycle for any valid n, f, γ. For simplicity, we provide details for the synchronous setting where the "claimed" input orderings of all nodes are taken into account. Abstractly, the algorithm inductively constructs transaction input orderings for each node in a way that all transactions will be part of the same Condorcet cycle. Note that this simply provides a theoretical example of unbounded chaining; as we saw earlier, in practical networks, we expect Condorcet cycles to be small in length.

Suppose that the system nodes are denoted by the set $\{0, \ldots, n-1\}$. To create a cycle of length $l \geq n$ (this is a non-optimal bound on the minimum length of a cycle with n nodes), the algorithm proceeds by interatively adding a new transaction to all input lists while maintaining the invariant that all current transactions form a Condorcet cycle. For the initial transaction tx_0, the input ordering txlist_j of each node j is taken simply as $[\mathrm{tx}_0]$. To add a later transaction tx_i, a set S_i of

Algorithm 1: Explicit input ordering with unbounded Condorcet chaining for n, f, γ

Let $N = \{0, \ldots, n-1\}$ be the set of all nodes.;
Let $\mathsf{txlist}_0, \ldots, \mathsf{txlist}_{n-1} \leftarrow [\mathrm{tx}_0]$;
$\mathsf{start} \leftarrow 0$;
for $i \in 1, 2, \ldots$ **do**
    $\mathsf{end} \leftarrow (\mathsf{start} + (\gamma n - f) - 1) \mod n$
    if $\mathsf{start} < \mathsf{end}$ **then**
        $S = \{\mathsf{start}, \ldots, \mathsf{end}\}$
    else
        $S = \{\mathsf{start}, \ldots, n-1\} \cup \{0, \ldots, \mathsf{end}\}$
    end
    for $j \in S$ **do**
        Insert tx_i right before tx_{i-1} in txlist_j
    end
    for $j \in N \setminus S$ **do**
        Append tx_i at the end of txlist_j
    end
    $\mathsf{start} \leftarrow (\mathsf{start} + 1) \mod n$
end

size $\gamma n - f$ is first chosen. S_1 is chosen to be the set $\{0, \ldots, \gamma n - f - 1\}$. The set S_i is obtained by cyclically rotating S_{i-1} modulo n. For a node $j \in S_i$, tx_i is inserted right before before tx_{i-1} in txlist_j; for other nodes $j \notin S_i$, tx_i is appended to the end of txlist_j. By doing so, we make sure that tx_i occurs before tx_{i-1} in at least $\gamma n - f$ lists.

Additionally, it is easy to see why tx_0 will occur before tx_l in at least $\gamma n - f$ lists through induction. First note that tx_0 is before tx_1 in $n - (\gamma n - f) = n(1 - \gamma) + f$ lists. Further, if tx_d occurs after tx_0 in some list, then for any $d' \geq d$, $\mathrm{tx}_{d'}$ will also occur after tx_0 in that list (since each transaction is added just before the last one or at the end of the list). Now, due to the cyclic rotation of the S_i, each tx_j will be after tx_0 in one more list (subject to a maximum of n lists) than tx_{j-1} was. Consequently, for any γ, f, when $l \geq n$, tx_0 will occur before tx_l in at least $\gamma n - f$ lists. When $\gamma = 1$, $l = n$ will be the first iteration that this happens.

Finally, putting this together, we can conclude that $[\mathrm{tx}_0, \mathrm{tx}_1, \ldots, \mathrm{tx}_l]$ forms a Condorcet cycle.