IEEE ICBC'24 1570982018

Playing the MEV Game on a First-Come-First-Served Blockchain

***Abstract*—Maximal Extractable Value (MEV) searching has gained prominence on the Ethereum blockchain since the surge in Decentralized Finance activities. In Ethereum, MEV extraction primarily hinges on fee payments to block proposers. However, in First-Come-First-Served (FCFS) blockchain networks like Algorand, the focus shifts to latency optimizations, akin to High-Frequency Trading in traditional finance. This paper illustrates the dynamics of the MEV game in an FCFS network, specifically Algorand. We introduce an arbitrage detection algorithm tailored to the unique time constraints of FCFS networks and assess its effectiveness. Additionally, our experiments investigate potential optimizations in Algorand's network layer to secure optimal execution positions.**

Our analysis reveals that while pool states' relevant updates occur approximately every six blocks on average, pursuing MEV at the block state level is not viable on Algorand, as arbitrage opportunities are typically executed within the blocks they appear. Our algorithm's performance under varying time constraints underscores the importance of timing in arbitrage discovery. Furthermore, our network-level experiments identify critical transaction prioritization strategies for Algorand's FCFS network. Key among these is reducing latency in connections with relays that are well-connected to high-staked proposers.

***Index Terms*—blockchain, maximal extractable value, decentralized finance, first-come-first-served, arbitrage**

I. Introduction

Blockchain networks offer profitable opportunities for various entities. While default economic incentives for consensus participants include block rewards and transaction fees, the recent surge in Decentralized Finance (DeFi) activities (with daily trading volumes on Decentralized Exchanges (DEXs) exceeding multi-billion USD [1]) has led to the emergence of a new incentive for strategically acting entities, which we know as Maximal Extractable Value (MEV) [2]. While this term generally refers to the value that can be made by entities like block proposers which have the privilege to determine transaction inclusion, exclusion, and ordering, value extraction is not limited to them, as MEV searcher activity dashboards such as libMEV [3] reflect a total profit of 64 million USD made by MEV searchers since the merge on Ethereum in September 2022.

Before private relays such as Flashbots[1] emerged on Ethereum, offering an off-chain sealed-bid auction for inclusion, MEV searching bots competed in Priority Gas Auctions (PGAs) on the public mempool to extract value [2]. As Ethereum is a blockchain where fees can influence the transaction ordering of proposers, while latency still played a role in being competitive in PGAs, such as to ensure your latest bid reaches the proposer on time, the primary determinant of the winner was the fee offered.

In blockchains where block proposers arrange transactions in the order received, known as First-Come-First-Served (FCFS), the dynamics for MEV searching differ, as demonstrated in [4], [5]. Unlike fee-based blockchains like Ethereum, in an FCFS network, the only way to prioritize is by propagating a transaction before competitors. Consequently, the available runtime for an MEV searching algorithm in FCFS networks is limited by the anticipated arrival time of a competing transaction, as opposed to almost the entire block time available to the searcher on a fee-based network.

This paper explores how the MEV extraction game can be played on an FCFS network, using Algorand as the case study, building on the initial research by Öz et al. [5]. Our methodology comprises two stages: *discovery* and *extraction*. The discovery stage employs a cyclic arbitrage detection algorithm. We focus on arbitrages, as they are shown to be a feasible MEV strategy on FCFS networks [4], [5]. We assess our algorithm's performance using historical Algorand data, considering the runtime constraints of fcfs) networks. In the next stage, similar to High-Frequency Trading in Traditional Finance, mev) extraction is about outpacing others in latency. We run experiments on a private Algorand network to identify key factors for prioritizing transaction propagation. Our research contributes to understanding MEV in FCFS networks, showcasing how algorithms can exploit profitable opportunities and highlighting the critical network characteristics for ensuring prioritized execution.

II. Background

In this section, we provide background information required for the working principals of the Algorand blockchain, DeFi activity on it with a focus on DEX protocols, and an overview of Automated Market Makers (AMMs).

A. Algorand Blockchain

For the detailed analysis and evaluation of MEV searching in FCFS, our focus is on the Algorand blockchain [6]–[8], introduced by Silvio Micali in 2017. Algorand utilizes a consensus mechanism called Algorand Byzantine Fault Tolerance Protocol (BA), providing instant finality, scalability (in the number of nodes and transactions per second), and avoiding soft forks [6]–[8]. It employs the Pure Proof-of-Stake (PPoS) for Sybil resistance, allowing anyone with at least one ALGO (the native token of Algorand) to join the consensus. Unlike Ethereum, Algorand does not reward consensus participants with fixed block rewards or transaction fees. However, discussions about changing incentives are ongoing[2].

Regarding high-level specifications, the system can handle around 8000 transactions per second and publishes blocks every 3.3 s in the v3.19.0 release[3]. The network consists of approximately 1100 nodes (relay and participation nodes)[4]. Participation nodes connect via relays, each connected to four randomly selected relays. The relays, in turn, forward messages to four relays and all incoming peers. Default connections on a relay

[1]Flashbots Auction: https://docs.flashbots.net/flashbots-auction/overview

[2]https://github.com/algorand/go-algorand/pull/5740

[3]https://github.com/algorand/go-algorand/releases/tag/v3.19.0-stable

[4]https://metrics.algorand.org/

are set to 2400. Configuration parameters may vary for each peer, as they are not enforced. Regular clients not participating in consensus also rely on connections via relays, but the number of such clients is unknown. It is expected that in the next year, Algorand will switch from the relay structure to a gossip-based Peer-to-Peer (P2P) layer similar to Ethereum[5].

Algorand handles scaling in the number of consensus participants by selecting a subcommittee from the total number of active participation nodes. One consensus round involves block proposer selection, soft vote, and certify vote, with a new committee selected at each step. Committees have varying sizes with 20 block proposers, 2990 parties in the soft vote, and 1550 in the certify vote [6]. The likelihood of selection correlates with stake amount, and members are unknown until they cast votes. The proposer selection step is crucial for MEV extraction, as the selected proposer's transaction sequence determines the extracted value. Fast connections to relays can aid MEV searching, but challenges arise with up to 20 proposers being eligible at every round. FCFS ordering is not protocol-enforced but comes with the official Algorand node implementation.

Algorand transactions include payments (native token transfers), Algorand Standard Asset (ASA) transfers (for other token types), and Algorand Smart Contract (ASC1) application calls. Assets are managed by ASA transactions, and ASC1, or applications, deploy functions in a Layer-1 network, written in Transaction Execution Approval Language (TEAL), an assembly-like language interpreted in the Algorand Virtual Machine (AVM). Each application has a unique ID, and call complexity may result in up to 256 inner transactions. Transaction cost is a fixed minimum of 0.001 ALGO per transaction, charged only for successful transactions. During network congestion, the fee strategy shifts to a dynamic cost model per byte. Congestion is determined by the number of transactions in a node's local mempool, leading to increased fees if a node is congested. Using multiple clients is crucial to gauge network capacity and adapt fees when congestion occurs over the whole network.

B. Decentralized Finance

Financial applications built on blockchains, often referred to as Decentralized Finance (DeFi), represent an emerging collection of applications that emulate services found in the traditional finance sector. By employing smart contracts, these applications on blockchains encompass a wide array of services, ranging from decentralized exchanges and options markets to lending protocols and tokenized assets. According to DefiLlama [9], as of December 2023, Algorand's DeFi ecosystem holds a Total Value Locked (TVL) of 82 million USD. This valuation has shown stability on an annual basis but exhibits growth over a longer period. Key indicators of blockchain activity include daily volume and the number of daily active addresses. Over the past year, the average daily volume was 5 000 000 ALGO, with around 36 000 daily active addresses, both metrics maintaining consistency year-over-year. In the realm of Algorand's DEX ecosystem, Tinyman[7] and Pact[8] each reported a TVL of 15 million USD. HumbleSwap[9], though smaller, also made a significant impact with a TVL of 3 million USD. Focusing on transaction volumes for December 2023, Tinyman (versions 1 and 2) showed an average daily volume of 729 000 USD, followed by Pact with 418 000 USD, and HumbleSwap at 41 000 USD [9].

C. Automated Market Makers

Conventional centralized cryptocurrency exchanges process over 80 billion daily in spot-volume[10], using Central Limit Order Book (CLOB) architecture. In CLOBs, a centralized entity controls customer asset custody and trade settlement, maintaining an enumerated order list for real-time buyer-seller matching. Orders are processed sequentially, with the central operator acting as an intermediary. In contrast, DEXs usually utilize smart contracts and an AMM model instead of on-chain order books due to their simplicity and efficiency in order matching without an intermediary, even in illiquid markets. AMMs manage liquidity pools, balancing various token reserves, while facilitating P2P trading at preset rates determined by a mathematical formula called the swap invariant. Constant Product Market Maker (CPMM), a common swap invariant variant used by popular DEXs such as Uniswap[11] on Ethereum and Tinyman, ensures the product of the assets reserves' in the liquidity pool remains constant unless there is liquidity provision or withdrawal, enabling automated price discovery. Users trade tokens with AMMs, paying a fee to liquidity providers (usually 0.3%) and receiving purchased tokens from the pool's reserves. When trading on DEXs, price slippage refers to the difference between the expected and actual execution prices of the trade. Expected price slippage is the predicted price change based on a trade's volume and AMM's liquidity and swap invariant. However, as expected slippage is calculated using historical blockchain data, fluctuations between transaction submission and execution can lead to unexpected slippage. Together, expected and unexpected slippages determine a trade's overall market impact.

III. Related Work

As our methodology for playing the MEV game on Algorand is twofold: opportunity discovery and MEV extraction on FCFS networks, we provide an overview of existing work on these subjects.

A. Profitable Opportunity Discovery

The discourse on profit-yielding opportunity discovery on blockchain networks includes a range of works focusing on cyclic arbitrage discovery [10]–[12], even with optimal routing [13], sandwiching [14], or more generalized strategies like imitation attacks [15]. Zhou et al. [10] provide an approach utilizing a greedy cycle detection algorithm paired with a gradual increment-based input searching. While they achieve sub-block time runtime, the algorithm is not optimized for efficiently finding an input and maximizing discovered profits as it operates on a limited asset set. Wang et al. [11] focus solely on constant-product markets on UniswapV2 [16] and show the consistent existence of greater than 1 ETH opportunities across blocks. Following, McLaughlin et al. [12] conduct a larger scale study and incorporate further market invariants such as UniswapV3 [17], which enables liquidity providers to bound liquidity to a tick range instead of the default full price range as in V2. Their arbitrage discovery study

[5]https://github.com/algorand/go-algorand/issues/5603
[6]https://github.com/algorandfoundation/specs/blob/master/dev/abft.md
[7]https://tinyman.org
[8]https://www.pact.fi
[9]https://www.humble.sh/
[10]https://www.coingecko.com/en/exchanges
[11]https://uniswap.org

on $5.5\,\text{million}$ blocks outputs approximately $4.5\,\text{billion}$ potential opportunities. However, they also show that, out of $20.6\,\text{million}$ arbitrages which yield over 1 ETH revenue, only $0.51\,\%$ of them are successfully executable. In contrast, close to $97\,\%$ fail due to reverting or incompatible tokens.

B. MEV on FCFS Networks

Carillo and Hu [4] analyze arbitrage MEV extraction on Terra Classic, an FCFS blockchain with fixed gas prices. Their study identifies more than $188\,000$ arbitrages, and highlights the success of searchers adopting a spamming strategy, involving sending multiple failed transactions for every successful one, to gain latency advantage. They also demonstrate the significance of a searcher's geographical position in the network for reducing transaction propagation latency and observing an opportunity generating transaction before the others by monitoring $400\,000$ transactions through their 84 nodes distributed to different locations.

Öz et al. in [5] investigate the applicability of transaction ordering techniques from fee-based blockchains to an FCFS blockchain like Algorand. Their empirical research, detecting $1.1\,\text{million}$ arbitrages across $16.5\,\text{million}$ blocks, reveals unique dynamics and extraction methods inherent to an FCFS network. Their results show the prevalence of network state backruns among all detected arbitrages. However, they also reveal that particular searchers profit from top-of-the-block positions, hinting at latency and transaction issuance timing optimizations. Although their on-chain data analysis does not offer new insights into latency games played between searchers and proposers, they highlight the use of strategic network clogging as a viable strategy in FCFS networks due to its low cost and show the existence of arbitrage clogs on Algorand.

IV. MEV Discovery

The initial stage of our methodology for engaging in the MEV game on Algorand's FCFS network involves identifying profitable opportunities. Given the substantial number of arbitrages on Algorand [5], our focus is on developing an algorithm to detect these opportunities. Prior research on Etheruem, such as [10], demonstrated real-time algorithmic detection using a greedy cycle detection method limited to a small set of assets. Their approach also included a less efficient method of input discovery through gradual increments. In contrast, our proposed algorithm is tailored to Algorand's specific time constraints, aiming to identify nearly all emerging arbitrage cycles and incorporating a straightforward yet effective input optimization technique.

To evaluate our algorithm's efficacy, we collect state data from Algorand and execute the algorithm subsequent to each block's finality. We present results in both unconstrained and time-constrained settings, with the latter being more relevant for the competitive dynamics of an FCFS network. This approach allows us to assess the algorithm's practicality and efficiency in a real-world blockchain environment.

A. Cyclic Arbitrage Detection Algorithm

Our methodology for detecting profitable arbitrage opportunities begins with a setup phase, where we translate the space of assets A and pools P into a multigraph $G = (V, E)$. Here, V represents the set of vertices, each corresponding to an asset $a \in A$, and E is the set of edges, where each edge denotes a pool $p_{ij} \in P$ that enables the exchange between assets a_i and a_j[12]. Given that G is a multigraph, we define $E_{ij} \subseteq E$ as the set of pools available for exchanging the two assets.

Assuming a subset $PA \subseteq A$ as the profit assets of interest, we employ a cycle detection algorithm on G to find the set of cycles C of length $l \in L$ for each profit asset $pa \in PA$, denoted as C^l_{pa}. Each trading cycle $c^l_{pa} \in C^l_{pa}$ comprises l swaps $\{s^1, \ldots, s^l\}$ where the input of s^1 and the output of s^l is the profit asset pa. Each swap s_{ij} between assets a_i and a_j can be implemented in $|E_{ij}|$ ways. Therefore, a cycle c^l_{pa} can have $N_{c^l_{pa}}$ different implementations, where $N_{c^l_{pa}} = \prod_{i=1}^{l} |E_{s^i}|$ and $I_{c^l_{pa}}$ represents the set of implementations, with $|I_{c^l_{pa}}| = N_{c^l_{pa}}$. The aggregate set of implementations for cycles of length l for a profit asset pa, denoted as I^l_{pa}, is defined as $I^l_{pa} = \bigcup_{c^l_{pa} \in C^l_{pa}} I_{c^l_{pa}}$. The comprehensive implementation set for all assets in PA is $I_{\text{total}} = \bigcup_{pa \in PA, l \in L} I^l_{pa}$.

The setup stage outputs I_{total}, leading to the arbitrage detection phase, which happens in real time after every proposed block b. In this phase, we apply the detection algorithm to a subset of cycle implementations $I^b_{\text{total}} \subseteq I_{\text{total}}$ related to our profit assets, specifically targeting cycles with updated pool reserves in block b. We ignore the cycles with only stale pools as we already evaluate the opportunities on them previously.

The algorithm is confined to operate within a predefined time window τ, which, for FCFS networks, depends on the arrival time of the first transaction, changing a relevant pool's state. In such networks, the desired position in a block can only be achieved by correctly timing the transaction issuance and propagation to the network. In fee-based blockchains such as Ethereum, $\min \tau$ is equal to block time (ignoring network propagation latency), as the targeted position can still be obtained by issuing a transaction with sufficient fees at any time before the block is mined.

While τ is not reached, for each cycle $i \in I^b_{\text{total}}$, we check if the product of involved pools' exchange rates' is greater than one, indicating an arbitrage. If so, we search for the profit-maximizing input for i using SciPy's[13] minimization function problem solver constrained by the involved pools' swap invariant. This approach maximizes our profitability objective function in a constraint-free nonlinear optimization landscape, employing solvers adept in numerical gradient approximation. In case the profit level of the arbitrage is more significant than a lower limit, we append it to the set of candidate arbitrages for block b, denoted as $\mathcal{A}^b_{\text{candidates}}$.

The final stage of our approach involves one of two strategies for arbitrage selection: a greedy strategy for maximizing profits (which we assumed as default in the rest of the analyses) and an FCFS strategy. The former strategy initially iterates over every candidate arbitrage in $\mathcal{A}^b_{\text{candidates}}$ and, after evaluating all, greedily selects and issues the most profitable, non-overlapping set, which we denote with $\mathcal{A}^b_{\text{greedy}}$. On the other hand, the FCFS strategy does not wait to check the profitability of every arbitrage in $\mathcal{A}^b_{\text{candidates}}$. It issues them as soon as their optimal input is discovered. While this strategy ensures rapid execution, it also lets go of maximizing profitability since early discovered arbitrages may invalidate to-be-realized, more lucrative opportunities. While there is no deliberate selection of arbitrages in FCFS strategy, we use $\mathcal{A}^b_{\text{FCFS}}$ to denote the set of arbitrages that will be executed.

[12]For the scope of this study, we ignore the pools which offer more than two assets.

[13]https://scipy.org/

Every arbitrage in $\mathcal{A}^b_{\text{greedy}}$ and $\mathcal{A}^b_{\text{FCFS}}$ interacts with a unique set of pools, as otherwise, one arbitrage could invalidate another one by changing the reserves of the intersecting pool.

B. Empirical Evaluation Setup

To test the performance of our algorithm, we constructed a historical state data collection setup. Leveraging the API capabilities of the Algorand node, we establish a process that continuously listens to our node by invoking the `/v2/status/wait-for-block-after/round` endpoint[14]. This method sets up an internal wait channel within the node that unblocks upon reaching the desired round, thus notifying us of new blocks. Concurrently, we utilize SDK utilities of various DEXs on the Algorand network to fetch reserve data of pools following the CPMM price invariant. This data, essential for arbitrage discovery, is stored for each round, maintaining a continuous and comprehensive market overview. ALGO token price data for revenue calculation is fetched from [18].

C. Limitations

In our methodology, we faced certain constraints that are important to consider. Firstly, there is no known forking tool on Algorand (such as Ganache[15] on Ethereum) which would allow us to construct the blockchain state at a desired block and execute a transaction. Thus, we built our pipeline for constructing historical states. Due to the limitation of this approach, we focused on only a subset of assets, pools, and DEX protocols. For example, we had to exclude the HumbleSwap DEX as it did not provide an SDK with low latency. Hence, the arbitrages we discover on the finalized blocks only represent a lower bound. Additionally, our approach focused on collecting state data on finalized blocks rather than a more comprehensive network-level data collection on the mempool, likely leading to underestimating total arbitrage opportunities. We also simplified our financial modeling, disregarding flash loan fees and assuming the immediate availability of initial assets, which might not reflect real-world trading conditions.

D. Results

We have tracked 136 assets, exchanged in 255 pools, on three different DEXs (TinymanV1, TinymanV2, Pact), starting from block 32 608 011 (Thu, 05 Oct 2023 00:49:42 GMT) until block 33 039 007 (Sat, 21 Oct 2023 21:05:40 GMT). In these 16 days, 430 996 blocks were built on the Algorand blockchain. In 30 828 (7.1 %) of them, reserves of at least one pool we tracked got updated, and we ran the arbitrage detection algorithm on it.

1) Unconstrained Arbitrage Discovery: Before analyzing the time-constrained performance of our algorithm, we let it run unconstrainedly to observe the maximum profitability of block state arbitrages. To benchmark its performance, we also calculate the executed arbitrage profits in the same block range, utilizing the heuristics defined in [5].

Figure 1 displays the revenue plots for arbitrages discovered through our algorithm, which we only ran on finalized block states (blue) versus the arbitrages executed in reality (green). The stark dominance of realized arbitrage revenues showcases that MEV searchers on Algorand promptly exploit arbitrage opportunities

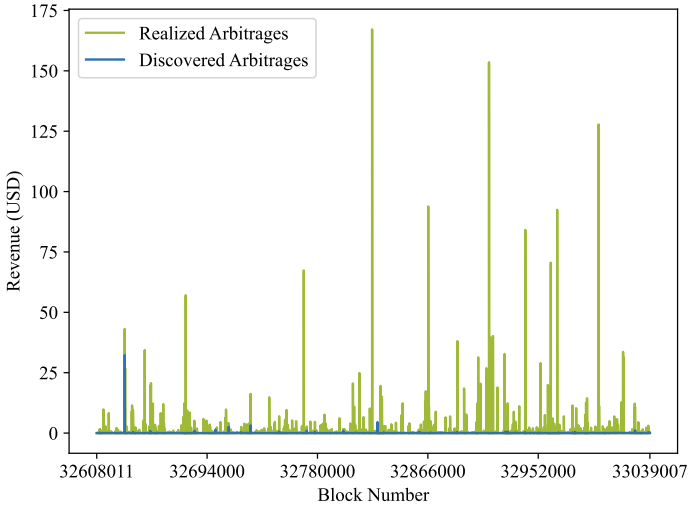


Fig. 1: Distribution of arbitrage revenue across the blocks in range 32608011 to 33039007. The blue plot reflects the revenue discovered on the block state by our algorithm, while the green plot shows the realized revenue by arbitrages executed in every block.

inside the block they emerge. Hence, in most cases, price discrepancies do not carry over to the next block (at most, we find a 32.2 USD opportunity on the block state). When we manually checked the most profitable ten arbitrages for the window between the position of the opportunity-creating transaction and their respective backruns, we found that the first backrun was always positioned right at the next position. The efficiency of MEV searchers on Algorand in arbitrage execution, in line with the results from [5], showcase that our initial attempt for searching MEV only at the block state level is a naive approach. In the future work, we plan to collect transaction data on the mempool directly and execute our algorithm after every trade on a pool relevant for us. This way, we can also detect the opportunities appearing in a block and find their optimal profitability. Currently, we cannot fully asses whether MEV searchers backrunning on the network-level run their strategies with optimized inputs.

2) Time-Constrained Arbitrage Discovery: The success of an arbitrage strategy depends on execution prior to an update in the reserves of the arbitraged pools. We have introduced τ to denote the time window before such an update occurs as part of a competing arbitrageur's transaction or by an innocent user trade. A competitive arbitrage discovery strategy needs to operate under τ. Hence, in this section, we measure our algorithm's performance with respect to a spectrum of τ values.

Our initial experiments on 430 996 Algorand blocks, in which only 7.1 % of them have an updated pool we track, show that pools relevant for our arbitrage detection algorithm are updated on median every six blocks (25 %: 2.0 , 75 %: 17.0); hence τ is close to 19.8 seconds (considering that Algorand block time is currently around 3.3 seconds[16]). See Figure 2 for a distribution of state update deltas for the examined date range. Interestingly, the max state update delta reaches 294 blocks ($\sim$ 16 minutes), although our algorithm does not detect any profitable block state arbitrage in this window.

[14]https://github.com/algorand/go-algorand/algod/api/server/v2/
[15]https://github.com/trufflesuite/ganache
[16]https://algoexplorer.io/

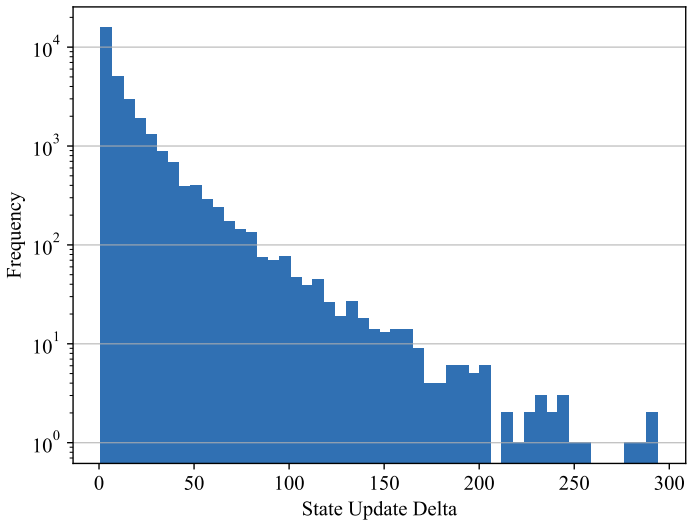


Fig. 2: Histogram of state update deltas among 430 996 blocks built between Thu, 05 Oct 2023 and Sat, 21 Oct 2023.

a) The Value of Time: We measure the profitability of our algorithm as a function of τ to observe the impact of available runtime window on discovered value. Although we have detected a median state update delta of six blocks ($\tau = 19.8$), due to the competitive nature of intra-block opportunities (see Section IV-D1), we conduct experiments on a range of $\tau \in [0.2, 19.8]$. Additionally, we consider $\tau = \infty$ to encapsulate the maximum profitability in that block.

Figure 3 displays the revenue difference % of tau values to maximum discoverable revenue when $\tau = \infty$. The plot legend also includes the mean (μ) difference. The results indicate that the discovered arbitrage revenue only significantly degrades when τ is 0.2 seconds-84.39 % less revenue found on average. On the other hand, almost maximum profitability is reached when τ is close to block time (3.3 seconds). While the revenue % difference we observe depends on the infrastructure we execute the algorithm to measure the runtimes and the size of the pool set we consider, our analysis yields an intuition about the positive influence of available runtime on the discovered value.

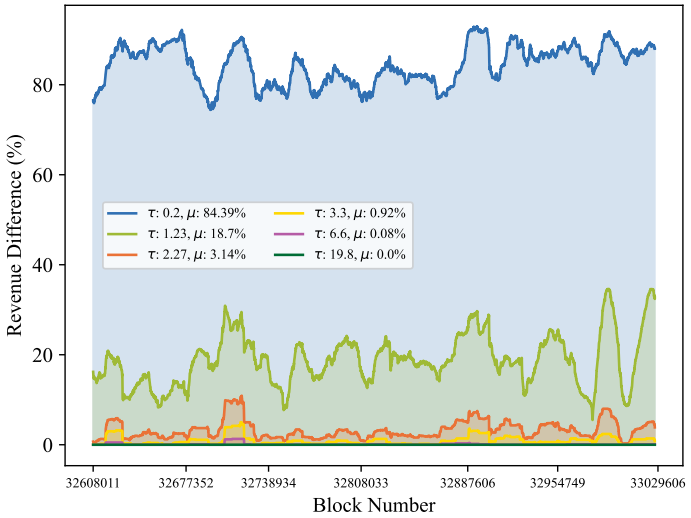


Fig. 3: Distribution of discovered revenue difference (in %) between τ values and $\tau = \infty$.

b) First-Come-First-Served: So far, we have adopted the profit-maximizing, greedy arbitrage selection strategy, as discussed in Section IV-A. However, as we have observed that MEV searchers on Algorand do not leave a significant amount of profits for block state arbitrage, we need to optimize the runtime of our algorithm even further to be competitive on the network-level arbitrage. To that extent, we adopt an FCFS strategy for arbitrage selection, which does not wait to consider all arbitrages available and select the most profitable ones but issues them as soon as their optimal input is calculated. While this strategy can yield less optimal revenue, it potentially saves valuable time.

Figure 4 displays the revenue difference % between FCFS and profit maximizing strategies, for different τ values. While the disparity is only 5.36 % when the algorithms are run for 0.2 seconds, the difference between the two strategies becomes more significant with increasing τ values. This is because, with more time, the profit-maximizing strategy discovers a wider set of opportunities and considers all of them when choosing the most profitable arbitrages to be issued. FCFS strategy, on the other hand, will always execute the first arbitrage it finds; hence, with increased time, the probability of finding an arbitrage that overlaps with an already taken one also increases. To minimize the revenue difference between the two approaches, the FCFS strategy requires applying a prioritization rule on the candidate arbitrage cycles before processing them. In our experiments, we sorted candidate cycles based on existing liquidity in involved pools, although more sophisticated rules can be developed by modeling the problem in a machine-learning domain.

V. MEV Extraction

In the MEV discovery section (see Section IV), we have shown that although the pools we track get updated on median every six blocks, we also observed that our algorithm fails to find many profitable opportunities as extracted value shows the efficiency of MEV searchers on Algorand in exploiting arbitrages as soon as they appear in a block. Since the name of the game is *latency optimization*, in this section, we demonstrate how transactions can

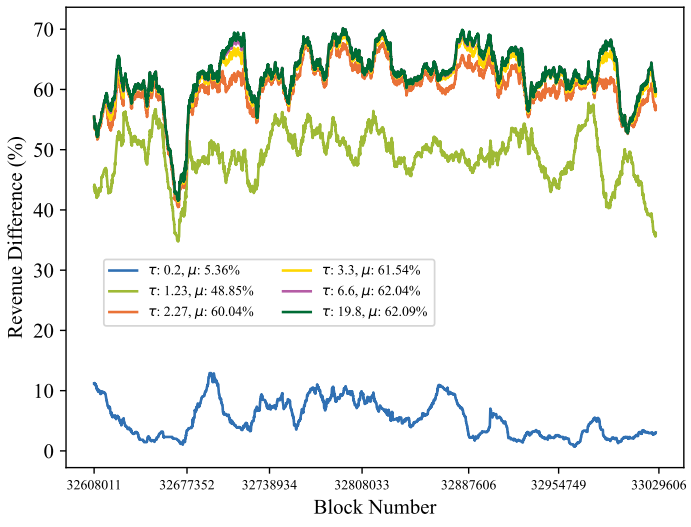


Fig. 4: Distribution of discovered revenue difference (in %) between profit-maximizing arbitrage selection and FCFS selection strategies, measured for varying τ values.

be prioritized by running experiments on a privately deployed Algorand network.

A. Network Experiments and Analysis

This section introduces the conducted network experiments, primarily focusing on the Algorand transaction ordering mechanism under simultaneous transaction attempts by competing entities. The central hypothesis driving these experiments examines the potential for a competitor to secure a transaction position in a block, thereby extracting arbitrage opportunities before others. The arbitrage trader must place their transaction ahead of the competition in a block, as subsequent transactions that compete for the same opportunity are likely to fail once the opportunity has been extracted. To understand the behavior of the network in practice, we run a sequence of experiments in a private network that helps us to verify the behavior on the mainnet of Algorand. The objectives of these network experiments are threefold:

- **Q1** How do the latency and transaction fees affect the transaction ordering when two different entities compete for prior transaction execution?
- **Q2** How does the connectivity between relay and participation nodes with differently distributed stakes affect the transaction ordering?
- **Q3** Is it possible to prioritize a transaction if we have visibility over the network topology and if so, how can we achieve this?

We set up a private network similar to Algorand mainnet to answer these questions instead of utilizing Algorand's testnet. Even though the testnet would also be costless, we require more visibility and control over the distribution of participation nodes and their stake in the testnet.

A private network provides granular control over the topology, in which we can introduce various latencies, distribute stakes among the participation nodes the way we want, and scale transaction sending to emulate various network conditions. We use the METHODA framework [19] that extends the ENGINE toolchain [20]. METHODA also implements the Algorand blockchain, making it easier to automate the Algorand network deployment process in a distributed environment relevant to our scenarios. In addition, it supports scalable experiments with a number of nodes and can emulate delays using Network Emulator (netem)[17]

1) Experiment Design: To collect insights and answer the defined questions, we devise three scenarios. These scenarios are realized using three distinct network topologies, each denoted using specific notations. These notations are defined as follows:

- $P = \{p_i \mid i = 1, \ldots, n_P\}$: Represents the set of participation nodes, where each p_i is an individual participation node and n_P is the total number of such nodes.
- $R = \{r_j \mid j = 1, \ldots, n_R\}$: Denotes the set of relay nodes, with each r_j being a relay node and n_R the total number of relay nodes.
- $NP = \{np_k \mid k = 1, \ldots, n_{NP}\}$: The set of non-participation nodes, where np_k is a non-participation node and n_{NP} is their total count.

The experimented network topologies are then formally represented as tuples (P, R, NP), encompassing the node sets and their connectivity relations. The three scenarios that we tested as part of this research work have been elaborated in Figure 5. ***Scenario I*** in Section V-A1 introduces a topology where we see two non-participating peers and a single relay, connected to a single participation node- np_2 has worse delay towards the r_1 ①. Therefore, we expect transactions issued through np_1 to be positioned in prior block positions even though np_2 pays more. Similarly, for ***Scenario II*** we introduce delay (②) from np_1 towards r_3 and between r_2 and r_3 Figure 5b. However, the participation node with higher stake p_1 should propose more often than p_2, placing np_2 in a worse position. Last, ***Scenario III*** places delay on the shortest path towards p_1 and p_2 (③) from np_2 Figure 5c. In this case, all participation nodes have equal stakes, thus, the same probability of winning the proposer selection round with the only difference being the Verifiable Random Function (VRF) value they generate.

[17]https://man7.org/linux/man-pages/man8/tc-netem.8.html

TABLE I: Summary of Observations for Scenarios 1, 2 and 3

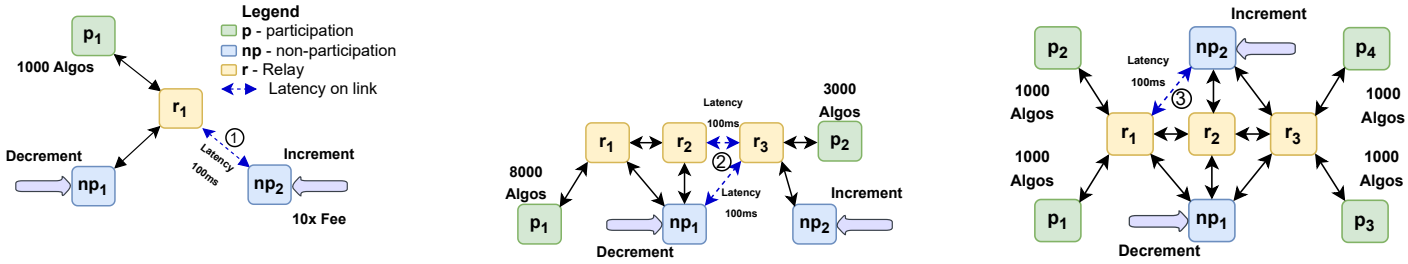
	Proposer	Blocks count	Prioritized Method	Frequency
Scenario 1	p_1	500	Decrement	100%
Scenario 2	p_1	377	Decrement	100%
	p_2	123	Increment	76.42%
Scenario 3	p_1	126	Decrement	96.03%
	p_2	113	Decrement	97.01%
	p_3	134	Increment	51.97%
	p_4	127	Decrement	54.80%

To emulate two MEV searchers S_1 and S_2, simultaneously trying to take an arbitrage opportunity, using the same detection algorithm, we deploy a simple, smart contract (see Algorithm 1) which tracks the state of a global variable $last\_executed$, and the winner searcher is the one which modifies this variable's value the first. We assume that S_1 and S_2 change the global state by calling functions *decrement* and *increment*, respectively.

Algorithm 1 Smart Contract Operations

```
1: Initialize global variable last_executed = ""
2: function INCREMENT
3:     last_executed ← "increment"
4: function DECREMENT
5:     last_executed ← "decrement"
```

2) Experiment Setup: Each Algorand node - participant, non-participant, and relay node is deployed using a combination of multiple physical machines and Linux containers (LXCs) interacting with each other. Each peer node runs in its own LXC container allocated the recommended HW specs - 16 vCPUs for a relay and 8 vCPUs for participation nodes. The LXC containers run on a node with 32cores/64threads AMD EPYC 7543 with 512 GB DDR 5 RAM. Depending on the scenario, we scale to additional physical nodes once we reach the available resources. The nodes are interconnected via a 10 GbE switch. Therefore, even if the delay between the container and physical machine differs, it is negligible compared to introducing delay by netem. We conducted our experiments using the Algorand protocol version *abd3d4823c6f77349fc04c3af7b1e99fe4df699f* and `go-algorand` in 3.18.1 release. During each experiment, we create 500 blocks corresponding to roughly 27.5 minutes of runtime.



(a) Scenario 1 (b) Scenario 2 (c) Scenario 3

Fig. 5: Network topology of three scenarios discussed in Section V-A with legend on in Figure a). Dashed blue arrows connecting peers corresponds to delay of 100 ms.

3) Experiment Findings: To highlight the findings from thee three scenarios, we follow how often the *decrement* method is invoked on node np_1 by S_1, while the *increment* method is simultaneously invoked on node np_2 by S_2. A detailed analysis of these results is provided below, with all results summarized in Table I highlighting the proposer node, block count, the method, and frequency for the corresponding method execution.

a) Scenario 1: In ***Scenario I*** we, observe a consistent prioritization pattern throughout all 500 iterations. The *decrement* function call directed to node np_1 by S_1 always reaches p_1 first over the *increment* call sent to node np_2 from S_2. Notably, this occurred despite the transaction to np_2 being attached with a fee tenfold that of the transaction to np_1. This outcome distinctly indicates that higher transaction fees are ineffective within Algorand's FCFS network to secure prioritization (unless, a network congestion strategy is adopted as described in [5]). Furthermore, the absence of additional latency between np_1 and relay r_1 resulted in the transactions of S_1 being consistently favored.

b) Scenario 2: Continuing with the observations in ***Scenario II*** , we see that node p_1 is chosen as the proposer in 377 out of 500 ($\sim$ 75 %) iterations. Hence, we confirm that proposer selection frequency is positively correlated to the stake in the network (p_1 : 72 %). Additionally, it is noted that when p_1 served as the proposer, the *decrement* function call is consistently prioritized. On the other hand, when p_2 is the proposer, the *increment* call received prioritization, although not every time. Despite node np_1 having advantageous network positioning due to its connections with all relays, the election of p_2 as the proposer enabled node S_2, using np_2, to propagate their transactions ahead of S_1. This advantage for S_2 can be attributed to the latency existing between np_1 and relay r_3. Nonetheless, as S_1 prevails in most cases, the findings signify the importance of a node's connectivity to a relay with higher-staked participants to achieve a prior position in the block.

c) Scenario 3: Lastly, ***Scenario III*** confirms that the proposers' observed selection frequency matches the theoretical selection frequency and its stake within the network. Additionally, two critical observations emerged from this scenario. First, in instances where p_1 and p_2 are elected as proposers, there is a heavy bias towards prioritizing the *decrement* call by S_1. This observation validates our earlier conclusions, underscoring latency as a pivotal factor influencing transaction order. The latency between np_2 and relay r_2 contributed to the delayed processing of the *Increment* method call sent to np_2 from S_2. Second, when p_3 and p_4 are elected as proposers, the data indicate an almost equal likelihood of giving preference to either searcher's call. This suggests that when searchers' utilized nodes' latency to relays and the respective participation nodes with equivalent stakes are similar, there is an equal chance of their competing transactions being executed first.

B. Network Analysis Findings

The scenarios, emulating key dynamics on the blockchain network, contribute to verifying Algorand's behavior under various conditions and answer the outlined questions ***Q1-Q3***. The summary of the key findings is as follows:

- Transaction fee does not play any role in prioritizing a transaction in the Algorand blockchain since the proposers have no incentive to reorder transactions based on fees attached as they do not get to keep them.
- Network latency is critical in ordering transactions in FCFS blockchains. For arbitrage traders in Algorand, this underscores the importance of minimizing the latency between their transaction nodes and the corresponding relay nodes. This strategic positioning is crucial for enhancing the likelihood of transaction precedence to their competitors.
- The proximity of a node to a high-staked proposer is significantly influential in transaction sequencing. Arbitrage traders aiming to exploit market opportunities should prioritize transmitting their transactions to well-connected relays with multiple high-staked nodes. Such an approach increases the probability of their transactions being processed earlier, thereby gaining a competitive edge in transaction execution.

VI. Conclusion

In this paper, we have provided a twofold approach for MEV searching on an FCFS blockchain, Algorand. Initially, we developed an algorithm for arbitrage detection and evaluated its performance using historical block state data, taking into account the time constraints of Algorand. Our findings indicate that although significant state updates between blocks are infrequent, MEV searchers on Algorand tend to close arbitrage opportunities within the same block they arise. This suggests that MEV detection should occur at the transaction level in the mempool, rather than at the block state level as initially attempted.

Subsequently, we performed experiments on a private Algorand network to identify key factors in transaction prioritization within an FCFS framework. Our results highlight the importance of minimizing latency, particularly in connections with relays that have well-connected links to multiple high-stake nodes.

Looking ahead, we plan to refine our arbitrage detection algorithm by considering a broader set of pools and applying it directly at the network level. Additionally, we aim to extend our network experiments to the Algorand mainnet to explore potential latency correlations between high-staked block proposers and successful MEV searchers. This future research will enhance our understanding of MEV dynamics in FCFS blockchains like Algorand.

REFERENCES

[1] "Top Decentralized Exchanges Ranked by Volume." [Online]. Available: https://www.coingecko.com/en/exchanges/decentralized

[2] P. Daian, S. Goldfeder, T. Kell, Y. Li, X. Zhao, I. Bentov, L. Breidenbach, and A. Juels, "Flash boys 2.0: Frontrunning in decentralized exchanges, miner extractable value, and consensus instability," in *2020 IEEE Symposium on Security and Privacy (SP)*, 2020, pp. 910–927.

[3] "libMEV." [Online]. Available: https://libmev.com

[4] F. Carrillo and E. Hu, "MEV in fixed gas price blockchains: Terra Classic as a case of study," Mar. 2023, arXiv:2303.04242 [cs]. [Online]. Available: http://arxiv.org/abs/2303.04242

[5] B. Öz, F. Rezabek, J. Gebele, F. Hoops, and F. Matthes, "A Study of MEV Extraction Techniques on a First-Come-First-Served Blockchain," Nov. 2023, arXiv:2308.06513 [cs] version: 2. [Online]. Available: http://arxiv.org/abs/2308.06513

[6] J. Chen and S. Micali, "Algorand," 2017.

[7] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich, "Algorand: Scaling byzantine agreements for cryptocurrencies," Cryptology ePrint Archive, Paper 2017/454, 2017, https://eprint.iacr.org/2017/454. [Online]. Available: https://eprint.iacr.org/2017/454

[8] ——, "Algorand: Scaling byzantine agreements for cryptocurrencies," in *Proceedings of the 26th Symposium on Operating Systems Principles*, ser. SOSP '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 51–68. [Online]. Available: https://doi.org/10.1145/3132747.3132757

[9] "DefiLlama." [Online]. Available: https://defillama.com/chain/Algorand

[10] L. Zhou, K. Qin, A. Cully, B. Livshits, and A. Gervais, "On the Just-In-Time Discovery of Profit-Generating Transactions in DeFi Protocols," in *2021 IEEE Symposium on Security and Privacy (SP)*, May 2021, pp. 919–936, iSSN: 2375-1207.

[11] Y. Wang, Y. Chen, H. Wu, L. Zhou, S. Deng, and R. Wattenhofer, "Cyclic Arbitrage in Decentralized Exchanges," Jan. 2022, arXiv:2105.02784 [cs, q-fin]. [Online]. Available: http://arxiv.org/abs/2105.02784

[12] R. McLaughlin, C. Kruegel, and G. Vigna, "A large scale study of the ethereum arbitrage ecosystem," in *32nd USENIX Security Symposium (USENIX Security 23)*. Anaheim, CA: USENIX Association, Aug. 2023, pp. 3295–3312. [Online]. Available: https://www.usenix.org/conference/usenixsecurity23/presentation/mclaughlin

[13] G. Angeris, A. Evans, T. Chitra, and S. Boyd, "Optimal Routing for Constant Function Market Makers," in *Proceedings of the 23rd ACM Conference on Economics and Computation*, ser. EC '22. New York, NY, USA: Association for Computing Machinery, Jul. 2022, pp. 115–128. [Online]. Available: https://dl.acm.org/doi/10.1145/3490486.3538336

[14] L. Zhou, K. Qin, C. F. Torres, D. V. Le, and A. Gervais, "High-frequency trading on decentralized on-chain exchanges," in *2021 IEEE Symposium on Security and Privacy (SP)*, 2021, pp. 428–445.

[15] K. Qin, S. Chaliasos, L. Zhou, B. Livshits, D. Song, and A. Gervais, "The blockchain imitation game," in *Proceedings of the 32nd USENIX Conference on Security Symposium*, ser. SEC '23. USA: USENIX Association, 2023.

[16] H. Adams, "Uniswap: A protocol for automated token exchange on ethereum," 2018, accessed: [Your Access Date Here]. [Online]. Available: https://uniswap.org/whitepaper.pdf

[17] Uniswap, "Uniswap v3 core," 2021, accessed: [Your Access Date Here]. [Online]. Available: https://uniswap.org/whitepaper-v3.pdf

[18] "Crypto API Documentation," 2023. [Online]. Available: https://www.coingecko.com/en/api/documentation

[19] F. Rezabek, K. Glas, R. von Seck, A. Aroua, T. Leonhardt, and G. Carle, "Multilayer environment and toolchain for holistic network design and analysis," 2023.

[20] F. Rezabek, M. Bosk, T. Paul, K. Holzinger, S. Gallenmüller, A. Gonzalez, A. Kane, F. Fons, Z. Haigang, G. Carle *et al.*, "Engine: Flexible research infrastructure for reliable and scalable time sensitive networks," *Journal of Network and Systems Management*, vol. 30, no. 4, p. 74, 2022.