slug: fcfs-and-front-running title: Can First Come First Served-Based Transaction Ordering Prevent Front-Running? authors: [ra] tags: [research] image: /img/fcfs-and-front-running.png hide_table_of_contents: false forum_link: https://collective.flashbots.net/t/can-first-come-first-served-based-transaction-ordering-prevent-front-running/892

Can First Come First Served-Based Transaction Ordering Prevent Front-Running?

Recently, consensus protocols have been proposed that aim to include transactions on a First Come First Served (FCFS) basis. In this article, we explore whether such FCFS-based ordering of transactions in consensus protocols can prevent front-running in permissionless blockchains.

Thanks to Alejo Salles and Quintus Kilbourn for feedback and review of this post!

Introduction & Problem Statement

The order in which transactions are submitted to mempools is usually not their finalized order in ledgers. The term *Maximum Extractable Value* (MEV) comprises various ways of profiting from actively manipulating the order of, inserting, and censoring transactions. Front-running is one well-known way of profiting from inserting transactions. Front-running causes negative side effects such as financial loss to users of decentralized finance applications as well as increased load on blockchain networks. Recently, there have been proposals for consensus protocols which aim to include transactions in the same order as they were generated. In this article, we explore whether such First Come First Served (FCFS)-based ordering of transactions in consensus protocols can prevent front-running, as could be inferred from recent work[^1][^2][^3]. Preventing front-running is particularly interesting, because the result of a transaction gets worse from the perspective of the sender if it was front-ran, and if front-running can be prevented so-called sandwiching attacks (i.e., another form of MEV) can equally be prevented.

Improving upon previous work[^1][^2], Kelkar, Deb, and Kannan introduce a consensus protocol that aims to provide FCFS-based ordering[^7] of transactions in permissionless blockchains[^3]. The consensus protocol guarantees that every transaction tx received before another transaction $tx\prime$ by a large fraction of nodes does not occur after $tx\prime$ in the finalized ledger. The consensus protocol works in the synchronous communication model and discretizes time into rounds such that transactions that arrive in the same round receive the same timestamp. Generally speaking, the synchronous communication model is used to reason about protocols in distributed systems. In the synchronous communication model, there exists some time bound Δ up to which the delivery of messages can be delayed.[^5] The authors prove that - in this setting - an adversary controlling up to a certain fraction of overall hash-power cannot manipulate the order of transactions, which would indicate that front-running can be prevented entirely. We argue that front-running a transaction may still be feasible, however, if transactions end up in the same round and therefore receive identical timestamps. For a real-world setting, the decisive question is: can the duration of a round[^11] be strictly smaller than the time it takes to create and distribute a front-running transaction? If the answer is yes, FCFS-based transaction ordering can indeed prevent front-running.

FCFS-Based Transaction Ordering & Front-Running

A FCFS-based consensus protocol guarantees that every transaction tx received before another transaction $tx\prime$ by a large fraction of nodes does not occur after $tx\prime$ in the finalized ledger. The main problem we see with this is that the front-runners abilities are not properly defined (furthermore, neither the network nor the application is properly modeled, and it is not clear how transactions that occur in the same round should be ordered). What is clear from the above definition is that the front-runner may run some fraction of nodes that can be used to lie about the perceived order of transactions. But what about other the nodes and the network in general? We will assume a global, Internet-like network over which nodes are distributed and a very weak adversary model for the front-runner. According to usual practice, we assume that clients send transactions to some nodes, who forward transactions to other nodes through a gossip protocol. The front-runner may control just a single node, for which it can choose the position in the network based on the position of other nodes. The front-

runner's node keeps connections to other nodes open. We assume that the front-runner has no control over the network (apart from choosing its node's position). Even though the front-runner is modeled as very weak (probably too weak for a real-world setting), we will show that front-running is possible under FCFS-based consensus.

Let's start with defining when front-running is (im)possible first. Front-running can be prevented if nodes are guaranteed to distinguish that the to-be-front-ran transaction originated before the front-running transaction. In other words, if the time it takes a front-runner to receive a front-runnable transaction, create a front-running transaction and send that to (a majority[^6] of) nodes is strictly greater than the time it takes nodes to receive the front-runnable transaction in the first place, then front-running can be completely prevented. Otherwise, front-running is possible. Let's define the factors just mentioned and put them into a nice-looking formula: - d_{E2E}^{nodes}: the time it takes from sending a transaction until it is received by (a majority of) nodes. - $d_{E2E}^{front-runner}$[^12]: the time it takes from sending a front-running transaction until it is received by (a majority of) nodes. For the sake of simplicity[^10], we assume that it is approximately the same as the time it takes the front-runner to receive a (potentially front-runnable) transaction. - d_{TX}: the time it takes a front-runner to create a front-running transaction. - Δ: parameter of a FCFS-based consensus protocol that determines the length of a round. It needs to be chosen large enough such that all nodes are guaranteed to receive a transaction within Δ. It furthermore determines the granularity to which nodes can agree what transaction originated before another.

$$ \tag 1 2 \cdot d_{E2E}^{front-runner} + d_{TX} - d_{E2E}^{nodes} < \Delta \implies \text{front-running possible} $$

As the front-runner is incentivized financially, but other nodes would just incur higher communication and computation cost, we assume that the front-runner will choose a better position within the network than the average node. In this way, the front-runner can receive transactions earlier than the average node and their transactions may also be received earlier by other nodes. We roughly estimate $d_{E2E}^{front-runner}$ to be on the order of 50 to 200 ms and d_{E2E}^{nodes} on the order of 100 to 1000 ms. The time it takes to create a front-running transaction d_{TX} can be roughly estimated to be on the order of a few ms to tens of ms, since just some quick analysis may be required before creating and signing the transaction. Δ presumably is quite large in permissionless settings over the Internet, because packet loss leads to timeouts, and retransmissions can take significant time so that Δ could be on the order of minutes (or even more). Inserting the rough estimations into Eq.1 results in $[-895 \text{ ms}, 350 \text{ ms}] < \text{several minutes}$. So front-running is clearly possible.

What if FCFS ordering protocols were improved such that they can deal with (some) packet loss across the network? Then, Δ could be reduced significantly - probably down to the order of seconds. That changes Eq.1 to $[-895 \text{ ms}, 350 \text{ ms}] < \text{several seconds}$, highlighting that front-running is still clearly possible.

What if the granularity of transaction ordering could be reduced from Δ down even further? Temporal ordering of events[^9] in distributed systems has been extensively studied early on[^13] in another context: clock synchronization protocols. We can use the results from that context to infer what is (im)possible in the context of transaction ordering. Contrary to common belief, no infinitely precise notion of time shared between nodes can exist in practice. The impossibility result by Lundelius and Lynch[^4] proves that it is not possible to synchronize the clocks of n nodes any better than $\epsilon (1 - \frac{1}{n})$, with ϵ representing the uncertainty in end-to-end delays. That result holds even if all clocks are assumed to have perfect oscillators (i.e., no clock drift) and nodes to be honest and fully interconnected. What that results tell us is that uncertainty in end-to-end delays is ultimately the decisive factor to the granularity of ordering transactions (not the maximum end-to-end delay) - even if all nodes act honestly, etc. Eq.1 then becomes: $$ \tag 2 2 \cdot d_{E2E}^{front-runner} + d_{TX} - d_{E2E}^{nodes} < \epsilon \implies \text{front-running possible} $$ End-to-end delay uncertainties affect the ordering of transactions as nodes cannot distinguish between a transaction originating later or just getting delayed more than some other transaction, if the time difference is sufficiently small.[^8] For a node to be able to order transactions correctly with certainty, the time difference between the reception times of the corresponding transactions must be greater than the end-to-end delay uncertainty. If the uncertainty can be reduced, transactions can be ordered with higher granularity. End-to-end delay uncertainties ϵ can be roughly estimated to be on the order of 100 to 300 ms and are influenced by a multitude of factors such as router buffers, switch buffers, operating system schedulers, imperfect clocks, asymmetric one-way delays, and changing network paths. To highlight how challenging end-to-end delay uncertainties are, non-intuitively, end-to-end delays cannot be measured precisely, even between just two directly connected hosts. In order to measure end-to-end delay precisely, both hosts would need to have perfectly synchronized clocks (which is not possible). Adding the estimation for end-to-end delay uncertainties to Eq.2 results in $[-895 \text{ ms}, 350 \text{ ms}] < [100 \text{ ms}, 300 \text{ ms}]$, which shows that while the inequation may be unsatisfied for some values, it is satisfied

for the majority of values. For this reason, while some front-running can be mitigated now, front-running still cannot be prevented completely - even though we are now operating close to the theoretical limits.

Front-Running Mitigation & Prevention

Building upon the time-based framework in this article, how could front-running mitigation with FCFS-based transaction ordering be improved? - Uncertainties in end-to-end delays ϵ could be decreased further. With more knowledge of or control over the network, end-to-end delay uncertainties can be reduced. For example, if the network position of nodes is known, the physical distance between nodes can be approximated and a minimum delay between nodes can be established, using the speed of light as lower bound for the delay, in order to reduce end-to-end delay uncertainties. If the network environment of nodes is trusted and can be controlled, the network may be modeled probabilistically and end-to-end delay uncertainties could be reduced further, but this may only be feasible in permissioned settings with full control over the network. The more knowledge one has of the network or the more one can control the network, the more end-to-end delay uncertainties can be reduced, but the more permissioned and trusted the whole setting becomes. So the extent to which end-to-end delays can be decreased in order to mitigate front-running in permissionless settings is limited. - By ensuring that $d_{E2E}^{nodes} \approx d_{E2E}^{front-runner}$, such that a front-runner cannot benefit significantly anymore from a better-than-average position in the network. This can be achieved by ensuring that nodes are (somewhat) fully interconnected, or by having clients themselves already send transactions to a majority of nodes. Both ways incur significant communication overhead, for nodes or for clients, which increases with the number of nodes and therefore does not scale. Nevertheless, it could mitigate front-running further if feasible. Eq.2 becomes $$ \tag{3} d_{E2E} + d_{TX} < \epsilon \implies \text{front-running possible} $$ d_{E2E} can be roughly estimated to be on the order of 200 to 400 ms. Inserting that estimation into Eq.3 results in $[205 \text{ ms}, 450 \text{ ms}] < [100 \text{ ms}, 300 \text{ ms}]$, showing that while this mitigation is effective to some extent, front-running still cannot be prevented completely.

Effective and generally applicable front-running mitigation appears to be a significant challenge. But, based on the insights from our analysis, (how) could front-running be prevented in the first place? - If the time it takes to create a front-running transaction d_{TX} can be increased, Eq.3 would immediately be unsatisfiable. That would already be the case if $d_{TX} \approx \epsilon$. To ensure that $d_{TX} \approx \epsilon$, clients would need to provide a proof that some time ($d_{TX} \gtrsim \epsilon$) has passed after they created a transaction. It would furthermore need to be ensured that clients created the proof indeed at the claimed point in time and not before. If such construction is feasible, it can prevent front-running. This front-running prevention comes at a cost, however, since user experience is harmed as users must provably wait before they can submit their transactions. With FCFS-based transaction ordering, the additional waiting time (and therefore UX cost) can be reduced from about three times the block time, which is the normal granularity of transaction ordering in blockchains, down to about ϵ. - Unfortunately, increasing the end-to-end delay d_{E2E} generally, which would affect both $d_{E2E}^{front-runner}$ and d_{E2E}^{nodes}, does not have the same effect as increasing d_{TX}, because increasing d_{E2E} usually increases the end-to-end delay uncertainties ϵ at the same time. If d_{E2E} could be increased significantly more than ϵ, front-running could be prevented. That does not seem feasible though in permissionless settings over the Internet. - Hiding the content of transactions from nodes until the order of transactions is fixed has the same effect as increasing d_{E2E} without increasing ϵ at the same time. Commit & reveal and threshold encryption are two distinct methods of hiding the content of transactions. Both also ensure that $d_{E2E}^{nodes} \approx d_{E2E}^{front-runner}$ so that a front-runner cannot benefit from a better-than-average position in the network. The decisive factor is the time between the commit and the reveal (or the encryption and the decryption), which can increase d_{E2E} to such an extent that Eq.3 is always unsatisfiable. In this way, hiding the content of transactions alone can prevent front-running, FCFS-based ordering of transactions just reduces the time required between the commit and the reveal (or the encryption and the decryption). Both methods have specific and shared challenges, however. Commit & reveal schemes introduce the so-called *free options problem* that incentivizes spam - if commit messages are free of charge and the reveal part cannot be enforced, users could opportunistically send commit messages and reveal only if it is in their favor to do so. Threshold encryption schemes, on the other hand, face efficiency challenges in terms of computation and communication. Furthermore, how a threshold of nodes can be incentivized to continuously act honest when financially incentivized to act dishonestly is an open research question. So while both methods could prevent front-running, they have their specific challenges, but there is also one shared issue among any method that hides the content of transactions, which becomes apparent when not only considering front-running but when taking other forms of MEV into account as well. At some point, the content of a transaction has to be revealed, and one node is in the privileged position to see the plain text transactions first and can therefore be the first to act on any MEV opportunities such as back-

running or arbitrage that emerge. So while the methods around hiding the content of transactions can prevent front-running, assuming their specific challenges are addressed, they do restrict access to other MEV opportunities to (one or more) nodes, which may be an undesirable property as access to the remaining MEV opportunities would not be democratic but only available to one (or a few) entities who would take all profit.

Discussion & Conclusion

In this article, we analyzed whether consensus protocols with FCFS-based ordering of transactions can prevent front-running in permissionless settings over the Internet. We showed that FCFS-based ordering of transactions alone can only mitigate front-running to a certain extent but cannot prevent it completely - even under a weak front-runner model, where the front-runner only controls a single node. This article highlighted the importance of the front-runner's position within the network, and how front-running mitigation can be improved if privileged positions within the network are hindered.

Based on our time-based framework, we discussed various ways to mitigate or prevent front-running. The discussion shed light onto two ways of how front-running could be prevented in the first place, i.e., having clients provably wait before their transaction can be included and hiding the content of transactions, why they could be effective, as well as their downsides. We discussed how FCFS-based ordering of transactions may reduce the downsides of specific front-running prevention mechanisms.

To conclude, end-to-end delay uncertainties fundamentally limit the granularity to which nodes can agree what transaction originated before another. By integrating FCFS-based transaction ordering into the consensus protocol, the granularity with which transactions can be ordered can be improved. The granularity achievable in permissionless real-world settings is so coarse grained, however, that front-running is only mitigated to some extent, and front-running opportunities largely remain. Improving the achievable granularity, such that front-running is largely mitigated, leads to more permissioned and trusted settings.

With that, thanks for reading, and looking forward to reading yourcomments! :zap: :robot_face:

[^1]: Mahimna Kelkar et al. “Order-fairness for byzantine consensus”. In: Annual International Cryptology Conference (CRYPTO). Springer. 2020, pp. 451–480.

[^2]: Mahimna Kelkar et al. “Order-Fairness for Byzantine Consensus”. Cryptology ePrint Archive, Report 2020/269. https://ia.cr/2020/269. 2020.

[^3]: Mahimna Kelkar, Soubhik Deb, and Sreeram Kannan. “Order-Fair Consensus in the Permissionless Setting”. Cryptology ePrint Archive, Report 2021/139. https://ia.cr/2021/139. 2021.

[^4]: Jennifer Lundelius and Nancy Lynch. “An upper and lower bound for clock synchronization”. In: Information and control 62.2-3 (1984), pp. 190–204.

[^5]: Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. “Consensus in the presence of partial synchrony.”. In: Journal of the ACM (JACM) 35.2 (1988): pp. 288-323.

[^6]: The exact percentage of nodes depends on the FCFS-protocol's details. We just loosely assume a 51% majority here.

[^7]: The authors actually call it *fair-ordering*, but we refrain from using that term as we find it to be vague and potentially misleading in the context of MEV. Instead, we use *FCFS-based ordering of transactions*, which we think is more appropriate.

[^8]: Since all nodes' observations' are affected by unknown, potentially unrelated delay uncertainties, majority voting as suggested in [^1] cannot help ordering transactions any better than those uncertainties allow --- generally speaking, majority voting can only be used against malicious node behavior but not against end-to-end delay uncertainties.

[^9]: In the context at hand, the sending of transactions would be*events* and the goal of a FCFS ordering protocol would achieve the temporal ordering of these.

[^10]: Receiving front-runnable transactions would probably be faster than sending a front-running transaction, making front-running even easier.

[^11]: In the synchronous communication model the duration of a round is lower bounded by Δ.

[^12]: Please note that $d_{E2E}^{front-runner}$ and d_{E2E}^{nodes} are not independent.

[^13]: Leslie Lamport. “Time, clocks, and the ordering of events in a distributed system”. In: Communications of the ACM, Volume 21, Issue 7. July 1978. pp 558–565