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

Burak Öz, Jonas Gebele, Parshant Singh, Filip Rezabek, Florian Matthes
School of Computation, Information and Technology, Technical University of Munich
{burak.oez, jonas.gebele, parshant.singh, filip.rezabek, matthes}@tum.de

***Abstract*—First-Come-First-Served (FCFS) transaction ordering has been discussed as a fairness approach against harmful Maximal Extractable Value (MEV) strategies. However, such an ordering mechanism promotes latency optimizations, similar to High-Frequency Trading in Traditional Finance. This paper examines the dynamics of the MEV extraction game in an FCFS network, specifically Algorand. We introduce an arbitrage opportunity detection algorithm tailored to Algorand's time constraints and assess its effectiveness. Our analysis reveals that while the states of exchange pools are updated approximately only every six blocks, pursuing MEV at the block state level is not viable, as arbitrage opportunities are typically closed within the block they appear. Additionally, we experiment on a private Algorand network to uncover latency optimization factors and show the importance of reducing latency in connections with relays well-connected to high-staked proposers.**

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

I. Introduction

The daily trading volume on Decentralized Exchanges (DEXs) exceeds multi-billion USD [1]. Such amount of activity on Decentralized Finance (DeFi) protocols has led to the emergence of transaction manipulation strategies known as Maximal Extractable Value (MEV) [2]. While this term generally refers to the value that can be captured by privileged entities like block proposers, who can determine transaction inclusion, exclusion, and ordering, value extraction is not limited to them, as MEV activity dashboard libMEV [3] reflects a total of $64\,\text{million}$ USD made by profit-seeking entities operating on the mempool, known as *MEV searchers*, since Ethereum's merge in September 2022.

In blockchains where block proposers arrange transactions in a block in the order received, thus First-Come-First-Served (FCFS), the dynamics for MEV searching differ compared to a fee-based blockchain like Ethereum [4]. While FCFS has been studied for achieving order-fairness [5], [6], it is also discussed to be shifting MEV extraction to a latency game [4], [7]. As the available window for searching MEV gets constrained by the expected arrival time of a competing transaction, MEV searchers have to compete like High-Frequency Traders in Traditional Finance and minimize latency with the source of the transaction, such as the network node of a DEX, and the block proposers' nodes.

In this paper, we demonstrate how the MEV game can be strategically played on a Layer-1, FCFS network, Algorand, extending the research by Öz et al. [4]. Unlike FCFS Layer-2 solutions such as Arbitrum, Algorand has a public mempool where searchers can monitor the pending transactions and a decentralized set of block proposers, making it an interesting network to study. To that extent, we first formally define a cyclic arbitrage opportunity discovery algorithm tailored to Algorand's network constraints. Then, we evaluate its performance on historical blockchain data we collect. Further, we present a private Algorand network setup and run experiments on it to identify latency optimization factors. Our research contributes to understanding the dynamics of MEV extraction in FCFS networks, showcasing how algorithms can discover opportunities when constrained on runtime and highlighting the critical network characteristics for prioritized execution.

II. Related Work

The discourse on arbitrage opportunity discovery includes studies mainly conducted on Ethereum. Zhou et al. [8] provide a greedy cycle detection algorithm that achieves a sub-block time runtime. However, their approach is not tailored to find an optimal input and maximize profits as it uses a gradual-increment method and operates on a limited asset set. Wang et al. [9] focus on constant-product markets on UniswapV2 [10] and show the consistent existence of larger than $1\,\text{ETH}$ opportunities across blocks. Following, McLaughlin et al. [11] conduct a larger scale study on $5.5\,\text{million}$ blocks and incorporate further market invariants such as UniswapV3 [12]. They find approximately $4.5\,\text{billion}$ worth of opportunities with only $0.51\,\%$ of the ones that yield over $1\,\text{ETH}$ revenue being successfully executable.

Carillo and Hu [7] and Öz et al. [4] conduct the first studies of MEV on FCFS blockchains. The former work analyzes arbitrage MEV extraction on Terra Classic and identifies more than $188\,000$ arbitrages, highlighting the success of MEV searchers adopting a spamming strategy. They also measure the latency on the network by monitoring $400\,000$ transactions through their distributed nodes to signify the importance of geographical positioning for hearing about transactions first. Öz et al. investigate the applicability of transaction ordering techniques observed in fee-based blockchains to FCFS blockchains with a focus on Algorand. They detect $1.1\,\text{million}$ arbitrages across $16.5\,\text{million}$ blocks and show the prevalence of network state backruns among all detected arbitrages. They also reveal that particular MEV searchers profit from top-of-the-block positions, hinting at latency and transaction timing optimizations. Their study presents network clogging as a viable MEV strategy in FCFS networks due to its low cost and finds arbitrage clogs executed on Algorand.

III. Finding Profitable Opportunities

In this section, we present our algorithm for finding profitable arbitrage MEV opportunities as they are shown to be frequently executed by MEV searchers on Algorand [4]. We propose a real-time algorithm tailored to network's specific time constraints, aiming to identify nearly all emerging arbitrage cycles and incorporate an efficient input optimization technique. We evaluate our algorithm's performance on historical state data.

A. Cyclic Arbitrage Detection Algorithm

Our algorithm 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 (we ignore the pools which offer more than two assets). 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 $I^b_{\text{total}} \subseteq I_{\text{total}}$ including our profit assets, targeting cycles with updated pool reserves in block b. We ignore the cycles with only stale pools as we have already evaluated them.

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 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 opportunity. If so, we search for the profit-maximizing input for i using SciPy's *minimize function* 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 $\mathcal{A}^b_{\text{candidates}}$.

Finally, we adopt one of two strategies for arbitrage selection: a greedy strategy for maximizing profits 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 trades off maximizing profitability since early discovered arbitrages may invalidate to-be-realized, more lucrative opportunities.

B. Empirical Evaluation Setup and Results

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. 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 [13].

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 executed 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 [4].

In Figure 1, we display a time series of 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 inside the block they emerge. Hence, in most cases, price discrepancies do not carry over to the next block. While the maximum realized revenue of an arbitrageur is 167.17 USD, we find, at most, a 32.2 USD opportunity on the block state that is fully closed in the subsequent six blocks. 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 located at the immediate following position.

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 under 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 s (block time $\approx$ 3.3 s). Interestingly, the max state update delta reaches 294 blocks ($\sim$16 min), although our algorithm does not detect any profitable block state arbitrage during this window.

The Value of Time: We measure the profitability of our algorithm as a function of τ to observe the impact of the available runtime window on the 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 III-B1), 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.

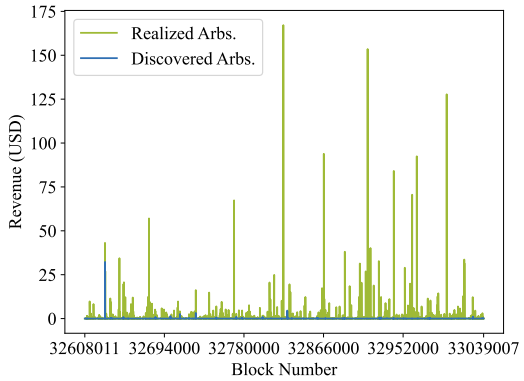


Fig. 1: Time series of arbitrage revenue discovered on the block state by our algorithm (blue) versus the total realized revenue by arbitrages executed in every block (green).

In Figure 2a, we plot the revenue difference percentage of τ values to maximum discoverable revenue when $\tau = \infty$, with the mean difference (μ) indicated in the legend. The results indicate that the discovered arbitrage revenue only significantly degrades when τ is very low (at $0.2\,\mathrm{s}$, $84.39\,\%$ less arbitrage revenue is found). On the other hand, almost maximum profitability is reached when τ is close to block time ($3.3\,\mathrm{s}$). 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.

First-Come-First-Served: So far, we have adopted the profit-maximizing, greedy arbitrage selection strategy. However, since MEV searchers on Algorand do not leave many opportunities for arbitraging on the block state, we need to optimize the runtime of our algorithm further to be competitive on the network-level arbitrage. Thus, 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.

We find that the disparity between greedy arbitrage selection strategy and FCFS is only $5.36\,\%$ when the algorithms are run for $0.2\,\mathrm{s}$ (see Figure 2b). However, the difference becomes significant with increasing τ values as, with more time, the greedy strategy discovers a broader set of opportunities and considers all of them when choosing the most profitable ones to be issued. FCFS strategy, on the other hand, always executes the first arbitrage it finds; hence, with increased time, the probability of finding arbitrages overlapping with the taken ones 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.

IV. Optimizing Network Latency

In FCFS networks, MEV extraction depends on *latency optimization* with the transaction source for observing an opportunity before the others and with block proposers for getting executed first. We conduct experiments on a privately deployed Algorand network to show how transactions can be prioritized and discuss the results in this section.

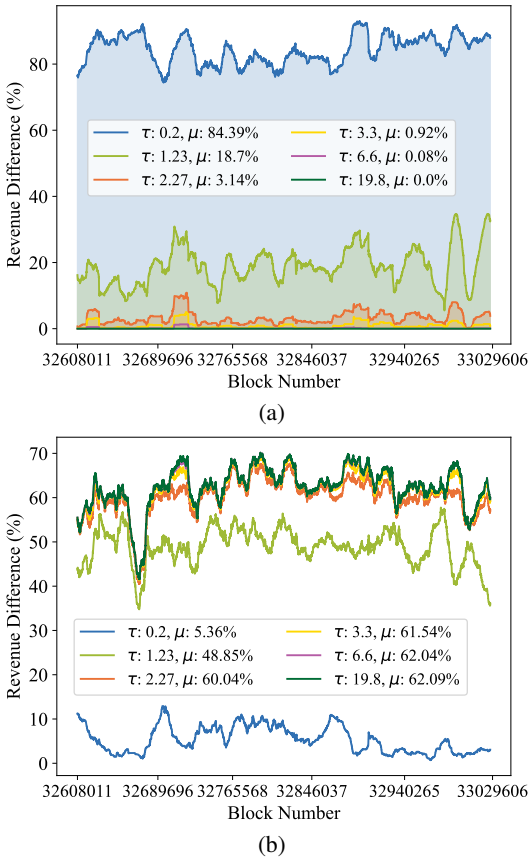


Fig. 2: Time series of discovered revenue difference percentage between the execution of our algorithm with τ and $\tau = \infty$ (2a) and between greedy arbitrage selection and FCFS strategies, measured for varying τ values (2b).

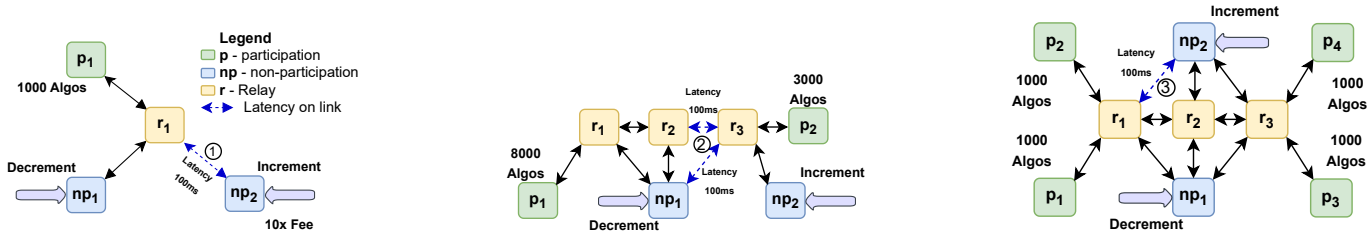
A. Network Experiments and Analysis

We first introduce the conducted network experiments, primarily focusing on the Algorand transaction ordering mechanism under simultaneous transaction attempts by competing entities. The objectives of these network experiments are threefold:

- **Q1** How do latency and transaction fees affect transaction ordering when different parties compete for their prior execution?
- **Q2** How does the connectivity between relay and participation nodes with various stakes affect the transaction ordering?
- **Q3** Is it feasible to prioritize a transaction if we have a view of the network topology and if so, how can we achieve this?

As we require visibility and control over the distribution of participation nodes and their stake, we set up Algorand as a private network. This allows us to control the topology, in which we can introduce latencies, scale transaction issuance to emulate various network conditions and allocate required stakes to participants. We use the METHODA framework [14] that extends the EnGINE toolchain [15]. METHODA implements the Algorand blockchain, automating the deployment process relevant to our scenarios. It also supports scalable experiments with more nodes and emulates delays using netem.

1) Experiment Design: We devise three scenarios, using distinct network topologies, as shown in Figure 3. ***Scenario I*** in Figure 3a introduces a topology with two non-participating peers and a single relay connected to a single participation node. We note that np_2 has worse delay towards r_1 (①), and we expect transactions issued through np_1 to be prioritized even though



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

Fig. 3: Network topology of three scenarios (3a, 3b, 3c) with a legend in 3a. Dashed blue arrows correspond to delay of 100 ms.

np_2 offers higher fees. Similarly, in ***Scenario II***, we introduce a delay (②) from np_1 towards r_3 and between r_2 and r_3 (see Figure 3b). However, since the participation node with a higher stake (p_1) is expected to propose blocks more often, np_2 should be in a worse position regarding transaction prioritization. Lastly, ***Scenario III*** assigns the same stake to every proposer while placing a delay on the shortest path towards p_1 and p_2 (③) from np_2 (see Figure 3c). As all participation nodes have equal stakes, they have the same probability of winning the proposer selection round, with the only difference being the Verifiable Random Function (VRF) value they generate. Thus, we expect np_1's transactions to be prioritized significantly in half of the experiments whenever p_1 or p_2 proposes a block.

To emulate two MEV searchers S_1 and S_2, simultaneously attempting to exploit an opportunity, we deploy a simple smart contract, 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.

2) Experiment Analysis: To highlight the findings from the three scenarios, we monitor 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.

In ***Scenario I***, we observe a consistent prioritization pattern throughout all 500 blocks. 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 by S_2. Notably, this occurs despite the transaction to np_2 has a tenfold fee that of the transaction to np_1. This result confirms the ineffectiveness of fees within Algorand's FCFS network for transaction prioritization.

Scenario II shows that node p_1 is chosen as the proposer in 377 out of 500 ($\sim$ 75 %) iterations, proportional to its stake in the network (p_1 : 72 %). We note that when p_1 proposed blocks, 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 being connected with all relays, the selection of p_2 as the proposer enables searcher S_2, using node np_2, to propagate their transactions ahead of S_1. This advantage for S_2 is due to the latency 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.

Scenario III also confirms that the proposers' selection frequency is proportional to its stake within the network. In instances where p_1 and p_2 are chosen, there is a heavy bias towards prioritizing the *decrement* call by S_1. This validates our previous findings, underscoring latency as a key factor influencing transaction order. When p_3 and p_4 are selected as proposers, the data show an almost equal likelihood of giving preference to either searcher's call.

	Proposer	Blocks	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%

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

B. Experiment Findings

The experimented scenarios contribute to verifying Algorand's behavior under various conditions and answer the outlined questions ***Q1-Q3***. The key findings are as follows:

- The default node implementation does not incorporate fee levels to prioritize transactions. Although the participation nodes can alter their source code, they have no such incentive unless they search MEV, as they do not keep the transaction fees.
- Network latency is critical in ordering transactions. For MEV searchers, this underscores the importance of minimizing the latency between their transaction and relay nodes.
- The proximity of a node to a high-staked proposer is significantly influential in transaction sequencing. MEV searchers should aim to transmit their transactions to well-connected relays with multiple high-staked nodes, as these nodes are expected to propose blocks more frequently.

V. Conclusion

This paper examines the intricacies of MEV extraction on FCFS networks with a study on Algorand. We propose an algorithm for arbitrage opportunity discovery and evaluate its performance on historical blockchain data. Our findings indicate that although state updates between blocks are infrequent, arbitrage opportunities are effectively closed in the block where they appear. We also show the impact of available runtime and arbitrage selection strategy on the discovered revenue. Subsequently, we perform experiments on a private Algorand network and highlight the importance of minimizing latency in connections to relays with well-connected links to multiple high-staked nodes. In future work, we plan to refine our 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 latency relations between high-staked block proposers and successful MEV searchers.

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] 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

[5] M. Kelkar, S. Deb, S. Long, A. Juels, and S. Kannan, "Themis: Fast, strong order-fairness in byzantine consensus," in *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 475–489.

[6] M. Kelkar, S. Deb, and S. Kannan, "Order-fair consensus in the permissionless setting," in *Proceedings of the 9th ACM on ASIA Public-Key Cryptography Workshop*, ser. APKC '22, New York, NY, USA, 2022, p. 3–14.

[7] F. Carrillo and E. Hu, "MEV in fixed gas price blockchains: Terra Classic as a case of study," Mar. 2023, arXiv:2303.04242 [cs].

[8] 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.

[9] Y. Wang, Y. Chen, H. Wu, L. Zhou, S. Deng, and R. Wattenhofer, "Cyclic Arbitrage in Decentralized Exchanges," in *Companion Proceedings of the Web Conference 2022*, ser. WWW '22. New York, NY, USA: Association for Computing Machinery, Aug. 2022, pp. 12–19. [Online]. Available: https://doi.org/10.1145/3487553.3524201

[10] H. Adams, "Uniswap: A protocol for automated token exchange on ethereum," 2018, accessed: 18.12.2023. [Online]. Available: https://uniswap.org/whitepaper.pdf

[11] 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

[12] Uniswap, "Uniswap v3 core," 2021, accessed: 18.12.2023. [Online]. Available: https://uniswap.org/whitepaper-v3.pdf

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

[14] F. Rezabek, K. Glas, R. von Seck, A. Aroua, T. Leonhardt, and G. Carle, "Multilayer environment and toolchain for holistic network design and analysis," Nov. 2023, arXiv:2310.16190 [cs] version: 2. [Online]. Available: https://arxiv.org/abs/2310.16190v2

[15] 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.