Lecture 13: Transaction Ordering and Fairness

Principles of Blockchains, University of Illinois,
Professor: Pramod Viswanath
Scribe: Soubhik Deb

March 11, 2021

Abstract

Consensus in blockchains this far refers to an ordered sequence of blocks. We have not paid attention to the ordering of the transactions within a block, which is left to the miner who assembles the block. The ordering is important because the fees associated with the transactions and the value of the native tokens at different points in the ledger can depend on the ordering. The importance of fair ordering is underscored by massive *front-running* attacks on the popular blockchain Ethereum. A fair ordering of transactions would mean that if one transaction was transmitted before another one then this ordering would be maintained in the final ledger. In this lecture we demonstrate a clean adaptation of the longest chain protocol to incorporate fair transaction ordering. We discuss how to quantify the level of fairness of the ordering (different nodes can see different ordering due to network delays, be they natural or through adversarial action) and show that the adaptation only adds a modest extra latency to the k-deep confirmation rule while guaranteeing high confidence of fair ordering.

Introduction

In centralized financial (CeFi) markets, the ordering in which market orders are executed in the exchanges play a crucial role in making a profit or loss of millions or potentially, billions of dollars. However, such high-stakes also incentivizes adversarial actors to engage in manipulating the execution ordering of these market orders. Such practices are termed as **frontrunning**. See Fig. 1 for an illustration of a frontrunning attack. Frontrunning is strictly monitored and prohibited in the centralized stock exchanges and past incidents of frontunning has resulted in fines worth millions of dollars by SEC and FBI.

With the rising prominence of the decentralized finance (DeFi) industry, a similar order manipulation via frontrunning has become very lucrative. Unlike in CeFi, this phenomenon has been exacerbated in DeFi due to the lack of any trusted authority that can regulate financial malpractices in the decentralized setting. As the order in which transactions are included in a block in the blockchain depends on the discretion of the miner of that block, the miners are generally incentivized to arrange those transactions earlier that earns them the most transaction fees. So, a client can pay a large transaction fee to have a frontrunning transaction ordered earlier in the blockchain. On the other hand, the clients that are victims of such frontrunning attacks have no way to know, verify and get redressal from such a nefarious ordering event. This has given rise to rampant order manipulation in DeFi as indicated by bots embedded inside public blockchains (e.g., Ethereum) and generating frontrunning transactions. According to one estimate, these frontrunning bots in Ethereum have proved to be very successful and have extracted a profit of more than $300 million since January 2020. Figure 2 illustrates a real-world example of frontrunning attack in DeFi in which a frontrunning bot takes advantage of the arbitrage in the exchange rates between different decentralized exchanges (DEX) to make a substantial profit. The immediate externality from frontrunning attacks

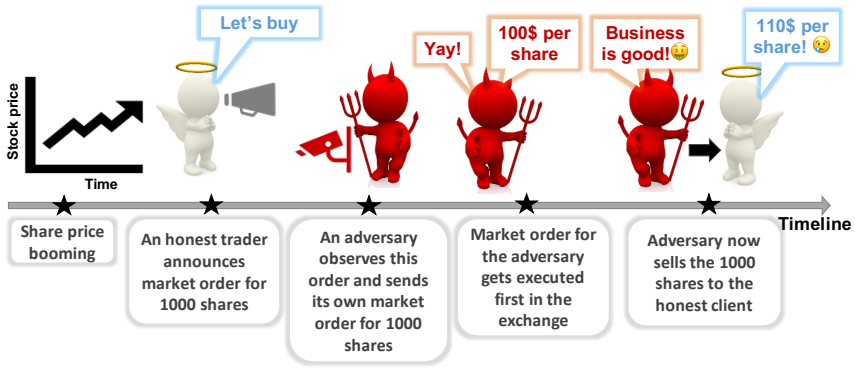


Figure 1: Suppose that the share price for some arbitrary company is increasing. After observing the buy order of the honest trader, the adversary creates its own buy order and gets it somehow executed before the former in the exchange. Adversarial trader could do so due to bribing, self-dealing, privy to inside information, etc. Now, the adversarial trader can sell these shares to the honest trader at a profit.

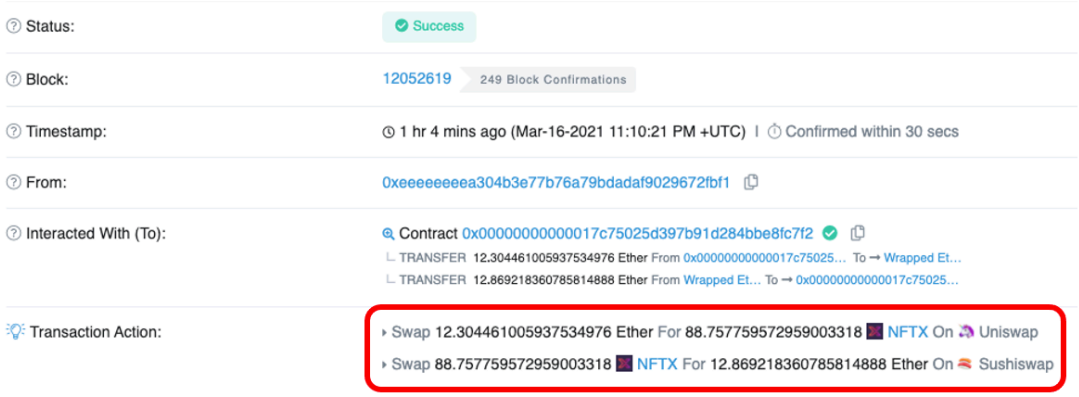


Figure 2: A bot sitting in the Ethereum executed couple of token swaps atomically using smart contract. The smart contract takes advantage of the arbitrage in the exchange rate between Ether (native token in Ethereum) and NFTX token (an ERC20 token) in two different DEXes Uniswap and Sushiswap. Seeing the increasing demand for NFTX in Sushiswap, the smart contract employs frontrunning to swap Ether for NFTX in Uniswap and then sells these NFTX tokens in the Sushiswap at a profit. In terms of the exchange rate between Ether and USD at the above timestamp, the bot executing this frontrunning made a profit of greater than $900 in a single atomic pair of transactions.

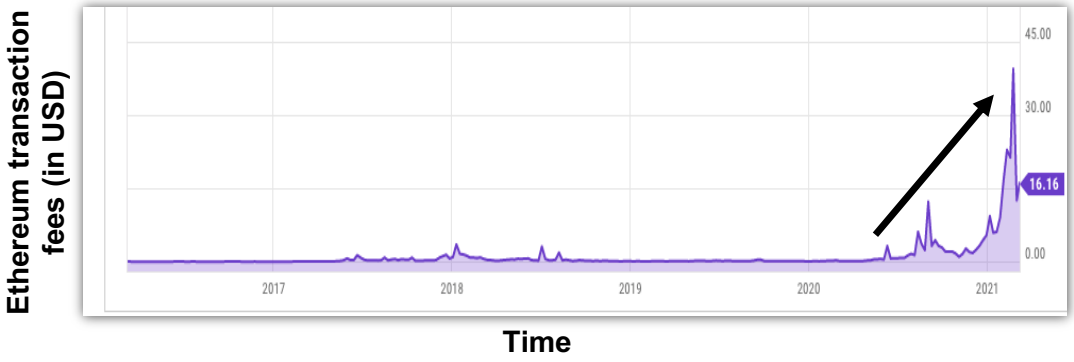


Figure 3: Ever since DeFi came into the mainstream in the summer of 2020, there has been an exponential increase in transaction fees in Ethereum, primarily, due to frontrunning.

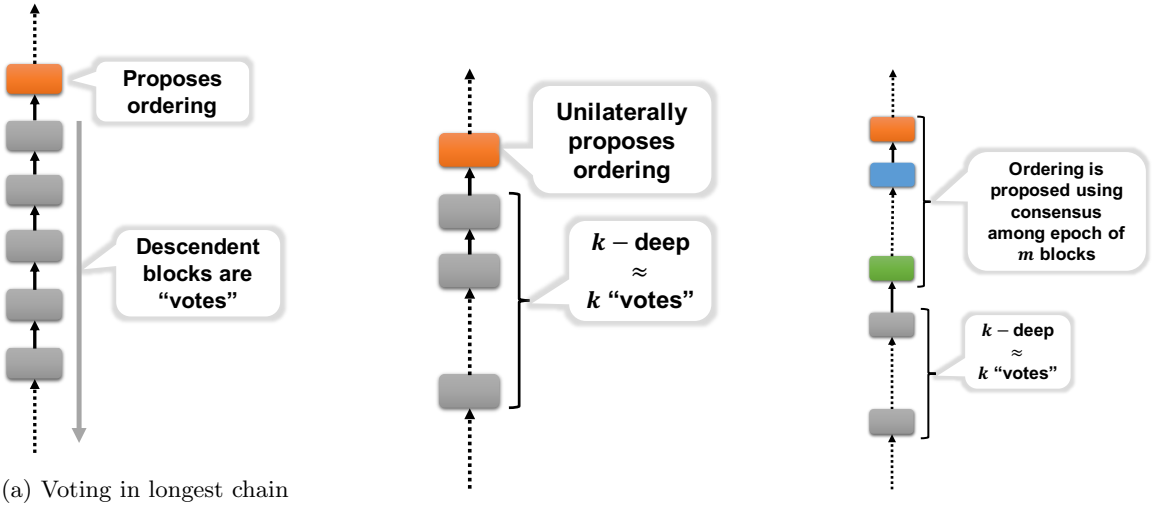
in DeFi is in the dramatic increase in the transaction fees (known as "gas"), as evident from Figure 3 for Ethereum. The increase in transaction fee occurs because there can be multiple competing bots trying to frontrun the *same* transaction: a frontrunning bot iteratively bids up its transaction fees in order for its frontrunning transaction to get ordered in a block by the miner ahead of the transactions of any of its competing bots or the original transaction which is being targeted for frontrunning. Additionally, there can be many failed transactions (because they ran out of gas) and reverted transactions from frontrunning which can occupy valuable space in the blockchain. Thus, these frontrunning attacks have resulted in negative externalities in Ethereum: network congestion in the peer-to-peer network and reduced block space usage in the chain. The frontrunning attacks have also contributed to the miner reaping as much profit (known as "miner-extractable value") in the form of transaction fees as possible from ordering-for-profit.

One possible solution is to impose a fixed fee that is agnostic to the actual value of that transaction (this is the essence of EIP-1559, a recent improvement proposal for Ethereum). However, a frontrunning bot can simply collude with a miner in an off-chain market and get its frontrunning transaction ordered earlier. So the challenge remains: can one design a fair-ordering consensus protocol that is provably secure against manipulation of transaction ordering? Of specific interest is a simple transaction ordering strategy that can be readily incorporated alongside the longest chain protocol. Such is the goal of this lecture.

Fair Ordering Protocol

In the longest chain protocol, each node has their own preferences on how the transactions should be ordered. The obvious question is how to reconcile all of these individual preferences to get one consistent ordering of transactions across all the honest nodes (we will call it fair-ordered ledger) from the fair-ordering protocol? Recall that as there are Byzantine adversaries in the system who can report fake ordering of transactions to the fair-ordering protocol, adequate security guardrails have to be present while reconciling views from different nodes so that Byzantine adversaries do not manipulate the ordering. Furthermore, the nodes participating in the longest chain protocol are pseudonymous (no fixed identity) and the participation level is varying.

Key idea. It is clear that the direct collection of preferences on the ordering of transactions from each node for reconciling all these preferences is not practical – the associated communication and storage complexities would be horrendous. However, the blocks in the longest chain can serve as natural intermediaries to sample from, and reconcile, the diverse ordering preferences. To be precise,



(a) Voting in longest chain protocol.

(b) $k-$deep confirmation.

(c) Proposal by consensus.

Figure 4: Pictorial illustration of the key idea in how to incorporate a simple transaction ordering strategy in longest chain protocol.

when a block B is mined by an honest miner, the miner orders the transactions inside the block according to its own preferences of the ordering of those transactions. Now, when another miner extends the chain from B we have seen that this miner action can be viewed as casting a "vote" of confirmation of the ordering of transactions proposed in block B; see Figure 4a. When there are k votes, i.e., B is buried k-deep in the longest chain, we can confirm the proposed ordering of transactions; see Figure 4b. Unfortunately all the descendant blocks of B are voting on the same specific ordering of the transactions in the block B, authored by the proposer of block B.

The key idea to get around this unilateral proposition of ordering of transactions is the following: have multiple descendant blocks (say, a sequence of of m blocks) come together and propose an ordering of transactions by consensus; see Figure 4c. With m sufficiently large, using chain-quality property of longest chain protocol (see Lecture 7), other nodes can be confident that the majority of the blocks in this sequence of m blocks have been mined by honest miners with high probability. Now, the ordering between any pair of transactions Tx_1 and Tx_2 in the fair-ordered ledger in any honest node is inferred based on whether "Tx_1 is ordered before Tx_2" or "Tx_2 is ordered before Tx_1" in the majority of the m blocks. The ordering of transactions obtained by consensus among the epoch of m blocks is confirmed after all these m blocks are $k-$deep, that is, k votes have been cast for this ordering. For notational purpose, if Tx_1 is ordered before Tx_2, then it is represented by $Tx_1 \rightarrow Tx_2$.

As mentioned earlier, each node could have their own preferences on how to order a set of transactions. The honest nodes deduce their preference of the ordering among these transactions from the order in which they received those transactions locally from the peer-to-peer (p2p) network. Owing to network delay or adversarial behavior in the network, different nodes might receive the same set of transactions from the p2p network in different orders and at different times; see Figure 5. These differences get reflected in the order in which the transactions are arranged in each of the m blocks in the epoch that are used for reaching consensus on their proposed ordering. Furthermore, a Byzantine adversary can arrange the transactions in arbitrary order in the block it mines so as to influence the final ordering obtained from consensus. However, owing to chain-quality property of the longest chain protocol, this ability of the adversary to do order-manipulation can be made negligible.

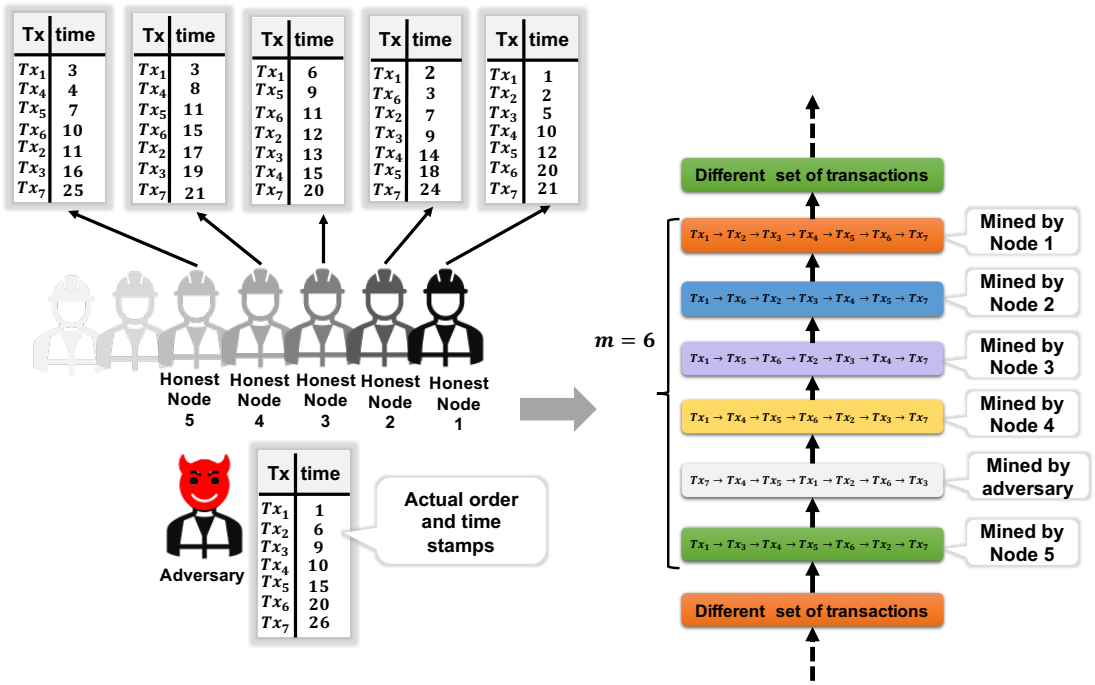


Figure 5: Consider a set of transactions $\{Tx_1, Tx_2, Tx_3, Tx_4, Tx_5, Tx_6, Tx_7\}$. Observe that the same set of transactions are received at different nodes in different orders and at different times. For the sake of convenience, we have illustrated the order and time in which this set of transactions are received for only five honest nodes and one adversarial node. Let the epoch required for reaching a consensus on proposing an ordering of transactions be of size $m = 6$. In the blocks mined by the honest nodes $1, 2, 3, 4$ and 5, the transactions are arranged in the order that they received those transactions locally. On the other hand, an adversarial node can disregard the order in which the transactions are received locally and arrange the transactions in the block it mined in any arbitrary order that suits its needs, like, arranging transactions in the order of decreasing transaction fees.

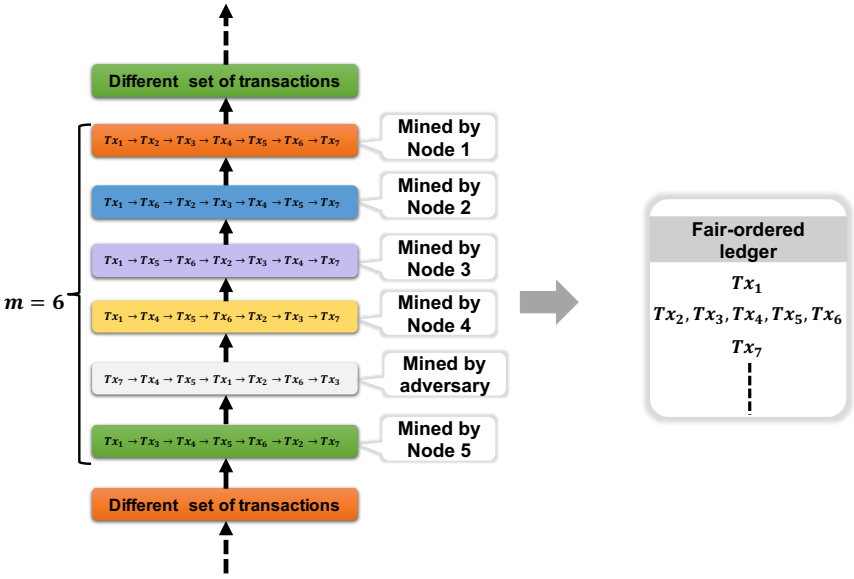


Figure 6: Pictorial representation of obtaining a fair-ordered ledger from the blockchain.

After all the m blocks in the epoch are $k-$deep, majority rule can be employed in order to reach consensus on the ordering on the transactions among the m blocks. Observe that, in Fig. 5, Tx_1 is preferred before other transactions in 5 out of 6 blocks in the epoch. So, in the fair-ordered ledger, Tx_1 will be ordered before other transactions. As for other transactions, $Tx_2 \to Tx_3$, $Tx_3 \to Tx_4$, $Tx_4 \to Tx_5$, $Tx_5 \to Tx_6$ and $Tx_6 \to Tx_2$ in majority of the blocks. However, this results in a paradoxical circular ordering. To circumvent this, one idea is to order all these transactions in a paradoxical ordering together in one batch in the fair-ordered ledger as illustrated in Fig. 6. Continuing ordering in this epoch-by-epoch manner using majority-voting rule and batching, a consistent fair-ordered ledger is obtained at each honest node in the system.

Pipelining. In the fair-ordering protocol presented above, all the blocks within an epoch orders the *same set* of transactions. This reduces the transaction throughput by a factor of m, the epoch size. What is required is that each transaction should be ordered in a "streaming" fashion. This can be accomplished by mandating that when a transaction is ordered in a block for the first time, this transaction must be included in the ordering of the transactions of the next $m-1$ descendent blocks. Thus, each transaction has its own epoch of m blocks where it is included in the ordering; see Figure 7.

How to order the transactions in the fair-ordered ledger whose epochs are overlapping? For example, in Fig. 7, notice that the epoch for transactions Tx_1, Tx_2, Tx_3, Tx_4 and the epochs for transactions Tx_5, Tx_6 are not the same but overlapping. We address this by requiring the protocol to construct m *semantic chains*. Each of these semantic chains are constructed by appending the transaction orderings from the blocks with same color. For example, semantic chain $\texttt{SemChain}_1$ (represented by orange color) is constructed in a streaming fashion by appending the transaction orderings in the orange-colored blocks. Now, as earlier, majority rule is employed in order to reach consensus on the ordering on the transactions among the m semantic chains.

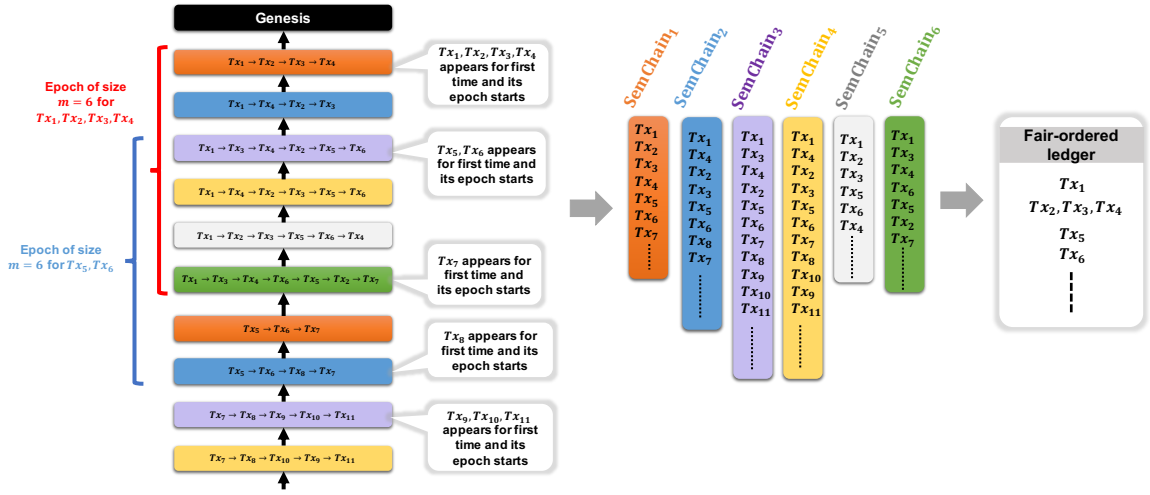


Figure 7: Transforming the fair-ordering protocol into a streaming protocol. In the longest chain, all blocks whose depth from the genesis modulo m are same are assigned same color. Each of the m semantic chains is constructed by appending the transaction ordering from the blocks with the same color. Observe that each transaction appears only once in each semantic chain. Majority rule is then employed to achieve a consensus on the ordering of the transactions in the fair-ordered ledger.

Fair Ordering Guarantees

The fair-ordering protocol described above outputs a globally-consistent ordering of transactions but how fair is this ordering and how to evaluate that fairness? what guarantees does this fair-ordered protocol make against any possible order manipulation by a Byzantine adversary? The ordering is fair as long as a majority of the m block proposers are honest; this is simply because the overall ordering is based on the majority of the individual m orderings. The fraction of honest miner blocks in the longest chain is known as *chain quality* CQ; see Equation 3 in Lecture 7. Now with PoW mining rate set slow enough, $CQ > \frac{1}{2}$ when $\beta < \frac{1}{3}$. The chain quality condition gets more accurate as m increases, with the fair-ordering protocol guarantees batch-order fairness for $\beta < \frac{1}{3}$ with probability approaching 1.

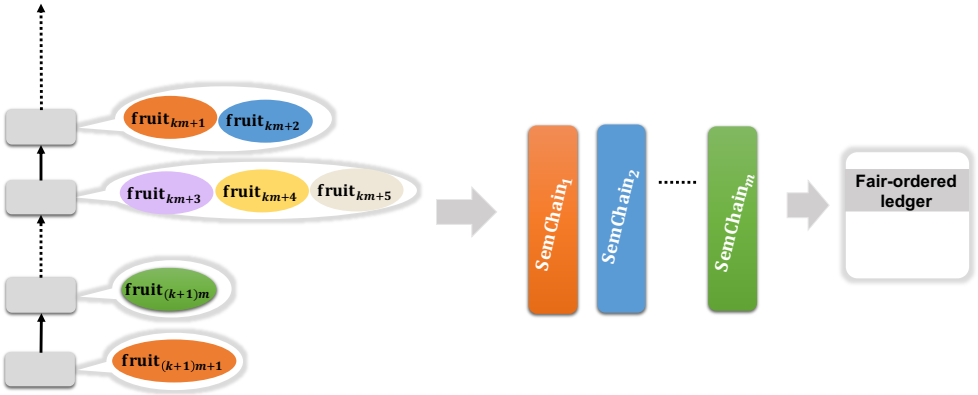
We have proposed batch order fairness as the metric to state the ordering guarantees. A detailed discussion about other possible metrics is in the appendix.

Increasing the adversarial threshold. The Fruitchains protocol was proposed in Lecture 7 to improve the chain quality to its optimal level; a natural question is how to adopt the fair-ordering protocol here to the Fruitchains setting. We do this by not having blocks of m different colors composing an epoch; instead now fruits of m different colors would compose an epoch (see Figure 8a). Now, each of the m semantic chains would be constructed by appending the ordering of transactions contained in fruits of same color.

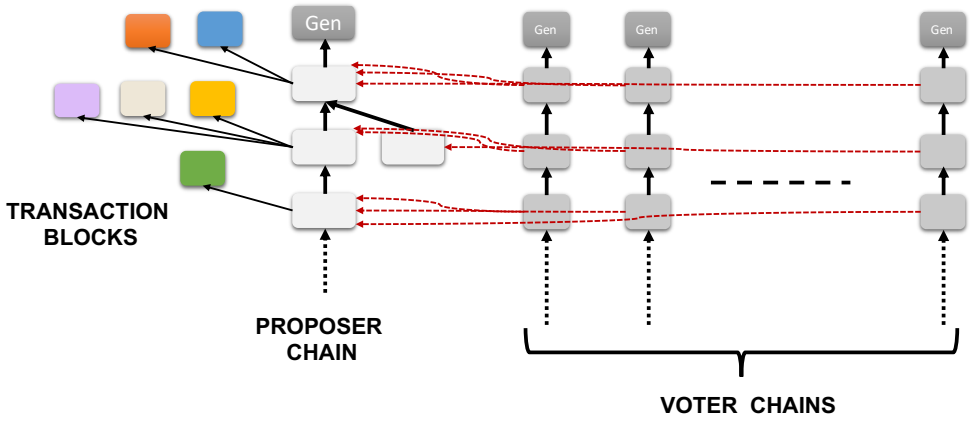
Reducing confirmation latency. In Lecture 9 we have studied the Prism protocol to incorporate multiple voting chains to significantly drop the latency of confirmation, which can be used in conjunction with the fair-ordering protocol; see Figure 8b.

Zero-Block Confirmation

Zero-block confirmation has been a topic of interest from the early days of Bitcoin, and touted as an important step towards the practicality of Bitcoin for day-to-day purchases. With zero-block confirmation property, *a node can confirm a transaction without even seeing a single mined block*



(a) Each of the m semantic chains will be constructed by appending the transaction ordering from the fruits with the same color.



(b) Multiple voter chains are used to reduce the latency for confirmation of blocks.

Figure 8: Improvements on the fair-ordering protocol.

that contains the transaction. To motivate the gravity of its importance, consider the following payment scenario: Alice wishes to buy a coffee from Carol's Coffee, using Bitcoin, which uses an underlying Nakamoto-style PoW blockchain. Alice pays 0.00001 Bitcoin for her coffee. However, for Alice's transaction to be confirmed by the network, it could take several PoW blocks to make sure the transaction is buried sufficiently deep in the chain. This could be in the order of hours, which makes Bitcoin based payment systems untenable for practical transactions. Now, it is possible that Carol's Coffee can immediately accept Alice's transaction and Alice can get her coffee quickly. But in the longest chain protocols like Bitcoin, ordering of transactions within a block is unilaterally decided by the miner of the block. So, Alice can soon after send a double spending transaction and an adversarial miner can order this double spending transaction from Alice before the transaction made by Alice to Carol's Coffee.

As described in previous sections, the fair-ordering protocol replaces this unilateral decision-making on the ordering of transactions with a consensus-based proposal on the ordering of transactions. As a consequence, the fair-ordering protocol guarantees the powerful zero-block confirmation property.

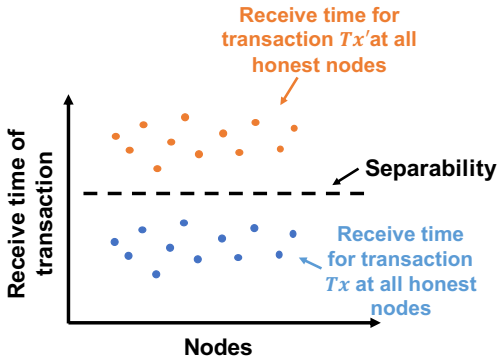
References

Flash Boys 2.0 was the first major work which brought attention to the phenomenon of frontrunning that is prevalent in DeFi. This work documented and quantified the widespread and rising deployment of arbitrage bots in blockchain systems, specifically in decentralized exchanges (DEXes). These two articles titled as "Ethereum is a dark forest" and "Escaping the dark forest" present interesting real-world case studies on the severity of frontrunning in Ethereum. The ubiquity of frontrunning bots necessitated the requirement for a new consensus property called *fair-ordering* for distributed systems. KZGJ and ZSCZA formalized the definition of fair-ordering in the context of Byzantine consensus and constructed protocols to achieve them. KDK introduced the first consensus protocol that achieves fair-ordering in permissionless setting.

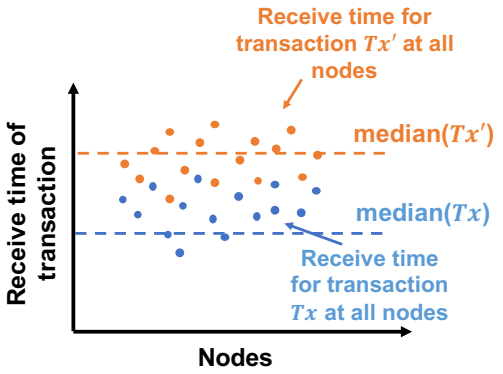
Possible Definitions for a Fair Ordering of Transactions

An honest node in the blockchain system derives fair-ordered ledger from the longest chain. In order to mitigate against manipulation of the order of these transactions, first we need to state a suitable definition for a fair transaction ordering that the ledger must satisfy. We briefly explain multiple candidate definitions that would illustrate the nuances of defining fair transaction ordering.

- **Send-order-fairness**. A strawman approach for ordering transactions would be based on the timestamps **when** the transactions were broadcast by the clients in the peer-to-peer network. For instance, if a transaction Tx was sent by a client before another transaction Tx' (possibly by another client), then Tx should appear before Tx' in the fair-ordered ledger of all honest nodes. However, this approach has a big problem: there needs to be a trusted way to timestamp a transaction at the client side. This would require some third-party trusted execution environments, e.g., Intel SGX which doesn't align with the vision of permissionless setting.
- **Time relative fairness or ordering linearizability**. This definition requires that if all honest nodes receive a transaction Tx before any honest node receives Tx', then Tx will be ordered before Tx' in the fair-ordered ledger of all honest nodes. In other words, if the **latest** receive time for Tx at any honest node is before the **earliest** receive time for Tx' at any honest node, then Tx should be ordered earlier. See Fig. 9a for an illustration. However, this definition is not sufficient to mitigate the aforementioned front-running attacks as it enforces no guarantees on transactions whose receive times are globally interleaved. To be more precise, an adversarial transaction that attempts to front-run an honest user transaction, is most likely



(a) For ordering linearizability. The receive time for transactions Tx and Tx' at each honest node are depicted by blue-colored and orange-colored dots, respectively.



(b) For median-fairness. The receive time for transactions Tx and Tx' at each node are depicted by blue-colored and orange-colored dots, respectively.

Figure 9: Pictorial representation for ordering linearizability and median fairness.

to end up getting interleaved with the user transaction in its receive times at different nodes. Consequently, for most practical front-running scenarios, this definition would be vacuous and not particularly useful.

- **Median fairness**. Under this definition, if the median of the set of receive times for Tx at all nodes is smaller than the median of the set of receive times for Tx' at all nodes, then Tx should be ordered before Tx' in the fair-ordered ledger of all honest nodes. See Fig. 9b for an illustration. Although this definition takes into consideration aforementioned interleaving, however, a protocol guaranteeing median fairness would be vulnerable to order manipulation due to even a single adversarial node reporting a flipped receive timestamp to the protocol. To be precise, consider two transactions Tx and Tx' and five nodes, node A, B, C, D and E, where E is the adversary that is Byzantine in nature. Tx is received by nodes $A, \cdots, E$ at rounds 1, 1, 4, 4, 2 while Tx' is received by the nodes at rounds 2, 2, 5, 5, 3. Now, all nodes have received Tx before Tx' and consequently, $median(Tx) < median(Tx')$. However, the adversarial node E could falsely report its receive times for Tx to be round 3 and that for Tx' to be round 2. Now, $median(Tx) = median(1, 1, 4, 4, 3) = 3 > median(Tx') = median(2, 2, 5, 5, 2) = 2$. Thus, even a single Byzantine adversary is able to invert the ordering of transactions Tx and Tx' by reporting false receive times.

- **receive-order-fairness**. Intuitively, recalling the aforementioned example of front-running, a sound definition for fair ordering of transactions must ensure that if a transaction Tx is **received before** the transaction Tx' by a **large fraction** of nodes in the blockchain system, then Tx must be **ordered before** Tx' in the fair-ordered ledger of an honest node. See Fig 10 for an illustration of this intuition. However, due to Condorcet paradox, receive-order-fairness is impossible to achieve. To understand this impossibility result, assume that there are currently N nodes - node 1, node 2, $\cdots$, node N, in the system. Suppose that the node 1 received transactions in the order $[Tx_1, Tx_2, \cdots, Tx_N]$ and any node $n \neq 1$ received transactions in the order $[Tx_n, \cdots, Tx_N, Tx_1, \cdots, Tx_{n-1}]$. Observe that for any pair of consecutive transactions Tx_n, Tx_{n+1}, $n \in [1, 2, \cdots, N-1]$, there are $N-1$ nodes that received Tx_n before Tx_{n+1}. Also, there are $N-1$ nodes that received Tx_{N-1} before Tx_1. This means that any consensus protocol satisfying receive-order-fairness must order Tx_1 before Tx_2, Tx_2 before Tx_3, $\cdots$, Tx_{n-1} before Tx_n and Tx_n before Tx_1, which is a contradiction. Thus, despite transitive ordering of

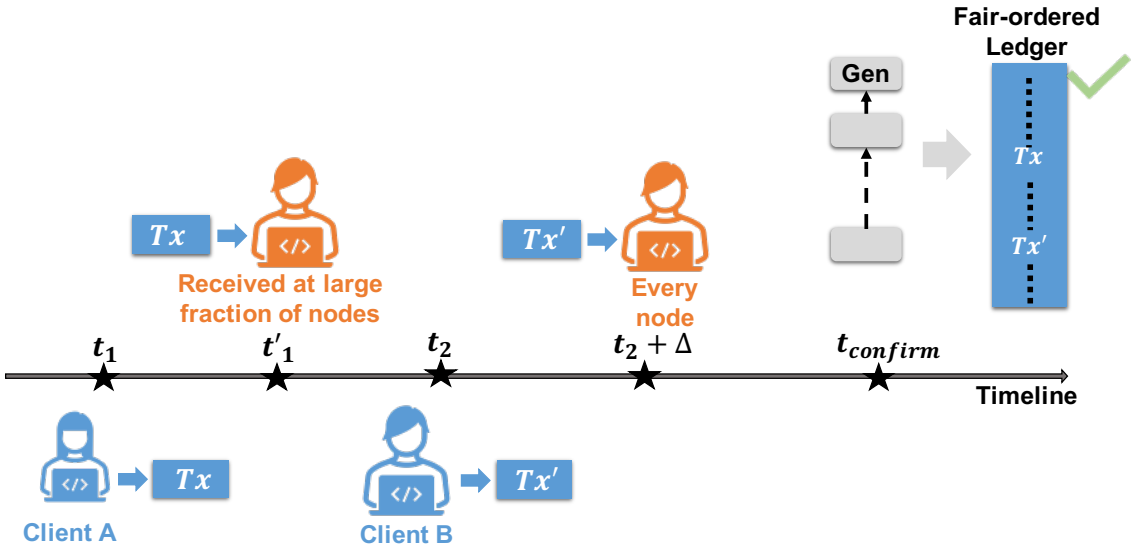


Figure 10: Suppose the transaction Tx, broadcast by client A at time t_1, is received by a large fraction of nodes at t'_1 before client B broadcasts the transaction Tx' at t_2. Then, when the blocks containing transactions Tx and Tx' are confirmed at $t_{confirm}$, $\gamma-$receive-order-fairness specifies ordering Tx before Tx' in the fair-ordered ledger of an honest node.

the transactions at each node, the collective final ordering is non-transitive and paradoxical.

- **Batch-order-fairness**. This definition states that if a transaction Tx is **received before** the transaction Tx' by a large fraction of nodes in the blockchain system, then the fair-ordered ledger will order Tx **no later** than Tx'. Observe that this definition is a relaxation on receive-order-fairness. Referring to the example given in receive-order-fairness, the paradoxical orderings due to Condorcet paradox can be sidestepped by ordering all the transactions in the paradoxical orderings together in the same batch and, thus, ordering them together in the fair-ordered ledger.