Transparent Transaction Ordering in Blockchain-based Collaborative Processes

Hassan Atwi[1], Tom Lichtenstein[2], Cesare Pautasso[1], Mathias Weske[2]

[1] Software Institute, Università della Svizzera italiana, Lugano, Switzerland
`{Hassan.Atwi,Cesare.Pautasso}@usi.ch`
[2] Hasso Plattner Institute, University of Potsdam, Potsdam, Germany
`{Tom.Lichtenstein,Mathias.Weske}@hpi.de`

Abstract. Blockchain technology offers a promising solution for overcoming trust-related challenges, such as transparency in interorganizational collaborative processes. However, since the ordering of transactions in a block can affect the outcome of a process instance, fairness and transparency concerns regarding block selection may arise. This paper presents a consensus-agnostic approach to promote transparency in transaction ordering in blockchain-based business processes. The approach allows participants to align the execution order of their transactions with their business objectives, effectively mitigating the lack of transparency associated with the arbitrary selection and ordering of transactions during block selection. Our experiments reveal that the increased transparency mitigates the risks of suppression and displacement attacks at the cost of introducing additional latency and cost.

Key words: Blockchain, Choreography, Block selection, Transaction ordering, Transparency

1 Introduction

Coordinating the execution of processes across organizational boundaries typically relies on a trusted, centralized party. Recently, blockchain technology has been studied as a potential infrastructure for cross-organizational collaboration allowing us to overcome this trust issue [17, 2, 9], by providing a decentralized, tamper-proof ledger of transactions. A blockchain is organized in blocks, each of which contains an ordered list of transactions, with new blocks being linked to the previous block using a cryptographic hash function [11]. New transactions are propagated through the network as pending transactions before being added to a block at the end of the blockchain.

While blockchain technology allows for transparency, ensuring fair and reliable ordering of transactions within blocks during block selection remains a challenge. In most blockchains, one leader node, which may be reassigned over time, is responsible for determining the inclusion and order of pending transactions for the next block [20]. Since transactions may represent interactions among collaborating parties, the order of transactions can affect the outcome of a blockchain-based collaborative process or the profitability of the interaction [4, 7, 16]. Therefore, malicious actors taking over the role of the leader node could exploit the freedom to order transactions to gain an advantage over their competitors. Given that in larger networks the pools of known pending transactions and the order in which they are received may differ for each node due to network latency, detecting fraudulent transaction orders remains a challenge [4, 15].

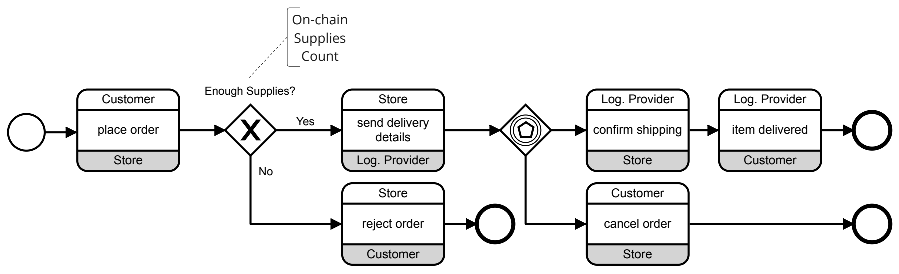


Fig. 1: Choreography diagram representing a simple supply-chain choreography.

To illustrate the impact of transaction order manipulation, we refer to a simple supply-chain choreography depicted in Fig. 1. This choreography orchestrates the interaction between a customer, a store, and a logistic provider. It begins with the customer placing an order at the store, triggering an inventory check. If supplies are insufficient, the customer is notified and the choreography ends. Otherwise, the store forwards delivery details to the logistic provider for shipping. In the latter case, the customer retains the option to cancel the order until the logistics provider confirms the shipping. Upon confirmation, the order is delivered to the customer.

The choreography is implemented on-chain using a smart contract, similar to other blockchain-based process execution approaches [17]. In such a decentralized setting where transaction orders are not centrally managed, the arbitrary nature of transaction ordering can significantly impact the fairness of the choreography. For instance, multiple customers compete for limited supplies from the same store, potentially leading to conflicts across instances of the choreography. Given an on-chain inventory check, the ordering of the transactions determines who acquires the limited supplies. In another scenario, a race condition may occur within a single instance at the event-based gateway, where the logistic provider and customer compete over who makes their choice first. In this case, the ordering of transactions decides if the order will be confirmed for shipping or canceled. While there is a natural race between multiple participants competing for shared and limited resources, malicious participants could perform front-running or censorship attacks [16], so that their transaction requests are prioritized, compromising the fairness in the choreography.

Existing solutions [1, 8, 12, 14] typically employ a random selection and ordering of transactions to maintain fairness. However, this can result in transaction orders not aligning with the participants' intentions. To address this challenge, this paper contributes a novel consensus-agnostic, process-aware transaction ordering approach. This approach allows participants to impact the order of associated transactions directly, ensuring fairness and transparency in block creation that aligns with the participants' intentions.

The paper is organized as follows: Section 2 describes the underlying concepts of fairness in block selection and blockchain-based collaborative processes and outlines related work. Section 3 introduces the transaction ordering approach as the main contribution of this paper. Section 4 evaluates the protocol's resilience to transaction suppression and displacement and discusses the approach in detail. Finally, Section 5 summarizes the paper and points out possible future research directions.

2 Preliminaries and Related Work

This section provides an overview of blockchain-based collaborative processes and fairness in block selection.

2.1 Blockchain-based Collaborative Processes

Blockchain technology offers a promising solution to enable collaboration between mutually distrustful organizations. Recently, model-driven approaches have been investigated to facilitate the design and execution of processes that span multiple organizations [17]. While various modeling languages are used to design blockchain-based collaborative processes, in this paper we refer to Business Process Model and Notation (BPMN) *choreography diagrams* [13].

Choreography diagrams (Fig. 1) represent interactions between two participants with *choreography tasks*, hereafter referred to as *tasks*, with participants with a white band being the *initiators.* Sequence flow arcs specify the order of tasks, while gateways express exclusive and concurrent behavior. The constraints on the order of execution are referred to as *control flow.* Choreography diagrams support two-way tasks with direct responses. We restrict this to one-way tasks, as two-way tasks can be represented by two consecutive one-way tasks without loss of generality. According to the lifecycle depicted in Figure 2, an *initialized* task must first be *enabled* with respect to the control flow constraints before it can be executed, resulting in the *completed* state. In addition, a task may be *skipped*, for example, in the case of exclusive behavior [18].

In a blockchain environment, the execution of a task is typically reflected by a blockchain transaction, hereafter referred to as *transaction.* In second-generation blockchains, the business logic defined by a choreography diagram can be enforced on-chain via *smart contracts* [17]. In the following, we refer to smart contracts dedicated to workflow logic enforcement as *workflow contracts.*

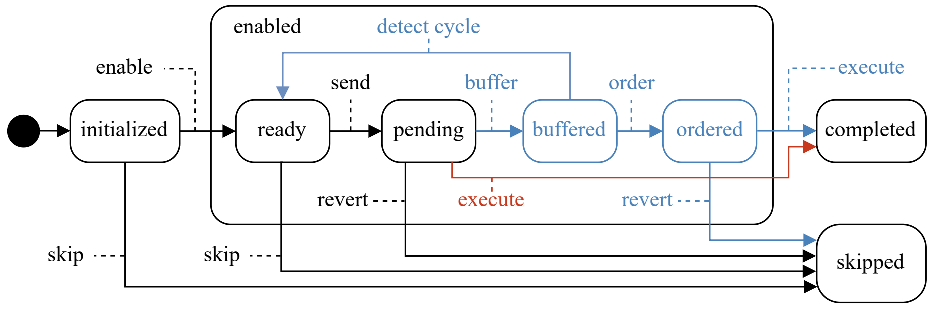


Fig. 2: State transition diagram representing the lifecycle of a choreography task in a blockchain environment using a traditional (red) and ordering contract-based (blue) implementation.

As a blockchain-based task depends upon the successful completion of a transaction, the lifecycle of a blockchain-based task includes additional states between the initial enablement and the completion, as illustrated in Figure 2. Each enabled task is initially in a *ready* state, signaling that a transaction can be sent to execute the task. Once a transaction is sent to the transaction pools of other blockchain nodes, the task is *pending*. In the following, we assume that each transaction sent will eventually be included in a block. In a traditional implementation (highlighted in red), once a transaction is included in a block, it is either *executed* or *reverted* according to the constraints of the workflow contract. The latter may occur if earlier transactions change the state of the workflow contract so that the current transaction conflicts with the workflow constraints, causing the task to be *skipped*. Thus, the course of a choreography can be affected by the execution order of the corresponding transactions [7]. A successful execution of the transaction, on the other hand, results in the task being *completed*.

2.2 Fairness in Block Selection

In current blockchain systems, the responsibility for *block selection*, i.e., selecting and ordering pending transactions to form a new block [1], is typically delegated to an elected *leader node*. Prominent protocols for leader election in public blockchains include: *proof of work* (PoW), as used by Bitcoin [11], which requires leader candidates to solve a computationally complex puzzle, and *proof of stake* (PoS), adopted by Ethereum in 2022 [19], which bases leader election on the respective balances of the nodes. Conversely, in private blockchain environments, leader election typically relies on voting, pseudorandom functions, or round-robin strategies among a predefined set of authority nodes. In the following, we will refer to the leader node as the *block proposer*.

Most common blockchain implementations grant the block proposer full control over the selection and ordering of transactions for the next block. In public blockchains, block proposers often optimize their block selection based on transaction fees to maximize their profit. This incentive can be exploited to perform front-running attacks [16], i.e., paying higher fees to place transactions first to gain an advantage over the competition. Since private blockchains generally do not charge transaction fees, front-running is only possible when block proposers use their privilege to arrange transactions to their advantage by displacing transactions or suppressing undesirable transactions, which raises the question of fairness in block selection in private networks [1]. Since latency or transmission errors can cause two nodes' transaction pools to diverge in terms of included transactions and transaction ordering, detecting malicious block proposers is particularly challenging in decentralized environments [4].

2.3 Related Work

Fairness in block selection received attention in recent blockchain literature. Kelkar et al. [4] identified *receive-order-fairness* as a relevant property for block selection, implying that the order of transactions in a block should reflect the transaction order observed by the majority of nodes. Due to Condorcet's paradox [3], receive-order-fairness is impossible to achieve in a decentralized setting. The authors propose *block-receive-order-fairness*, which ensures that transactions are not assigned to separate blocks in an order that conflicts with the observations of the majority. Wendy et al. [5] propose blockchain-agnostic preprotocols executed in parallel to the blockchain to address order fairness by having validator nodes agree on a block before proposing it to the blockchain. Similar to [4], the authors focus on ensuring order fairness between blocks. In [8], the authors introduce FairLedger, a permissioned protocol in which each node of a predefined committee has an equal opportunity to add its transactions to the block, while all nodes must be aware of all transactions to be added to the next block. The order of the transactions within the block remains undetermined.

Furthermore, Asayag et al. [1] propose Helix, a consensus protocol that uses encrypted transactions to complicate transaction suppression in block selection. In addition, the authors propose a hash-based pseudorandom ordering of transactions to ensure fair block selection. The protocol is extended in [14] with a joint block validation procedure that refines the assessment of the leader node's honesty by comparing the included transactions with those known to a committee of validator nodes. The extension also includes the concept of pending transaction declarations, which allows nodes to periodically query other nodes for their local transaction pool, further complicating transaction suppression. Another extension to Helix is presented in [12], which introduces a reputation system for leader and validator nodes. Finally, Sokolik et al. [15] propose *age-aware fairness*, which aims to limit transaction latency by reserving a predefined space in each block for senior transactions.

This work complements existing work by introducing a blockchain-agnostic approach that achieves fair block selection by 1) providing resilience to suppression, 2) making transaction ordering collaborative and transparent, 3) detecting conflicts in ordering decisions, and 4) allowing nodes to resolve them.

3 Transparent Transaction Ordering

To ensure transparency in the ordering of transactions within blockchain-based collaborative processes, we propose an approach to achieve on-chain transaction ordering based on process information derived from the choreography model. The derived information helps in enabling process-aware transaction ordering. This involves selecting orderers based on the ongoing interactions within the process model.

The approach mainly relies on the introduction of an ordering smart contract, which serves as a preprocessor to organize transaction orders before execution. This approach is consensus-agnostic: it does not depend on the specific consensus protocol adopted by the blockchain network. However, it is assumed that the consensus protocol fairly redistributes the role of the block proposer among the nodes after each added block. As illustrated in Fig. 3, there are three main phases to the approach: buffering, ordering, and execution. We refer to a run through all three phases as an epoch. In the following, we provide a detailed description of the steps within an epoch. In addition, the impact of the approach on the task lifecycle compared to traditional implementations is discussed.

3.1 Buffering Phase

In typical blockchain-based process engine architectures [9], transactions update a workflow contract's state when included in a block, allowing the block proposer to control the execution order. We propose an alternative approach using an ordering contract as a relayer between participants and the workflow contract. Participants issue meta-transactions[1] that are buffered in the ordering contract rather than executed immediately. This buffering phase delays choreography tasks state transitions by introducing an intermediate *buffered* state (Figure 2). If a block is compromised by a malicious actor, buffered transactions can be resubmitted by its original sender in subsequent blocks before execution. This provides participants with an extended window for submitting transactions.

The buffering phase needs to be bounded so that all meta-transactions are executed within a certain time limit. However, due to the decentralized execution of smart contracts, deterministic time measurements are impossible. Oracles [10] could address this limitation by providing access to external sources of precise timing information. Depending on oracles for such essential information poses a risk of undermining the core decentralized foundations of blockchain, as the trust is placed in the hands of the timekeeping party [6]. A reliable alternative to the timestamp is the block number. Thanks to the consensus protocol used to maintain a strict block ordering along the chain, any attempt to manipulate the number of blocks is certain to fail. In our approach, the ordering contract uses the block number to introduce temporal awareness into the transaction ordering process. The block number helps establish a specific duration of the window during which the ordering contract buffers meta-transactions. Participants can

[1] https://docs.openzeppelin.com/learn/sending-gasless-transactions

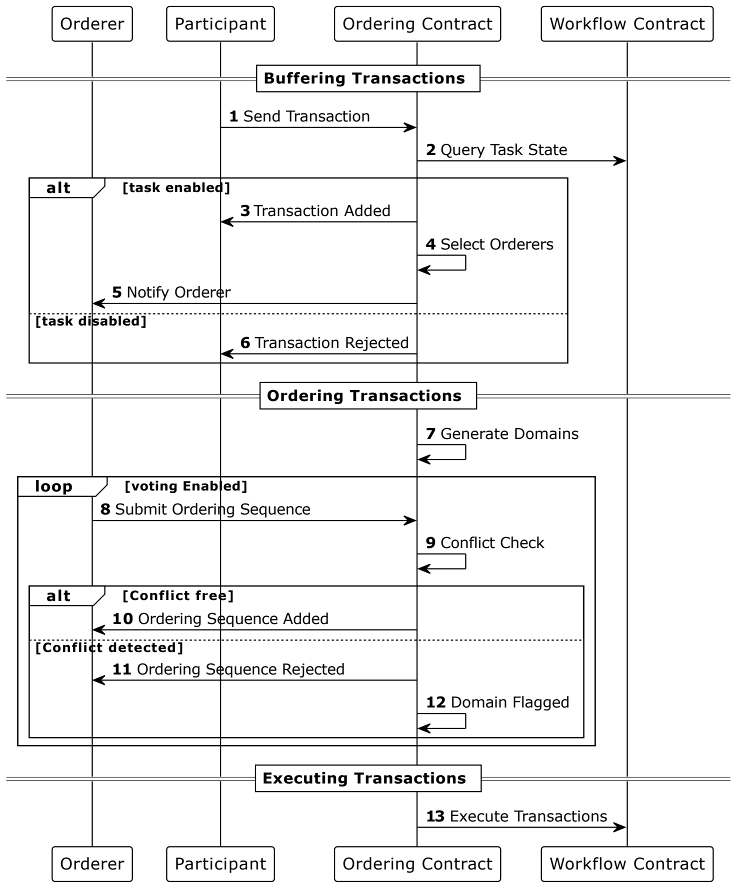


Fig. 3: An epoch consisting of the buffering, ordering, and execution phases, represented as a sequence diagram.

determine the number of blocks dedicated to this buffering period. By extending the buffering phase, participants maximize their opportunities to submit meta-transactions, which can reduce the risk of malicious attacks.

3.2 Ordering Phase

In each epoch, a set of orderers is selected from the choreography participants. The role of an orderer grants the ability to cast votes on the desired sequence for arranging the buffered transactions of an epoch. Notably, each orderer is limited to ordering only the buffered transactions corresponding to the task where they serve as either the sender or the receiver. During the ordering phase of the epoch, the ordering contract stops accepting meta-transactions and proceeds with accepting ordering sequences from the chosen orderers. Once all ordering sequences are collected, the ordering contract combines each ordering sequence, creating a merged list that reflects the consensus on the global ordering among all participating orderers.

To facilitate the explanation of the orderer selection process, we represent the buffered transactions as follows (Fig. 4). Each node in the graph corresponds to a participant. In a choreography instance, each participant assumes a specific role, which may vary from one choreography instance to another. Nodes are connected by edges if a buffered transaction represents a task involving both participants. The set of orderers for each epoch is the subset of participants involved in two or more buffered transactions. Hence, let V be the set of nodes in the graph where each $v \in V$ represents a participant, and N_v is the set of neighbors of the node v. The set of orderers can be expressed as: $O = \{v \mid v \in V \wedge |N_v| > 1\}$. This simple strategy employed for selecting orderers will result in a group of intermediaries who act as *mediators* between two or more participants, thereby establishing them as neutral parties responsible for enforcing the most suitable order. In a plain implementation, there is a risk that if two transactions race against each other, one of the senders might be the block proposer determining the order. Our strategy ensures instead that the mediator chooses the order. For instance, as shown in Fig. 4, in $Epoch_{n+1}$, $store_1$ serves as a neutral party ordering transactions for both customers. Transactions occurring in isolation, such as tx_5 in domain 2, lack any form of competition, eliminating the need for an orderer. Consequently, isolated transactions are inserted into the block regardless of their ordering.

As each orderer individually submits their vote for a preferred sequence of transactions, cycles may result from their combined preferences. The existence of cycles in collective preferences introduces an additional layer of complexity to the ordering process, which could potentially affect other instances by halting the entire network. To mitigate this impact, the ordering contract isolates each interaction graph during an epoch into separate domains. This isolation ensures that if one domain faces cycles in voting, the other domains can proceed with execution independently, thereby deterring any broader impact on the entire network.

Each domain is isolated from the others, as they do not share any common transactions. Orderers exclusively order the transactions within their respective domains. Consequently, the global order between domains is not a concern, since they are not interconnected within an epoch. For instance, in Fig. 4, $Epoch_n$ is depicted with two distinct domains, namely ‘Domain 1’ and ‘Domain 2’. Each domain independently establishes its order of transactions. The final ordering sequence of ‘Domain 1’ comprises transactions tx_1, tx_2, tx_3, and tx_4, arranged according to the specific orders within ‘Domain 1’. ‘Domain 2’ encompasses only one transaction, tx_5, thus no ordering is required. The global order of $Epoch_n$ results from combining the orderings of both domains. The ordering contract disregards the order between the two domains in the global order since they do not share any common transactions.

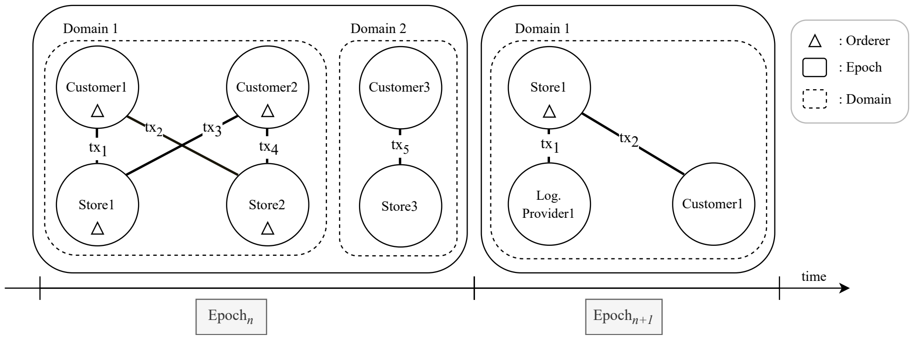


Fig. 4: Interaction graphs within two subsequent epochs.

To demonstrate the possibility of a cycle occurring within the ordering preferences, we take domain 1 (Fig. 4) as an example. Each orderer casts their preference on the ordering sequence. $customer_1$ submits their preference with the sequence $tx_2 > tx_1$, i.e., tx_2 over tx_1, $store_1$ submits the sequence $tx_1 > tx_3$, $customer_2$ submits the sequence $tx_3 > tx_4$, and finally, $store_2$ submits the sequence $tx_4 > tx_2$. The combined sequence of all submissions, based on their submission order, results in the final order $tx_2 > tx_1 > tx_3 > tx_4 > tx_2$. This result serves as a notable instance of the Condorcet paradox [3]. Despite the validity of local orderings, their combination introduces conflict into the collective order.

In case a cycle is detected, the transaction is reverted, and the associated domain is flagged as conflicting. Note that a cycle in the ordering presents a special case: Since all conflicting transactions are reverted, the states of the workflow contracts do not change. Therefore, on the choreography level, the states of the corresponding tasks are not skipped but return to the ready state, as illustrated in Figure 2. This way, participants have the opportunity to resubmit the conflicting transactions so that they can be buffered and ordered again. Otherwise, if a valid order is found, the task advances to the *ordered* state.

3.3 Execution Phase

In the final stage of the epoch, the ordering contract executes the transactions that do not involve conflicting domains by forwarding them to the workflow contract. According to the lifecycle (Figure 2), it is still possible for a task to be skipped at this point. This may happen when earlier transactions change the state of the workflow contract so that the transaction of the task no longer satisfies the workflow constraints. Otherwise, the transaction is executed, changing the state of the workflow contract and causing the corresponding task to reach the completed state. A new epoch begins once all phases are completed.

4 Evaluation and Discussion

The evaluation scenario comprises two instances of the choreography introduced in Fig. 1 denoted by C_1 and C_2. The participants are distributed as follows: In instance C_1, the participants are $\{customer_1, log.\ provider_1, store_1\}$, while in instance C_2, the participants are $\{customer_2, log.\ provider_2, store_1\}$. We execute the scenario in 500 epochs, where competition among participants occurs in two ways: across instances and within instances. Across instances, the two customers race for placing orders for the limited supplies at the same store. We limit the scenario to only one available supply item per epoch; thus, only one customer emerges as the winner in each epoch. The competition within an instance emerges at the event-based gateway, where the customer and the logistics provider compete to cast their choice first, i.e., to confirm the shipment or to cancel the order.

4.1 Setup

To conduct the evaluation, we established a network comprising five nodes running Quorum[2] clients. The network operates on the QBFT[3] consensus algorithm, which is an enterprise-grade consensus protocol recommended for permissioned networks. With QBFT as the adopted consensus mechanism, the block proposer is selected in a round-robin fashion. The consensus protocol is configured to generate a new block every five seconds. To ensure a full representation of participants in the network, we allocate each node to a participant, meaning that each participant possesses a dedicated node. Since a participant has complete control over a node, there is potential to inject malicious behavior into the block creation procedure, e.g., to execute front-running attacks.

Given the unpredictability of participants' intentions in a network, we categorize the nodes into three distinct groups:

1. **Honest Nodes**: These nodes operate without malicious intent and propose a block with the order of transactions as locally observed.
2. **Displacement Nodes**: These nodes prioritize their own transactions by displacing the ones submitted by other participants.
3. **Suppression Nodes**: These nodes suppress all transactions competing with their transactions.

Throughout the experiment, $customer_1$ and $log.provider_2$ play the role of malicious nodes attempting to implement both strategies, namely suppression and displacement, individually in two distinct experiments. Across instances, $customer_1$ will target $customer_2$ to obtain the most supplies. Within their own instance, $customer_1$ will target $log.provider_1$ to maximize canceled choices, while

[2] `https://github.com/Consensys/quorum`

[3] `https://docs.goquorum.consensys.io/configure-and-manage/configure/consensus-protocols/qbft/`

log.provider$_2$ will target *customer*$_2$ to ensure the most confirmed choices. The remaining participants serve as honest nodes.

In our evaluation, we distinguish two setups for invoking choreography tasks: a plain setup, and an ordering contract (OC) setup. In the plain setup, participants send transactions directly to the workflow contract without buffering. In the OC setup, all transactions must pass through the ordering contract, where they are buffered, ordered, and finally executed in the workflow contract. In this experiment, we implemented a straightforward ordering strategy where orderers distribute transactions fairly among participants in a rotating sequence. In each epoch, the priority changes so that each participant's transaction gets a chance to be processed first in turn. Depending on business objectives, the selected orderers could implement more sophisticated strategies. To mitigate the effect of malicious attacks, we implemented a buffer of two blocks for this experiment. This gives honest nodes a window of two blocks to submit their transactions, thus reducing the likelihood of censorship. To have a reproducible experiment, we generate an event log that contains the initial order of interactions and randomly injected delays for 500 epochs. The inclusion of delays serves the purpose of injecting noise into the network, thereby ensuring that the selection of block proposers is not consistently aligned with the order of transactions.

The prototype implementation, including the smart contracts, the event logs, and their analysis results are available on GitHub [4].

4.2 Results

We report the results of the experiment conducted for each setup with the two attack types, i.e., transaction suppression and displacement. In addition, we compare the performance between the setups in terms of latency and cost.

Across Instances. Fig. 5 illustrates the results of supply distribution between the two customers. In a plain setup, the malicious node (*customer*$_1$) could secure the majority of supplies using both types of attacks, primarily through suppression. When employing the ordering contract setup, the supply distribution becomes more evenly distributed between the customers.

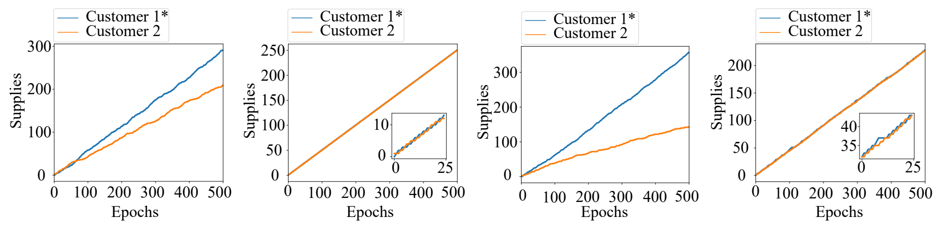
Within Instances. Fig. 6 and Fig. 7 show the number of times each path following the deferred choice was selected within the instances. The malicious nodes (*customer*$_1$ in C_1 and *log. provider*$_2$ in C_2) won most of the races in both instances by applying the two types of attacks. On the other hand, the ordering contract setup enforced an even distribution of deferred choices despite the malicious nodes' actions.

Performance. In terms of performance, as illustrated in Fig. 8, the duration of epochs in the ordering contract setups, on average, exceeds that of the plain setups by a factor of four regardless of the attack type. The cost associated with

[4] `https://github.com/Hassan42/Transparent-Transaction-Ordering`

ordering contract setups was at least nine times that of the plain setup. This outcome is expected, given that the ordering approach involves the execution of additional on-chain logic.

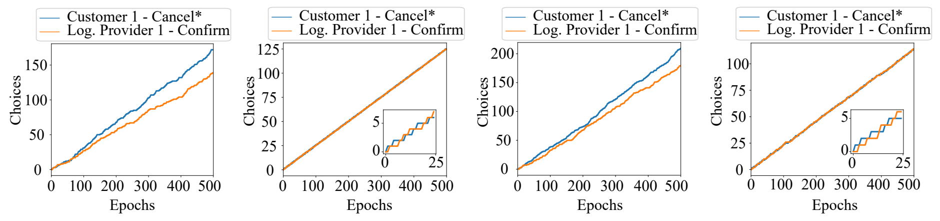
The results indicate that the ordering contract facilitates a process-aware ordering of transactions by selecting the appropriate orderer throughout the choreography's execution. Even in a malicious node-free environment, the ordering contract leverages the transparency and domain-specific ordering of business transactions. However, the trade-off for achieving process-aware transaction ordering is an increase in latency and cost.



(a) Plain Setup Displacement Attack (b) OC Setup Displacement Attack (c) Plain Setup Suppression Attack (d) OC Setup Suppression Attack

*) $customer_1$ is the malicious participant.

Fig. 5: Distribution of supplies between customers across instances.



(a) Plain Setup Displacement Attack (b) OC Setup Displacement Attack (c) Plain Setup Suppression Attack (d) OC Setup Suppression Attack

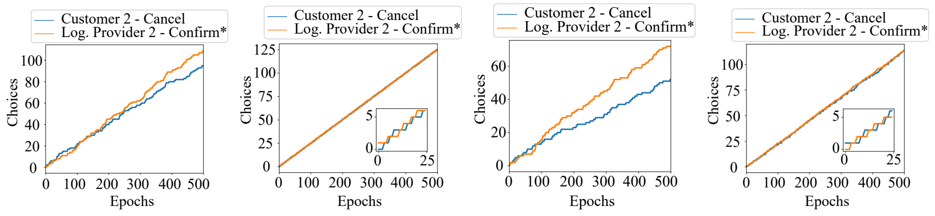
*) $customer_1$ is the malicious participant.

Fig. 6: Distribution of deferred choices within instance C_1.

4.3 Discussion

Based on the experiment's findings, we will discuss the approach regarding liveness, consistency, cost, and latency, with each attribute providing insights into the functionality and constraints of the proposed approach.

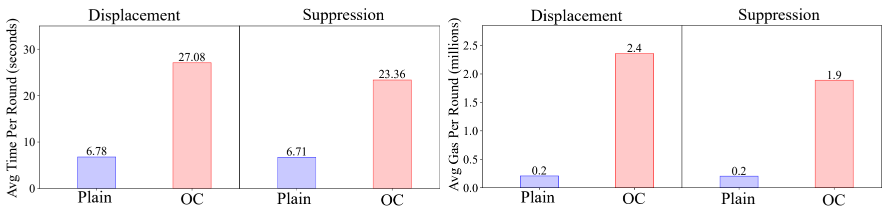
Liveness. Complete assurance of liveness is not guaranteed, as certain ordering sequences may result in conflicts with undefined duration. Resolving such conflicts often requires conflicted participants to compromise on their preferences.



(a) Plain Setup Displacement Attack (b) OC Setup Displacement Attack (c) Plain Setup Suppression Attack (d) OC Setup Suppression Attack

*) $log.provider_2$ is the malicious participant.

Fig. 7: Distribution of deferred choices within instance C_2.



(a) average epoch duration (b) average gas consumption

Fig. 8: Trading-off fairness against performance (a) and gas consumption (b) under suppression and displacement attacks

In our approach, the ordering contract emits an event and flags such cases, enabling conflicting participants to resubmit the transactions. To avoid blocking the entire network, the ordering contract minimizes the potential damage by segregating buffered transactions into domains, thereby ensuring partial liveness in the network. Another crucial factor influencing liveness is the continuous generation of blocks by the network hosting the ordering contract, even for empty blocks. Regular block generation is crucial to the buffering phase in order to maintain regular time intervals and the eventual termination of the phase, even if no new transactions are sent.

Consistency. The inherent properties of the blockchain guarantee a consistent ordering of all transactions included in a block. Executing the ordering procedure on-chain ensures a uniform view among all participants. Hence, in our approach, each phase in an epoch remains visible and consistent for all participants.

Cost. Due to the extra on-chain logic introduced by the ordering contract, executing blockchain-enabled choreographies becomes more costly compared to a plain implementation. Therefore, this approach is better suited for private networks rather than public ones, where transaction costs and gas prices are not as significant.

Latency. Incorporating the ordering contract into workflow execution increases latency, primarily caused by transaction buffering. The duration of transaction buffering can be adjusted through a collective agreement among participants as the network progresses. However, reducing the length of the transaction buffer can facilitate displacement or suppression attacks, as fewer block proposers propagate the transactions for an epoch. Further exploration is needed to develop a dynamic approach for tuning the epoch length to balance the trade-off between latency and the level of protection against malicious attacks.

5 Conclusion and Future Work

This paper proposes a novel transparent approach for ordering transactions using contextual information provided by choreography models. The approach mainly relies on buffering transactions and arranging them on-chain before they are forwarded to a smart contract implementing the underlying business logic. The additional measures reduce or in some cases even eliminate the impact of front-running attacks, e.g., transaction suppression or displacement. While the evaluation demonstrates significant improvements in fairness and transparency compared to the baseline, this outcome comes at the expense of increased latency and cost. The suggested approach lays the foundation for additional experiments involving scenarios with a larger number of participants as well as future enhancements and additional investigations into optimizing epoch length selection, exploring new ordering strategies, and embedding process awareness in the underlying consensus protocol.

Acknowledgement

Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – 450612067, and by the Swiss National Science Foundation (SNSF) under project ”Flexible Choreographies in Multi-chain Environments“ (196958).

References

1. Asayag, A., Cohen, G., Grayevsky, I., Leshkowitz, M., Rottenstreich, O., Tamari, R., Yakira, D.: A fair consensus protocol for transaction ordering. In: 26th IEEE International Conference on Network Protocols. pp. 55–65 (2018)
2. Dwivedi, V., Norta, A.: Auto-generation of smart contracts from a domain-specific xml-based language. In: Intelligent Data Engineering and Analytics: Proc. of the 9th International Conference on Frontiers in Intelligent Computing: Theory and Applications (FICTA 2021). pp. 549–564. Springer (2022)
3. Gehrlein, W.V.: Condorcet's paradox. Theory and Decision **15**(2), 161–197 (1983)
4. Kelkar, M., Zhang, F., Goldfeder, S., Juels, A.: Order-fairness for byzantine consensus. In: Micciancio, D., Ristenpart, T. (eds.) 40th Annual International Cryptology Conference. LNCS, vol. 12172, pp. 451–480. Springer (2020)

5. Kursawe, K.: Wendy, the good little fairness widget: Achieving order fairness for blockchains. In: 2nd ACM Conference on Advances in Financial Technologies. pp. 25–36 (2020)
6. Ladleif, J., Weske, M.: Time in blockchain-based process execution. In: 24th International Enterprise Distributed Object Computing Conference. pp. 217–226 (2020)
7. Ladleif, J., Weske, M.: Which event happened first? deferred choice on blockchain using oracles. Frontiers Blockchain **4**, 758169 (2021)
8. Lev-Ari, K., Spiegelman, A., Keidar, I., Malkhi, D.: Fairledger: A fair blockchain protocol for financial institutions. In: 23rd International Conference on Principles of Distributed Systems. vol. 153, pp. 4:1–4:17 (2019)
9. López-Pintado, O., García-Bañuelos, L., Dumas, M., Weber, I., Ponomarev, A.: Caterpillar: A business process execution engine on the ethereum blockchain. Software: Practice and Experience **49**(7), 1162–1193 (2019)
10. Mühlberger, R., Bachhofner, S., Castelló Ferrer, E., Di Ciccio, C., Weber, I., Wöhrer, M., Zdun, U.: Foundational oracle patterns: Connecting blockchain to the off-chain world. In: Proc. Blockchain and Robotic Process Automation Forum (at BPM). pp. 35–51 (2020)
11. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system. Tech. rep. (2008), `https://bitcoin.org/bitcoin.pdf`
12. Nassar, M., Rottenstreich, O., Orda, A.: Communication-aware fairness in blockchain transaction ordering. In: 23rd IEEE International Conference on High Performance Switching and Routing. pp. 175–182 (2022)
13. Object Management Group (OMG): Business Process Model and Notation (BPMN), Version 2.0.2 (2014), `https://www.omg.org/spec/BPMN/2.0.2/`
14. Orda, A., Rottenstreich, O.: Enforcing fairness in blockchain transaction ordering. Peer-to-Peer Netw. Appl. **14**(6), 3660–3673 (2021)
15. Sokolik, Y., Rottenstreich, O.: Age-aware fairness in blockchain transaction ordering. In: 28th IEEE/ACM International Symposium on Quality of Service (2020)
16. Torres, C.F., Camino, R., State, R.: Frontrunner jones and the raiders of the dark forest: An empirical study of frontrunning on the ethereum blockchain. In: 30th USENIX Security Symposium. pp. 1343–1359 (2021)
17. Weber, I., Xu, X., Riveret, R., Governatori, G., Ponomarev, A., Mendling, J.: Untrusted business process monitoring and execution using blockchain. In: 14th International Conference on Business Process Management. LNCS, vol. 9850, pp. 329–347 (2016)
18. Weske, M.: Business Process Management - Concepts, Languages, Architectures, Third Edition. Springer (2019). https://doi.org/10.1007/978-3-662-59432-2
19. Wood, G.: Ethereum: A secure decentralised generalised transaction ledger. Ethereum project yellow paper (2014), `https://ethereum.github.io/yellowpaper/paper.pdf`
20. Xiao, Y., Zhang, N., Lou, W., Hou, Y.T.: A survey of distributed consensus protocols for blockchain networks. IEEE Communications Surveys & Tutorials **22**(2), 1432–1465 (2020)