Themis: Fast, Strong Order-Fairness in Byzantine Consensus

Mahimna Kelkar
Cornell Tech

Soubhik Deb
University of Washington

Sishan Long
Cornell Tech

Ari Juels
Cornell Tech

Sreeram Kannan
University of Washington

***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 enforces the strongest notion of fair ordering proposed to date. It also achieves standard liveness, rather than the weaker notion of previous work with the same fair ordering property.**

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 practical settings 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.

I. Introduction

Decentralized Finance (DeFi), meaning the deployment of financial instruments on blockchains, has attracted substantial interest in recent years, with over 50 billion USD locked in Ethereum DeFi as of August 2022 [3]. Unfortunately, while DeFi continues to gain popularity, a long line of work [12], [14], [20], [30], [39] 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. Indeed, in both permissioned consensus protocols, e.g., PBFT [11] and Hotstuff [37], 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 [9], [18], [19], [21], [22], [38] has proposed 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. Intuitively, this style of fairness seeks to guarantee a specific ordering in the finalized ledger based on how transactions arrive into the network. These notions are different and in many cases stronger than past ordering properties such as causal ordering [7], [31] 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 [27]). The new line of work on fair ordering attempts to tackle transaction ordering at a more fundamental level; notably, [18], [19], [38] all found exciting connections of the fair ordering problem to social choice theory.

The fair ordering landscape. Existing fair-ordering protocols, however, have serious practical limitations. The Aequitas protocol from [19] has impractically high $\mathcal{O}(n^3)$ communication complexity and is also only able to provide a weaker liveness property. Protocols from subsequent works require $\mathcal{O}(n^2)$ communication but suffer from significant other shortcomings.

The protocol in [9] is only shown to provide liveness when *all nodes are honest* (see Section VII). Moreover, as we show in this work (see Section VI), there are subtle *censorship issues* in Pompē [38] and that the fairness property satisfied by both Pompē [38] and Wendy [21] is *significantly weaker* than one from Aequtias. Table I shows a few comparison points.

This work presents a new protocol Themis, which we term to be the first fair-ordering protocol that can be practically deployed. Themis achieves the same strong fairness property as Aequitas, guarantees liveness, and our implementation incurs minimal cost over Hotstuff [37], a widely used state-of-the-art consensus protocol without any fair-ordering guarantees.

A. Themis *Overview and Contributions*

Themis operates in a partially synchronous setting with a committee of n nodes of which at most f may be arbitrarily adversarial where $n \geq 4f+1$. We implement Themis on top of Hotstuff [37], a widely used leader-based protocol and show that it incurs minimal performance overhead.

Themis**: Fair-ordering property.** Themis achieves the *batch-order-fairness* property proposed by Kelkar et al. [19]. Informally, batch-order-fairness[2] (Definition III.1) 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$'$. While [19] also introduced a stronger receive-order-fairness property where tx must be

[1] We use "fairness" to mean *fairness of transaction ordering* or *fair ordering*, although the term has been used in the past for unrelated notions (e.g., fair PoW mining [29]).

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

Protocol	Transaction Ordering	Comm. Complexity Optimistic	Comm. Complexity Worst	Corruption	Liveness	Censorship Resistance	Synchronized Clocks?
Aequitas [19]	**γ-batch-order-fairness** (Definition III.1)	$\mathcal{O}(n^3)$	$\mathcal{O}(n^3)$	$n > \frac{4f}{2\gamma-1}$ **(5)**	**Weak**	**Yes**	**No**
Wendy [21]	Timed-Relative-Fairness**(1)** (Section VI-A)	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$	$\boldsymbol{n \geq 3f+1}$	**Standard**	**Yes**	Yes**(2)**
Pompē [38]	Ordering Linearizability**(1)** (Section VI-A)	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$	$\boldsymbol{n \geq 3f+1}$	**Standard**	**No**	Yes
Quick-Fairness [9]	**κ-differential order-fairness**(3)	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$	$n \geq 3f+\kappa+1$	**Only when all nodes are honest**		**No**
Themis (This Work)	**γ-batch-order-fairness**(4)	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$	$n > \frac{4f}{2\gamma-1}$ **(5)**	**Standard**	**Yes**	**No**
SNARK-Themis (This Work)		$\boldsymbol{\mathcal{O}(n)}$	$\boldsymbol{\mathcal{O}(nf)}$				

TABLE I: 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. An entry in **red** indicates a significant shortcoming.

(1) *Fair separability* (Definition VI.1) captures both notions (see Section VI-A). Our experiments in Section VI show why this notion is significantly weaker than the one achieved by Themis and is likely not enough to handle adversarial order-manipulation in real-world settings; **(2)** In Wendy, if honest local clocks are far apart (e.g., several seconds apart), then their fairness definition will never apply and therefore no fairness guarantees can be provided; **(3)** We show that κ-differential-order-fairness is simply a reparameterization of batch-order-fairness (see Section VII and Appendix D); **(4)** Since we consider a total ordering in Themis, transactions within a batch are ordered contiguously for execution (see Section III-C and Section IV); **(5)** The fairness properties of both Aequitas and Themis are parameterized by γ but even for the weakest variant, i.e., corresponding to $\gamma = 1$, they are still stronger than those of Wendy [21] and Pompē [38]. For $\gamma = 1$, Aequitas and Themis both require $n \geq 4f+1$ but also work with $n \geq 3f+1$ if only crash-faults are present.

ordered strictly before tx′, it was shown to be impossible without strong synchrony assumptions.

Informally, *batches* arise as a result of non-transitive *Condorcet* cycles [2] in message receipt times across nodes (see Section III for more details); the relaxation is minimal in the sense that when there are no cycles, the stronger receive-order-fairness property can be satisfied.

We find however that these cycles can extend for arbitrarily long; this is ultimately responsible for the liveness problems with Aequitas. To get around this, we notice that in a typical deployment for Themis (e.g., for smart contracts), all transactions (even those in a batch) will require a total ordering for execution. Here, unlike Aequitas which can totally order transactions in a cycle only after all transactions in it are seen (which can take arbitrarily long), in Themis, using our technique of *unspooling* (see Section III), we can output a batch part-by-part while still ensuring that all transactions in the same batch are output in an uninterrupted sequence. As a consequence, Themis imposes a total ordering on transactions, but one that respects batch-order-fairness.

Themis further guarantees another useful property: for any two consequent 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. This property also provides resistance against a particular kind of frontrunning attack—where the adversary wants to *immediately precede* a targeted user transaction.

We find that the fairness property supported in Themis (even for the weakest version, i.e., corresponding to $\gamma = 1$) is stronger in practice than other ordering notions proposed in Zhang et al. [38] and Kursawe [21]. This is showcased through our suite of fairness experiments.

Themis **design** (Section IV). Themis can be bootstrapped from any leader-based consensus protocol with minimal design changes: 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 block 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.

To construct a fair proposal, 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 Aequitas protocol from [19] which achieves only a weaker liveness notion due to the possible "chaining" of Condorcet cycles (see Section III). 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. The technique is also of general interest: it can, e.g., be retrofitted to [19] to achieve standard liveness there too.

Theoretical SNARK-based design (Section IV-D and Appendix B-B). We also describe a more theoretical design—SNARK-Themis—which makes use SNARKs to achieve optimistic $\mathcal{O}(n)$ communication complexity. This is particularly notable since we find that other fair-ordering protocols cannot easily be made optimistically $\mathcal{O}(n)$ even with the use of SNARKs. This makes SNARK-Themis the first such fair-ordering protocol; it's asymptotic communication complexity is in fact optimal and equivalent to state-of-the-art protocols that lack any fair ordering guarantees, e.g., Hotstuff [37].

Implementation and benchmarks (Section V). We implement the $\mathcal{O}(n^2)$ version of Themis, and show that its integration with Hotstuff's codebase [1] results in small performance overhead in practice; our implementation with $n = 30$ nodes was still able to achieve a latency of roughly 53ms and a peak throughput of $52,719$ transactions per second, which should suffice for most applications. Notably, Themis scales in the same way as Hotstuff when n is increased. Furthermore, any overhead almost entirely vanishes when nodes are geo-distributed. Themis' source code is available at: https://github.com/anonthemis/themis-src-anon.

Suite of fairness experiments (Section VI). There exists no prior work on practical measurement or empirical comparison of fair-ordering protocols. A key contribution of our work, therefore, is a systematically conceived suite of experiments to quantify the practical impact of both fairness definitions and protocols and understand their design tradeoffs. We consider both honest settings as well as broad classes of adversarial attacks that are common in practice. We study Themis through this lens and show that it provides significantly better fairness properties compared to other alternatives. We believe that our fairness suite will be useful for any future work on fair ordering protocols. We showcase three experiments:

1) *Ideal setting.* We quantify the strength of different fair ordering properties in an ideal honest setting (with no adversarial nodes) to understand the best-case scenario.

2) *Frontrunning and insertion attacks.* We evaluate resilience against network-layer insertions—attempts to maliciously insert transactions at the network layer (i.e., even before the consensus begins) through e.g., frontrunning.

We first prove that Themis does not allow any frontrunning under a natural assumption that the network respects triangle inequality. This result complements our experiments using a real network. As a concrete datapoint, for 100 geo-distributed nodes, for the fairness notions in [21], [38], up to 94% of the node connections in the network are still susceptible to adversarial frontrunning. In contrast, for the fairness notion in Themis, the number goes down to 2.8% in the simplest protocol parameter choice, and just 0.16% for the optimal parameter.

3) *Adversarial reordering.* We evaluate robustness to reordering attacks, i.e., attempts to maliciously reorder transactions compared to the honest execution. As a concrete datapoint, in setting with 101 nodes, it is easier to reorder in a median-timestamp based protocol (which essentially abstracts out the fairness components of [21], [38]) with just 5 adversarial nodes, than in Themis even given 25 nodes.

As a separate point, we also evaluate resistance to *censorship* (Section VI-F). In particular, we show a subtle censorship issue with Pompē [38], due to its use of an ordering phase prior to the actual consensus protocol.

II. Preliminaries

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 [13]; 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.

Graph terminology and algorithms. We make use of common graph algorithms to reason about the ordering dependencies between transactions. Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ denote 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.

$\mathcal{G}$ is 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 can be dropped when the context is clear. Recall that an SCC 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 [34]. 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 [26].

III. Building Blocks

Our starting point is the fairness definitions of Kelkar et al. [19], and their Aequitas fair-ordering protocol. Our experiments show that their fairness property is stronger than the ones from other works, and therefore the focus of this work is to achieve the same strong property while fixing two problems with Aequitas: (1) the lack of standard liveness; and (2) the high communication complexity.

A. Aequitas Background

Batch-order-fairness. While Kelkar et al. [19] defined several fairness variants, the primary notion considered and subsequently realized by their leaderless Aequitas protocol, was *batch-order-fairness*, parameterized by $\frac{1}{2} < \gamma \leq 1$.

Definition III.1 (γ-batch-order-fairness)**.** Suppose that tx and tx$'$ are received by all nodes. If γn nodes received tx before tx$'$ locally, then all honest nodes output tx *no later than* tx$'$.

A stronger notion, receive-order-fairness (exactly as Definition III.1, but now, tx must be output *before* tx$'$), was also considered. An impossibility result, however, rules out realization except in specific synchronous settings. The impossibility arises from the Condorcet paradox [2] in voting theory.

Abstractly, this allows for non-transitive global preferences even if each party's local preferences are transitive. 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.

Consequently, the global "fair" transaction ordering (according to receive-order-fairness) could contain cycles (even in a non-adversarial setting), which we call *Condorcet cycles* (we will often simply use the term cycle). Batch-order-fairness sidesteps this problem by enabling transactions to be output in *batches* (transactions within a batch can still be totally ordered at e.g., the application layer), and subsequently ignoring unfairness resulting from any cyclic orderings in the same batch. Importantly, **the batch relaxation is only used for transactions in the same cycle**.

Aequitas overview. 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. FiFo broadcast [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 outputs them when, informally, they no longer have any dependencies.

Aequitas realizes batch-order-fairness and circumvents the receive-order-fairness impossibility 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).

B. Aequitas Technical Challenges

Weak-liveness. A crucial problem with the Aequitas protocol, however, is that it was proven to only guarantee a *weak* notion of liveness (given as Definition A.1 for completeness). We find that once again, the presence of Condorcet cycles proves problematic here. In particular, we find that Condorcet cycles can "chain" together and form larger ones that can extend for arbitrarily long (in the worst case); informally, two cycles can be chained by having them share a transaction. This means that transactions input much later can become part of the same cycle (although this can happen because of the specific input transaction orderings and not necessarily the adversary). In other words, specific input orderings could prevent transactions from being output for arbitrarily long.

Although Kelkar et al. [19] do not formally define these cycles, they are implicit in their weak-liveness definition. We show that, intuitively, an equivalent formulation of weak-liveness is that liveness for a transaction is guaranteed *only after its entire cycle is complete*, which may take arbitrarily long in the worst case (i.e., it cannot be a function of Δ).

In contrast, (standard) liveness in a partially synchronous network guarantees that transactions are output within a finite time dependent on the fixed (but unknown) parameter Δ.

Technical challenges and insights. Fixing the two aforementioned problems (weak-liveness and communication complexity) with Aequitas results in some technical challenges that required novel insights. Surprisingly, we find that the two problems are intertwined in a way that solving one actually requires solving the other.

As a first step, we prove that since Condorcet cycles can extend for arbitrarily long, if the protocol waits for all transactions in a cycle to be seen before they are output, it cannot achieve standard liveness; in fact, here, Aequitas' notion of weak-liveness is the best possible property.

To get around this, we notice that in a standard SMR deployment (e.g., for smart contracts), all transactions (even those in the same batch) will need to be *totally ordered* for execution. Note that this is compatible with batch-order-fairness and here, Aequitas will enforce any total ordering within a batch after it has been *fully seen*. Surprisingly, it turns out that designing the protocol from the ground up with the final total ordering in mind allows us to get around the liveness problems. In so doing, we introduce *batch unspooling*, which intuitively allows for a batch to be output *part-by-part* (i.e., some transactions in a batch output before the rest) while still ensuring that all transactions in the batch are output *contiguously* (i.e., all transactions in a batch are output in an uninterrupted sequence). Towards this goal, our work needed to establish a **novel understanding of Condorcet cycles**.

Second, to reduce the communication cost, we use a common technique of routing communication through a *leader*. This turns out to be quite tricky however; a naïve design actually leads to higher cost. This happens because the presence of arbitrary-length Condorcet cycles can result in empty proposals even from honest leaders. To solve this, we introduce our technique of *deferred ordering* which enables the unspooling of a batch *across leaders*. In particular, deferred ordering allows for a leader proposal, in some cases, to contain partially ordered transactions which will be be totally ordered by a subsequent leader. Notably, we can still guarantee that the total ordering will be finalized *only based on the network delay* and therefore no weak-liveness problems arise.

C. Novel Understanding of Condorcet Cycles

While understanding cyclic ordering dependencies between transactions, we need to account for the fact that some replicas may be adversarial (and claim to have received transactions in a different order), as well as the fact that up to f honest nodes may not be considered (since we need to work in a partially synchronous setting).

Condorcet cycles. We formally define (weak)-Condorcet cycles below as they lead to a key piece of our protocol design; these also turn out to be hidden within the weak-liveness definition of Kelkar et al. [19].

Definition III.2 (Condorcet Cycle)**.** A list $[\text{tx}_1, \ldots, \text{tx}_l]$ is a (weak)-Condorcet cycle of length l if the following holds: For all $i \in \{1, \ldots, l\}$, where $\text{tx}_1 = \text{tx}_{l+1}$, at least $n(1-\gamma)+1$ **honest** nodes have received tx_{i+1} before tx_i.

Remark 1. While not important for our paper, we can also define a stronger version based on the transaction ordering nodes

"claim" to have received within the protocol. In particular, in the above definition, 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 $[\text{tx}_1, \ldots, \text{tx}_l]$ will become a *strong*-Condorcet cycle. The threshold $\gamma n - f$ is used here since it is guaranteed to hold when γn nodes receive tx_{i+1} before tx_i. Unless specified, we will use cycles from Definition III.2.

Impossibility of liveness for entire batches. We start by showing a constructive example of an input transaction ordering that results in Condorcet cycles of *arbitrary length (and time)* for any parameters n, $f \geq 1$ and γ. This is done by continuously *chaining* together smaller Condorcet cycles to form larger ones. As an informal but illustrative example, a cycle $[\text{tx}_1, \text{tx}_2, \ldots, \text{tx}_l]$ can be formed through smaller cycles $[\text{tx}_1, \text{tx}_2, \text{tx}_3]$, $[\text{tx}_3, \text{tx}_4, \text{tx}_5], \ldots, [\text{tx}_{l-2}, \text{tx}_{l-1}, \text{tx}_l]$ where subsequent cycles are formed such that they share a common transaction. The overall construction is somewhat non-intuitive, but we give a general algorithm in Appendix A to explicitly build these arbitrary length cycles. In fact, the algorithm also shows something stronger, namely the construction of arbitrary-length *strong*-Condorcet cycles.

As a consequence, if a protocol waits for the entire cycle to be seen before transactions within it are output (as Aequitas does), then standard liveness is impossible to achieve.

Batch *unspooling*. We get around this impossibility using a new technique. In typical use cases for consensus protocols (e.g., a smart-contract setting), it is necessary to *totally order* all transactions for execution. For this, Aequitas supports enforcing any total ordering for transactions within a batch. Still, this is done only at the end *after* the entire cycle has been seen. The final ordering can now be thought of being linearly partitionable into cycles; in particular, $[\text{tx}_1^{(1)}, \ldots, \text{tx}_{l_1}^{(1)}, \text{tx}_1^{(2)}, \ldots, \text{tx}_{l_2}^{(2)}, \ldots]$ where $\text{tx}_1^{(i)}, \ldots, \text{tx}_{l_i}^{(i)}$ are part of the same cycle.

Perhaps surprisingly, we find that embedding the total ordering requirement *within the protocol design* itself allows us to provide (standard) liveness while still *keeping the same fairness guarantees*. We call this technique *batch unspooling*. Unspooling allows us to output transactions within a cycle *without waiting for the full cycle* to be seen. Nevertheless, we can guarantee that transactions within the same cycle will be output *contiguously*, i.e., no transaction from a later cycle will be ordered before all transactions from the current cycle are ordered. This allows us to **achieve the same fairness property** as Aequitas (in the scenario where final execution requires a total ordering) while providing standard liveness.

We note that figuring out when the current cycle ends brings back the same weak-liveness problem, but our technique of batch unspooling makes it so that this is no longer required.

IV. Themis Description

We now describe our protocol Themis in detail. Themis can be bootstrapped from *any existing leader-based protocol* and will endow it with fair ordering with minimal design changes.

Overall design. Recall that standard leader-based protocols allow the leader to *unilaterally* choose its block proposal—only the validity of transactions and not their ordering is checked by the replicas. Themis will achieve fairness 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 the given replica ordered by their receive times.

Looking ahead, to enable the unspooling of a batch across multiple leaders, we will introduce our technique of *deffered ordering*. In line with this, we need to describe two algorithms for an honest leader to execute: `FairPropose` for constructing (partial) proposals, and `FairUpdate` for any updates to previous proposals. In our core Themis design, the leader will also send all the local orderings to all replicas to enable checking the fairness of its proposal. As a more theoretical alternative, within SNARK-Themis, we leverage SNARKs to reduce the asymptotic communication complexity of this step.

Finally, we also describe an algorithm `FairFinalize`, that allows the replicas to extract a fair transaction ordering from fully specified proposals (we will elaborate on this later).

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 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. In other words, Themis does not need to rely on synchronized clocks.

A. Constructing the Leader Proposal

The leader starts by checking the validity of the local replica orderings by checking their signatures. Then, to construct its proposal, the leader executes $\texttt{FairPropose}(\mathcal{L})$, where $\mathcal{L}$ is a set of $n - f$ replica orderings. Abstractly, the goal of this algorithm is to propose as many transactions as possible while making sure that the exclusion of any transaction does not violate fairness.

Replica ordering notation. For a set $\mathcal{L}$ of $n - f$ orderings, we use $\text{tx} \in_k \mathcal{L}$ (resp. $\text{tx} \notin_k \mathcal{L}$) to denote that tx is present in at least (resp. less than) k orderings in $\mathcal{L}$. We use $\text{tx} \prec_{(\mathcal{L},k)} \text{tx}'$ to denote that tx appears before tx' in at least k orderings in $\mathcal{L}$. We use $\mathsf{Weight}_{\mathcal{L}}(\text{tx}, \text{tx}')$ to denote the maximum value k such that $\text{tx} \prec_{(\mathcal{L},k)} \text{tx}'$.

Deferred ordering intuition. Abstractly, the final total transaction ordering can be thought of as being partitioned into cycles. While enabling batch unspooling across multiple leaders, we still need to guarantee that no transaction from a *later cycle* is output before all transactions from the current cycle are output. Towards this, based on the local replica orderings, the leader can split transactions into three types: *solid*, *blank*, and *shaded*. We use *non-blank* to label a transaction that is either solid or shaded. Very roughly, this delineation enables the following:

1) Solid transactions are the ones that are present in many replica orderings and can be sequenced within the current proposal. This property can be guaranteed for tx if $\text{tx} \in_{n-2f} \mathcal{L}$.

2) Blank transactions are present in too few replica orderings and can be *excluded* from the current proposal. In particular, we can guarantee that if tx is blank, then it *cannot be part of a previous cycle* than any transaction in the current proposal; in other words, excluding it will not result in any unfairness. This happens when tx $\notin_{n(1-\gamma)+f+1} \mathcal{L}$.

3) Shaded transactions take a more indeterminate form. These are transactions tx such that tx $\in_{n(1-\gamma)+f+1} \mathcal{L}$ but tx $\notin_{n-2f} \mathcal{L}$. In many cases, we can rule out the possibility of tx occurring in a previous cycle than all transactions in the current proposal, in which case tx can safely be excluded. In some cases however, current information does not allow for this possibility to be ruled out. When this happens, we must include the shaded transaction within the current proposal.

Doing so can result in a somewhat different challenge. If two or more such shaded transactions are included, then the ordering *amongst them* may not be clear yet. Our deferred ordering technique applies exactly here. It allows proposing a partial ordering for shaded transactions which will be completed at a later point when they are present in $n-2f$ replica orderings. Notably, this happens only based on the actual network delay for these transactions to be received, and is therefore independent of other transactions.

Leader proposal algorithm. We now detail the algorithm `FairPropose`. Since some transactions may be partially ordered, we will have `FairPropose` output a *dependency* graph $\mathcal{G}$ that contains ordering dependencies within transactions in the current proposal. Later, through `FairFinalize`, replicas will extract out the final fair ordering from this graph.

The dependency graph $\mathcal{G}$ is constructed as follows: First, a vertex is added for each non-blank transaction. An edge (tx, tx') is added to the graph whenever tx $\prec_{(\mathcal{L},\text{thresh})}$ tx$'$ where $\text{thresh} = n(1-\gamma)+f+1$. Intuitively, this threshold is chosen so that an edge (tx, tx') signifies that tx$'$ *cannot be in an earlier cycle* than tx. Only one of (tx, tx') or (tx', tx) is added however, i.e., if both conditions are satisfied, the one 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 $k, k' \geq \mathsf{thresh}$, 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 $k, k' < \mathsf{thresh}$. When this happens, an edge will be added by a later proposal.

Next, we need to figure out which shaded transactions need to be included in the proposal and which ones can be excluded (recall that all solid transactions will be proposed). In particular, the proposal will contain exactly those shaded transactions that have a path (in $\mathcal{G}$) to a solid transaction.

For this, we compute the condensation of $\mathcal{G}$ and topologically sort it. Let V be the last vertex in this sorting that contains a solid transaction. Then, `FairPropose` outputs the graph obtained by removing all tx from $\mathcal{G}$ where $\mathsf{SCC}(\text{tx})$ occurs after V in the topological sorting; everything after this needs to be excluded. The details are given in Fig. 1.

Proposal properties. The output of `FairPropose`$(\mathcal{L})$ is guaranteed to contain all solid transactions in $\mathcal{L}$ and no blank transactions. Further, it contains exactly those shaded transactions that contain an outgoing path into a solid transaction within the dependency graph $\mathcal{G}$. Further, there is exactly one edge between any two vertices, except when both vertices are shaded (in which case there might be none). We prove this graph structure in Lemma B.1. Future leaders cannot modify

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

`// Propose a fair ordering for new transactions.`
On input a set $\mathcal{L}$ containing the local 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 $\mathcal{G}^*$ and its topological sorting S.
2) **Output Fair Ordering**
   - Let V be the last vertex of S that contains a solid transaction.
   - Remove those tx from $\mathcal{G}$, that are in vertices after V in S.
   - Output $\mathcal{G}$.

Fig. 1: Leader Proposal Algorithm

Algorithm `FairUpdate`$(\mathcal{L}_{\text{updates}})$

`// 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 tx $\in_{n-2f} \mathcal{L}_{\text{updates}}$, $k \geq k'$ and $k \geq n(1-\gamma)+f+1$ then add the edge (tx, tx') to $\mathcal{E}_{\text{updates}}$.
   - Output $\mathcal{E}_{\text{updates}}$.

Fig. 2: Update Algorithm

existing edges but will be able to add missing edges between shaded transactions. As mentioned before, our deferred ordering technique of proposing a "partial graph" is crucial as it allows us to avoid the weak-liveness problem.

To show that this graph proposal is consistent with batch-order-fairness, we now need to show that any transaction exclusion does not violate fairness. In particular, we prove (in Lemma B.3) that any transaction that is excluded *cannot be in a previous cycle* than any transaction that was included. As a consequence, this implies that cycles are output *contiguously*, i.e., all transactions from the current cycle will be output before any transaction from a subsequent cycles.

Update algorithm. The algorithm `FairUpdate` 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 receiving $\mathcal{L}_{\text{updates}}$ containing $n-f$ such replica orderings, 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 tx $\prec_{(\mathcal{L},n(1-\gamma)+f+1)}$ 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 Fig. 2.

The complete leader proposal can now be defined as $B = (\mathcal{G}, \mathcal{E}_{\text{updates}}, \pi = (\mathcal{L}, \mathcal{L}_{\text{updates}}))$. Replicas will use π to verify

fairness; in SNARK-Themis, this can be done with a SNARK.

Consensus integration. The consensus design changes required are minimal. To enable the leader to construct its proposal, all replicas will first submit their local transaction orderings. The leader will now construct its proposal B and use the underlying consensus protocol to confirm it. Within this, replicas will also verify the *fairness* of the proposal (using π) before providing their signature. Since π contains all the replica orderings, the communication cost of the protocol will be $\mathcal{O}(n^2)$; this can theoretically be reduced to $\mathcal{O}(n)$ (when the leader is honest) by making π a SNARK instead. As a standard optimization, only the proposal hash needs to be signed. Note that other parts of the underlying protocol, e.g., leader election and view-change will remain exactly the same. Fig. 7 (Appendix B) provides a complete description.

Replica verification. Replicas can easily verify the fairness of the block $B = (\mathcal{G}, \mathcal{E}_{\text{updates}}, \pi)$. In particular, given all the orderings within $\pi = (\mathcal{L}, \mathcal{L}_{\text{updates}})$, replicas can simply run `FairPropose` and `FairUpdate` themselves to ensure that the proposed $\mathcal{G}$ and $\mathcal{E}_{\text{updates}}$ are both correct.

B. Finalizing the Fair Ordering

Fully specified proposals. Recall that leaders can propose partial graphs for which missing edges will be added by future leaders. A proposal is *fully specified* if its graph is a *tournament*, i.e., there exists exactly one edge between each 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 Aequitas. This is because missing edges in $\mathcal{G}$ will be added (making it fully specified) as soon as the transactions are received by the network and an honest leader is elected.

Transactions in fully specified proposals can be totally ordered and output by Themis. The algorithm `FairFinalize` describes how honest replicas can extract this final ordering.

Finalization algorithm. Given a sequence of proposals $B_1, \ldots, B_l$ agreed upon by the underlying consensus protocol whose transactions have not been fairly ordered yet, the first step is to add any missing edges; in particular, if tx and tx$'$ are transactions in $B_i.\mathcal{G}$ such that $(\text{tx}, \text{tx}') \in B_j.\mathcal{E}_{\text{updates}}$ for some $j > i$, then add the edge (tx, tx') to $B_i.\mathcal{G}$.

After the updates, let k be the largest index such that all $B_i.\mathcal{G}$ $(i \leq k)$ are now fully specified. Intuitively, these are the blocks whose transactions will be ordered by `FairFinalize`.

To order transactions in $B.\mathcal{G}$, we first compute the condensation graph $B.\mathcal{G}^*$ by collapsing its SCCs and then find the topological sorting $[V_1, V_2, \ldots, V_c]$. Note that each of V_1 to V_c represent an SCC within $B.\mathcal{G}$. To output the final total ordering, we need to now order transactions within each V_i. As an optimization, transactions whose SCC as well as all SCCs that precede it contains only solid transactions can also be finalized immediately even before $B.\mathcal{G}$ is fully specified.

Observe that it is sufficient to order transactions in each V_i arbitrarily (since intuitively they represent cyclic dependencies) using an agreed-upon function; a similar technique is used in Aequitas. As a better alternative however, we show how to take advantage of useful hidden properties within the graph to provide stronger fairness guarantees, which we describe next.

Stronger fairness guarantees within SCCs. In particular, instead of arbitrarily ordering transactions in each SCC, we will make use of Hamiltonian cycles (a graph cycle that visits every vertex exactly once) to order transactions in such a way that it connects to the ordering in other SCCs.

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 $B_j.\mathcal{E}_{\text{updates}}$, 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, e.g., by removing the edge with the smallest weight). 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 $H_{i1}, \ldots, H_{i(l_i-1)}, H'_{il_i}$.

Fig. 3: FairFinalize Algorithm

Intuitively, the upshot is that we can now *cleanly connect* the boundaries of subsequent Condorcet cycles. 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 e.g., alphabetically (as done in Aequitas [19]).

A nice practical consequence of this property is strong resilience against a particular kind of front-running attack—specifically the one where the adversary wants to place its transaction *immediately before* an honest user's transaction.

C. Formal Themis Results

We will now formally define 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. The proofs are deferred to Appendix B-A. We start with the fairness properties.

Theorem IV.1 (Themis fairness)**.** *At any time, let* $[\text{tx}_1, \ldots, \text{tx}_l]$ *be the total transaction ordering output by* Themis*. Then, the following properties are satisfied:*

1) (Batch-order-fairness) Intuitively, the ordering can be linearly partitioned in batches that satisfy batch-order-fairness. In particular, there are indices $1 = i_1 < \cdots < i_k = l + 1$ *such that* $[C_1, \ldots, C_{k-1}]$*, where each* $C_j = \{\text{tx}_{i_j}, \ldots, \text{tx}_{i_{j+1}-1}\}$*, satisfies* γ*-batch-order-fairness. Importantly, (similar to Aequitas), these batches are minimal in the sense that they are of size one when there are no cyclic ordering dependencies.*

2) (Consequent-transaction fairness) For each $i \in \{1, \ldots, l-1\}$*, it holds that* tx_i *was received before* tx_{i+1} *by at least* $n(1-\gamma)+1$ *honest nodes.*

Corollary IV.1.1. Themis *satisfies receive-order-fairness when there are no Condorcet cycles.*

We also show that Themis satisfies the standard SMR consistency and liveness properties.

Theorem IV.2. Themis *satisfies consistency and liveness.*

Audit friendliness. Themis' fairness properties are also quite friendly for auditing. In particular, an optimistic fast path can be used where the leader constructs the proposal as normal but does not send any correctness proof to the replicas; instead, the proposal 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.

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 $\gamma = 1$) in settings with only crash faults (instead of explicit adversarial behavior). Interestingly, the same protocol will work for Byzantine faults but at the cost of allowing the leader node to censor transactions.

D. SNARK-Themis

In Appendix B-B, we show a more theoretical design, SNARK-Themis, which makes use of generic arguments of knowledge (specifically, SNARKs [4], [16] for NP languages) in a black-box way to verify computation.

Through this, SNARK-Themis is able to achieve optimistic $\mathcal{O}(n)$ communication complexity. Notably, we find that existing fair-ordering protocol designs cannot easily satisfy the same property even using SNARKS. This makes SNARK-Themis the *first such fair-ordering protocol*; the communication here is asymptotically optimal and equivalent to state-of-the-art consensus protocols without fair-ordering guarantees.

V. Implementation and Benchmarks

We ran an extensive set of experiments for Themis that we detail in this section. In addition to the standard performance benchmarks (Section V-A) for throughput and latency, we also design a suite of fairness experiments in Section VI which are useful in quantifying the extent of fair transaction ordering. We begin with an overview of our implementation below.

Implementation details. Our implementation for Themis is bootstrapped from the Hotstuff protocol [37]: a state-of-the-art leader-based protocol for the partially synchronous setting. For this, we started from the authors' open-source libhotstuff codebase [1] (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.

A. Performance and Benchmarks

Experimental setup. For our performance experiments, 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 an equal number of nodes in each region. To demonstrate scalability, we vary the total number of nodes n in the system from 5 to 100. 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.

Latency and throughput. We report on the latency and throughput of Themis and compare them to standard Hotstuff run as a baseline. The results are shown in Fig. 4. We experimented with two blocksize (denoting the number of transactions in a proposal) parameters $\beta = 50$ and $\beta = 400$.

Overall, Themis provides very comparable performance to Hotstuff. Notably, we found that both latency and throughput of Themis scale in the same way as Hotstuff as the number of nodes increase. Using a larger blocksize increases the performance gap between the two protocols; this is because Themis needs to build a graph with $\mathcal{O}(\beta^2)$ edges. In the single datacenter setting, this gap was found for blocksize $\beta = 400$. This difference is not fundamental however, and we highlight that the computation can be parallelized and therefore, by using more cores per node, the performance of Themis can be made comparable to Hotstuff even for a large β. An optimized implementation of the graph algorithms used by Themis could also further boost the performance.

More importantly, we found that this difference vanishes in the geo-distributed setting due to already larger communication latency. In fact, we found that performance of both systems was identical even for very large blocksizes ($\beta = 1200$).

In essence, we expect the performance of Themis to be sufficient for most applications that require its fairness properties.

Performance comparison to other fair-ordering protocols. We do not directly compare the performance of Themis to other fair-ordering protocols: this is primarily due to lack of any comparable protocol implementation. For instance, Aequitas as well as the protocol from [9] do not come with an implementation, while an open-source implementation of Wendy only provides simulations and is not yet integrated with the consensus layer. Pompē provides an implementation using Hotstuff as the underlying protocol; their benchmarks also show comparable performance to Hotstuff (similar to Themis), and even better performance in some geo-distributed deployments, due to a fast *ordering* phase that precedes the actual consensus layer. However, since this ordering phase itself creates censorship issues within Pompē, a performance comparison to Themis would not be on an equal footing.

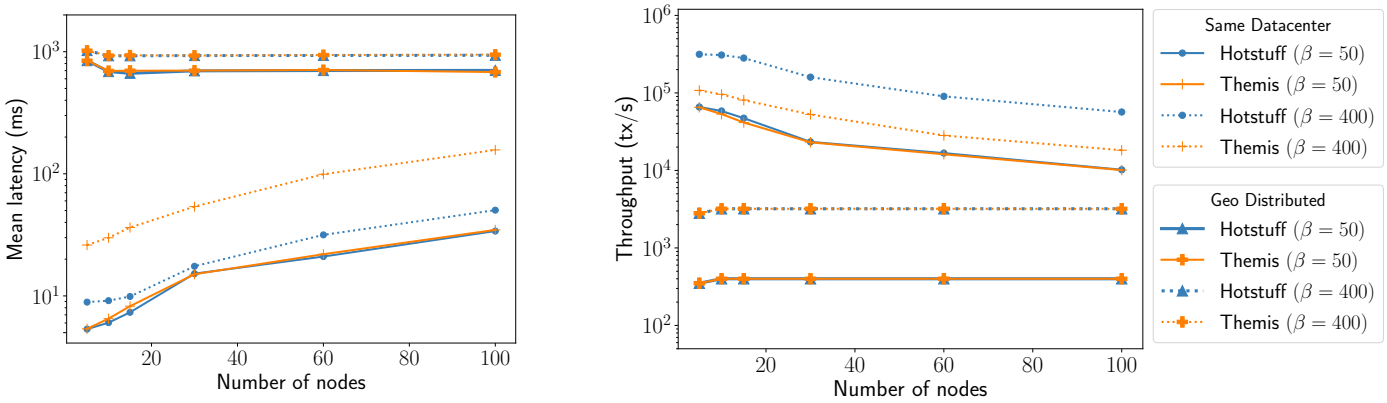


Fig. 4: Performance comparison of Themis and Hotstuff in both the same-datacenter and geo-distributed settings.

VI. Suite of Fair Ordering Experiments

To better analyze and compare different fair-ordering definitions and protocols on an equal footing, we propose a general suite of *fairness experiments*. Through these, we compare the fairness definitions from [21], [38] as well as receive-order-fairness, and batch-order-fairness. We also compare Themis to existing fair-ordering protocols.

Comparison axes. Our analysis comprises primarily of three axes—each targeted at specific setting or attack vector. We start by considering an *ideal* setting where all nodes are honest to understand the best-case scenario (Section VI-C). Our next two experiments simulate adversarial environments. In Section VI-D, we analyze susceptibility to *insertion* (e.g., front-running) attacks. In Section VI-E, we evaluate robustness to *reordering*—to what extent adversarial nodes can influence the final ordering by reordering honest user transactions.

As a related analysis point, in Section VI-F, we also evaluate censorship-resistance from a formal standpoint; here, we show a subtle censorship issue with Pompē [38].

A. Fair-Ordering Definitions

Fair ordering definitions from [21], [38]. Zhang et al. [38] and Kursawe [21] consider similar definitions of fairness which we consolidate into a single property *fair separability*.

Definition VI.1 (Fair Separability)**.** If *all* honest nodes receive tx before *any* honest node receives tx′, then all honest nodes output tx before tx′.

This property is called *timed-relative-fairness* in [21]. It is also stated as a desideratum in [15] although no protocols achieving this property are given. The *ordering-linearizability* notion considered in [38] is similar except that it is only applied when tx and tx′ are both output by the protocol. In particular, ordering-linearizability considers it acceptable if only tx′ is ouput 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′. By doing so, the Pompē 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 VI-F.

Comparing batch-order-fairness on an equal footing. While batch-order-fairness allows transactions to be output together, this is done only for those in the same cycle, and even within a batch, a total ordering will be enforced later for execution. Therefore, to quantify the strength of batch-order-fairness in the most conservative way, in all our comparisons, we consider the final *total execution ordering* guaranteed by batch-order-fairness, i.e., exactly the fairness property of Themis.

For instance, if an adversary can adversarially get tx *executed* before tx′ (e.g., through frontrunning or order-manipulation), we count it as a victory for the adversary and a failure of batch-order-fairness and our protocol. In fact, even if the adversary is able to place two transactions into the same cycle when they should not have been, we will still consider it a success for the adversary. We emphasize that even with this conservative approach, our results highlight the strength of batch-order-fairness and Themis in preventing order-manipulation attacks.

B. Simulation Environment

We created an environment to simulate the creation and network broadcast of transactions to better understand the effect of different parameters. This is useful for both our ideal setting (Section VI-C) and adversarial reordering (Section VI-E) experiments. Transactions are generated by a *sending process* with the time delays between consecutive transactions sampled from the distribution GenerationDist. For each transaction, we simulate when it would reach different consensus nodes by sampling the network latency from a distribution NetworkDist. Let $\mathsf{Send}(\mathrm{tx})$ denote the time that tx was generated and $\mathsf{Recv}(i, \mathrm{tx})$ denote the time that tx is received by replica i.

We instantiate each distribution as an exponential distribution, which is standard in networking literature. The rate of generation or arrival of messages is modeled as a Poisson process, which is equivalent to the intervals between messages (i.e., GenerationDist) being an exponential distribution. Separately, the network delay is also usually modeled as an exponential distribution. 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.[3]

The network ratio r represents how quickly new transactions are created compared to their network propagation time.

[3]The choice of $\mu = 1$ is w.l.g. since exponential distributions satisfy the scaling property — If $X \sim \mathsf{Exp}(1/\mu)$, then $kX \sim \mathsf{Exp}(1/k\mu)$.

It serves as a proxy for how far apart 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 (say 10 or 100) is typically more reflective of a geo-distributed setting.

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 include a discussion in Appendix C. In our experiments, we assume the finest possible granularity since this is the most challenging setting, but we note that in some cases, a coarser granularity may be acceptable.

Transaction comparisons. 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.

C. Ideal Setting Comparison

We first seek to understand the utility of different fair-ordering properties *in an ideal, non-adversarial setting*. Here, the only influence is from how far nodes are and any randomness in network propagation. In this ideal setting, we ask how close we can get to the magical first-in-first-out property of ordering transactions based on when they were *sent*.

Ideal ordering summary. We vary r from 10^{-2} to 10^{3}, and measure the number of transaction pairs that are correctly accounted for by different fairness definitions, i.e., their ordering is consistent with the sending process: tx is ordered before tx′ when $\mathsf{Send}(\mathsf{tx}) < \mathsf{Send}(\mathsf{tx}')$. Fig. 5 shows our results. To quantify the usefulness of batch-order-fairness conservatively, we consider it a failure of the definition if transactions end up in the same cycle. Overall, we still find that both receive-order-fairness and batch-order-fairness ensure closer to ideal ordering than fair separability for all values of r.

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. Further, observe that having a larger n makes each definition fare better.

We also find that receive-order-fairness and batch-order-fairness perform identically except for when $\gamma \approx 0.5$. The deviation between the two definitions occurs due to (strong)-Condorcet cycles (since we are in an ideal setting); intuitively, this shows that these cycles are infrequent. To underscore this, we further investigate these cycles; we find that cycles are rare, and are of small length even when they do arise (see Fig. 8 in Appendix C). This effectively demonstrates that even though the stronger receive-order-fairness is impossible without network synchrony, in many practical settings, the performance of batch-order-fairness is almost identical.

D. Network Level Insertion Attacks and Frontrunning

We now 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: On 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 can frontrunning take place before transactions are received by all nodes, i.e., even before the consensus protocol begins.

While causal ordering or privacy techniques can help hide transaction data before ordering, thus thwarting targeted frontrunning (i.e., based on transaction content), 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 privacy 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 [27]).

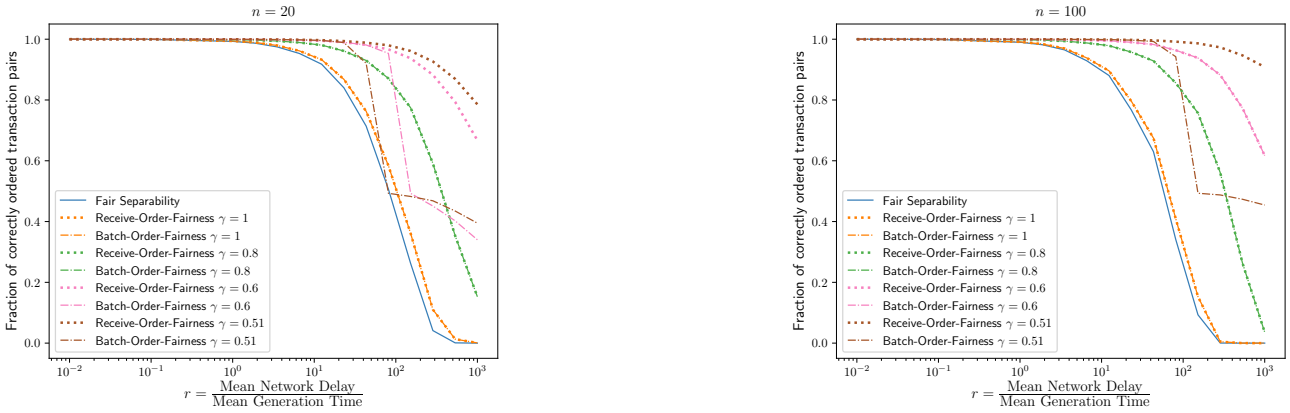
Formal frontrunning analysis in a natural network setting. As a concrete result, we show how Themis as well as the order-fairness definitions more broadly prevent frontrunning in a natural network setting where triangle inequality is respected. Abstractly, we prove that in such a network model, an adversary cannot force a frontrunning transaction tx_{adv} into the same Condorcet cycle as an honest transaction tx, and consequently will not be able to get tx_{adv} executed earlier. This serves to complement our experimental analysis of frontrunning in real network settings. The full details are given in Appendix C-B.

Remark 2 (Full adversarial control). We remark that if the adversary has full network control of when transactions (both honest and adversarial) are *input* to honest nodes, then network-level frontrunning is trivial in any protocol. Therefore our experimental analysis needs to be limited to real-world networks rather than ones with full adversarial control.

More specifically, the usual consistency and liveness properties remain intact even if the adversary has such a full network control. Interestingly, even the fair-ordering property holds based on the input orderings, but since the adversary can essentially manipulate the input orderings of honest nodes themselves given full network control, the property of the protocol no longer reflects any intuitive notion of fairness.

Indeed, this limitation led to Kelkar et al. [19] formalizing two networks: the (standard) *internal* network (for communication amongst consensus nodes) and the *external* network (for all transaction submission). The adversary has full control over the internal network but does not control the external network. [19] notes that such a power would be similar in spirit to controlling e.g., the user's access to the internet. This ensures that the input orderings of honest nodes cannot be manipulated by the adversary. We highlight that similar assumptions of the adversary not having full control over transaction submission are present in all previous works [18], [21], [38].

Experiment details. We used a real network rather than a simulation for this experiment to understand the possibility of frontrunning in practice. Through measurement of communication latency between nodes, 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



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

Fig. 5: Comparison of the fraction of transaction pairs correctly ordered by different fairness notions as a function of r.

depends on many other factors — for instance, the expected profit of a particular frontrunning strategy, the variance in network latency, the presence of competing entities etc.

Frotrunnable pairs. We now describe frontrunnable pairs for the fairness notions considered. 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 (A, B) is frontrunnable since 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 (receive/batch)-order-fairness, (A, B) is frontrunnable 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. For order-fairness, we 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).

Experimental results. We started with a simple toy setup by deploying one node each in five different AWS regions. We defer the details to Appendix C and directly proceed here with our analysis using a large scale network with 250 servers.

For this, we used the publicly available WonderProxy dataset [36] which contains ping times (in both directions) between each server pair 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 compute the number of frontrunnable pairs. We average the results over 100 randomly chosen committees. Table II reports on our findings for the number of frontrunnable pairs for different fairness definitions.

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

TABLE II: 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 dataset [36]. We chose $n = 20, f = 4$ and $n = 100, f = 24$ for the second and third row respectively.

For fair separability, on average 82% (for $n = 20$) and 94% (for $n = 100$) pairs were found frontrunable. 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.

E. Robustness to Adversarial Reordering

In this section, we evaluate to what extent adversarial nodes can influence the final transaction ordering through *reordering*, i.e., by claiming within the protocol to have received user transactions in a different order than they were actually received.

Experiment intuition. All fair-ordering properties guarantee some robustness to adversarial reordering. For instance when all honest nodes receive tx before any honest node receives tx$'$, fair separability guarantees that tx will be output before tx$'$ no matter what the adversary does. Yet this says nothing about *how often* the fairness property will apply for transaction pairs or what happens when the property *does not apply* (e.g. is the protocol fairness all or nothing or does it smoothly degrade). Understanding this will paint a more complete picture of the overall robustness of a protocol to adversarial reordering.

Dependence on transaction *closeness*. Zhang et al. [38] 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

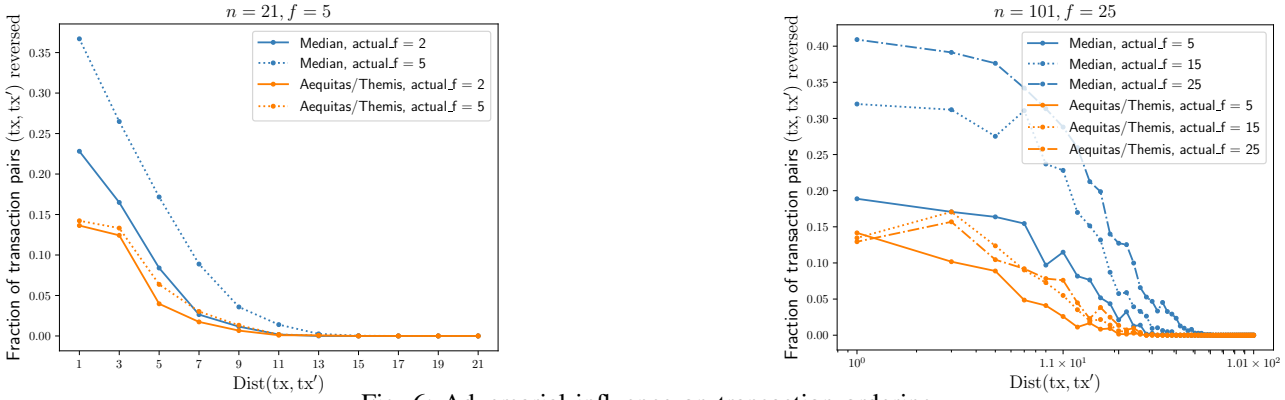


Fig. 6: Adversarial influence on transaction ordering

— close in the sense that even their honest ordering could have been inverted by small network delays. Effectively, this would signify that the impact of the adversary is similar to that of small network fluctuations.

As a proxy for "closeness," we use the distance metric $\mathsf{Dist}(\mathrm{tx}, \mathrm{tx}') = |\#_{(\mathrm{tx}<\mathrm{tx}')} - \#_{(\mathrm{tx}'<\mathrm{tx})}|$ where $\#_{(x<y)}$ denotes the number of nodes that received x before y. Dist varies from $(n \bmod 2)$ to n in increments of 2. A small Dist implies a "fragile" pair, where the number of nodes that received tx earlier is roughly the same as the number that received tx′ earlier, while a large Dist means that one of the 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 yet powerful 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. It generalizes attempting to flip the ordering of a single transaction pair. 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.

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 total ordering) of a transaction pair with a given distance. As a more accurate comparison, for Themis, even if the adversary is able to put transactions into the same cycle when they should not have been, we consider as a failure for our protocol.

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. Fig. 6 shows the results.

We compare Themis (with $\gamma = 1$) to a simple *median* protocol which outputs transactions sorted in ascending order by their median timestamp. Note that this abstracts exactly the relevant fairness component of Pompē [38] and Wendy [21].

Reordering insights. 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 the $n = 101$ figure is in log-scale to better highlight reordering for transactions pairs with small distance (since reordering was absent for large Dist).

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 Dist, 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.

Still, overall, we found that for a given actual number of corruptions, it is much 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 through a small number of corruptions than it was against Themis using the maximum number of corruptions. Further, we also note that for this specific adversarial strategy, the additional gain against Themis through extra corruptions is small.

F. Censorship Resistance and Attacks

Censorship resistance is defined as the property that if an honest client sends a transaction to the consensus nodes, then it will eventually be output by the protocol. While many existing protocols provide censorship resistance (e.g., [28]), we emphasize that on its own this does not guarantee that the *ordering* of transactions is "fair."

In our setting, we find that there can be inherent tradeoffs between liveness and censorship resistance for some existing fair-ordering designs—in particular, if an earlier phase is used to first *order* transactions before consensus.

Censorship attack on [38]. Pompē [38] uses a pre-protocol (specifically an ordering phase prior to consensus) to compute the median timestamp for transactions, with the fair ordering being taken according to the 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 VI.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 timestamp for tx 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 *throughout* the protocol on the time for an honest node to complete the ordering phase, which cannot be assumed in the partially synchronous setting. The synchrony requirement is also especially of concern in the common setting where clients (rather than nodes) submit transactions since the bound would now have to involve the client's connection.

While this was partially acknowledged in [38] as a possible way a Byzantine node could selectively chose which of its transactions to submit to the consensus phase, the stronger censorship attack vector was 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 transaction) is in charge of accumulating timestamps from the nodes in the ordering phase before proceeding to the consensus phase, as [38] notes, it is not possible to distinguish between whether the transaction was sent to the consensus phase in time or not. In fact, because of this, a view change also cannot be triggered in order to get back censorship resistance. Consequently, either there will be a liveness failure (by potentially waiting forever), or censorship resistance will not be achieved.

From our analysis, the core 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 along with Themis and Aequitas [19] are not susceptible to such censorship; if a transaction is received by all honest replicas, it will not be censored, and moreover will be sequenced in a fair order.

VII. Related Work

While our experimental suite contains several useful comparisons, we use this section to highlight a few more nuances.

Causal ordering and privacy. Prior work in classical distributed systems [7], [31] had considered a limited notion of preventing reordering based on *transaction content*, but as mentioned earlier (and also identified by [18], [19], [21]), 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 [5], time-lock encryption, verifiable delay functions (VDFs) [33], or even multi-party-computation (MPC) among the protocol nodes [24]. 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 our knowledge lacking formal analysis on their properties.

Many of these protocols also randomly order transactions within an epoch. As noted in, e.g., [18], such techniques are susceptible to flooding attacks (e.g., [27]) where now with high probability, some adversarial transaction will be be unfairly ordered ahead of the honest 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.

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 $\mathcal{O}(n^2)$ communication while Aequitas 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 but expensive compiler to make any standard consensus protocol fair, Themis can minimally modify any leader-based protocol to give it the same notion of fairness.

We note that the structure of the dependency graph constructed by Themis is close but somewhat different than the one from Aequitas. We include a brief discussion in Appendix D. Further, although both Aequitas and Themis satisfy batch-order-fairness, Themis's use of Hamiltonian cycles allows for connecting different Condorcet cycles appears to provide a stronger notion of "fairness within a batch." This implies that there are other subtleties within the graph structure that are not captured by the fairness definition. We leave the problem of uncovering these to future work.

Comparison to Cachin et al. [8], [9]. Concurrent to this work, Cachin et al. [8] defined κ-differential-order-fairness (for $\kappa \geq 0$), which states that if $b(\mathrm{tx}_1, \mathrm{tx}_2) > b(\mathrm{tx}_2, \mathrm{tx}_1) + \kappa + 2f$, where $b(x, y)$ denotes the number of honest nodes that receive x before y, then tx_1 will be delivered no later tx_2. In Theorem D.1, we prove that this definition is simply a reparameterization of batch-order-fairness. The original protocol described in [8] contained a major bug which was later fixed in a subsequent revision [9]. Unfortunately, this revised protocol offers significantly weaker security; the proof for liveness only works when *all nodes are honest*, which likely makes it less useful in any practical setting.

VIII. Conclusion

We introduced Themis, a consensus protocol that satisfies a strong fair transaction-ordering property, 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.

References

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

[2] Condorcet paradox, Accessed Aug. 2021. https://wikipedia.org/wiki/Condorcet_paradox.

[3] DeFi Pulse, Accessed Aug. 2021. defipulse.com.

[4] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In ITCS, page 326–349, 2012.

[5] Lorenz Breidenbach, Phil Daian, Ari Juels, and Florian Tramèr. To sink frontrunners, send in the submarines, 2017. https://hackingdistributed.com/2017/08/28/submarine-sends/.

[6] Benedikt Bünz and Ben Fisch. Transparent snarks from dark compilers. In EUROCRYPT, pages 677–706, 2020.

[7] Christian Cachin, Klaus Kursawe, Frank Petzold, and Victor Shoup. Secure and efficient asynchronous broadcast protocols. In CRYPTO, pages 524–541, 2001.

[8] Christian Cachin, Jovana Micic, and Nathalie Steinhauer. Quick order fairness. In FC, 2022.

[9] Christian Cachin, Jovana Micic, Nathalie Steinhauer, and Luca Zanolini. Quick order fairness, 2021. arXiv:2112.06615.

[10] Paul Camion. Chemins et circuits hamiltoniens des graphes complets. Comptes Rendus de l'Académie des Sciences de Paris, 249:2151–2152, 1959.

[11] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance. In OSDI, pages 173–186, 1999.

[12] Philip Daian, Steven Goldfeder, Tyler Kell, Yunqi Li, Xueyuan Zhao, Iddo Bentov, Lorenz Breidenbach, and Ari Juels. Flash boys 2.0: Frontrunning in decentralized exchanges, miner extractable value, and consensus instability. In IEEE S&P, pages 585–602, 2020.

[13] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Consensus in the presence of partial synchrony. J. ACM, 35(2):288–323, 1988.

[14] Shayan Eskandari, Seyedehmahsa Moosavi, and Jeremy Clark. Sok: Trans- parent dishonesty: Front-running attacks on blockchain. In FC, pages 170–189, 2019.

[15] Juan Garay and Aggelos Kiayias. Sok: A consensus taxonomy in the blockchain era. In CT-RSA, pages 284–318, 2020.

[16] Jens Groth. On the size of pairing-based non-interactive arguments. In EUROCRYPT, pages 305–326, 2016.

[17] Chi Ho, Danny Dolev, and Robbert van Renesse. Making distributed systems robust. In OPODIS, pages 232–246, 2007.

[18] Mahimna Kelkar, Soubhik Deb, and Sreeram Kannan. Order-fair consensus in the permissionless setting. https://eprint.iacr.org/2021/139.

[19] Mahimna Kelkar, Fan Zhang, Steven Goldfeder, and Ari Juels. Order-fairness for Byzantine consensus. In CRYPTO, pages 451–480, 2020.

[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, the good little fairness widget: Achieving order fairness for blockchains. In ACM AFT, pages 25–36, 2020.

[22] Klaus Kursawe. Wendy grows up, 2021. https://vega.xyz/papers/Wendy_Grows_Up.pdf.

[23] Leslie Lamport, Robert Shostak, and Marshall Pease. The Byzantine generals problem. TOPLAS, 4(3):382–401, 1982.

[24] Yunqi Li, Sylvain Bellemare, Mikerah Quintyne-Collins, and Andrew Miller. Honeybadgerswap: Making mpc as a sidechain, 2021. https://medium.com/initc3org/honeybadgerswap-making-mpc-as-a-sidechain-364bebdb10a5.

[25] Mary Maller, Sean Bowe, Markulf Kohlweiss, and Sarah Meiklejohn. Sonic: Zero-knowledge snarks from linear-size universal and updatable structured reference strings. In CCS, page 2111–2128, 2019.

[26] Y. Manoussakis. A linear-time algorithm for finding hamiltonian cycles in tournaments. Discrete Appl. Math., 36(2):199–201, 1992.

[27] Alex Manuskin. The fastest draw on the blockchain: Ethereum backrunning, 2020. https://medium.com/@amanusk/the-fastest-draw-on-the-blockchain-bzrx-example-6bd19fabdbe1.

[28] Andrew Miller, Yu Xia, Kyle Croman, Elaine Shi, and Dawn Song. The honey badger of BFT protocols. In ACM CCS, pages 31–42, 2016.

[29] Rafael Pass and Elaine Shi. Fruitchains: A fair blockchain. In PODC, pages 315–324, 2017.

[30] Kaihua Qin, Liyi Zhou, and Arthur Gervais. Quantifying blockchain extractable value: How dark is the forest? In IEEE S&P, pages 198–214, 2022.

[31] Michael K. Reiter and Kenneth P. Birman. How to securely replicate services. ACM Trans. Program. Lang. Syst., 16(3):986–1009, 1994.

[32] Srinath Setty. Spartan: Efficient and general-purpose zksnarks without trusted setup. In CRYPTO, pages 704–737, 2020.

[33] StarkWare. Presenting: Veedo a stark-based vdf service. https://medium.com/starkware/presenting-veedo-e4bbff77c7ae.

[34] Robert Tarjan. Depth-first search and linear graph algorithms. SIAM J. Comput., 1(2):146–160, 1972.

[35] Riad S. Wahby, Ioanna Tzialla, Abhi Shelat, Justin Thaler, and Michael Walfish. Doubly-efficient zksnarks without trusted setup. In IEEE S&P, pages 926–943, 2018.

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

[37] Maofan Yin, Dahlia Malkhi, Michael K. Reiter, Guy Golan-Gueta, and Ittai Abraham. Hotstuff: BFT consensus with linearity and responsiveness. In PODC, pages 347–356, 2019.

[38] Yunhao Zhang, Srinath Setty, Qi Chen, Lidong Zhou, and Lorenzo Alvisi. Byzantine ordered consensus without Byzantine oligarchy. In OSDI, pages 633–649, 2020.

[39] Liyi Zhou, Kaihua Qin, Christof Ferreira Torres, Duc V Le, and Arthur Gervais. High-frequency trading on decentralized on-chain exchanges. In IEEE S&P, 2021.

Appendix A
Further details on Condorcet cycles

We provide an explicit algorithm (Algorithm 1) for generating an arbitrarily large strong-Condorcet cycle for any valid n, f, γ (which also immediately implies the result for (weak)-cycles). For simplicity, we provide details for the synchronous setting where the "claimed" input orderings of all nodes are taken into account, but this can also be extended to only using $n - f$ orderings. Abstractly, the algorithm inductively constructs transaction input orderings for each node in a way that all transactions will be part of the same strong-cycle.

Suppose that the system nodes are denoted by the set $\{0, \ldots, n-1\}$. To create a strong-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 iteratively adding a new transaction to all input lists while maintaining the invariant that all current transactions form a strong-cycle. For the initial transaction tx_0, the input ordering txlist_j of each node j is taken simply as $[\mathsf{tx}_0]$. To add a later transaction tx_i, a set S_i of 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

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 $S = \{\mathsf{start}, \ldots, n-1\} \cup \{0, \ldots, \mathsf{end}\}$;
  for $j \in S$ **do** Insert tx_i right before tx_{i-1} in txlist_j;
  for $j \in N \setminus S$ **do** Append tx_i at the end of txlist_j;
  $\mathsf{start} \leftarrow (\mathsf{start} + 1) \mod n$
end

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 strong-Condorcet cycle.

Weak-liveness in terms of cycles. Looking at the weak-liveness definition from Kelkar et al. [19], it is easy to see that any tx′ part of the same cycle as tx will need to be added to T. This means that if the cycle extends for arbitrarily long, new transactions will keep being added to T, delaying the delivery of transactions in that cycle. The Aequitas protocol, therefore, can only guarantee liveness after the current cycle completes, which may take arbitrarily long. While the weak-liveness definition mentions one honest node, we note that Aequitas achieves a stronger property (for a general γ) where $n(1-\gamma)+1$ is used as the threshold instead.

Definition A.1 (Weak-liveness [19])**.** Consider tx received by all protocol nodes and define the set T built recursively as follows: (1) Add tx to T; (2) For any $m \in \mathsf{T}$, add to T all m' such that at least one honest node has received m' before m. At any point, if all honest nodes have received all transactions in T, then tx will eventually be delivered by all honest nodes.

APPENDIX B
Themis DETAILS

Fig. 7 contains a complete description of Themis from the perspective of both the leader and the replica nodes.

A. *Proofs for* Themis

In this section, we formally prove the theorem statements from the main text. We begin with some helpful lemmas.

Lemma B.1 (Dependency Graph Properties)**.** *The following 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* $(\mathrm{tx}, \mathrm{tx}')$ *and* $(\mathrm{tx}', \mathrm{tx})$ *is present in the output graph.*

Proof: Let $\mathcal{G}$ be the dependency graph constructed by the algorithm `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 solid transaction.

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 $\mathrm{tx} \prec_{\mathcal{L}, n(1-\gamma)+f+1} \mathrm{tx}'$ or $\mathrm{tx} \prec_{\mathcal{L}, n(1-\gamma)+f+1} \mathrm{tx}'$ holds. Since only one edge is added even when both holds, we conclude that exactly one of the edges $(\mathrm{tx}, \mathrm{tx}')$ and $(\mathrm{tx}', \mathrm{tx})$ is contained in the final graph output. ∎

Complete Themis **Description**

Round leader protocol:

1) Wait for $n - f$ correctly signed lists $(\mathsf{List}_i, \mathsf{Update}_i)$ from replicas. Let $\mathcal{L} = \{\mathsf{List}_i\}$ and $\mathcal{L}_{\text{updates}} = \{\mathsf{Update}_i\}$
2) Run `FairPropose`$(\mathcal{L})$ and `FairUpdate`$(\mathcal{L}_{\text{updates}})$ to get $\mathcal{G}$ and $\mathcal{E}_{\text{updates}}$ respectively.
3) Run the underlying Consensus protocol to agree on a valid $B = (\mathcal{G}, \mathcal{E}_{\text{updates}}, \pi = (\mathcal{L}, \mathcal{L}_{\text{updates}}))$ and append it to the list of agreed upon blocks.

Replica protocol (for N_i):

1) Let List_i denote the ordered list of transactions received by N_i that are not yet part of any leader proposal. Let Update_i denote the ordered list of transactions received by N_i that are correspond to vertices with missing edges in some previous leader proposal.
2) Send $\mathsf{List}_i, \mathsf{Update}_i$ to the current round leader and verify that the leader's response is a valid proposal (by checking the SNARK proofs).
3) Participate in the underlying Consensus protocol to agree on a block $B = (\mathcal{G}, \mathcal{E}_{\text{updates}})$ and append it to the list of agreed upon blocks.

Constructing the final ordering (for N_i):

At any point, N_i can do the following to append new transactions to its final ordering:

1) Let B_j be the first block whose transactions have not yet been output in the final ordering by N_i, and let B_k be the final block.
2) Run `FairFinalize` with $B_j, \ldots, B_k$ to obtain an ordering $[\mathrm{tx}_p, \ldots, \mathrm{tx}_q]$ of transactions.
3) Append $\mathrm{tx}_p, \ldots, \mathrm{tx}_q$ to the final transaction ordering.

Fig. 7: Complete Description of Themis

Lemma B.2. *If* `FairPropose`$(\mathcal{L})$ *outputs a solid transaction* tx *and does not output* tx′ *(and* tx′ *has not been previously proposed), then there are at least* $n(1-\gamma)+1$ *honest nodes that have received* tx *before* tx′*.*

Proof: If tx′ is blank, 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′. ∎

Lemma B.3 (Contiguous Cycles)**.** *If* `FairPropose`$(\mathcal{L})$ *outputs* tx *and does not output* tx′ *(and* tx′ *has not been previously proposed), then either* tx *is an earlier cycle than* tx′ *or* tx *and* tx′ *are in the same cycle. Further, if* tx′ *is received before* tx *by* γn *nodes, then* tx *and* tx′ *are in the same cycle.*

Proof: We first show that if tx is output and tx′ is not, then it cannot be the case that tx′ is in an earlier cycle. To see why, assume for contradiction that tx′ is an earlier cycle. This means that there is no sequence of transactions [tx =

$\mathsf{tx}_0, \mathsf{tx}_1, \ldots, \mathsf{tx}_l = \mathsf{tx}']$ such that for each j it holds that tx_j was received before tx_{j+1} by $n(1-\gamma)+1$ honest nodes. But this contradicts the fact that tx was output and tx' was not, and therefore we conclude that tx' cannot be in an earlier cycle.

Now, for the case when tx' is received before tx by γn nodes, 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 B.2), which contradicts the given condition. Therefore tx must be shaded and consequently, it was included in the proposal since there is a sequence of transactions $[\mathsf{tx} = \mathsf{tx}_0, \mathsf{tx}_1, \ldots, \mathsf{tx}_l]$ such that the graph contains each of the edges $(\mathsf{tx}_i, \mathsf{tx}_{i+1})$ 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 $[\mathsf{tx}_0, \mathsf{tx}_1, \ldots, \mathsf{tx}_l, \mathsf{tx}']$ is a cycle. ■

We are now ready to prove the main theorems for Themis.

Proof of Theorem IV.1: 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 B.3, we know that tx' must be in the same or later 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 B.3, we know that tx' must be in the same 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 cycle as tx or a later one.

The above cases show that the final transaction ordering final transaction ordering can be split into contiguous cycles. In other words, either tx will be output before tx' or some contiguous transaction sequence containing both tx and tx' will be a cycle. Consequently, there are indices $1 = i_1 < \cdots < i_k = l+1$ such that $[C_1, \ldots, C_{k-1}]$, with batches $C_j = \{\mathsf{tx}_{i_j}, \ldots, \mathsf{tx}_{i_{j+1}-1}\}$, satisfies γ-batch-order-fairness.

Now, to show consequent-transaction fairness, in the output ordering, consider transactions tx_j and tx_{j+1} where tx_{j+1} is output immediately after tx_j. The following cases arise:

(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, $(\mathsf{tx}_j, \mathsf{tx}_{j+1})$ is an edge in the 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 $(\mathsf{tx}_j, \mathsf{tx}_{j+1})$ is already an edge. 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 B.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. ■

It is easy to see that, similar to Aequitas, the batches are minimal in the sense that they are of size one when there are no cycles. Consequently, in such a case, Themis satisfies the stronger receive-order-fairness notion. We now show that Themis satisfies consistency and (standard) liveness.

Proof of Theorem IV.2: 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. ■

B. SNARK-Themis *Details*

We now provide details for our SNARK-Themis protocol. Intuitively, for this, instead of forwarding the replica local orderings to all replicas, the leader will now create a SNARK to prove correctness of its proposal.

SNARK preliminaries. 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 , 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], [25]). We do not require any zero-knowledge property.

SNARK proof for the leader proposal. 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 $\texttt{FairPropose}(\mathcal{L})$ outputs $\mathcal{G}$ where $\mathcal{L} = \{\mathsf{List}_{i_1}, \ldots, \mathsf{List}_{i_{n-f}}\}$. The string representation can be decided upon deterministically, for example, by choosing vertices in alphabetical order. We use general-purpose SNARKS for the language L to prove correctness. A similar correctness proof can be constructed for the update list $\mathcal{E}_{\text{updates}}$.

Finally, the overall proposal block B by the leader will be the tuple $(\mathcal{G}, \mathcal{E}_{\text{updates}}, \pi = (\pi_1, \pi_2))$ where $\mathcal{G}$ is the graph for newly proposed transactions, $\mathcal{E}_{\text{updates}}$ is the set of update edges, 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 no longer depends on the number of nodes, enabling the underlying consensus algorithm to achieve communication complexity $\mathcal{O}(n)$.

SNARKs for Wendy [21] and Aequitas [19]. Wendy and Aequitas both require an $\mathcal{O}(n^2)$ communication phase where

all nodes gossip transactions to all others. This is critically necessary for censorship resistance, and we found no easy way to improve the communication complexity using SNARKs.

Improved complexity for Pompē [38]. The Pompē protocol has a communication complexity of $\mathcal{O}(n^2)$. The general 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, in standard deployments, *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 clients. Since the CRS size is dependent on n for known constant-size SNARKs, this means that in the standard setting where clients and nodes can be distinct, Pompē will not have optimistic communication linear in n.

While SNARKs without trusted setup can be used instead, current state-of-the-art constructions [6], [32], [35] 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.

Appendix C
Experiment Details

We provide experiment details deferred from the main text.

Network ratio and 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. A granularity of g (defined below) 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.

Definition C.1 (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.

A. Ideal Setting Details

Comparing receive and batch order-fairness. For $n = 100$, note the plots for the two definitions 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 $n/2$—for large n, this effect is not seen until γ gets close to 0.5.

Abstractly, this deviation 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? We found that in practical settings, Condorcet cycles are small, and therefore, effectively the two definitions are quite similar.

Cycle size for order-fairness. We find the number of transactions that are present in (strong)-Condorcet cycles of each size in the ideal setting. Fig. 8 contains 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.

B. Frontrunning Experiment Details

Toy AWS setting. We start with a simple geo-distributed network with one node each in five different 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 III contains ping times averaged over 1000 samples.

Out of 20 total node pairs (A, B), we found that 10 are frontrunnable (i.e., A can frontrun transactions originating from B) in the case of fair separability. As a concrete example based on Table III, 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. On the other hand, there were no instances of triangle inequality being broken in our toy setting, and as a consequence, there were no frontrunnable pairs in the case of order-fairness.

Frontrunning analysis. We now provide an analysis of how Themis can handle frontrunning under a common network model where triangle inequality is respected. In a network where triangle inequality holds, an adversary cannot see an honest transaction and then create and send an adversarial transaction (i.e., an attempt to frontrun) in a way that any honest node receives the adversarial transaction first. In other words, if tx is an honest transaction and m is an adversarial transaction sent afterwards, then all honest nodes should receive tx before m. In this network setting, we can show that Themis prevents frontrunning—i.e., the adversarial transaction will be ordered strictly after the honest transaction it is trying to frontrun. We show this through the following lemma.

Lemma C.2. *An adversary cannot force an adversarial transaction m to be in the same cycle as any honest transaction under the above mentioned network model.*

Proof: To see why, consider a transaction dependency graph $\mathcal{G}^*$ that contains an edge from tx to tx$'$ is a *majority* of nodes claim to have received tx earlier. First note that

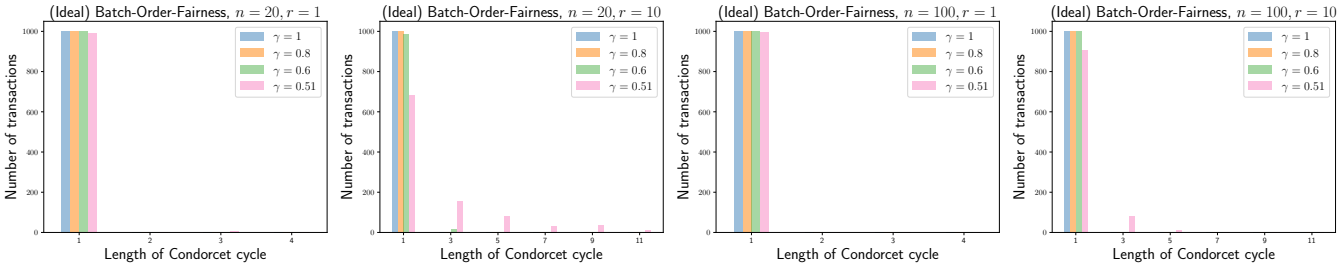


Fig. 8: Comparison of the number of transactions in a cycle of specific lengths. A value of 1 means that there is no cycle.

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

this graph is complete (i.e., there is one edge between any transaction pair), and that any cycle in the actual dependency graph will also be a cycle in this graph.

Now, since $\mathcal{G}^*$ is complete, if a cycle within $\mathcal{G}^*$ contains m, then m is also contained within a cycle of length 3, i.e., there exists some x, y such that $x \to y \to m \to x$ is a cycle. Note that this is directly true for Themis since its dependency graph is already complete.

Now, if x, y are both honest user transactions, then by our network assumption, all honest users have received y before m and m before x. This implies that no honest node will have received x before y and so the cycle cannot exist. On the other hand, w.l.g, if only x is honest, then again by our network assumption, all honest users must have received m before x and x before y. Therefore, no honest node has received y before m and therefore again the cycle cannot exist. ■

This also shows that for Themis, and all the order-fairness variants more broadly, assuming triangle inequality holds, an adversary will not be able to get its transaction ordered before the honest user's transaction.

Appendix D
Additional Insights on Related Works

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

Equivalence of differential-order-fairness [9] and batch-order-fairness. The following theorem shows an equivalance between the two definitions:

Theorem D.1. *Consider specific n, f. A protocol satisfies κ-differential-order-fairness ($\kappa \geq 0$) if and only if it also satisfies γ-batch-order-fairness where $\gamma \geq \frac{1}{2} + \frac{3f+1+\kappa}{2n}$.*

Proof: Define $b(\mathrm{tx}, \mathrm{tx}')$ to be the number of honest nodes that receive tx before tx′. Notice that γ-batch-order-fairness implies γ'-batch-order-fairness for any $\gamma \leq \gamma' \leq 1$.

($\Longrightarrow$). Suppose that a protocol Π satisfies γ^*-batch-order-fairness for $\gamma^* \in (\frac{1}{2}, 1]$ such that Π does not satisfy the property for any $\gamma < \gamma^*$. Now, consider any transactions $\mathrm{tx}_1, \mathrm{tx}_2$ such that $b(\mathrm{tx}_1, \mathrm{tx}_2) - b(\mathrm{tx}_2, \mathrm{tx}_1) > \kappa + 2f$. Since we have $b(\mathrm{tx}_1, \mathrm{tx}_2) + b(\mathrm{tx}_2, \mathrm{tx}_1) \geq n - f$, adding the two, we get $2 \cdot b(\mathrm{tx}_1, \mathrm{tx}_2) \geq n + f + \kappa + 1$, or equivalently, $b(\mathrm{tx}_1, \mathrm{tx}_2) \geq \frac{n+f+\kappa+1}{2}$.

Now, since Π satisfies γ^*-batch-order-fairness where γ^* is minimal, if $\frac{n+f+\kappa+1}{2} = \gamma^* n - f$, then Π will order tx_1 no later than tx_2, i.e., it will also satisfy κ-differential-order-fairness. This implies $\gamma^* = \frac{1}{2} + \frac{3f+1+\kappa}{2n}$.

($\Longleftarrow$). Suppose that a protocol Π satisfies κ-differential-order-fairness. Define $\gamma^* = \frac{1}{2} + \frac{3f+1+\kappa}{2n}$. Now, consider two transactions tx_1 and tx_2 such that $\gamma^* n$ nodes have received tx_1 before tx_2. This means that $\gamma^* n - f = \frac{n+f+1+\kappa}{2}$ honest nodes have received tx_1 before tx_2, i.e., $b(\mathrm{tx}_1, \mathrm{tx}_2) \geq \frac{n+f+1+\kappa}{2}$. Therefore, $b(\mathrm{tx}_2, \mathrm{tx}_1) \leq (n - f) - \frac{n+f+1+\kappa}{2} = \frac{n-3f-\kappa-1}{2}$. Consequently, $b(\mathrm{tx}_1, \mathrm{tx}_2) - b(\mathrm{tx}_2, \mathrm{tx}_1) \geq 2f + \kappa + 1$, i.e., $b(\mathrm{tx}_1, \mathrm{tx}_2) > b(\mathrm{tx}_2, \mathrm{tx}_1) + 2f + \kappa$ which means that Π will deliver tx_1 no later than tx_2. Therefore, Π satisfies γ^*-batch-order-fairness, and in turn also for any $\gamma \geq \gamma^*$. ■

Remark 3 (κ parameterization). First notice that the $\kappa = 0$ and $n = 3f+1$ case corresponds to the $\gamma = 1$ case for batch-order-fairness. Note that the protocol from [9] works for $n = 3f+1$ only when all nodes are honest, and in this setting, Aequitas and Themis both also satisfy $\gamma = 1$ batch-order-fairness.

For a given κ, if tx_1 and tx_2 are such that $b(\mathrm{tx}_1, \mathrm{tx}_2)$ and $b(\mathrm{tx}_2, \mathrm{tx}_1)$ differ by more than $\kappa + 2f$, then their fairness is "considered" by the protocol. To provide the same fairness as batch-order-fairness for a $\gamma = \frac{1}{2} + \epsilon$ where ϵ is small, the minimum acceptable difference between $b(\mathrm{tx}_1, \mathrm{tx}_2)$ and $b(\mathrm{tx}_2, \mathrm{tx}_1)$ needs to also be made small, for which the only option is to make f much smaller than n. This corresponds exactly to the bound for batch-order-fairness where the corruption threshold

reduces as γ tends to $\frac{1}{2}$.

As an example, when $n = 3f + 1$, the only κ that makes sense is $\kappa = 0$, which corresponds only to $\gamma = 1$ (i.e., the weakest version) in the batch-fairness context. This means that to get a stronger fairness property, the only option is to reduce the corruption ratio f/n and this will in turn also correspond to a stronger γ (i.e., closer to 1/2).