Condorcet Attack Against Fair Transaction Ordering

Mohammad Amin Vafadar
University of Alberta, Canada

Majid Khabbazian
University of Alberta, Canada

Abstract

We introduce the *Condorcet attack*, a new threat to fair transaction ordering. Specifically, the attack undermines batch-order-fairness, the strongest notion of transaction fair ordering proposed to date. The batch-order-fairness guarantees that a transaction `tx` is ordered before `tx`$'$ if a majority of nodes in the system receive `tx` before `tx`$'$; the only exception (due to an impossibility result) is when `tx` and `tx`$'$ fall into a so-called "Condorcet cycle". When this happens, `tx` and `tx`$'$ along with other transactions within the cycle are placed in a batch, and any unfairness inside a batch is ignored.

In the Condorcet attack, an adversary attempts to undermine the system's fairness by imposing Condorcet cycles to the system. In this work, we show that the adversary can indeed impose a Condorcet cycle by submitting as few as two otherwise legitimate transactions to the system. Remarkably, the adversary (e.g., a malicious client) can achieve this even when all the nodes in the system behave honestly. A notable feature of the attack is that it is capable of "trapping" transactions that do not naturally fall inside a cycle, i.e. those that are transmitted at significantly different times (with respect to the network latency). To mitigate the attack, we propose three methods based on three different complementary approaches. We show the effectiveness of the proposed mitigation methods through simulations, and explain their limitations.

2012 ACM Subject Classification Security and privacy → Distributed systems security

Keywords and phrases Transaction ordering, fairness, Condorcet cycle

Digital Object Identifier 10.4230/LIPIcs.AFT.2023.15

1 Introduction

The first blockchain application, Bitcoin, emerged in the midst of the financial crisis of 2008, caused in part by the excessive trust placed in centralized institutions. Blockchain technology changed this. In blockchain, there is no central authority or intermediary controlling the entire system. Instead, transactions are validated and included through a consensus mechanism among the participating parties. Decentralization also promotes transparency and reduces the possibility of fraud or corruption since all transactions are publicly recorded and visible to all participants on the network.

Despite the decentralized nature of blockchain systems, the ordering of transactions is carried out in a centralized manner; the miner/validator who creates a block determines the ordering of transactions within the block. This gives too much power to a single entity as the success and profitability of a transaction can be determined by the order in which the transaction appears in a block [6, 1, 8, 9, 16]. For instance, when a Non-Fungible Token (NFT) is dropped in a given block, transactions positioned earlier in the block have a higher chance of acquiring the NFT compared to those placed later.

To address this issue, several existing works [12, 20, 11, 4, 10, 13] proposed decentralized methods for handling transaction ordering, where instead of a single node, a committee of nodes collectively decide on the ordering of received transactions. At the core of these methods, each node in the system reports a list of transactions in the order the node has

received them. The system then generates and agrees on a "fair" ordering by taking the reported orderings into account.

Finding a fair ordering is not trivial. For instance, suppose that for any two transactions $\texttt{tx}_1$ and $\texttt{tx}_2$, we require $\texttt{tx}_1$ to be placed before $\texttt{tx}_2$ if a large majority of nodes in the system claim to have received $\texttt{tx}_1$ before $\texttt{tx}_2$. Despite being a primitive requirement, no method can provide a guarantee due to an impossibility result rooted in social choice theory [2]. As an example, consider a system consisting of three nodes, where each node has received three transactions: $\texttt{tx}_1$, $\texttt{tx}_2$, and $\texttt{tx}_3$. Suppose the nodes report the ordering as $[\texttt{tx}_1, \texttt{tx}_2, \texttt{tx}_3]$, $[\texttt{tx}_2, \texttt{tx}_3, \texttt{tx}_1]$, and $[\texttt{tx}_3, \texttt{tx}_1, \texttt{tx}_2]$. In this case, $\texttt{tx}_1$ is reported to be before $\texttt{tx}_2$ by two nodes (i.e. the majority), $\texttt{tx}_2$ is reported to be before $\texttt{tx}_3$ by two nodes, and $\texttt{tx}_3$ is reported to be before $\texttt{tx}_1$ by two nodes. This essentially creates a cycle, referred to as *Condorcet cycle* [5], which prevents any final ordering from respecting the views of the majority on how transactions should be ordered.

The existing fair ordering methods adopt a relaxed approach to ordering transactions inside a Condorcet cycle. For instance, Cachin et al. in Quick-Fairness [4] do not mention any ordering mechanism for such transactions, and Kelkar et al. in Aequitas [12] suggest a simple alphabetical ordering. This relaxed approach is, perhaps, due to two reasons: 1) it is not possible to guarantee fair ordering of transactions inside a cycle; 2) Condorcet cycles occur infrequently in practice, and when they do occur, they usually involve transactions that are received around the same time by the nodes in the system. Nevertheless, in this work, we show that Condorcet cycles deserve more attention as they can be created "artificially" by adversaries through what we refer to as the *Condorcet attack*. An interesting feature of the Condorcet attack proposed in this work is that it can be conducted by a client outside the system. In particular, the attack can be effectively executed even when all the nodes in the system are honest!

As will be explained later, in the Condorcet attack, an adversarial client sends a small number of transactions to different nodes in the system. The adversary chooses the timing and order of these transactions to create a Condorcet cycle that traps many honest transactions in it (a Condorcet cycle with only the adversary's transactions in it is all but harmless to the system.). This cycle has to be broken by the leader in a leader-based method in order to establish a total ordering. Even if the leader is honest, the act of breaking the cycle could change the order of honest transactions, which would have otherwise been appropriately ordered[1].

Defending against the Condorcet attack is not straightforward. It is partly because it is challenging to differentiate between honest transactions and otherwise valid transactions that are submitted with the intention of creating a cycle. It becomes notably more challenging to safeguard the system when, in addition to the adversarial client outside the system, the leader and possibly a fraction of the nodes in the system are adversarial. Nevertheless, in this work, we propose three mitigation techniques based on three different approaches. The proposed techniques complement each other and can work together harmoniously to maximize resistance against the attack.

In summary, we make the following contributions. We introduce a framework for a new type of attack (Condorcet attack) against fair transaction ordering. We show that the attack can be highly successful in trapping honest transactions in a cycle. To mitigate the attack, we propose three techniques based on three different complimentary approaches, and show

[1] Kelkar et al. [11] consider it a success for an adversary if the adversary places two transactions into the same cycle when they should not have been.

the effectiveness of the technique through simulations.

2 Related Work

The classical approach to mandating fair transaction ordering is through *secure causal ordering*, a method introduced by Birman and Reiter in 1994 [17], and later improved by Cachin et al. in 2001 [3]. This method uses encryption to conceal the content of transactions during the ordering process, and allows decryption of transactions only after the order of transactions is finalized. This prevents an adversary from observing the content of transactions during the ordering process, thereby effectively eliminating attacks such as the sandwich attack [16] that rely on inspecting transaction contents. However, the method is unable to prevent "blind front-running attacks" where, for instance, the adversary's sole objective is to order her transaction first (to, for example, get priority in purchasing a token). In addition, the method cannot prevent attacks based on transactions' metadata, as metadata (such as the source of a transaction) is not encrypted.

The second approach to mandating fair transaction ordering involves a first-come, first-served strategy. This approach is complementary to the first approach and has been the focus of several recent studies. The existing methods that follow this strategy can be broadly classified into two categories: timestamp-based methods and batch-based methods. Timestamp-based methods are computationally inexpensive but require synchronized clocks. Batch-based methods, on the other hand, offer stronger fairness than timestamp-based methods, but can tolerate fewer adversarial nodes.

Timestamp-based Methods. An example of a timestamp-based protocol is Pompe [20] due to Zhang et al. Pompe introduces a notion of fairness called the *ordering linearizability*. This notion stipulates that if the highest timestamp of a transaction $\mathtt{tx}$ is less than the lowest timestamp of a transaction $\mathtt{tx}'$ among honest nodes, then $\mathtt{tx}$ must be ordered before $\mathtt{tx}'$ in the final order of transactions. Although it can enforce ordering linearizability, Pompe suffers from censorship issues, as noted in [11].

Kursawe's Wendy protocol [13] is another timestamp-based protocol that defines a notion of fairness called *timed-relative-fairness*. This notion requires that if all honest nodes received a transaction $\mathtt{tx}$ before time τ, and transaction $\mathtt{tx}'$ after τ, then $\mathtt{tx}$ must be ordered before $\mathtt{tx}'$.

Batch-based Methods. Aequitas [12] by Kelkar et al. is a batch-based method proposed for fair transaction ordering. Aequitas enforces a fairness notion known as the *γ-batch-order-fairness*. The notion requires that if two transactions $\mathtt{tx}$ and $\mathtt{tx}'$ are received by all nodes in a system with n nodes, and γn nodes received $\mathtt{tx}$ before $\mathtt{tx}'$, then all honest nodes must output $\mathtt{tx}$ no later than $\mathtt{tx}'$. Aequitas suffers from high communication complexity of $\mathcal{O}(n^3)$, and can guarantee only a weak notion of liveness, one of the two pillars of consensus security.

The second batch-based method is Quick-Fairness [4] proposed by Cachin et al. This method enforces a fairness notion called the *κ-differential order-fairness*. This notion mandates that if the number of nodes that have received transaction $\mathtt{tx}$ before $\mathtt{tx}'$ exceeds $\kappa + 2f$ for some $\kappa \geq 0$, then $\mathtt{tx}$ should be ordered no later than $\mathtt{tx}'$, where f is the maximum number of adversarial nodes in the system. Kelkar et al. [11] show that this notion of fairness is indeed a re-parameterized version of the γ-batch-order-fairness notion. They also demonstrate that the Quick-Fairness protocol satisfies fairness only when all nodes are honest.

Kelkar et al. addressed the shortcomings of Aequitas in their protocol called Themis [11].

Themis satisfies the γ-batch-order-fairness notion, and solves the liveness problem of Aequitas. Moreover, SNARK-Themis variant offers a communication complexity of $\mathcal{O}(n)$ and standard Themis offers a communication complexity of $\mathcal{O}(n^2)$ instead of $\mathcal{O}(n^3)$ offered by Aequitas. In addition, it satisfies a more generalized notion of fairness than the one used in Quick-Fairness and a stronger notion of fairness than those used in the existing time-based methods. For these reasons, in our work, we focus on Themis and Aequitas as the state-of-the-art fair transaction ordering methods.

3 Model

System. We consider a permissioned system with a committee of n nodes. The nodes receive transactions directly from clients, and submit the list of their received transactions together with the order in which the transactions were received to a special node called the leader. The leader collects the lists of transactions from the nodes, and proposes a final ordering using a pre-decided fair-ordering protocol. The leader in the system is not fixed, and can change through a pre-determined protocol.

Fair Ordering. We adopt the batch-order-fairness from [12, 11], the strongest notion of fair ordering proposed to date. For a parameter $\frac{1}{2} < \gamma \leq 1$, the batch-order-fairness specifies that if a fraction γ of nodes receive a transaction `tx` before receiving another transaction $\texttt{tx}'$, then `tx` must be placed in the order before $\texttt{tx}'$, with exceptions allowed only if `tx` and $\texttt{tx}'$ are within the same Condorcet cycle (Condorcet cycles are defined in Section 4). Transactions within a cycle are placed in a batch, and are ordered by a method that we refer to as *batch-ordering scheme*. The existing fair ordering protocols either do not specify a batch-ordering scheme or propose a simple one (e.g., an alphabetical-based scheme [12]).

Network. The network utilizes public key infrastructure and secure digital signatures for communications. As in [12], we consider two networks: the (standard) internal network (for communication amongst nodes in the system) and the external network (for clients to transmit their transactions to the system).

We assume that the network operates under partial synchrony [7], meaning that there is a network delay Δ (not known to the nodes) that limits the amount of time it takes for messages to be delivered between nodes.

Adversary. We consider an adversary who has control over $f \geq 0$ out of n nodes, and also possesses at least one client capable of submitting transactions to the system. The adversary can deviate arbitrarily from the protocol. The adversary does not have control over the external network, but may have full control over the internal network, hence can delay and reorder messages up to the bound Δ.

4 Preliminaries

Graph Terminology. We use $G = (V, E)$ to denote a graph with the set of vertices V and the set of edges E. In this work, each vertex represents a transaction, therefore, we use the terms vertices and transactions interchangeably. Unless otherwise specified, we use an unweighted and directed graph. In the case of a weighted graph, the weight or cost associated with the edge $(u, v) \in E$ is represented by $w(u, v)$.

A *tournament graph* is a directed graph where every pair of distinct vertices is connected by a directed edge in either of two possible directions. A *Strongly Connected Component (SCC)* in a graph is a maximal subgraph in which there is a path from every vertex to every

other vertex. A *condensation graph* is obtained from the original graph by combining its SCCs into a single vertex. A *Directed Acyclic Graph (DAG)* is a directed graph that contains no cycles, meaning it is possible to move from one vertex to another along the directed edges, but it is not possible to return to the original vertex by following a sequence of directed edges. A *topological sort* is an ordering of the vertices in a DAG such that for every directed edge (u, v), vertex u appears before vertex v. In other words, if there is a directed edge from vertex u to vertex v, then u must appear before v in the topological sort. A *Hamiltonian Path* is a path in a graph that passes through every vertex exactly once. A *Hamiltonian Cycle* is a cycle in a graph that passes through every vertex exactly once.

Themis. Themis is the latest ordering method which achieves batch-order-fairness in the presence of an adversary who controls up to $f < \frac{(2\gamma-1)n}{4}$ nodes out of n nodes. Themis categorized received transactions into three different categories.

- Solid Transactions: A transaction is solid if it has been received by at least $n - 2f$ nodes. A solid transaction is one that has been received by enough honest nodes that the leader can unambiguously include it in the current proposal while respecting the fairness guarantees.
- Blank Transactions: A transaction is blank if it has not been received by at least $n(1 - \gamma) + f + 1$ nodes. A blank transaction has not been received by enough nodes yet, hence excluding it from the current proposal will not violate fairness with respect to transactions that are included.
- Shaded Transactions: A shaded transaction is a transaction that is neither solid nor blank. A shaded transactions is received by enough nodes to be included to preserve fairness, but not enough nodes to finalize its position in the current proposal.

Themis is a leader-based method and works in three phases, as described below.

- Phase 1 (Fair Propose): The Fair Propose phase is the first phase of the algorithm, where each node proposes a set of transactions and their local orderings to the leader. The leader then uses the local orderings of $n - f$ nodes to build a dependency graph. In the dependency graph, an edge from a vertex v_1 to v_2 indicates that the transaction v_1 should be placed before the transaction v_2. From the dependency graph, the leader then computes the condensation graph and its topological sorting to output a fair ordering.
- Phase 2 (Fair Update): The Fair Update phase is the second phase of the algorithm, where the leader node updates the ordering for previous proposals. This is necessary since this is part of the deferred ordering technique, and new transactions may depend on previously proposed transactions, and these dependencies need to be accounted for in the ordering. The Fair Update algorithm takes the local transaction orderings of $n - f$ nodes for previously proposed shaded transactions as input and outputs the updated dependencies.
- Phase 3 (Fair Finalize): The Fair Finalize phase is the third and final phase of the algorithm, where a sequence of proposals is finalized into a final ordering. The Fair Finalize algorithm updates the graphs for each proposal and computes the condensation graphs and their topological sorting. It then retrieves the final transaction ordering for each proposal based on the Hamiltonian cycles of the vertices in the sorted condensation graphs.

Condorcet Cycles. As mentioned above, Themis constructs a dependency graph, a directed graph where each vertex represents a transaction, and an edge from a vertex v_1 to v_2 indicates that the transaction corresponding to v_1 should be placed before the transaction

corresponding to v_2. We refer to any cycle in this dependency graph as a *Condorcet cycle*. We note that cycles can occur in a dependency graph because of the Condorcet paradox [11].

5 Condorcet Attack

In this section, we present the framework of the Condorcet attack. The attack aims at trapping honest transactions (i.e., transactions submitted by honest clients) inside a Condorcet cycle. If there is no effective batch-ordering scheme in place (e.g., if the batch-ordering scheme is alphabetical-based as suggested in [12]), this can change the ordering of the honest transactions even when all the nodes in the system are honest.

An adversary can take different strategies to impose a Condorcet cycle. For instance, suppose that the adversary controls f nodes, including the leader, in the system. The adversary then controls f local orderings, and can manipulate these orderings in a way to create a cycle. In the simulation section, we show that this strategy can not only create a cycle but also chain the cycles to involve more honest transactions. Nevertheless, the length of these cycles is typically small and the chain usually breaks rather quickly. As a result, this strategy is not effective in trapping distant transactions[2] (e.g., two transactions whose times of submission are separated by a multiple of the average network latency).

Another strategy, which is the one we take in this work, is to create a Condorcet cycle by injecting (valid) transactions into the system following a pre-described pattern. This can be done by an adversarial client outside the system, and can be effective even when all the nodes in the system are honest. The attack will be more effective in creating cycles and bypassing potential countermeasures if the adversary controls a fraction of nodes in the system (see Example 5).

The immediate damage of imposing a Condorcet cycle, as mentioned earlier, is that it can change the true ordering of honest transactions. In addition to this, the attack may be used to conduct other malicious activities; for instance, the adversary can create a cycle and then with the help of an adversarial leader can try to place its own transaction in desired positions in the final ordering.

▶ **Example 1.** Let $\mathcal{P} = \{P_1, P_2, P_3\}$ be a partition of nodes, where P_1, P_2 and P_3 are three parts with almost equal size. In this simple example, the adversary $\mathcal{C}$ uses/injects two transactions $\mathtt{A}, \mathtt{B}$ (i.e., $\mathcal{S} = \{\mathtt{A}, \mathtt{B}\}$). In the initialization phase, $\mathcal{C}$ sends the transaction $\mathtt{A}$ and then $\mathtt{B}$ to all the nodes in part P_1, and sends the transaction $\mathtt{B}$ to all the nodes in part P_2 (it sends no transactions to the nodes in part P_3). Then, after the pause period, $\mathcal{C}$ sends $\mathtt{A}$ to all the nodes in part P_2, and transaction $\mathtt{A}$ and $\mathtt{B}$, in that order, to all the nodes in part P_3. Suppose that during the pause phase, the nodes receive three honest transactions $\mathtt{tx}_1$, $\mathtt{tx}_2$, and $\mathtt{tx}_3$ all the in that order. The local ordering of transactions at each node will be then:

$$
\begin{aligned}
P_1 &: \quad [\mathtt{A}, \mathtt{B}, \mathtt{tx}_1, \mathtt{tx}_2, \mathtt{tx}_3] \\
P_2 &: \quad [\mathtt{B}, \mathtt{tx}_1, \mathtt{tx}_2, \mathtt{tx}_3, \mathtt{A}] \\
P_3 &: \quad [\mathtt{tx}_1, \mathtt{tx}_2, \mathtt{tx}_3, \mathtt{A}, \mathtt{B}]
\end{aligned}
$$

Note that without the adversarial client $\mathcal{C}$ disturbing the system (i.e., without transactions $\mathtt{A}$ and $\mathtt{B}$), the system would have had an easy job of ordering the honest transactions as all the nodes in the system have received the honest transactions in the same order, i.e. [$\mathtt{tx}_1$, $\mathtt{tx}_2$, $\mathtt{tx}_3$]. Because of the adversary's transactions $\mathtt{A}$, $\mathtt{B}$, and $\mathtt{C}$, however, we have a cycle now

[2] The analysis of why this occurs is left for future work.

as illustrated in Figure 1. In this figure, an edge from a transaction tx to a transaction tx′ indicates that the majority of the nodes have received tx before tx′.

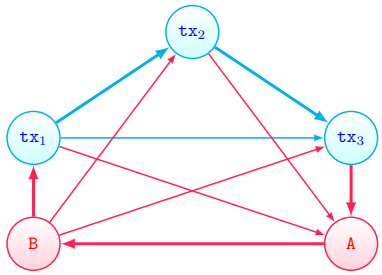


Figure 1 A Condorcet cycle created using two transactions A and B

Attack Framework. In this section, we provide a general construction that encompasses the different variants of the Condorcet attack. Let $\mathcal{C}$ be a client controlled by the adversary, and $\mathcal{S}$ be a set of arbitrary but valid transactions created by $\mathcal{C}$. Let $\mathcal{P}$ be a partition of the nodes in the system. In its general form, the Condorcet attack is executed in three phases:

- Phase 1 (Initialization): In this phase, the client $\mathcal{C}$ sends a number of transactions from the set $\mathcal{S}$ to each node in the system. The set of transactions sent to a node can be different from that sent to another node. More specifically, the client $\mathcal{C}$ assigns a subset $\mathcal{S}_i$ of $\mathcal{S}$ (possibly an empty subset) to each part $\mathcal{P}_i$ in the partition $\mathcal{P}$. It then determines an ordering for each subset $\mathcal{S}_i$, and sends the transactions in $\mathcal{S}_i$ to all the nodes in part $\mathcal{P}_i$ with the determined order.
- Phase 2 (Pause): In the second phase, the attacker waits for a specific amount of time, referred to as the pause time, for the honest transactions to be received by the nodes. The adversary can trap more transactions within a cycle as the pause time increases. However, the pause time should be limited to a single consensus round in the system as the attack should not extend across multiple consensus rounds.
- Phase 3 (Finalization): The third and final phase is the finalization phase, where the attacker completes the Condorcet cycle by sending a new set of transactions to each part in the partition. More specifically, the client $\mathcal{C}$ assigns a subset $\mathcal{S}'_i$ of $\mathcal{S}$ (typically a different subset than $\mathcal{S}_i$, used in the initialization phase) to each part $\mathcal{P}_i$ in the partition $\mathcal{P}$. It then determines an ordering for each subset $\mathcal{S}'_i$, and sends the transactions in $\mathcal{S}'_i$ to all the nodes in part $\mathcal{P}_i$ with the determined order.

▶ Remark 2. In practice, nodes in the system may receive some honest transactions during the initialization and/or finalization phases. These transactions may or may not get trapped in the Condorcet cycle. Based on our simulation results, however, the vast majority of honest solid transactions during the pause time fall into the Condorcet cycle.

▶ Remark 3. A potential issue that can impact the success of the Condorcet attack is that the external network may deliver the transactions injected by the adversary out of order. For instance, in Example 1, the transactions A and B may be received out of order by the nodes in part P_1, in which case a cycle does not occur. If the network is prone to packet reordering, then to improve its success, the adversary can execute multiple Condorcet attacks concurrently through what we refer to as *cloning*.

Cloning. Packet reordering can happen in a network because of various factors such as network congestion, routing algorithms, and the physical distance between the source and

the destination. To conduct a successful Condorcet attack, it is important that nodes receive the injected packets in the order they were transmitted; a deviation from the intended order may result in the failure of the attack.

To increase the success probability of the attack in the presence of network reordering, the adversary can send cloned transactions to the nodes: Instead of sending a single transaction $\mathtt{A}$, the adversary sends multiple clones of the transaction. For instance, in Example 1, the adversary can send $\mathtt{A}_1$ and $\mathtt{A}_2$ instead of $\mathtt{A}$, and sends $\mathtt{B}_1$ and $\mathtt{B}_2$ instead of $\mathtt{B}$. Essentially, the adversary interleaves the execution of two Condorcet attacks (for better results, the adversary can interleave several instances of the attack). Then, if the network does not change the order of the transactions, the nodes in parts P_1, P_2, and P_3 will receive transactions as follows:

$$
\begin{aligned}
P_1 &: \ [\mathtt{A}_1, \mathtt{A}_2, \mathtt{B}_1, \mathtt{B}_2, \mathtt{tx}_1, \mathtt{tx}_2, \mathtt{tx}_3] \\
P_2 &: \ [\mathtt{B}_1, \mathtt{B}_2, \mathtt{tx}_1, \mathtt{tx}_2, \mathtt{tx}_3, \mathtt{A}_1, \mathtt{A}_2] \\
P_3 &: \ [\mathtt{tx}_1, \mathtt{tx}_2, \mathtt{tx}_3, \mathtt{A}_1, \mathtt{A}_2, \mathtt{B}_1, \mathtt{B}_2]
\end{aligned}
$$

In Section 7.3, we show that cloning can significantly increase the success rate of the Condorcet attack in the presence of network reordering.

Impact on Current Solutions. The current fair transaction ordering protocols either do not offer a batch-ordering scheme (e.g. [4]) or offer a primitive one (e.g. [12]). For instance, the proposed batch-ordering scheme in Aequitas [12] is alphabetical ordering. Therefore, if an adversary creates a Condorcet cycle, as in Example 1, the honest transactions will be ordered alphabetically rather than by the time of their arrival.

Themis [11], proposes a more thoughtful batch-ordering scheme. In this scheme, a Hamiltonian cycle is built and then used to order transactions in the cycle. The latest version of Themis at the time of writing this work suggests to break the weakest link in the Hamiltonian cycle in order to convert it into a Hamiltonian path. We use this version of Themis in our work. In the best-case scenario, the order of honest transactions in the Hamiltonian cycle is preserved. Even in this case, the final ordering of these transactions can change because the Hamiltonian cycle has to be converted into a path by breaking the cycle at one point. It is at this point where honest transactions can be divided into two groups. The ordering of the honest transactions within each group remains correct, but the ordering of any two transactions from different groups will be incorrect. Therefore, similar to [4] and [12], Themis is vulnerable to the Condorcet attack even if all the nodes (including the leader) in the system are honest.

To combat the Condorcet attack, a natural approach is to use a strong batch-ordering scheme. For instance, in Example 1, we can observe that all the nodes report $\mathtt{tx}_1$ before $\mathtt{tx}_2$, and all the nodes report $\mathtt{tx}_2$ before $\mathtt{tx}_3$, whereas only two third of the nodes report $\mathtt{A}$ before $\mathtt{B}$. In this example, the weakest link is between adversarial transactions, and breaking it (as suggested by Themis) does not change the true ordering of the honest transactions. This solution works for the scenario described in Example 1. However, this solution may not work in other settings, for example when the adversary controls a faction of nodes in the system (see Example 5).

6 Mitigation

Despite its simplicity, it is not straightforward to completely defeat the Condorcet attack. In the following, we present three mitigation techniques based on three different approaches to hinder an adversary from successfully executing the attack. We elaborate on the strength of each technique and confirm it through simulations later in Section 7. We also explain the

limitation of each technique, i.e. under what settings/assumptions the technique may not be effective.

An interesting feature of the proposed mitigation methods is that they do not conflict with each other, thus in practice, they can be applied together for the maximum defense against the attack. Another interesting feature of the proposed mitigation methods is that they can be easily applied to Themis, which is currently the strongest fair-ordering solution in the literature. We elaborate on this when we cover each proposed mitigation.

6.1 Ranked Pairs Batch-ordering

The approach we take in our first proposed mitigation is to use a strong batch-ordering scheme to order transactions within a batch. Formally, a batch-ordering scheme is a method that takes as input a strongly connected (possibly weighted) directed graph $G = (V, E)$, and returns an ordering of the vertices V. The strongly connected graph represents the transactions that are in a batch/cycle.

The candidate for our batch-ordering scheme is *ranked pairs*, an electoral system developed by Nicolaus Tideman in 1987 [19]. Ranked pairs satisfies many natural and well-studied axiomatic properties in social choice theory[3] and is resistant to certain manipulations including adding, deleting and changing a fraction of orderings reported by nodes [15]. In ranked pairs, the ordering is essentially achieved by choosing a maximal subset E' of E in the inputted graph $G = (V, E)$ with high weights such that $G' = (V, E')$ is a DAG. The DAG is then used to establish an ordering of the vertices V.

More specifically, our ranked pairs batch-ordering scheme takes as input a weighted directed graph $G = (V, E)$. Let $E_1 = E$. In step i, $i \geq 1$, the algorithm selects an edge $(u, v) \in E_i$ with the highest weight[4]. It then sets the order $u \prec v$, unless this violates the transitivity of the orders decided in previous steps. Finally, it sets $E_{i+1} \leftarrow E_i \backslash \{(v_i, v_j)\}$, and terminates if $E_{i+1} = \emptyset$.

We note that the idea in the above batch-ordering scheme is to establish an ordering using the strongest edges in G. This will be an effective defense against the Condorcet attack if the ordering of the honest transactions has "strong support" in the system. In a special case where all the nodes are honest, and all support/report the same ordering of honest transactions, the Condorcet attack can be fully prevented as stated in the following theorem.

▶ **Proposition 4.** *Suppose that the Condorcet attack succeeds in creating a Condorcet cycle. Let* $\mathtt{tx}_1, \mathtt{tx}_2, \ldots, \mathtt{tx}_m$ *be the set of honest transactions in the Condorcet cycle. Suppose that all the nodes in the system are honest and report* $\mathtt{tx}_i$ *before* $\mathtt{tx}_j$ *for every* $1 \leq i < j \leq m$*. Then the proposed ranked pairs batch-ordering scheme returns the true ordering of the honest transaction, that is it orders* $\mathtt{tx}_i$ *before* $\mathtt{tx}_j$ *for every* $1 \leq i < j \leq m$*.*

Proof. Let $G = (V, E)$ be the graph with V representing the transactions in the Condorcet cycle, and the weight of each edge $(u, v) \in E$, represented as $w(u, v)$, be equal to the number of nodes that reported u before v. Let $u_1, u_2, \ldots, u_m$ be the vertices in V that represent the honest transactions. Let $E_f \subseteq E$ be the set of all edges with the full support of the nodes,

[3] besides Schulze, ranked pairs is the only existing electoral system that satisfies anonymity, Condorcet criterion, resolvability, Pareto optimality, reversal symmetry, monotonicity, and independence of clones [18].

[4] When there are multiple edges with the highest weight, one can be chosen according to a fixed tie-breaking method.

that is

$$E_f = \{e \in E | w(e) = n\},$$

where n is the number of nods in the system. Since all the nodes in the system have the same view on the ordering of the honest transactions, we get that $(u_i, u_j) \in E_f$ for every $1 \leq i < j \leq m$. We note that the sub-graph $G' = (V, E_f)$ of G is cycle free, as otherwise there will be a cycle in the ordering of individual nodes. The ranked pairs batch-ordering algorithm first chooses all the edges in E_f before proceeding with other edges in E. When the algorithm covers all the edges in E_f the true ordering of the honest transactions will be set, and cannot be changed by the remaining steps of the algorithm. ◀

Limitation. Proposition 4 considers an ideal scenario where 1) all the nodes are honest, and 2) they all report the honest transaction in the same order. If one of the above two conditions does not hold, however, the Condorcet attack may be able to create a cycle (see the following example).

▶ **Example 5.** Consider a system with $n = 5$ nodes. Let $\mathtt{tx}_1, \mathtt{tx}_2, \mathtt{tx}_3$ be three honest transactions. An adversarial client $\mathcal{C}$ can create a Condorcet cycle of the form

$$\begin{aligned}
N_1 &: \quad [\mathtt{A}_1, \mathtt{A}_2, \mathtt{A}_3, \mathtt{A}_4, \mathtt{tx}_1, \mathtt{tx}_2, \mathtt{tx}_3] \\
N_2 &: \quad [\mathtt{A}_2, \mathtt{A}_3, \mathtt{A}_4, \mathtt{tx}_1, \mathtt{tx}_2, \mathtt{tx}_3, \mathtt{A}_1] \\
N_3 &: \quad [\mathtt{A}_3, \mathtt{A}_4, \mathtt{tx}_1, \mathtt{tx}_2, \mathtt{tx}_3, \mathtt{A}_1, \mathtt{A}_2] \\
N_4 &: \quad [\mathtt{A}_4, \mathtt{tx}_1, \mathtt{tx}_2, \mathtt{tx}_3, \mathtt{A}_1, \mathtt{A}_2, \mathtt{A}_3] \\
N_5 &: \quad [\mathtt{tx}_3, \mathtt{tx}_2, \mathtt{tx}_1, \mathtt{A}_1, \mathtt{A}_2, \mathtt{A}_3, \mathtt{A}_4]
\end{aligned}$$

where $\mathtt{A}_1, \mathtt{A}_2, \mathtt{A}_3, \mathtt{A}_4$ are the transactions submitted by $\mathcal{C}$. Note that all the nodes, except Node 5, report the order $[\mathtt{tx}_1, \mathtt{tx}_2, \mathtt{tx}_3]$, while node 5 reports $[\mathtt{tx}_3, \mathtt{tx}_2, \mathtt{tx}_1]$ (Node 5 is either controlled by the adversary or is an honest node who has simply received the transactions in this order). If we run the proposed ranked pairs batch-ordering scheme on this cycle, the returned order of honest transactions may be incorrect. It is because the edge between any pair of transactions has a weight of 4 in the dependency graph. As a result, an edge between two honest transactions such as $\mathtt{tx}_1$ and $\mathtt{tx}_2$ may be eliminated in the ranked pairs method, which would result in $\mathtt{tx}_2$ and $\mathtt{tx}_3$ to be ordered before $\mathtt{tx}_1$. As for Themis, if we use the proposed method by Yannis Manoussakis [14] (as suggested by Themis), we get the Hamiltonian cycle $(\mathtt{A}_1, \mathtt{A}_2, \mathtt{A}_3, \mathtt{A}_4, \mathtt{tx}_1, \mathtt{tx}_2, \mathtt{tx}_3)$. All the edges in this cycle have the identical weight of four, hence there is no distinct weakest edge. Therefore, Themis may remove any of the edges in the cycle. If the removed edge is between two honest transactions, the final ordering of honest transactions would be incorrect. We remark that both the ranked pairs batch-ordering scheme and Themis would order honest transactions correctly if Node 5 order honest transactions as $[\mathtt{tx}_1, \mathtt{tx}_2, \mathtt{tx}_3]$. This example, therefore, shows that the adversary has more power in modifying the order of honest transactions if (in addition to injecting transactions) it controls a number of nodes in the system (e.g. Node 5 in this example).

▶ Remark 6. To use the proposed ranked pairs batch-ordering scheme in Themis, we can simply replace the Hamiltonian-based batch-ordering scheme of Themis with the ranked pairs batch-ordering scheme in the FairFinalize algorithm. We remark that the weight information of the dependency graph is available within the FairFinalize algorithm, thus this replacement is possible.

6.2 Post-decryption Resolution

In secure causal ordering, as mentioned earlier, transactions are ordered while they are encrypted, and get decrypted only once a total ordering is committed [17, 3]. This prevents an adversary from observing the contents of transactions while they are being ordered, hence eliminating those front-running attacks (e.g. the sandwich attack [16]) that must examine the content of transactions.

To mitigate the Condorcet attack, we propose to maintain the above strategy, except we leave the ordering of transactions inside a Condorcet cycle to after they are decrypted. Note that after the decryption of these transactions, an adversary cannot impose a change to the ordering as 1) there is already a consensus on the set of transactions that must be included, thus the adversary cannot add or remove any transaction to the set; 2) the ordering of the transactions is performed locally at each node using a pre-determined algorithm. In other words, it is too late for the adversary to manipulate the ordering of transactions, although the contents of transactions are disclosed.

Once the transactions within a cycle are decrypted, their contents are disclosed, enabling them to be partitioned into independent groups (i.e., transactions inside different groups are independent of each other). Each group can then be ordered independent of the others. By implementing this measure, the adversary is unable to manipulate the ordering of honest transactions if the adversary's transactions are independent of honest transactions. This is because the adversary's transactions will not fall within any group that includes honest transactions. Note that we still need to order the groups themselves (i.e. which group comes first, which comes second, and so on). As transactions across various groups have no effect on one another, the groups can be safely ordered using a pre-determined algorithm such as ranked pairs as described in Section 6.1.

▶ Remark 7. In the Themis protocol, we can apply the above post-decryption resolution method within the FairFinalize algorithm: If transactions `A` and `B` are independent, the edge between them in the dependency graph can be safely removed.

Limitation. The post-decryption resolution prevents the adversary from manipulating the order of honest transactions if the adversary's transactions are independent of the honest transactions. In certain scenarios, however, the adversary may be able to create dependencies. For instance, consider a situation where a popular NFT is dropping in a block currently being formed. Given the high demand, many transactions are transmitted with the intention of acquiring this NFT. Recognizing this, the adversary can execute the Condorcet attack by using transactions that fall within the same dependency group as those attempting to acquire the NFT.

Another limitation of the post-decryption resolution is the computational burden it places on the system to identify dependencies between transactions.

6.3 Broadcast

In the Condorcet attack, the adversary follows a well-structured three-phase strategy: in the first phase, the adversary sends a set of transactions, then pauses in the second phase, and then finishes the attack by sending another round of transactions in the third phase. The idea behind our third mitigation technique is to disturb/break the above pattern by broadcasting transactions inside the system as soon as they arrive at an honest node. Because of the broadcast, the adversary's transactions that were submitted in the first phase will propagate in the system, which can nullify the adversary's target in the third phase since the

transactions that the adversary transmits in the third phase have already been received by the nodes (thus their order has already been decided by the nodes).

In Section 7.5, we observe that this strategy proves highly effective in mitigating the suggested Condorcet attack. However, it is important to note that this strategy does incur increased communication overhead as a drawback. For instance, in Themis, nodes transmit transactions only to the leader as opposed to broadcasting in the network by themselves. Therefore, when applied to Themis, the above strategy will increase Themis's communication overhead (although it does not increase Themis's quadratic communication complexity).

Limitation. The main limitation of the above broadcast-based mitigation technique is that it will be ineffective if the adversary has strong control over the internal network. For instance, in Themis and Aequitas, it is assumed that the adversary controls all message delivery in the internal network, and can delay messages up to a bound Δ. If Δ is large enough (e.g., if it is larger than the duration of the Condorcet attack) then the adversary can circumvent the proposed mitigation by delaying all the broadcast transactions so they are delivered only after the attack is complete.

7 Simulation

To assess the impact of the Condorcet attack, as well as the effectiveness of the proposed mitigation methods, we conduct a series of experiments through simulations. In this section, we present the results of these experiments.

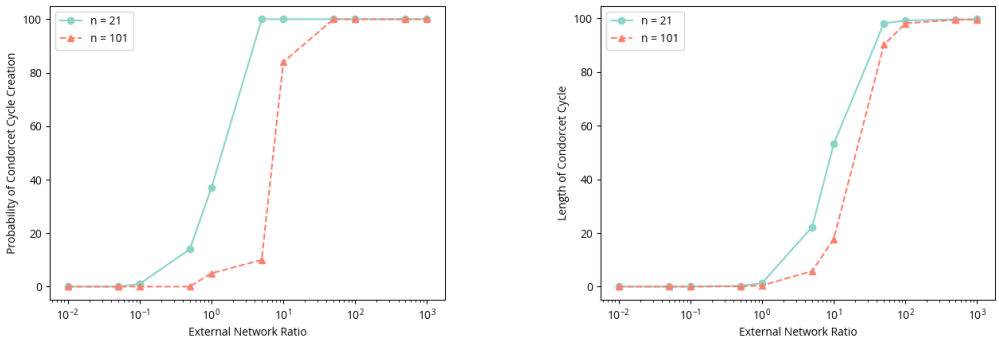
Environments. Our simulation encompasses four environments. The first environment captures the honest setting, where all the nodes and clients are honest, thereby eliminating the possibility of a Condorcet attack. Even in this environment, Condorcet cycles can occur. Therefore, we are interested to know if our proposed ranked pairs batch-order scheme can more effectively order transactions within a cycle than the Hamiltonian-cycle-based scheme used in Themis.

In the second environment, all the nodes in the system are honest, but there is an external adversary, who conducts the Condorcet attack from outside the system. In this environment, we are interested to evaluate the success rate and impact of the Condorcet attack (i.e., how many honest transactions the adversary can trap within a cycle).

In the third environment, we introduce packet reordering to the external network. We evaluate the impact of this on the success rate of the Condorcet attack. We also observe how the cloning method can help the adversary to improve its success rate.

The last environment that we consider is similar to the second environment, except this time we guard the system using the proposed mitigation methods. In this environment, we measure the impact of the Condorcet attack in order to examine the strength of the proposed mitigation methods.

Clients. We use a sending process to submit all the clients' transactions to the system. The sending process transmits transactions in sequence at discrete times t_i, $i \geq 0$. At each time instance, the process sends (n copies of) the transaction of a given client to all the n nodes in the system. Each copy of the transaction will arrive at its destination node with a random delay drawn independently from a distribution named `NetworkDist`. We refer to this distribution as the network latency. We use another distribution, `GenerationDist`, to determine the delay between two consecutive time instances (i.e. $t_{i+1} - t_i$ follows the `GenerationDist` distribution). Similar to [11], we set both `GenerationDist` and `NetworkDist` to exponential distributions with means of one and r, respectively. We refer to



(a) The chance of a Condorcet cycle

(b) Number of transactions in cycles

Figure 2 Condorcet cycles in the honest environment

r as *external network ratio*. One can think of r as the expected number of clients who transmit transactions within a time frame equal to the average network latency.

Themis Variant. In our simulations, we use the practical Themis variant with the communication complexity of $\mathcal{O}(n^2)$, instead of the the SNARK-Themis variant. In our simulations, all transactions are eventually received by each node in a single round. Therefore, the choice of γ does not have any impact on the simulation results (hence, we simply set $\gamma = 1$). We used the latest version of Themis, which breaks the Hamiltonian cycle by removing the weakest link. The weakest link is the link that has the least weight or support in the Hamiltonian cycle. To construct a Hamiltonian cycle, we used the proposed method by Yannis Manoussakis [14] as suggested by Themis.

7.1 Honest Environment

Honest Environment Setting. In this environment, all the nodes and clients are honest, and consequently, there is no Condorcet attack. Nevertheless, as shown in Figure 2, Condorcet cycles can occur particularly when the external network ratio is greater than one.

To obtain the results plotted in Figure 2, we varied the external network ratio from 0.01 to 1000. For each given network ratio, and each network size of $n = 21$ and $n = 101$, we conducted 100 simulation runs. In each run, the sending process transmitted 100 transactions (at 100-time instances drawn from the `GenerationDist` distribution). Once every node received all the transmitted transactions, we proceeded to generate the dependency graph using the Themis algorithm. By examining the graph (i.e. extracting strongly connected components) we then identified all the Condorcet cycles.

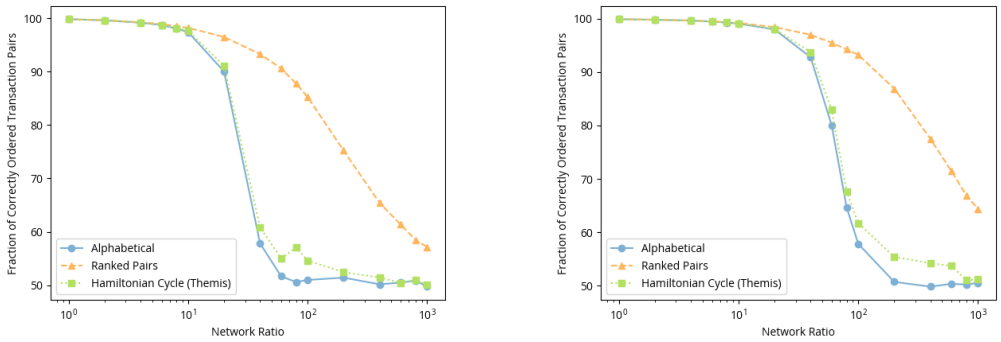
Cycle Length. An interesting observation from Figure 2 is that when the external network ratio is less than about one, Condorcet cycles rarely occur. As the external network ratio becomes larger than one, however, Condorcet cycles start to appear. For high values of the external network ratio, as depicted in Figure 2, Condorcet cycles not only occur frequently but also include many of the transmitted transactions. Overall, this observation suggests a critical threshold at which the system's behavior, with respect to creating Condorcet cycles, significantly changes.

Condorcet Cycles Categories. We refer to Condorcet cycles that are not created by an adversary as *natural Condorcet cycles*. Conversely, we call a Condorcet cycle adversarial if it is created by an adversary. In Section 6.1, we proposed a ranked pairs batch-ordering

scheme to handle the ordering of transactions within an adversarial Condorcet cycle. Later in this section, we demonstrate that the proposed scheme indeed alleviates the severity of the Condorcet attack.

Ranked Pairs Performance. Here, we show (Figure 3) that the proposed ranked pairs batch-ordering scheme is also a good candidate for ordering transactions within a natural Condorcet cycle. Consequently, even in an honest environment, we can improve fairness in ordering transactions by replacing the existing batch-ordering schemes (i.e., the alphabetical scheme, and the Hamiltonian-based scheme of Themis) with the proposed ranked pairs batch-ordering scheme.

Batch Ordering-Schemes Performance Comparison. In Figure 3, the external network ratio (the x-axis) ranges from 1 to 1000; this is the range in which Condorcet cycles naturally occur. The y-axis shows the fraction of transaction pairs that are ordered correctly according to their transmission time. Each data point in Figure 3 is the average of values obtained over 100 simulation runs. The data presented in this figure demonstrate the superiority of the proposed ranked pairs batch-ordering scheme for two network sizes of $n = 21$ and $n = 101$.



(a) The number of nodes is $n = 21$ **(b)** The number of nodes is $n = 101$

Figure 3 Fraction of correctly ordered transactions in the honest environment

7.2 Adversarial Environment

Adversarial Environment Setting. In the existing adversarial environments in the literature, there is often at least one (typically up to $f = \theta(n)$) adversarial node in the system. In our adversarial environment, in contrast, all the nodes in the system can be honest. There is, however, an adversarial client in our environment who orchestrates the Condorcet attack from outside the system.

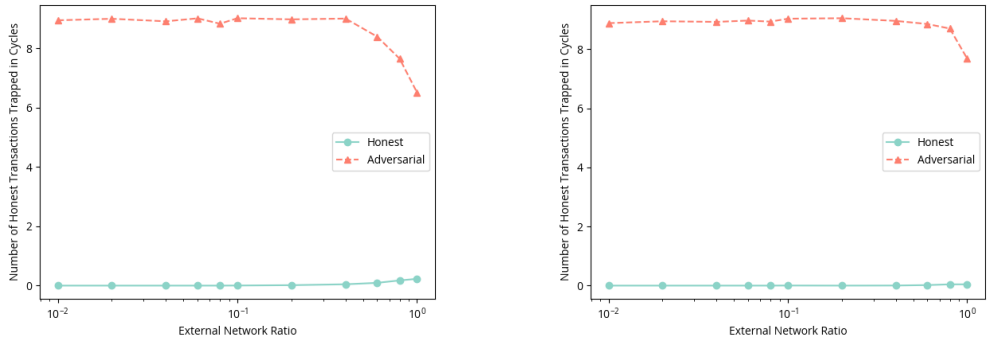
In this section, we evaluate the performance of the Condorcet attack in this environment. In particular, we measure the success rate of the attack in the number of honest transactions it can trap within a cycle. The measurement is carried out for external network ratios r less than one, as natural Condorcet cycles are rare in this regime, particularly when $r \ll 1$. This allows us to assess the strength of the attack in creating cycles in a setting where Condorcet cycles do not naturally happen.

In our simulation, we simply use two adversarial transactions to create the Condorcet cycle as described in Example 1. We set the pause time of the Condorcet attack to $\tau \in \{10, 50\}$ times the mean of the `GenerationDist` distribution. This means that, on average, τ honest

transactions are transmitted to the system during the pause time.

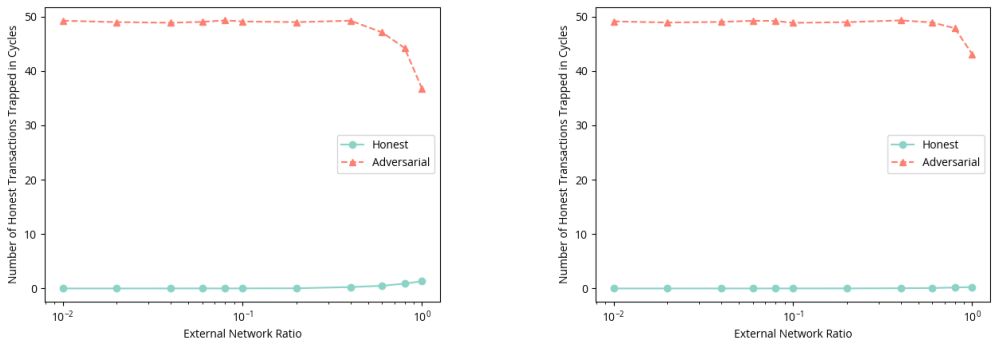
In parallel to the transmissions of honest transactions, the two adversarial transactions are transmitted to create a Condorcet cycle. Once all transactions are received by the nodes, we calculate two separate dependency graphs: one considering the adversarial transactions, and one ignoring them. By comparing these two dependency graphs, we then assess the impact of the attack on the final ordering.

Condorcet Attack Performance. Figures 4 and 5 show the average number of the honest transactions that the attack can trap within cycles over two different settings: $\tau = 10$ and $\tau = 50$. As shown, for a wide range of external network ratios, the attack can trap nearly all the honest transactions that are transmitted during the pause time (about 9 honest transactions in the setting $\tau = 10$, and nearly 49 honest transactions in the setting $\tau = 50$). This demonstrates the strength of the attack, considering that, on average τ honest transactions are submitted to the system during the pause time (and the attack traps nearly all of them).



(a) $\tau = 10$, $n = 21$ **(b)** $\tau = 10$, $n = 101$

Figure 4 Number of honest transactions trapped in Condorcet cycles for $\tau = 10$.



(a) $\tau = 50$, $n = 21$ **(b)** $\tau = 50$, $n = 101$

Figure 5 Number of honest transactions trapped in Condorcet cycles for $\tau = 50$.

7.3 Network Reordering

In the Condorcet attack, the adversary sends a sequence of transactions in a particular order to create a cycle. The external network may, however, change the order of transactions transmitted, which can, in turn, reduce the attack's success rate. To evaluate this, we performed simulations over a network which changes the order of two consecutively transmitted transactions with probability $0 \leq p \leq 0.5$. For each value of p, we performed 1000 runs of simulations. The success rate of the attack was set to the fraction of runs in which the attack successfully trapped the honest transactions in a Condorcet cycle.

Using the above setting, we conducted two instances of the Condorcet attack. The first instance uses two adversarial transactions `A` and `B` as in Example 1, and takes the following pattern:

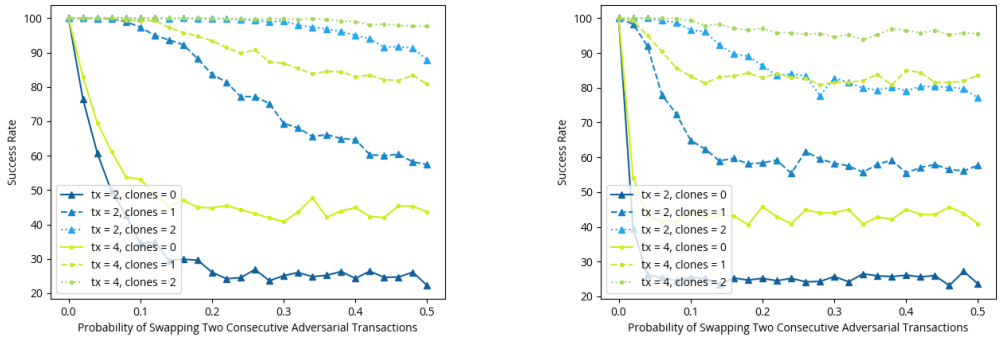
$$\begin{aligned} P_1: &\ \texttt{A}, \texttt{B}, \text{Pause} \\ P_2: &\ \texttt{B}, \text{Pause}, \texttt{A} \\ P_3: &\ \text{Pause}, \texttt{A}, \texttt{B} \end{aligned}$$

As illustrated in Figure 6, this instance is sensitive to network reordering (the success rate of the attack drops quickly with p). As shown in the figure, the attack's success rate increases when we use the second instance of cloning described below.

In our second instance (denote as $\texttt{tx} = 4$ in Figure 6), the adversary partitions nodes into four parts P_1, P_2, P_3 and P_4, and uses four transactions (`A`, `B`, `C` and `D`) instead of two, in the following pattern:

$$\begin{aligned} P_1: &\ \texttt{A}, \texttt{B}, \text{ Pause }, \texttt{C}, \texttt{D} \\ P_2: &\ \texttt{B}, \texttt{C}, \text{ Pause }, \texttt{D}, \texttt{A} \\ P_3: &\ \texttt{C}, \texttt{D}, \text{ Pause }, \texttt{A}, \texttt{B} \\ P_4: &\ \texttt{D}, \texttt{A}, \text{ Pause }, \texttt{B}, \texttt{C} \end{aligned}$$

This instance of the Condorcet attack is more robust against network reordering as demonstrated in Figure 6. As in the first instance, the success rate of the instance can be boosted using the cloning method. In particular, note that the second instance together with a single clone is almost fully resistant to network transaction reordering.



(a) The number of nodes is $n = 21$ **(b)** The number of nodes is $n = 101$

Figure 6 Impact of network reordering on the success of the Condorcet attack.

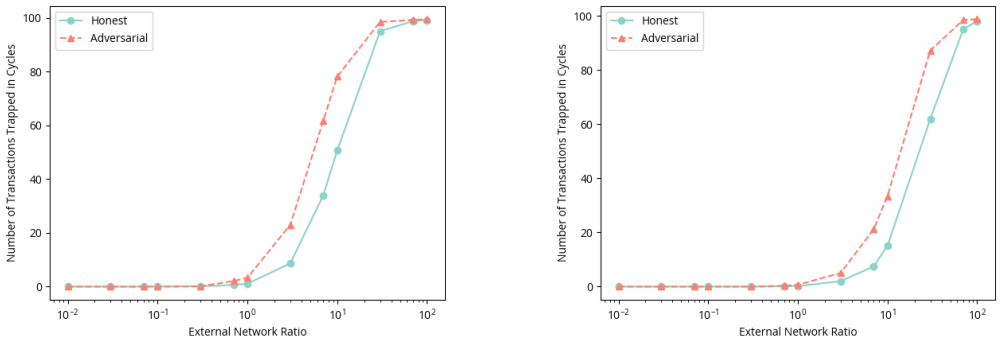
7.4 A Non-Injective Condorcet Attack

Injecting transactions into the system is a key component of the proposed Condorcet attack. Without this component, an adversary has limited power in creating cycles even when the adversary controls the leader and a faction of all the nodes in the system.

To illustrate the above point, we conducted simulations over two networks with sizes: $n = 21$ and $n = 101$. In our simulation, the adversary controls the maximum fraction of nodes, including the leader, allowed by Themis (a quarter of nodes minus one). All these nodes report the order of their received transactions in reverse, in a strategy to create Condorcet cycles[5]. The external network ratio is varied from 0.01 to 100 to capture a wide range of network conditions. The total number of transmitted transactions is set to 100.

To evaluate the impact of the above strategy in creating cycles, we created two dependency graphs. The first graph represents the scenario where the adversarial nodes reverse their orderings, whereas the second graph represents the scenario where the adversarial nodes report the true ordering. Figure 7 shows the results of our simulation.

Non-Injective Condorcet Attack Performance. As shown in Figure 7, the adversary's attempts to create cycles are largely unsuccessful in the region where the external network ratio is less than one. We note that in this region, the average temporal gap between two different transaction transmissions is more than the average network latency. In particular, when $r \ll 1$ (i.e., when transactions are transmitted far apart in time with respect to the network latency), honest nodes in the system have a clear view of the true ordering of transactions. In this region, the adversary is all but powerless in creating cycles[6], as evident in Figure 7. In contrast, in the same region, an external adversary can create a cycle using the proposed Condorcet attack, even when all the nodes in the system are honest.



(a) The number of nodes is $n = 21$ **(b)** The number of nodes is $n = 101$

Figure 7 The non-injective attack has limited power in creating cycles.

[5] We note that this may not be an optimum strategy to create Condorcet cycles. Nevertheless, we believe that an optimum strategy (which may be computationally intractable) may not be significantly more successful than the adopted strategy. We leave the validation of this claim for future work.

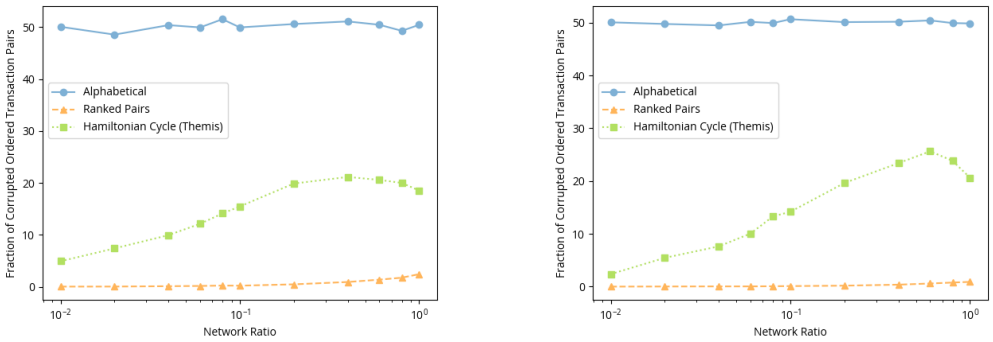
[6] When $r > 1$ (i.e., in the region where Condorcet cycles naturally emerge) the adversary achieves some degree of success in creating larger cycles than naturally occur.

7.5 Mitigation

In this section, we evaluate the performance of our mitigation methods in preventing or minimizing the impact of the Condorcet attack.

Ranked-pairs-based Mitigation Method. To evaluate the effectiveness of this mitigation, we conducted a simulation over two network sizes of $n = 21$ and $n = 101$. We set the pause time of the attack to 10 times the mean of `GenerationDist`, and set the total number of honest transactions to 20. We varied the external network ratio r from 0.001 to 1. Recall that in this range of external network ratio (i.e., $r < 1$), Condorcet cycles do not emerge naturally; rather they are created by the Condorcet attack. To evaluate the true impact of our ranked-pars mitigation method, therefore, we focused on this region.

Ranked Pairs Mitigation Performance. Figure 8 compares the performance of our proposed ranked-pairs-based mitigation method to the Hamiltonian-based method used in Themis, and the simple alphabetical method. The results show that the proposed ranked-pairs method achieves a low error rate, indicating that it can effectively order honest transactions correctly even when they fall in a Condorcet cycle. In contrast, the Themis algorithm's error rate increases as the network ratio increases, and reaches as high as about 25%. The error rate in the case of alphabetical ordering is 50%. Note that a random ordering method can, on average, correctly orders 50% of all the pairs of transactions. In this sense, the worst-case transaction ordering error is 50%, which is the case for the alphabetical method (this method is essentially a random ordering method).



(a) The number of nodes is $n = 21$ **(b)** The number of nodes is $n = 101$

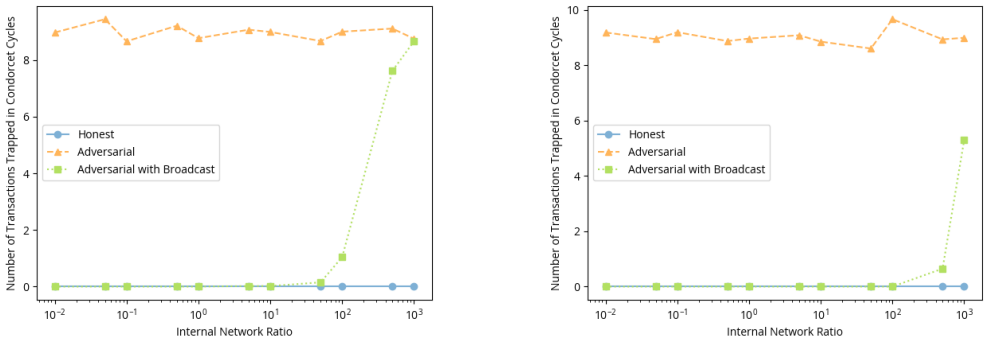
Figure 8 The performance of the proposed ranked pairs-based mitigation method

The Broadcast-based Mitigation Method. To evaluate the effectiveness of the broadcast-based mitigation method, we conducted simulations using two network sizes: $n = 21$ and $n = 101$. We introduced a new exponential distribution called `InternalNetworkDist`, which captures the random delays experienced by messages within the internal network. Specifically, we sample from `InternalNetworkDist` to determine the delay between sending a transaction from one node to another node. This is in contrast to `NetworkDist`, which is used to determine the random delays between a client and a node in the external network.

In our simulation, we set the mean of `InternalNetworkDist` to r'. We refer to r' as the *internal network ratio*. In our simulations, we set τ to 10 times the mean of `GenerationDist` (i.e. $\tau = 10 \cdot r$), and set the total number of honest transactions to 20. We fixed the external network ratio to $r = 0.1$, to ensure that no natural Condorcet cycles were created, and varied the internal network ratio r' from 0.01 to 1000.

Broadcast Environment Categories. We analyzed the number of honest transactions trapped in a Condorcet cycle under three different settings. In the first setting, referred to as the "honest setting", nodes did not broadcast and the adversary did not conduct a Condorcet attack. In the second setting, nodes still did not broadcast, but the adversary attempted a Condorcet attack. Finally, in the last setting, the adversary launched an attack while the nodes employed the broadcasting method to mitigate it.

Broadcast Mitigation Performance. Figure 9 shows the result of our simulations in the above three settings. The results demonstrate that the proposed broadcast-based mitigation is highly effective in preventing the adversary from creating a Condorcet cycle and trapping honest transactions. This can be attributed to two key factors: Firstly, the mitigation strategy disrupts the completion of the pause phase, thereby preventing honest transactions from being trapped in a Condorcet cycle. When the internal network ratio r' is smaller than the pause time, almost no transactions are trapped. Interestingly, even when r' exceeds the pause time, the adversary cannot achieve the same level of performance. It is because the broadcast of transactions with the internal network can still somewhat disturb the ordering of adversarial transactions. This reduces the success rate of the attack as the specific ordering of adversarial transactions is crucial for creating a Condorcet cycle. If, on the other hand, the adversary has enough control over the internal network to delay transactions as much as the pause time, it can circumvent the proposed broadcast-based mitigation as the adversary can enforce the ordering of its transactions within the internal network by delaying all the messages.



(a) The number of nodes is $n = 21$ **(b)** The number of nodes is $n = 101$

Figure 9 The performance of the proposed broadcast mitigation method

8 Conclusion

Condorcet cycles can occur naturally. While these natural cycles may not significantly disrupt fairness in the system since transactions falling within these cycles are typically received around the same time, the artificial creation of Condorcet cycles can lead to significant unfairness in the system. In this paper, we showed that even with all nodes in the system behaving honestly, it is relatively simple to generate such artificial cycles. Furthermore, we demonstrated that these created cycles possess significant power, as they can trap transactions submitted at widely different times that would not naturally fall within a cycle.

To address this attack, we proposed three mitigation methods using different approaches. These methods complement one another and can be employed collectively to fortify the defensive measures against the attack. Through simulations, we showcased that despite their described limitations, the proposed mitigation methods can substantially reduce the adverse impact of the Condorcet attack.

References

1. Carsten Baum, James Hsin-yu Chiang, Bernardo David, Tore Kasper Frederiksen, and Lorenzo Gentile. Sok: Mitigation of front-running in decentralized finance. *IACR Cryptol. ePrint Arch.*, page 1628, 2021. URL: https://eprint.iacr.org/2021/1628.
2. Felix Brandt, Vincent Conitzer, Ulle Endriss, Jérôme Lang, and Ariel D. Procaccia, editors. *Handbook of Computational Social Choice*. Cambridge University Press, 2016. doi:10.1017/CBO9781107446984.
3. Christian Cachin, Klaus Kursawe, Frank Petzold, and Victor Shoup. Secure and efficient asynchronous broadcast protocols. In Joe Kilian, editor, *Advances in Cryptology - CRYPTO 2001, 21st Annual International Cryptology Conference, Santa Barbara, California, USA, August 19-23, 2001, Proceedings*, volume 2139 of *Lecture Notes in Computer Science*, pages 524–541. Springer, 2001. doi:10.1007/3-540-44647-8_31.
4. Christian Cachin, Jovana Micic, Nathalie Steinhauer, and Luca Zanolini. Quick order fairness. In *Financial Cryptography and Data Security - 26th International Conference, FC 2022, Grenada, May 2-6, 2022, Revised Selected Papers*, Lecture Notes in Computer Science, pages 316–333. Springer, 2022. doi:10.1007/978-3-031-18283-9_15.
5. M. d. Condorcet. Essay on the application of analysis to the probability of majority decisions. *Paris: Imprimerie Royale*, 1785.
6. Philip Daian, Steven Goldfeder, Tyler Kell, Yunqi Li, Xueyuan Zhao, Iddo Bentov, Lorenz Breidenbach, and Ari Juels. Flash boys 2.0: Frontrunning, transaction reordering, and consensus instability in decentralized exchanges. *CoRR*, abs/1904.05234, 2019. URL: http://arxiv.org/abs/1904.05234, arXiv:1904.05234.
7. Cynthia Dwork, Nancy A. Lynch, and Larry J. Stockmeyer. Consensus in the presence of partial synchrony. *J. ACM*, 35(2):288–323, 1988. doi:10.1145/42282.42283.
8. Shayan Eskandari, Seyedehmahsa Moosavi, and Jeremy Clark. Sok: Transparent dishonesty: Front-running attacks on blockchain. In Andrea Bracciali, Jeremy Clark, Federico Pintore, Peter B. Rønne, and Massimiliano Sala, editors, *Financial Cryptography and Data Security - FC 2019 International Workshops, VOTING and WTSC, St. Kitts, St. Kitts and Nevis, February 18-22, 2019, Revised Selected Papers*, volume 11599 of *Lecture Notes in Computer Science*, pages 170–189. Springer, 2019. doi:10.1007/978-3-030-43725-1_13.
9. Lioba Heimbach and Roger Wattenhofer. Sok: Preventing transaction reordering manipulations in decentralized finance. *CoRR*, abs/2203.11520, 2022. arXiv:2203.11520, doi:10.48550/arXiv.2203.11520.
10. Mahimna Kelkar, Soubhik Deb, and Sreeram Kannan. Order-fair consensus in the permissionless setting. In Jason Paul Cruz and Naoto Yanai, editors, *APKC '22: Proceedings of*

the 9th ACM on ASIA Public-Key Cryptography Workshop, APKC@AsiaCCS 2022, Nagasaki, Japan, 30 May 2022, pages 3–14. ACM, 2022. doi:10.1145/3494105.3526239.

11 Mahimna Kelkar, Soubhik Deb, Sishan Long, Ari Juels, and Sreeram Kannan. Themis: Fast, strong order-fairness in byzantine consensus. *IACR Cryptol. ePrint Arch.*, page 1465, 2021. URL: https://eprint.iacr.org/2021/1465.

12 Mahimna Kelkar, Fan Zhang, Steven Goldfeder, and Ari Juels. Order-fairness for byzantine consensus. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology - CRYPTO 2020 - 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17-21, 2020, Proceedings, Part III*, volume 12172 of *Lecture Notes in Computer Science*, pages 451–480. Springer, 2020. doi:10.1007/978-3-030-56877-1_16.

13 Klaus Kursawe. Wendy, the good little fairness widget: Achieving order fairness for blockchains. In *AFT '20: 2nd ACM Conference on Advances in Financial Technologies, New York, NY, USA, October 21-23, 2020*, pages 25–36. ACM, 2020. doi:10.1145/3419614.3423263.

14 Yannis Manoussakis. A linear-time algorithm for finding hamiltonian cycles in tournaments. *Discret. Appl. Math.*, 36(2):199–201, 1992. doi:10.1016/0166-218X(92)90233-Z.

15 David C. Parkes and Lirong Xia. A complexity-of-strategic-behavior comparison between schulze's rule and ranked pairs. In Jörg Hoffmann and Bart Selman, editors, *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence, July 22-26, 2012, Toronto, Ontario, Canada*. AAAI Press, 2012. URL: http://www.aaai.org/ocs/index.php/AAAI/AAAI12/paper/view/5075.

16 Kaihua Qin, Liyi Zhou, and Arthur Gervais. Quantifying blockchain extractable value: How dark is the forest? In *43rd IEEE Symposium on Security and Privacy, SP 2022, San Francisco, CA, USA, May 22-26, 2022*, pages 198–214. IEEE, 2022. doi:10.1109/SP46214.2022.9833734.

17 Michael K. Reiter and Kenneth P. Birman. How to securely replicate services. *ACM Transactions on Programming Languages and Systems*, 16(3):986–1009, 1994. doi:10.1145/177492.177745.

18 Markus Schulze. A new monotonic, clone-independent, reversal symmetric, and condorcet-consistent single-winner election method. *Soc. Choice Welf.*, 36(2):267–303, 2011. doi:10.1007/s00355-010-0475-4.

19 T. N. Tideman. Independence of clones as a criterion for voting rules. *Social Choice and Welfare*, 4(3):185–206, September 1987. doi:10.1007/bf00433944.

20 Yunhao Zhang, Srinath T. V. Setty, Qi Chen, Lidong Zhou, and Lorenzo Alvisi. Byzantine ordered consensus without byzantine oligarchy. In *14th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2020, Virtual Event, November 4-6, 2020*, pages 633–649. USENIX Association, 2020. URL: https://www.usenix.org/conference/osdi20/presentation/zhang-yunhao.