A COMPARATIVE ANALYSIS OF ETHEREUM GAS PRICE ORACLES' PERFORMANCE AND A MECHANISM FOR ETHEREUM GAS PRICE PREDICTION POST-EIP-1559

by

Ibrahim Barada

A thesis submitted in partial fulfillment
of the requirements for the degree

of

Master of Science

in

Computer Science

MONTANA STATE UNIVERSITY
Bozeman, Montana

December 2022

ACKNOWLEDGEMENTS

I would like to express my gratitude to Dr. Mike Wittie and Dr. David Millman for their supervision, guidance, and valuable input throughout my research journey. Also, I would like to acknowledge my friends and family for their continued support and motivation. Finally, I would like to thank Gianforte School of Computing at Montana State University for offering me the Benjamin Scholarship and the opportunity to research and explore emerging technologies.

TABLE OF CONTENTS

TABLE OF CONTENTS – CONTINUED

LIST OF TABLES

LIST OF FIGURES

Figure Page

LIST OF FIGURES – CONTINUED

LIST OF ALGORITHMS

ABSTRACT

Blockchain transactions compete for limited space in blockchain blocks. Miners prefer to include transactions with higher fees into new blocks. Ethereum released EIP-1559 as an upgrade for its transaction pricing mechanism. The improvement proposal aims to stabilize the transaction pricing mechanism and improve the predictability of gas prices. In the context of Ethereum, gas price oracles predict fees such that transactions submitted at those fees make it into a block within a target delay. In practice, however, Ethereum gas price oracles are inaccurate, which makes it difficult for distributed applications to operate predictable services in terms of price and performance. To understand and measure oracle accuracy we define new gas prediction performance metrics. We demonstrate that oracles underprice transactions, causing them to miss the delay target. We also show that oracles overprice transactions, causing them to meet the delay target, but at a higher-than-necessary cost. As a result of oracles inaccuracies, users tend to either wait longer or pay more than sufficient gas prices for a transaction to get into a block. We provide a comparative analysis of five gas price oracles pre and post-the release of EIP-1559 showing their performance in terms of accuracy of acceptance, underpricing, and overpricing. We also discuss the factors that influence oracle accuracy and the effects of those inaccuracies in terms of time and money wasted. We apply our predefined metrics to study the performance of oracles pre and post-EIP-1559. We observe that EIP-1559 improved the transaction acceptance rate and shortened acceptance delays. On the other hand, we observe that EIP-1559 increased transaction overpricing. The current gas price prediction mechanisms required further investigation after the release of EIP-1559. Hence, we devised a new mechanism to predict gas prices of EIP-1559-compatible transactions on the Ethereum blockchain. The mechanism allows users to calculate gas prices based on the current block utilization and base fee. We measured the probability of acceptance, time wasted, and money wasted and noticed an increase in the probability of acceptance and a decrease in both time and money wasted in comparison to the currently existing oracles.

1

INTRODUCTION

Blockchain throughput, in terms of transactions per second, is fundamentally limited by the time it takes to disseminate and validate new blocks [22]. To ensure fair sharing of blockchain resources and incentivize miners to validate transactions, users pay a fee to include their transactions into a block. These fees, however, are volatile due to changes in transaction volume [5] and cryptocurrency price [29]. To help blockchain users submit transactions that enter the blockchain within the expected time and at a reasonable price, users tend to consult with gas price oracles. Gas price oracles provide gas price predictions based on historical blockchain data and they follow multiple mechanisms to predict gas prices under different conditions. In the classic pricing approach, users submit transactions with a gas price that the user sets; a.k.a. the bid, predicted by the oracle. However, the classic pricing mechanism results in drawbacks, such as transaction overpayment and long acceptance delays [18]. Consequently, Ethereum moved to a new pricing procedure to control the volatility of the classic pricing system. Unfortunately, oracles predictions are less accurate than advertised [1], making it difficult for distributed applications (Dapps) to operate predictable services in terms of price and performance. In efforts to address this problem, we first define performance metrics that allow us to measure oracle prediction accuracy. Then, we apply our metrics to five publicly available oracles and presented a comparative analysis. After providing a comparative analysis between oracles pre and post-EIP-1559, we concluded that the protocol improves oracle's performance.

Furthermore, we investigated the prediction mechanisms applied by oracles and devised a new mechanism based on block utilization and the latest base fee. The motivation behind our algorithm comes from the fact that the protocol maintains some level of stability by

increasing and decreasing the base fee with respect to the block's utilization.

The expected impact of this dissertation is allowing users to make informative decisions when submitting transactions on the Ethereum blockchain. We also aim to provide a new prediction mechanism for EIP-1559-compatible transactions that reduces the time and money spent on transactions while providing accurate acceptance results. We organize the rest of this dissertation as follows. In Ch. 1, we provide an overview of Ethereum pool mechanics and discuss different types of gas price oracles and prediction mechanisms. We also introduce 3 performance metrics: accuracy of acceptance, underpricing and overpricing. Additionally, we quantify the underpricing and overpricing metrics in terms of time and money wasted, respectively. Then, we apply our proposed metrics to 5 publicly available oracles and present a comparative analysis pre and post-EIP-1559. In Ch. 2, we devise a new prediction mechanism for EIP-1559-compatible transactions. The algorithm predicts gas prices with respect to the current block utilization and base fee. We calculate the probability of acceptance of our algorithm and we notice an increase compared to other oracles. We also calculate the time and money wasted for transactions that consult with our oracle and we conclude that our prediction algorithm wastes less time and money compared to oracles of similar nature. Conclusions are drawn in Ch. 4.

3

PERFORMANCE METRICS FOR ETHEREUM GAS PRICE ORACLES AND A COMPARATIVE ANALYSIS OF ORACLES PERFORMANCE PRE AND POST-EIP-1559

Introduction

In the context of Ethereum, a transaction fee is the amount of *gas* spent by a transaction on operations within the Ethereum Virtual Machine (EVM) multiplied by the *gas price* specified in the transaction. Gas price is also loosely correlated with how long it takes for a transaction to be accepted into a new block, as miner implementations prefer to include higher price transactions in blocks to maximize block rewards [11].

To help blockchain users submit transactions that enter the blockchain within the expected time and at a reasonable price, gas price oracles provide gas price predictions based on historical blockchain data.

As part of the EIP-1559 release [9], Ethereum updated its transaction pricing technique which previously operated as a first-price auction mechanism. EIP-1559 aims to reduce rapid spikes in gas prices and allows oracles to improve gas price predictability. In the classic pricing approach, a transaction had a single gas price fee that the user sets as part of the bidding process. However, EIP-1559 removed the gas fee and introduced 3 different fees instead. EIP-1559 introduced the base fee, max fee and priority fee. The base fee is the minimum gas price required for a miner to consider a transaction into the blockchain. For example, any transaction priced below the base fee will not make into a block. In efforts to reduce sudden gas price spikes, the protocol automatically updates the base fee with respect to block utilization. For example, if the network is highly utilized, the base fee will increase and consequently users will submit less transactions. On the other hand, if the network is under utilized, the base fee will decrease encouraging users to submit more transactions. The max fee is the maximum price a user is willing to pay for a transaction to be accepted

in the blockchain. For example, if the base fee increases while the transaction is still in the waiting pool, the max fee ensures that the transaction meets the new base fee requirement. The max fee plays a vital role during times of gas price fluctuation. For example, if the base fee increases during transaction submission, the max fee covers the extra price up to a limit defined by the user. Finally, the priority fee is the price paid by the user to incentivize the miner to select a transaction. The priority fee in EIP-1559 is similar to the concept of the tip, it is optional and the amount decided by the user as an incentive to the miner.

In response, some gas price oracles upgraded their prediction mechanisms and released EIP-1559-compatible gas price tools that predict both the max and priority fees.

Currently, the accuracy of the performance of the gas price oracles is not well understood. Pierro et al. analyzed the accuracy of EthGasStation oracle and found out that transactions submitted with predicted gas prices fail to be included in the Ethereum block within the expected delay [1]. However, their work does not compare the accuracy of multiple oracles. Their work also does not analyze the occurrence of cases where transactions enter a block within the expected delay but at a higher-than-necessary price. Other works have developed improvements to the accuracy of price prediction methods used in existing oracles, but these methods work only for very low expected delays and did not result in hosted services available to blockchain users [10, 25, 32]. Therefore, we observe that there have been limited studies on gas price oracles performance metrics pre-EIP-1559 and nothing, to our knowledge, on post-EIP-1559. As a result, blockchain users do not know which publicly available oracle to rely on and how much error to anticipate.

In this chapter, we revisit the question of gas price oracle performance metrics. We propose new gas price oracle performance metrics consisting of three levels of analysis; accuracy of acceptance, underpricing and overpricing. Our metrics address the issues with other oracle measurement papers by providing an overall analysis of oracles in terms of acceptance, delays, and overpaying. We also apply the metrics to pre and post-EIP-1559

systems allowing us to study the effects of EIP-1559 on gas price prediction mechanisms. The first metric is the accuracy of acceptance. The metric analyzes the transactions that match an oracle's prediction and checks if the transactions make it to the block under the conditions advertised by the oracle. Second, the underpricing metric analyzes the transactions that made it to the blockchain, but the delay is longer than expected. This metric allows us to calculate the average time wasted per transaction for each oracle and tier pre and post-EIP-1559. Finally, the overpricing metric studies the transactions that make it to the block on time but at a more than-required price. The overpricing metric allows us to calculate the average amount of extra money paid for each oracle and tier. The metric shows the amount of money that users could have saved if the oracles were perfectly accurate. Based on our defined metrics, we compare the performance of gas price oracles before and after the release of EIP-1559. We collect and analyze a dataset of 44,332 Ethereum blocks, 18,742,478 transactions, and 180,779 gas price predictions collected between April 8, 2021, and April 15, 2021. This data consists of Ethereum transactions before EIP-1559. As for post-EIP-1559, we collect and analyze 58,690 Ethereum blocks, 15,429,102 transactions, and 91,061 gas price predictions sampled between February 16, 2022, and March 14, 2022. Based on our data sets we compare the accuracy of four publicly available oracle APIs (ETHGasStation [16], Etherchain [12], Anyblock [2], Web3py [30] and Blocknative [7])). Our results show that post-EIP-1559, EthGasStation produced highly accurate transaction delay times, but users consulting EthGasStation ended up overpaying the most. In general, we also observe that EIP-1559 increased the accuracy of gas price oracles in terms of delay times, gas price percentile, and probability of acceptance. On the other hand, we observe that EIP-1559 increased transaction overpricing inaccuracy and consequently increased the amount of money wasted.

We organize the rest of this chapter as follows. In Section 2, we provide an overview of Ethereum pool mechanics and discuss different types of gas price oracles and prediction

mechanisms. Section 2 focuses on related work on different gas prediction mechanisms and performance evaluation metrics. In Section 2 we describe our Ethereum node setup and data collection. Section 2 presents our comparative evaluation of gas price oracles pre and post-EIP-1559. Finally, we conclude in Section 2.

Background

Understanding the process that miners follow to generate new blocks is crucial to define our performance metrics and further analyzing oracle accuracy. EIP-1559 updated the pool mechanics and thus it is crucial to understand both pre and post-EIP-1559 to comprehend the effects of EIP-1559 on gas price predictability. Further, oracles follow different mechanisms to predict gas prices. Since EIP-1559 updated the pricing mechanism, oracles also changed their prediction mechanisms accordingly. Understanding each oracle's prediction approach allows us to formulate our performance metrics. Consequently, we determine the accuracy of the oracles based on their prediction approach.

Pool Mechanics

Miner nodes add a transaction to the blockchain after gossiping about it amongst themselves. For instance, when a miner receives a transaction, it validates it and moves it into the executable (pending) transaction pool. Miners sort the transactions in the pool by gas price. Miners try to include high-value transactions in their blocks, but may also remove transactions that have been included in blocks by other miners, or because they are replaced in the pool by higher-valued transactions.

EIP-1559 updated the pricing mechanism and consequently, miners follow a new approach to select transactions and form blocks. Post-EIP-1559 transactions set two gas price values; max fee and priority fee. Accordingly, miners check if the max fee covers the base fee proposed by the system. Afterward, the miner will sort the transactions that met

the base fee requirement by their priority fee. The priority fee plays the role of the incentive for a miner to select transactions [9].

EIP-1559 is the first major update to the first-price auction mechanism on any major blockchain [26]. EIP-1559 introduced new gas price metrics; base fee, max fee, and priority fee. The base fee is the minimum amount of gas required for a miner to consider a transaction into the next block whereas the max fee is the maximum amount of gas a user is willing to spend on a transaction [33]. For instance, the base fee could change during a transaction submission, and the max fee ensures that the transaction price meets the minimum requirement. Further, the priority fee is a fee paid by the user to incentive the miner to choose the transaction upon the next block creation. In addition to predicting the gas price for classic transactions, most oracles developed new systems to retrieve the max fee and the priority fee for EIP-1559 transactions as well. The Ethereum community expected that the new transaction fee mechanism (TFM) will decrease transaction delay [8].

Gas Price Oracles

Gas price oracles aim to predict the gas price for new transactions so that they will remain in miner pools long enough to be included in a block. Oracles generally predict prices in a number of *tiers*, for example, `instant`, `fast`, `average`, and `slow` depending on the oracle's nomenclature. In this paper, we are researching three different types of oracle prediction mechanisms; time-based, percentile-based, and probability-based oracles. Each type guarantees acceptance into the blockchain under predefined conditions.

ETHGasStation [17], Etherchain [13], and Web3.py [30] are examples of time-based oracles. They specify their tiers in terms of expected transaction acceptance delay (TAD). The oracles predict a gas price for transaction acceptance within a delay target. The delay target for transactions submitted at the `instant` price point is one or two blocks, or under 30 seconds from pool arrival time. The `fast` tier generally means within 2 minutes, `average`

5 minutes, and `slow` refers to 30 minutes [17] [13].

On the other hand, percentile-based oracles, such as Anyblock [3], predict the gas price based on a guaranteed percentile in the transaction pool. The higher the transaction percentile the more likely it will be included in a block. Percentile-based oracles specify prediction tiers in terms of accepted transaction percentile(ATP), or the percentile the transaction price achieves within the past 200 blocks of the block that accepts it. The price of transactions submitted at the `instant` tier are expected to be at the 99^{th} percentile, `fast` at the 90^{th} percentile, `average` at the 60^{th} percentile, and `slow` at the 30^{th} percentile [3].

We also study the probability-based approach. In this approach, the oracles provide a probability of acceptance with each predicted gas price. Blocknative [6], a probability-based oracle, refers to their tiers as "confidence of acceptance". The price of transactions submitted at the `instant` tier has a 99% chance of being accepted, `fast` has 90% chance, `average` has 80% chance, and `slow` has 70% chance [6].

Related Work

Defining metrics to measure oracle performance accuracy is crucial for distributed application end-users. However, not much work has been published to define oracles' performance metrics yet. Weber et al. was the first to analyze the relationship between gas price and transaction acceptance delay [31]. They showed the diminishing marginal return of higher gas prices on transaction delay. They also measured transaction failure rates and proposed transaction resubmission mechanisms based on dynamic price adjustments. Sousa et al. performed a similar analysis using Pearson correlation to argue that gas price does not strongly correlate with transaction acceptance [11]. These results may be understood in the context of transaction pool mechanics, where miners prefer higher-priced transactions for inclusion in blocks, but do not follow the strict priority.

Lacking a metric for oracle accuracy, however, their work did not investigate the Granger

causality of environmental factors on the accuracy of oracle predictions [27]. Pierro et al. extended that work to show that ETHGasStation misses the transaction acceptance delay more often than the advertised 2% of the time [1]. They also showed that gas prediction accuracy improves with more frequent predictions based on more up-to-date data.

Blockchain researchers performed several studies on the stability of gas prices after EIP-1559. In their attempt to study the stability of the Ethereum gas price ecosystem post-EIP-1599, Leonardos et al. mentioned that the introduction of the base fee indeed stabilized gas prices and guaranteed a system convergence under various conditions [24]. Leonardos et al. Lyapunov arguments to show that the Ethereum system stabilizes in most cases. Also, they explain that the system could become chaotic under extreme base fee settings [24].

Closest to our paper is the results by Liu et al., who conducted an empirical analysis of multiple user experience elements and concluded that EIP-1559 improves gas price fee estimations, decreases gas price variance in blocks, and reduces transaction delay [26]. They explain that pre-EIP-1559, users had to pay the entirety of their bids, regardless of the network congestion and block utilization. Hence, the risk of overpaying increases when the network's congestion is lower than expected. However, the new pricing mechanism allows users to set a cap on the payment. Therefore, if the base fee drops and the user pays more, the network refunds the extra gas back to the user [26]. Consequently, Liu et al. claim that this mechanism makes fee estimation easier. However, the paper does not cover priority fee estimation where users overpay miners for a transaction that could have made it with less priority fee. Further, Liu et al. measured the intra-block variance of gas prices pre and post-EIP-1559 and found that the variance decreases after the introduction of EIP-1559. The paper claims that the decrease in intra-block variance is a demonstration of the decline of gas price spikes and the stability of the Ethereum gas ecosystem after the new transaction pricing mechanism. Also, Liu et al. measured the waiting time for transactions pre and post-EIP-1559 by calculating the difference between the time the transaction first appeared

in the pool and the time the transaction first appeared in a block. The paper shows that the waiting time decreases and claims that the improvement in waiting time corresponds to better gas estimation. As for delay time, we do it differently in our analysis. Liu et al. cover the waiting time in general, however, in our paper we will focus on the extra time waited beyond what the gas price oracles predicted. Then, we measure gas price predictability and oracles performance pre and post-1559 accordingly.

Data Collection

To help interpret the results, we briefly describe our data collection process and data structure. To analyze oracle's performance and the effect of EIP-1559 on gas price predictability, we collect data pre and post-EIP-1559. Ethereum released EIP-1559 on August 2021, hence, we collected two sets of data; one dataset before August 2021 and another one after the go-live of the improvement proposal.

Pre-EIP-1559, we collected a total of 44,332 Ethereum blocks and 18,742,478 transactions, and 180,779 gas price predictions collected between April 8, 2021, and April 15, 2021. Post-EIP-1559, we collected 58,690 Ethereum blocks, 15,429,102 transactions, and 91,061 gas price predictions between February 16, 2022, and March 14, 2022.

Post-EIP-1559, we did not collect the data continuously during the mentioned dates, however, we collected samples between mid-February and mid-March to ensure that we collect transactions during different states of the network. For example, we collected samples during high network utilization and low network utilization. For pre-EIP-1559 transactions, we collect gas prices and use them for future comparison with oracles predictions. However, post-EIP-1559, we collect two types of gas fees; base, max, and priority fees, and use them to analyze oracle prediction performance.

We maintain a local full Geth node instance to ensure that our network is up to date and contains the complete blockchain data. Hence, we could further analyze the data and

draw conclusions.

Block and Transaction Data

To obtain block and transaction pool data we deployed a local Geth node and enabled the **eth** and **txpool** namespaces of Geth JSON-RPC API [19, 20, 21]. We query the Geth node three times a second using the `eth-blockNumer` and `txpool-content` methods and record the data in the **Block** and **Transaction** tables shown in Table 2.1.

Oracles Data

We collected data from five oracles: ETHGasStation, Etherchain, Anyblock, Web3.py, and Blocknative. We access ETHGasStation, Etherchain, Anyblock, and Blocknative through their REST APIs [2, 12, 16] and Web3.py through standalone tool [30] and parse the gas price for each tier. Blocknative was excluded in our pre-1559 analysis since the tool does not support classic Ethereum transactions. On the other hand, Web3.py was excluded from our post-EIP-1559 analysis since the tool is not compatible with post-EIP transactions. We query each endpoint when the Geth node detects a new block (around 13 sec on average) and record the data in the **Oracle** table as shown in Table 2.1.

Environment Data

Finally, we collect the Ethereum environment data, specifically, the price of Ethereum in USD and the network hash rate. We obtain these from Etherscan's API [14]. We record the data in the **Environment** table in Table 2.1.

Table	Field	Source	Description
Block	`no`	Geth API	Block number
	`hash`	Geth API	Hash of block fields
	`ts`	Geth API	Timestamp of block creation
	`tx_list`	Geth API	List of transaction hashes in the block
Transaction	`hash`	Geth API	The hash of the transaction
	`block_hash`	Geth API	Accepted transactions block hash
	`time_in`	Geth API	Time the transaction entered the pool
	`time_out`	Geth API	The time the transaction left the pool
	`gas_price`	Geth API	The gas price of a classic transaction
	`max_fee`	Geth API	The maximum gas price
	`priority_fee`	Geth API	The maximum priority fee
Oracle	`name`	Oracle API	The name of the oracle
	`tier`	Oracle API	The name of a price prediction tier
	`max_fee`	Oracle API	Gas price predicted by the oracle
	`priority_fee`	Oracle API	Priority fee predicted by the oracle
	`ts`	Query script	Time of prediction
Environment	`net_hash_rate`	Etherscan	The combined hash rate of miners
	`eth_usd`	Etherscan	The price of Ethereum in USD
	`ts`	Query script	The time at which queries for the data

Table 2.1: Database schema of the collected data.

Evaluation

Number of Transactions

To illustrate the integration of data in the `Oracle` and `Transaction` tables, we compute the relative transaction counts using price predictions from the oracles and tiers pre and post-release of EIP-1559. It is not possible to know whether a transaction submitted at a price predicted by an oracle was submitted after consulting the oracle, but we follow the assumption of Pierro et al.that it might as well have been [1].

Figure 2.1 shows the number of transactions submitted at the price predicted by each oracle and tier. The figure is split into two sub-figures presenting the data pre and post-EIP-1559. The x-axis lists the oracles, while the y-axis shows the different prediction tiers. The shade of each cell represents the number of transactions submitted at the gas price predicted by the oracle for the tier at transaction submission time. For example, before EIP-1559, the number of transactions for Anyblock `slow` is given by:

```
SELECT count(*)
FROM Oracle AS O, Transaction AS T
WHERE O.name=‘anyblock‘ AND O.tier=‘slow‘ AND
O.gas_price=T.gas_price AND
T.ts>O.pred_ts AND T.ts<O.next_pred_ts
```

The query ensures that the matched transactions happened between the oracle's current and next prediction window.

On the other hand, since Ethereum updated its gas price methodology and introduced max and priority fees, the number of matched transactions is obtained differently. For example, the number of transactions matching Blocknative's `fast` is given by:

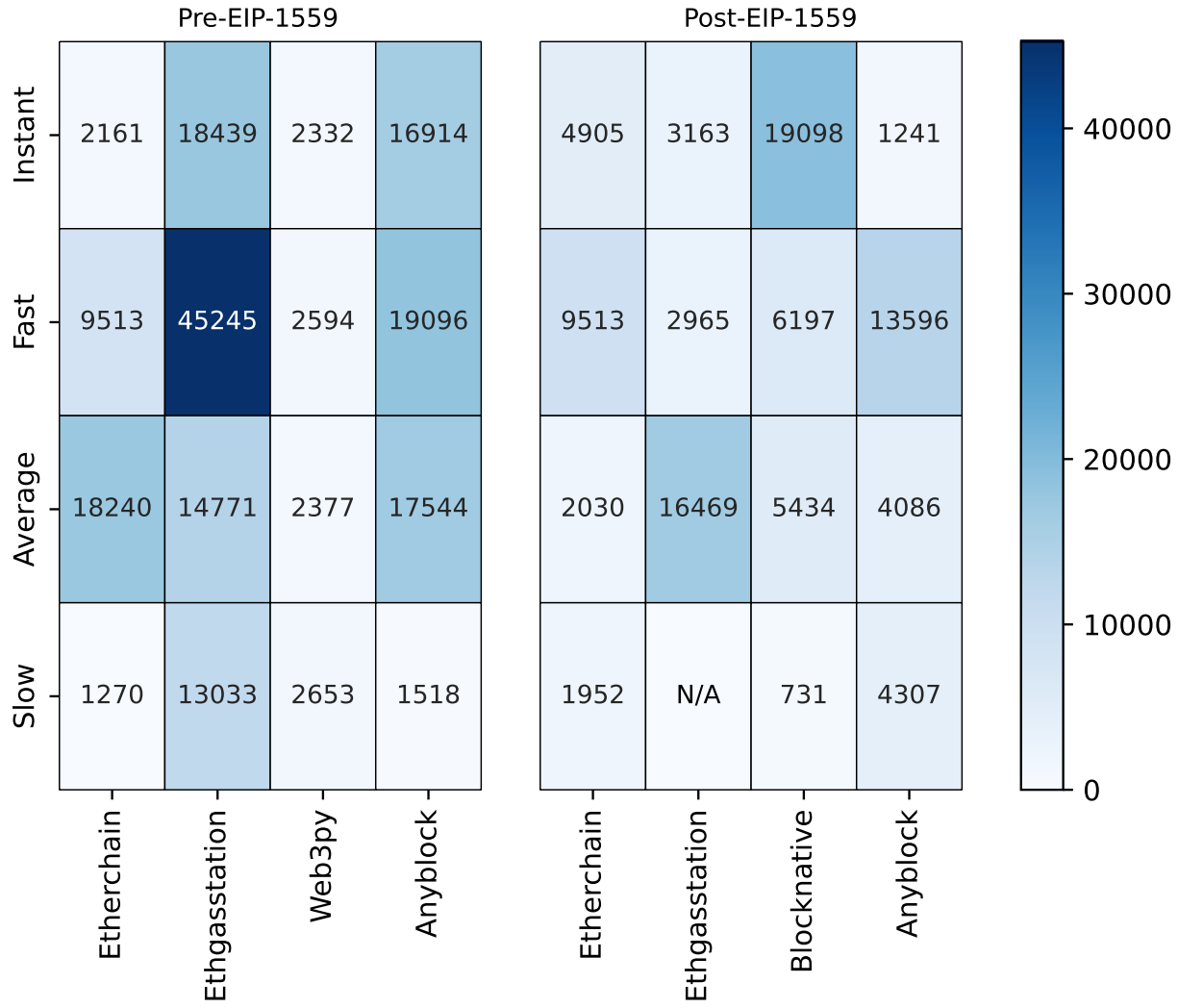

Figure 2.1: The number of transactions that match a gas price prediction for each oracle and tier.

```
SELECT count(*)
FROM Oracle AS O, Transaction AS T
WHERE O.name='blocknative' AND O.tier='fast' AND
O.max_fee=T.max_fee AND O.priority_fee=T.priority_fee AND
T.ts>O.pred_ts AND T.ts<O.next_pred_ts
```

We observe that EthGasStation's fast tier was consulted the most before EIP-1559 and Blocknative's instant tier was consulted the most after the proposal. The data we collected does not corroborate the results presented by Pierro et al., who claims that users choose the `fast` and `average` tiers of EthGasStation only 6.3% of the time [1]. We observe that EthGasStation was consulted significantly more before EIP-1559 for the instant and fast tiers. Specifically, we observe that users choose the `average` 16.1% of the times before EIP-1559. Whereas, users utilize the same tier 72.8% of the time after EIP-1559. We suspect that the difference is related to EthGasStation's decision to remove the slow tier in their 1559-compatible release. Consequently, users that want to save up on transaction prices tend to switch to the average tier as the new cheapest option. After EIP-1559, users tend to consult with Blocknative's instant tier the most. Remarkably, Blocknative's `instant` tier accounts for 60.7% of transactions matching Blocknative's predictions. On the other hand, the rest of the oracles' `instant` accounts for less than 16% of their total prediction matches.

Figure 2.1 shows that we have a large number of transactions matching oracles prediction and hence we can perform further analysis.

Gas Oracle Accuracy

To analyze and compare oracle accuracy, we first need to come up with its precise definition. Different oracles predict gas prices with respect to transaction delay,

e.g. EthGasStation, Etherchain, and Web3.py, or the percentile of the accepted transaction price, e.g. Anyblock, or the probability of acceptance, e.g. Blocknative and Anyblock. Depending on the type of oracle we consider, we can measure its accuracy in terms of transaction acceptance delay (TAD), accepted transaction percentile (ATP), and probability of acceptance (POA).

We measure TAD as the difference between transaction submission time and the creation time of the block containing the transaction. For an oracle o and tier t we can query for a TAD distribution $D_{o,t}$ as

SELECT B.ts - T.sub_ts

FROM Block AS B, Transaction AS T, Oracle AS O

WHERE T.hash in B.tx_list AND

O.name=o AND O.tier=t AND

O.gas_price=T.gas_price

Figure 2.2 shows the TAD for transactions in Etherchain's `instant` tier. The y-axis in Figure 2.2 marks the transaction acceptance delay in seconds. The x-axis marks the number of transactions accepted at the predicted price sorted by their respective TAD values. The solid blue lines show the TAD values, while the dashed red lines show the target TAD values for Etherchain's instant tier. The dashed red line is set at 30 seconds as defined by Etherchain [13]. We observe that TAD values fall above and below the target tier value. When the TAD line lies above the target line the proposed gas price is too low and transactions take longer than the target to make it into a block. This is the case of a gas oracle *underpricing* its prediction. When the TAD line lies below the target line the oracle-proposed gas price is too high, meaning that transactions submitted at a lower price could have still made it into a block within the target tier delay. This is the case of a gas oracle *overpricing* its prediction.

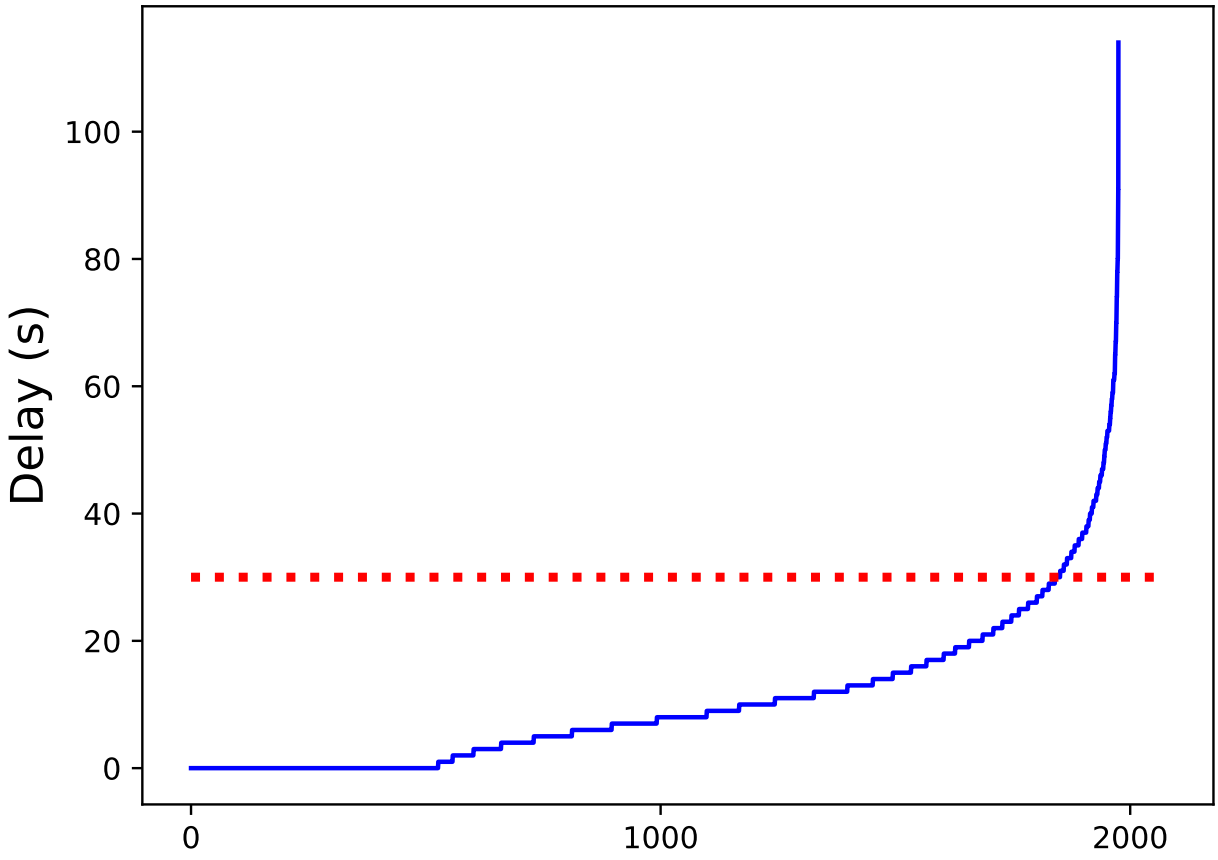

Figure 2.2: Example of TAD distributions vs transaction count for Etherchain `instant`.

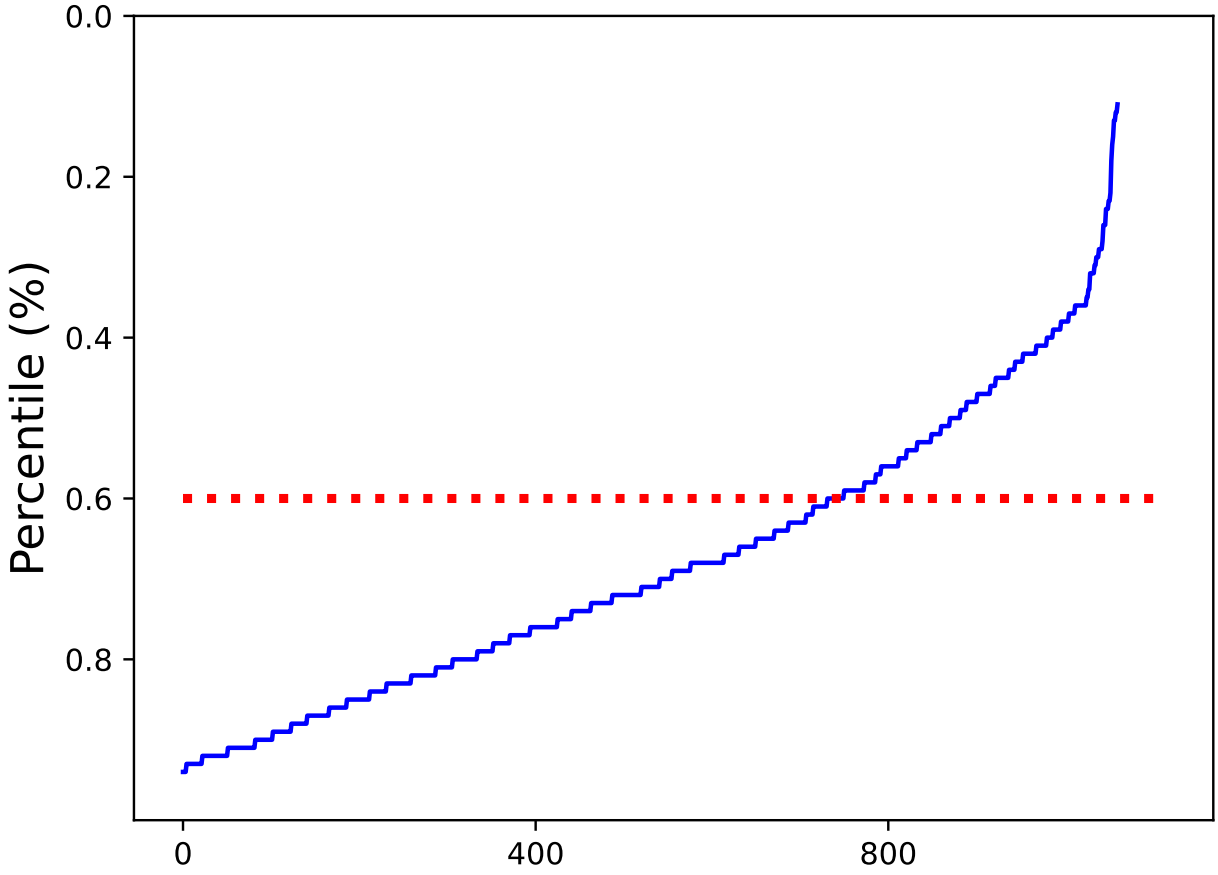

Figure 2.3: Example of ATP distributions vs transaction count for Anyblock `average`.

The ATP query is more complex than the TAD query, but it computes the percentile of each accepted transaction gas price with respect to the gas prices of the accepted transactions in the 200 preceding blocks, including the transaction's block. We base the ATP calculation on 200 blocks since that is the time horizon of input data to both ETHGasStation and Anyblock [3, 15, 27].

Figure 2.3 shows the ATP distributions for transactions in Anyblock's `average` tier. The y-axis (reversed) in Figure 2.3 marks the percentile of the transaction's gas price with respect to all pending transactions in the pool. The x-axis marks the transactions accepted at the

predicted price sorted by ATP values. The solid blue lines show the ATP values, while the dashed red lines show the target percentile indicated by Anyblock's average tier. Anyblock's average tier prediction guarantees that the transaction will be in the 60^{th} percentile [3] and hence the dashed red line is set to 0.6. Similar to TAD, when the ATP plot lies above the target line, the oracle-proposed gas price is too low and transactions place at a lower than promised percentile a.k.a. *underpricing*. On the other hand, when the ATP plot lies below the target line the oracle-proposed gas price is too high, meaning that transactions submitted at a lower price could have attained promised percentile. This means that transaction is *overpriced*.

We also measure the probability of acceptance (POA) as the ratio of accepted transactions to transactions that matched the gas price prediction. To find the probability of acceptance, we first find a group of transactions matching the same prediction. Then, we check the number of transactions in the selected group that made it to the next block. Lastly, we calculate the percentage of accepted transactions to matched transactions. For example, if 100 transactions consulted an oracle, and only 60 of those transactions made it to the next block, then the probability of acceptance at the predicted price is 60%. After calculating the probability of acceptance of all matched transactions, we find the mean of our all instances.

Figure 2.4 shows the probability of acceptance distributions for transactions matching Blocknative's `fast` tier. The y-axis marks the probability of acceptance in percent. The x-axis marks the group of transactions matching the predicted price at each studied instance. We sort the instances on the x-axis by the calculated probability of acceptance. The x-axis marks the transactions recorded at each accepted sorted by their POA values. The solid blue lines show the POA values, while the dashed red lines show the target probability of acceptance indicated by Blocknative's fast tier. Each point in the blue line corresponds to an instance where a group of transactions matched Blocknative's fast tier. In this study,

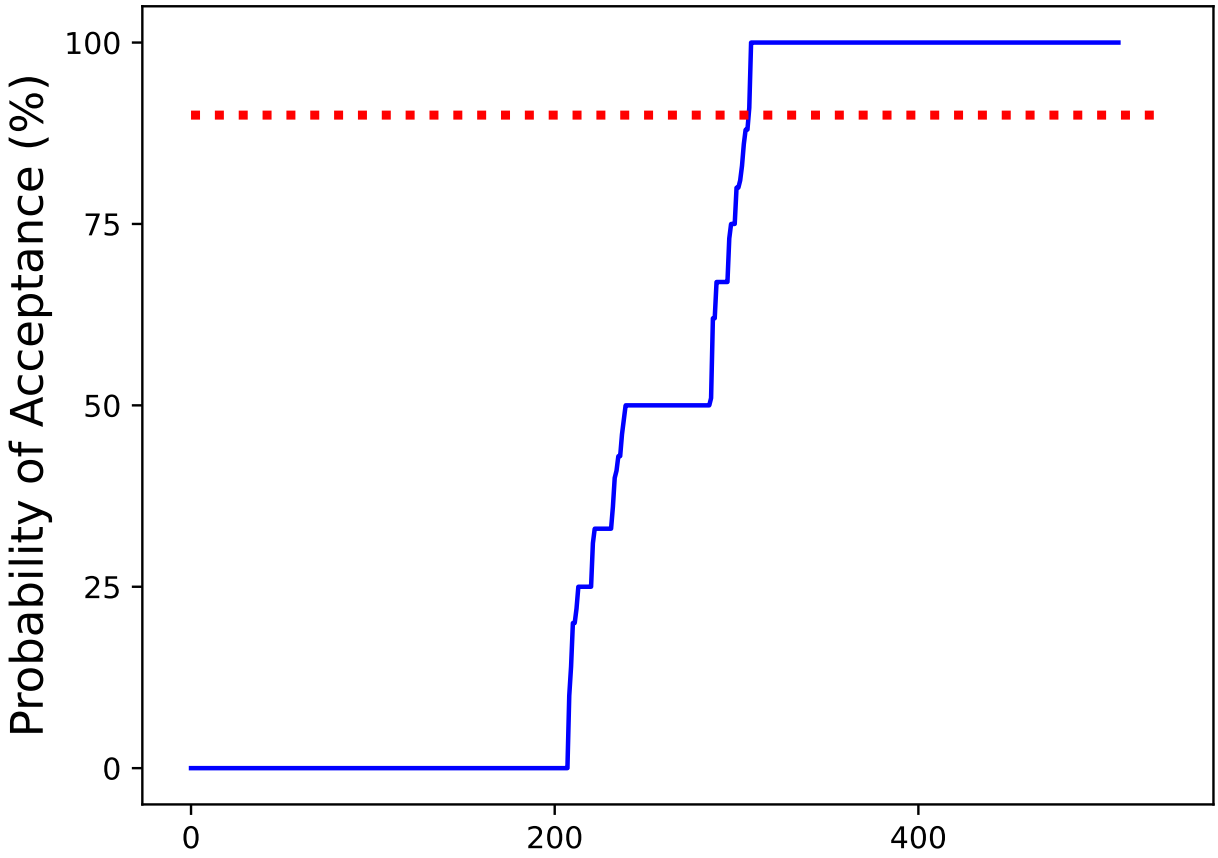

Figure 2.4: Example of POA distributions vs transaction count for Blocknative `fast`.

we analyzed 500 instances. Blocknative indicates that if a user submits a transaction with a gas price matching their fast tier prediction, there is a 90% chance that a miner selects the transaction in the next block.[6] Therefore, when the plot lies above the target line, the oracle-proposed gas price is too high and transactions place at a higher than promised probability of acceptance a.k.a. *overpricing*. Similarly, when the plot lies below the red line, the transaction is underpriced and the probability of acceptance is below what the oracle promised.

Based on the examples of underpricing and overpricing predictions illustrated in Figure 2.2, Figure 2.3, and Figure 2.4, we propose to compute oracle accuracy as follows.

- First, we focus on comparing the number and types of mispredictions. To compare oracle and tier combinations in spite of differing counts, we use proportions. Recall that given a binary predicate P, the *Iverson bracket* $[P]$ is 1 when P is true and 0 otherwise. For the set of oracles O and tiers T, we have an observed distribution of TAD values $D_{o,t}$ and the target TAD $r_{o,t}$ for $o \in O$ and $t \in T$. We compute the proportion of transactions suffering from underpricing and overpricing, respectively

$$p^-_{o,t} = \frac{\sum_{d \in D_{o,t}} [r_{o,t} < d]}{|D_{o,t}|}, \quad p^+_{o,t} = \frac{\sum_{d \in D_{o,t}} [r_{o,t} > d]}{|D_{o,t}|}.$$

  Pierro et al. [1, 27] noted that gas price oracles report a margin of error of 2%. We were not able to verify that number in oracle documentation. As such, we consider any transaction whose TAD differs from the target as suffering from either underpricing or overpricing.

- Next, we focus on magnitudes. To compare TAD values across the different oracles and tiers, we normalize all values to $[0, 1]$ using min-max rescaling:

$$\widehat{D}_{o,t} = \frac{D_{o,t} - \min_{o' \in O, t' \in T} D_{o',t'}}{\max_{o' \in O, t' \in T} D_{o',t'} - \min_{o' \in O, t' \in T} D_{o',t'}}$$

Similarly, for target transaction delay:

$$\widehat{r}_{o,t} = \frac{r_{o,t} - \min_{o' \in O, t' \in T} D_{o',t'}}{\max_{o' \in O, t' \in T} D_{o',t'} - \min_{o' \in O, t' \in T} D_{o',t'}}$$

- We compute the mean underpricing and overpricing magnitude

$$m^{-}_{o,t} = \frac{\sum_{d \in \widehat{D}_{o,t}} (d - \widehat{r}_{o,t})[\widehat{r}_{o,t} < d]}{\sum_{d \in \widehat{D}_{o,t}} [\widehat{r}_{o,t} < d]},$$

$$m^{+}_{o,t} = \frac{\sum_{d \in \widehat{D}_{o,t}} (\widehat{r}_{o,t} - d)[\widehat{r}_{o,t} > d]}{\sum_{d \in \widehat{D}_{o,t}} [\widehat{r}_{o,t} > d]}$$

  as the mean difference between scaled transaction delay and scaled target delay.

- Finally, we compute oracle accuracy, or rather its *inaccuracy* as the mean, normalized area above and below the target delay. The underpricing inaccuracy is $i^{-}_{o,t} = p^{-}_{o,t} \times m^{-}_{o,t}$ and overpricing inaccuracy $i^{+}_{o,t} = p^{+}_{o,t} \times m^{+}_{o,t}$ for each oracle and tier. The underpricing and overpricing inaccuracy allow us to understand how badly oracles miss their tier TAD targets during our observation period. The inaccuracy values increase when more transactions miss tier TAD, or when fewer do so, but more significantly.

- To compute oracle inaccuracy based on ATP and POA, we simply replace TAD with ATP or POA in the above steps.

Figure 2.5 and Figure 2.6 show the underpricing and overpricing inaccuracy for the different oracles and tiers pre and post-the release of EIP-1559. The x-axis lists the oracles and the y-axis the tiers. The shade of each cell shows the calculated inaccuracy, $i^{-}_{o,t}$ in Figure 2.5 and $i^{+}_{o,t}$ in Figure 2.6.

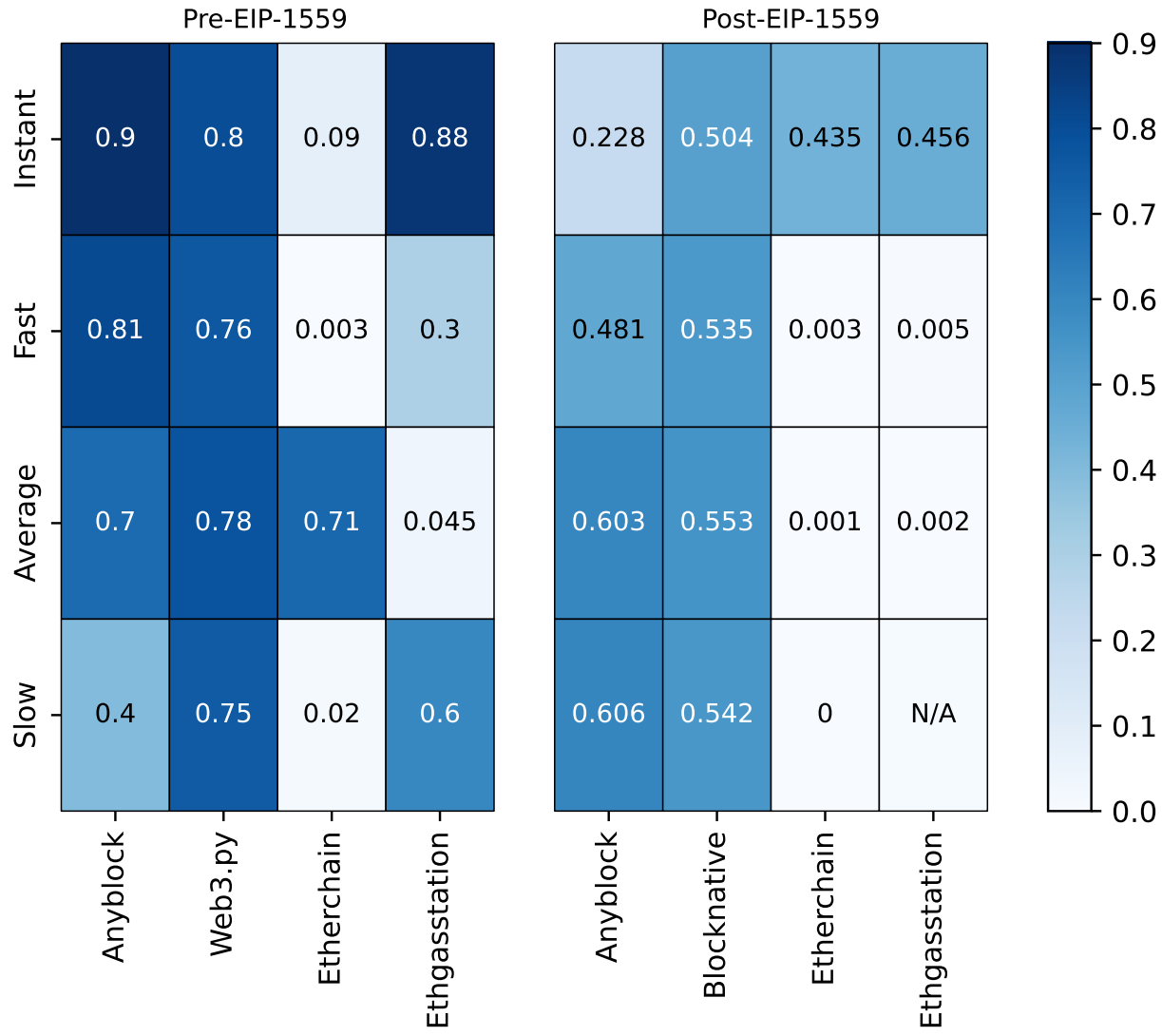

Figure 2.5: Underpricing inaccuracy $i^{-}_{o,t}$.

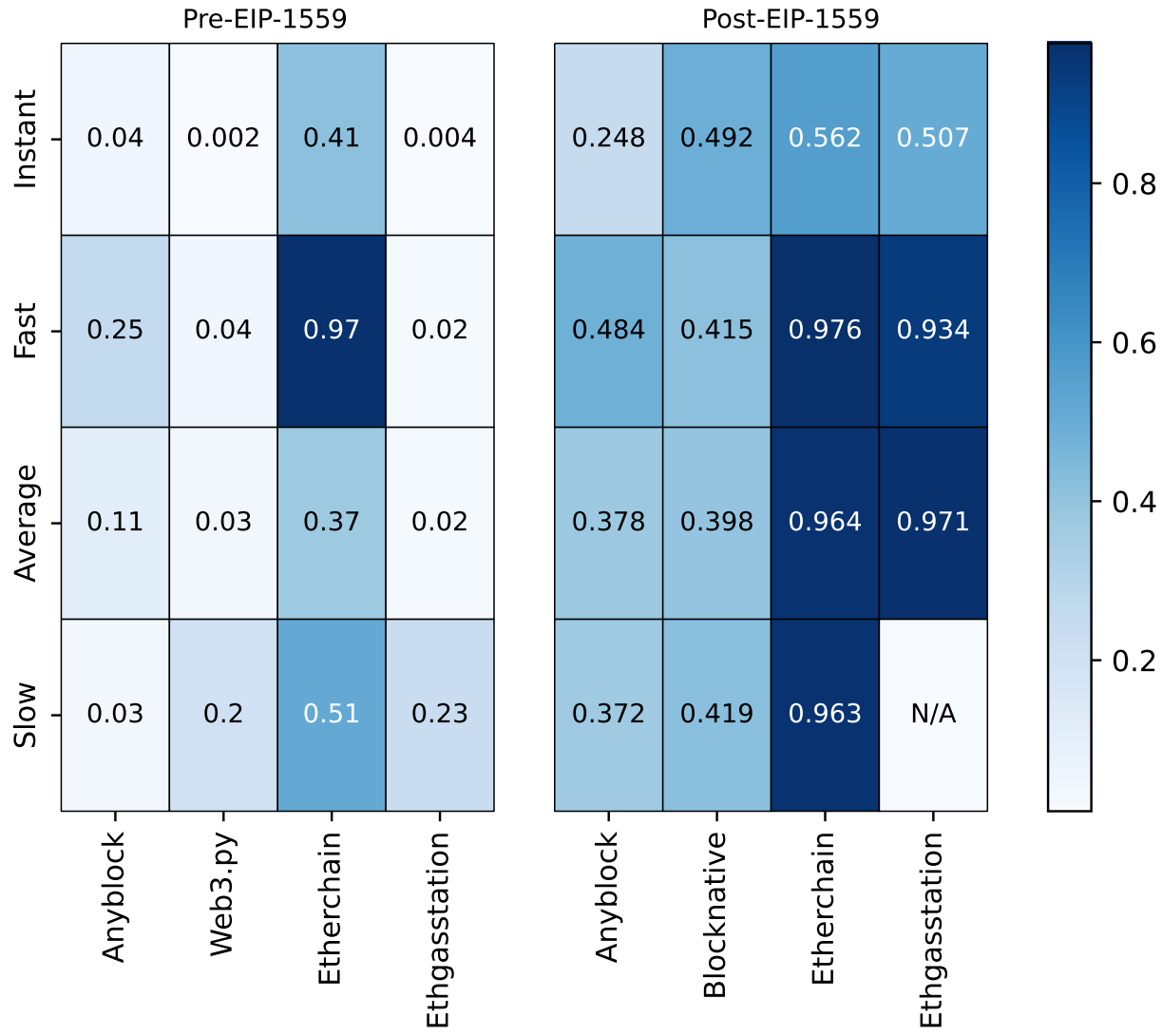

Figure 2.6: Overpricing inaccuracy $i^{-}_{o,t}$.

Notable is the absence of EthGasStation's `slow` tier after the release of EIP-1559 as shown by the N/A values. Further, Figure 2.5 indicates that pre and post-EIP-1559, Etherchain underprices transactions the least but overprices transactions the most. We suspect that Etherchain increases its predicted prices to insure the acceptance of the transaction in the promised time interval resulting in short delays and more than required payments. EthGasStation's results are very similar to those of Etherchain. On the other hand, we observe that Anyblock tends to do the opposite. Pre and post-EIP-1559, Anyblock's underpricing inaccuracy are higher than most of the oracles. On the other hand, Anyblock's overpricing inaccuracy is lower than the rest of the oracles. Thus, Anyblock tends to predict lower than required gas prices resulting in longer waiting times but the user avoids paying extra gas prices. We analyzed Web3.py pre-EIP-1559 and observed that the oracle highly underprices its predicted transactions. We also studied Blocknative post-EIP-1559 and observed that it is close to Anyblock's post-EIP results. Lastly, we observe that oracles with similar prediction mechanisms such as TAD, ATP, and POA tend to have close results.

In Figure 2.5, before EIP-1559, the tiers generally show a greater degree of underpricing inaccuracy as they miss the target TAD and ATP more often and to the greatest extent. On the other hand, we observe a significant improvement in Anyblock's and EthGasStation's instant tier where the inaccuracy dropped by 0.67 and 0.42 respectively. On the other hand, we observe that Etherchain's instant tier inaccuracy increased by 0.34. While the `average` and `slow` predictions generally make their TAD and ATP targets more often, the notable exceptions are Anyblock `slow` and Web3.py `average`.

We also observe in Figure 2.6 that, comparatively, overpricing is less of an issue before EIP-1559 with the exception being Etherchain's `fast` tier. However, following the release of EIP-1559, overpricing inaccuracy is remarkably higher. Transactions could have made their target TAD, ATP, and POA at a lower price. For instance, we observe that all tiers of Anyblock, Etherchain, and EthGasStation are more overpriced after EIP-1559. Recall

we remove Web3.py from the post-1559 analysis and include Blocknative in the post-1559 analysis only. Therefore, we cannot compare the oracles pre and post-protocol but we present their values for performance measurement. However, Web3.py and Blocknative have a distribution similar to the rest of the oracles in their respective analyses.

Time Wasted In this subsection, we represent the underpriced transactions as to the time wasted. We analyze the duration of time spent by the user waiting beyond the oracle's promised acceptance time. After Submission, transactions reach a pool that miners pick from to construct blocks. Whenever an oracle forecasts an acceptance time, a transaction should not wait in the pool beyond that deadline. For time-based oracles, we calculate the time wasted for underpriced transactions as the difference between the actual acceptance time and the expected acceptance time. However, for percentile-based and probability-based oracles, the expected time of acceptance is the creation time of the next block, so, we found the block number when the transaction reaches the pool and then check to see if the transaction made it to the next created block.

In Figure 2.7, we present the time wasted for each oracle and tier pre and post-EIP-1559. On the x-axis, we group the tiers by oracles and separate oracles by dashed lines. On the y-axis, we used a box plot to demonstrate the distribution of time wasted across underpriced transactions in seconds. Also, we highlight the mean of each distribution for comparison. For instance, EthGasStation's instant box-plot mean decreased from 28 seconds before EIP-1559 to 25 seconds per transaction after EIP-1559; this is the lowest time wasted amongst oracles after EIP-1559. On the other hand, EthGasStation's average box-plot mean increased from 16 seconds before EIP-1559 to 8 minutes per transaction after EIP-1559; this is the highest time wasted amongst oracles after EIP-1559. Both time-based oracles, Etherchain and EthGasStation, wasted less time before EIP-1559. EIP-1559 states that the base fee will increase and decrease by up to 12.5% after blocks are more than 50% full [9]. Consequently,

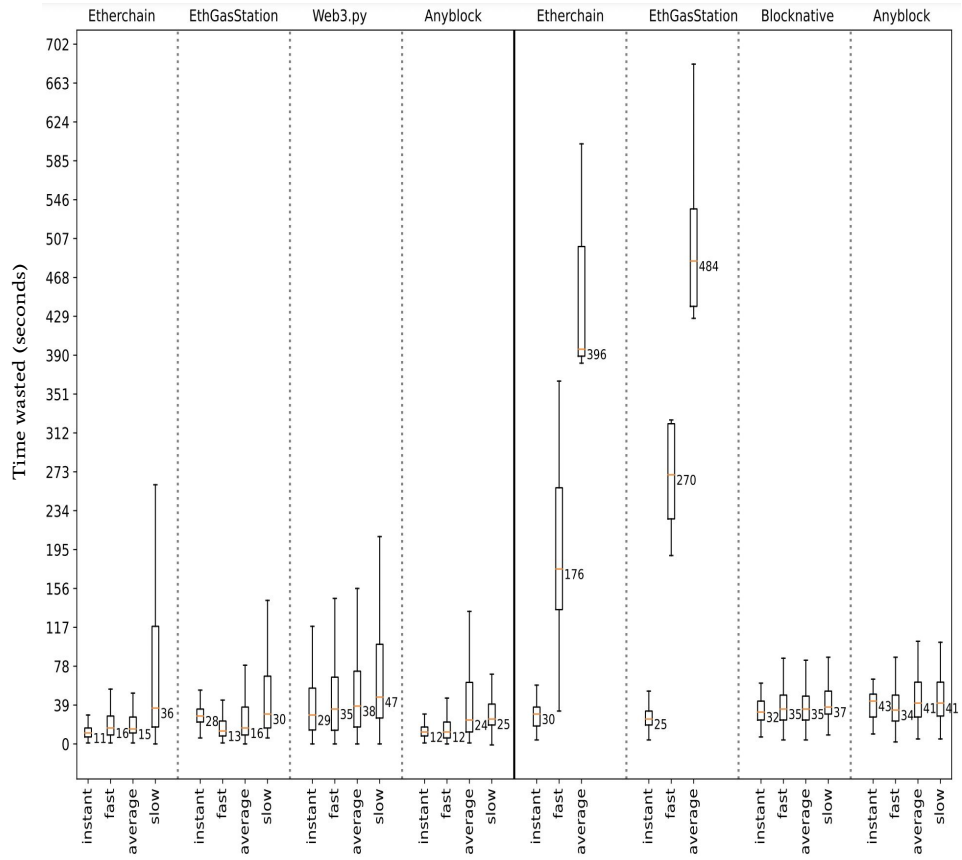

Figure 2.7: Gas price oracles time wasted in seconds.

time-based oracles can predict gas prices for the next few blocks more accurately. For example, if the current block is full, then the next gas price will increase by at least 12.5% giving the oracles a basis for prediction. However, due to constant fluctuation in block utilization, it becomes imprecise to predict further into the future. Therefore, time-based oracles tend to waste more time when predicting slower tiers post-EIP-1559. EIP-1559 introduced *base fee* which is the minimum gas price required to make it to the next block. Consequently, the user is now aware of the minimum price to get into the blockchain. Hence, overpricing inaccuracy is decreased. Although overpricing occurs less post-EIP-1559, we observe that when it happens, the average time wasted is higher pre-EIP-1559. Moreover, we noticed that post-EIP-1559, Anyblock, and Blocknative have a close distribution amongst their tiers. On average, Anyblock transactions wait for approximately three additional blocks to be accepted while Blocknative transactions wait for two blocks. Lastly, we observe that, after EIP-1559, Blocknative predictions waste time the least whereas EthGasStation tends to waste the most.

Money Wasted In this subsection, we analyze the effect of overpriced transactions on the extra price paid for transactions under the oracle's promised conditions. As mentioned by Laurent et al. [23], for a transaction to enter the pool, the user must pay a gas price. The higher the price, the higher the chances of acceptance into the next block. However, some of these accepted transactions are potentially overpriced and could have made it with less money. Therefore, we calculate the money that could have been saved under the same conditions. To calculate the money wasted, we find the block containing the transaction and then scan through the rest of the block to get the transaction with the minimum amount of gas spent. We then find the difference between the minimum amount and the actual amount spent in Gwei. Then, we calculate the money wasted as the difference between the gas price paid versus the minimum gas price needed for that block. Notably, we perform the

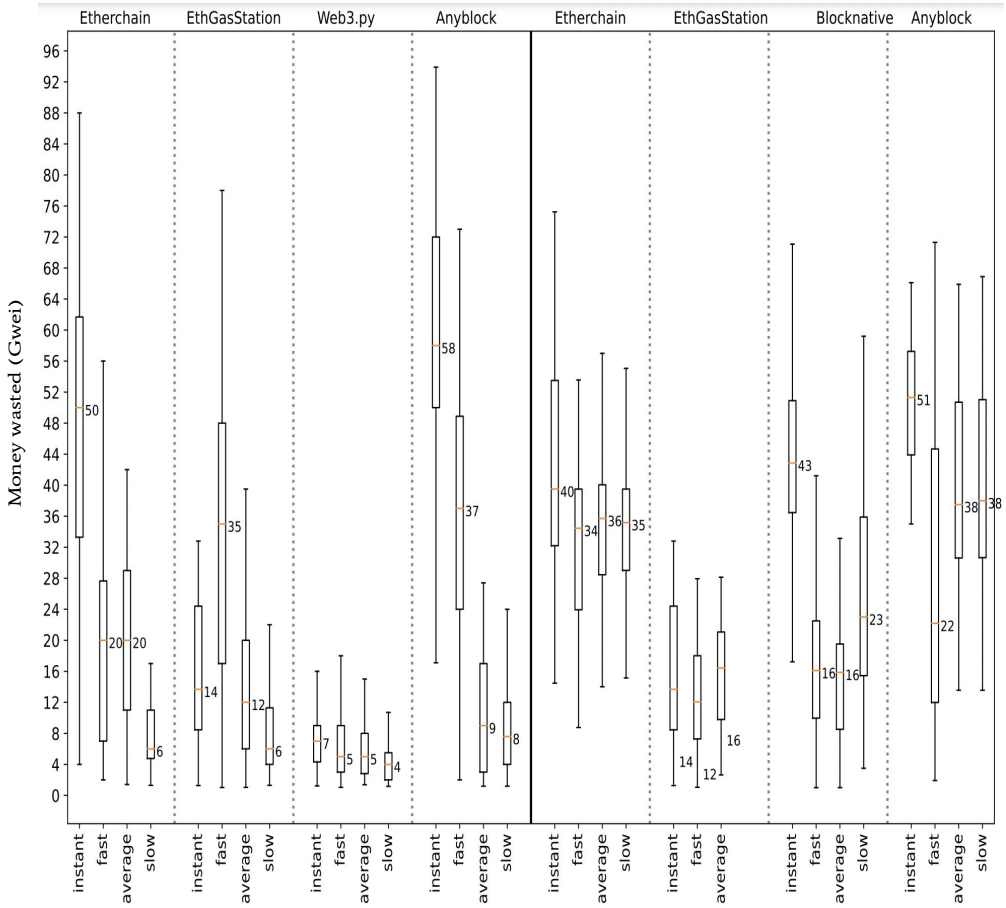

Figure 2.8: Gas price oracles money wasted in Gwei.

comparison for other transactions that reached the pool around the same time and accepted in the same block.

In Figure 2.8, we present the money wasted for each tier in Gwei for oracles pre and post-EIP-1559. On the x-axis, we group the tiers by oracles and separate them by dashed lines. On the y-axis, we used a box plot to demonstrate the distribution of money wasted by overpriced transactions. Also, we highlighted the mean of each distribution for comparison. We noticed that Anyblock's instant tier is the highest amongst the oracles in terms of

money wasted. Before EIP-1559, Anyblock's instant tier wasted an average of 58 Gwei per transaction. This average dropped to 51 Gwei after the release of EIP-1559, but still the highest across the oracles. Also, Web3.py wasted the least amount of money before EIP-1559, but we could not analyze its behavior afterward due to the unavailability of predictions post-EIP-1559. Furthermore, Ethgasstation's fast-tier transactions pay the least extra price, post-EIP-1559, with an average of 12 Gwei wasted per transaction. It is also worth mentioning that Etherchain's instant tier and EthGasStation's fast tier decreased money wasted per transaction by 10 Gwei and 23 Gwei respectively after the introduction of the protocol. On the other hand, the rest of Etherchain's and EthGasStation's tiers wasted more money post-EIP-1559.

Block Utilization By design, the probability of a transaction getting accepted in the next block increases with the gas price paid. Namely, the faster the tier, the more expensive a transaction becomes. Ergo, with higher payments, users could potentially be overpaying. Intuitively, the instant tier wastes more money than the fast, the fast tier wastes more money than the average, and so on. However, in Figure 2.8 we noticed that Blocknative's and Anyblock's slow tiers waste more money than their fast tier. We studied this edge scenario closely and realized that the block utilization when a user selects the fast tier was higher than the utilization when the slow tier is selected. We concluded that in times of high blockchain traffic, the fee to get accepted into the block increases rapidly. In response, end-users increase their payments to boost their chances by attempting to reach the minimum gas price needed. Hence, the fast tiers end up wasting less money. On the contrary, users select slower tiers in times of low traffic where blocks are underutilized. As per EIP-1559, when a block is not highly utilized, the gas price automatically drops. Hence, users pay extra for a transaction that could potentially make it to the block with a lower price.

We demonstrate in Figure 2.9 the block utilization when a miner selects a transaction

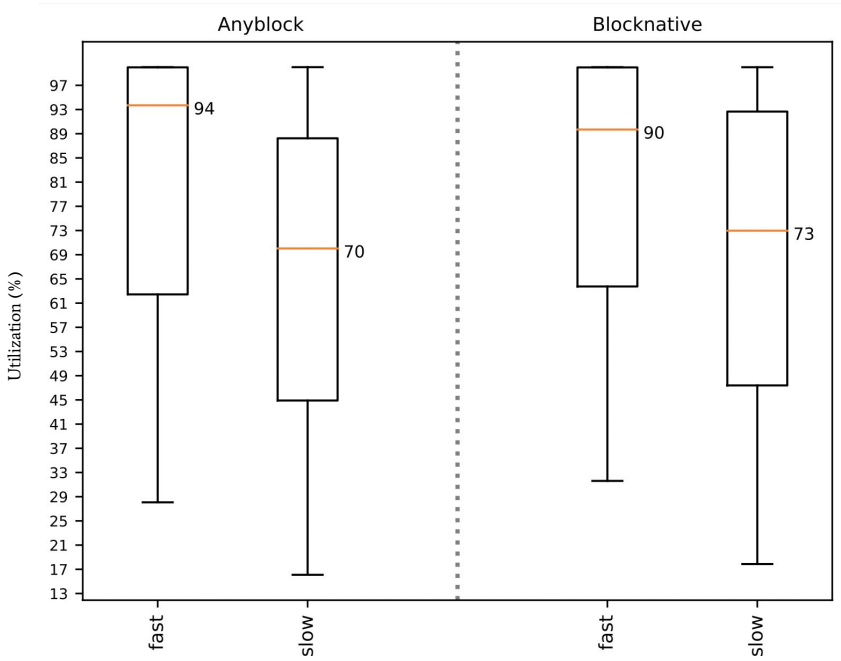

Figure 2.9: Examples of the percentage of block utilization for overpriced transactions.

in the fast and slow tiers of Anyblock and Blocknative after the release of EIP-1559. On the x-axis, we group the fast and slow tiers by the oracles and separate them by a dashed line. On the y-axis, we used a box plot to demonstrate the distribution of the block utilization of the overpriced transactions. Also, we highlighted the mean of each distribution for comparison. For Anyblock, we can see that the average block utilization for the fast tier is 94% whereas in the slow tier the average is 70%. Similarly, Blocknative's average block utilization for the fast tier is 90% whereas the slow tier is 73%. We observe that the fast tiers are full. Therefore, transactions will less likely be overpriced since the money paid is needed to make it into the next block. On the other hand, the slow tiers were underutilized during submission. Hence, users pay more than sufficient prices to get into the next block.

Transaction Acceptance Rates

So, to measure the accuracy of acceptance, we collect transactions matching an oracle's prediction. Then, we find the block where the oracle expects the transaction to be accepted and compare it to the block where the transaction ended up.

In Figure 2.10, we present the accuracy of acceptance of the oracles pre and post-EIP-1559. We group the bar plots into regions describing each oracle's tier. On the x-axis, we list the gas price prediction tiers. On the y-axis, we show the ratio of calculated accuracy with respect to the accuracy advertised by the oracle. We calculate the accuracy as the ratio of the expected accuracy versus the calculated accuracy. In general, we observe an overall improvement in the accuracy of acceptance after EIP-1559. Etherchain, EthGasStation, and Anyblock's accuracy of acceptance increased across all tiers except for Etherchain's instant tier. Also, we observe that the overpricing mentioned in Figure 2.10 reflects the accuracy of acceptance. For instance, Etherchain and EthGasStation's high accuracy of acceptance after EIP-1559 is a result of increasing gas price predictions, a.k.a overpricing. For instance, Anyblock's instant tier expected accuracy is 99%, but the calculated accuracy turned out

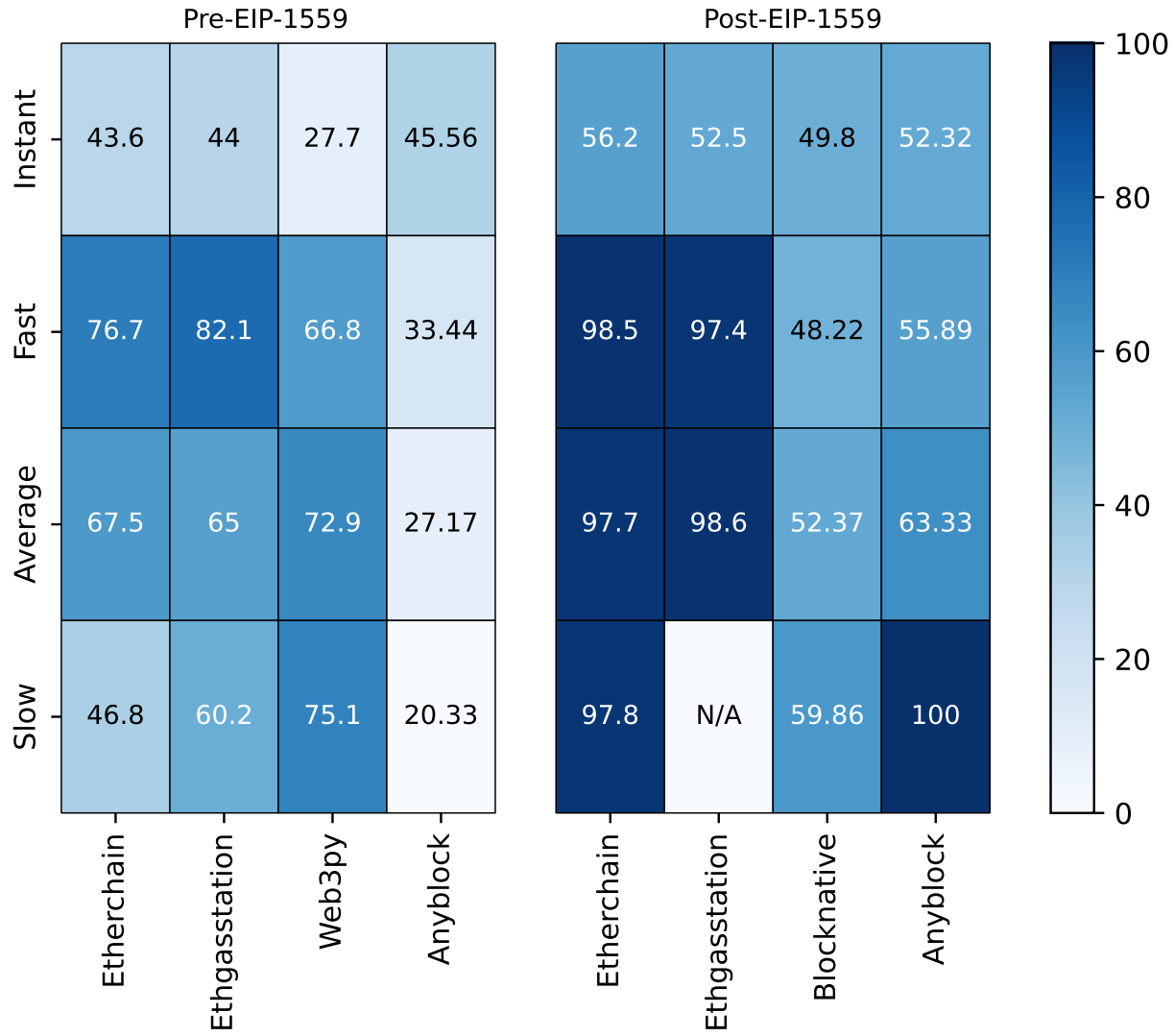

Figure 2.10: Gas price oracles accuracy of acceptance.

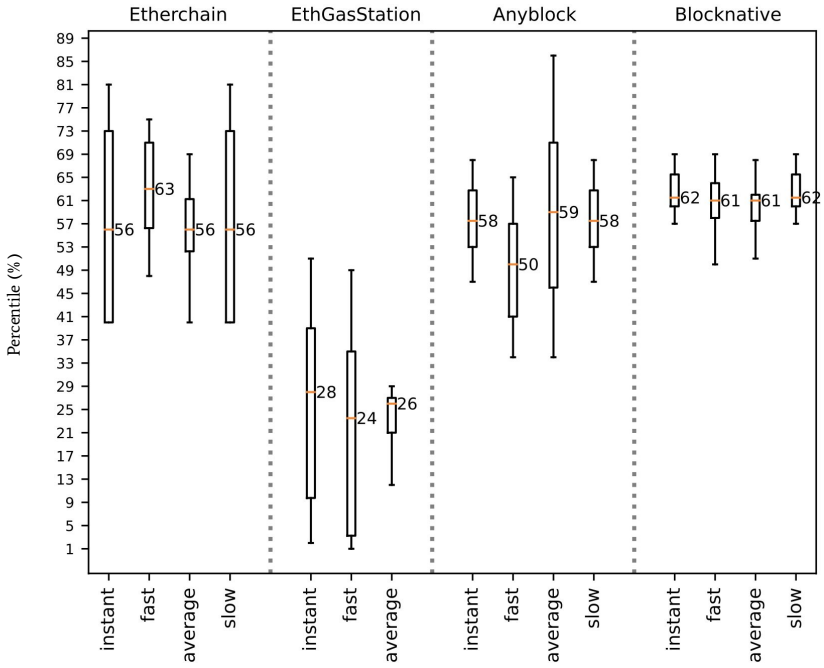

Figure 2.11: Example of gas price percentile of transactions that were removed from the pending pool.

to be 51.8%. Accordingly, the oracle is 52% accurate. On the other hand, the slow tier expected accuracy is 30%. We found out that 37.5% of the slow transactions are accurate. Ergo, the results exceed expectations with slow-tier transactions making this tier the most accurate across oracles.

Furthermore, we noticed that some transactions did not make it to any future blocks. We analyzed those transactions and came to the following observations: Blocks are almost full and transactions are continuously falling in a low percentile group.

In Figure 2.11, we present the percentile of the gas paid with respect to other pending transactions in the pool. We group the box plots into four regions representing the oracles. On the x-axis, we list the gas price prediction tiers. On the y-axis, we show the percentile of

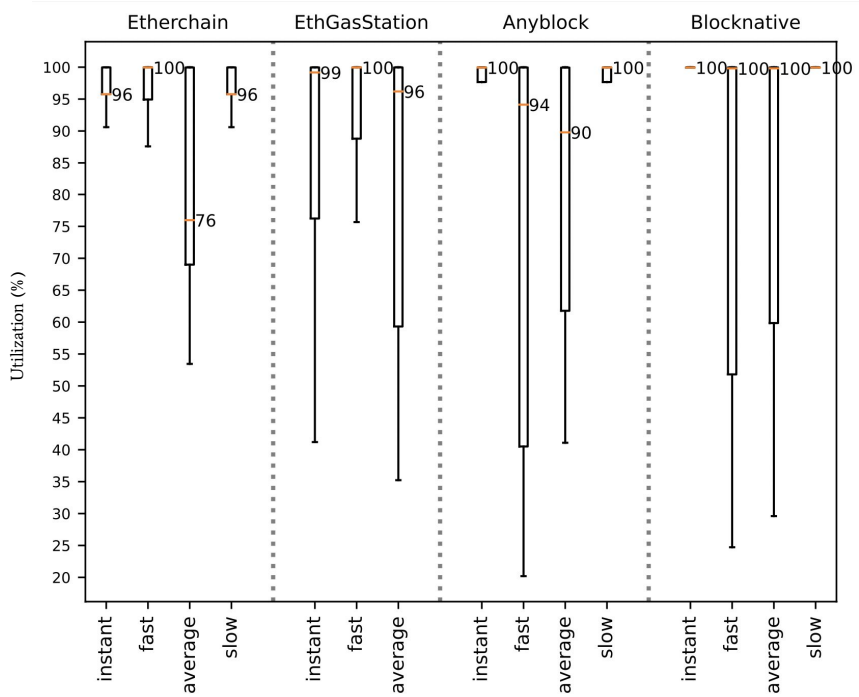

Figure 2.12: Example of block utilization when transactions left the pending pool.

gas price predicted when a transaction does not make it into any future block. For instance, EthGasStation's gas prices did not cross the 30^{th} percentile making it very unlikely to be selected by a miner.

In Figure 2.12, we present another potential reason behind the transactions not being accepted in any block. We present block utilization analysis as box plots showing the distributions of block utilization across different oracles and tiers. We group the plots into four regions representing the oracles. On the x-axis, we list the gas price prediction tiers and on the y-axis, we show the block capacity when the unaccepted transactions were present in the pending pool. For instance, when users submit transactions after consulting with Blocknative, the block utilization is at a 100%. Also, the average block utilization during all instances is 97%. We observe that the unaccepted transactions were very underpriced for

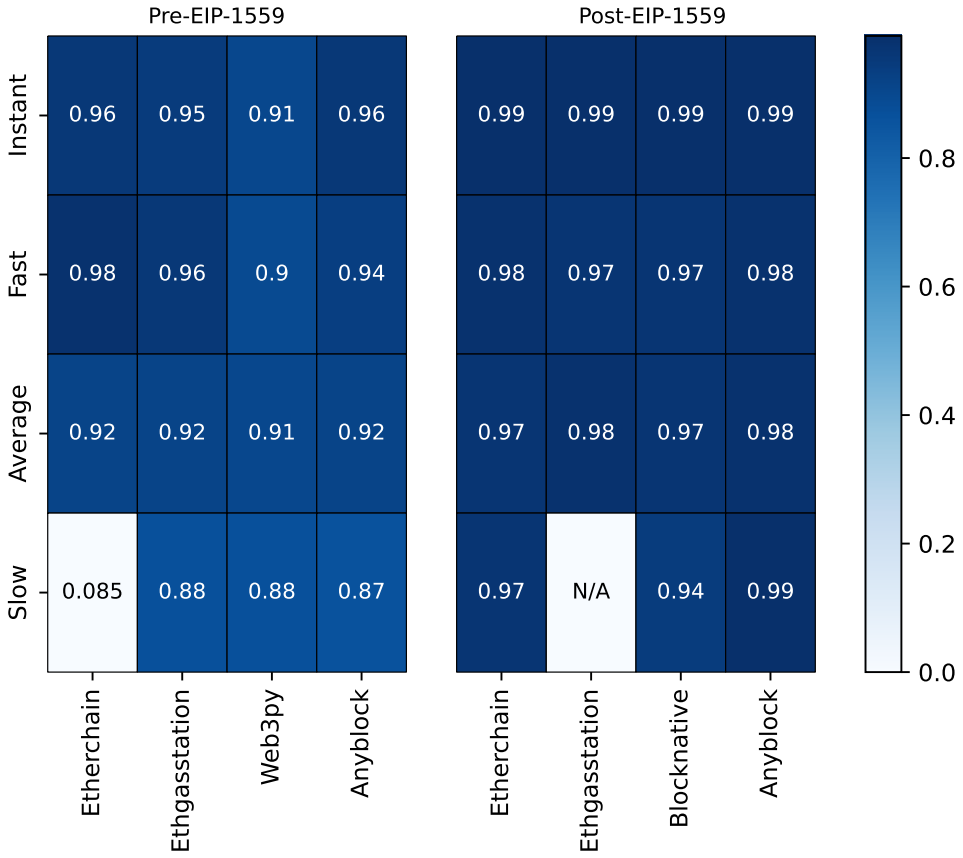

Figure 2.13: Percentage of transactions eventually accepted for each oracle and tier.

times of high block utilization leading them to a lower gas price percentile and hence not making it into the blockchain.

We also investigate the number of transactions that eventually made it into a block. This includes underpriced transactions, overpriced transactions, and transactions with perfect prediction. Figure 2.13 shows the percentage of transactions submitted at the price of each oracle and tier that eventually made it into a block pre and post-EIP-1559. The x-axis lists the oracles, while the y-axis the prediction tiers. We observe transactions in tiers with higher prices are more likely to be accepted into the blockchain. EthGasStation, Etherchain, and Anyblock performed significantly better after the release of EIP-1559. For

example, the instant tier of mentioned oracles has a 99% eventual acceptance rate. In other words, if a user submits a transaction at the instant tier price prediction, there is a 99% chance the transaction will make it to the blockchain at some point. Also, we noticed that Anyblock and EthGasStation are both leading in terms of eventual acceptance rate pre and post-Ethereum's update to the gas pricing mechanism. Notably, the results indicate that EIP-1559 improved the eventual acceptance of transactions.

Conclusions

In this chapter, we analyzed and compared the performance of Ethereum gas price oracles in terms of their accuracy of acceptance, underpricing, overpricing, and transaction eventual acceptance rates. We based our analysis on new oracle accuracy metrics that capture both underpricing and overpricing of transactions allowing us to quantify the metrics in terms of time and money wasted. We apply our performance metrics to oracles before and after the introduction of EIP-1559. We observe that EIP-1559 decreases the number of underpriced transactions and increases the number of overpriced transactions. We also observe that EIP-1559 improved the overall performance of oracles by increasing acceptance accuracy, especially for the fast and instant tiers. At the same time, the accuracy and transaction acceptance rates of existing oracles leave much to be desired.

YARDSTICK: GAS PRICE ORACLE FOR ETHEREUM GAS PRICE PREDICTION FOR EIP-1559-COMPATIBLE TRANSACTIONS

Introduction

The duration of time it takes a miner to add a transaction to the blockchain is dependent on the transaction's gas price. Ethereum transaction gas price plays an important role in the duration of time it takes a miner to accept a transaction into the blockchain. The amount of gas a user pays depends on how fast the transaction needs to be accepted into a block [18]. These prices vary due to network congestion and thus make a gas price prediction uneasy [5]. Consequently, blockchain users tend to consult with publicly available gas price oracles to ensure their transactions are accepted into the next available block with a minimal gas price.

Before the London Fork and the release of EIP-1559, Ethereum followed an auction mechanism to price transactions [4]. To maximize profit, miners select transactions with higher bids and add them to a block. In August 2022, Ethereum moved to a new pricing procedure to control the volatility of the classic pricing system. The London upgrade aims to stabilize gas prices and avoid sudden spikes in the network [4]. Buterin et al. developed a new gas pricing mechanism and introduced base, max, and priority fees [9].

EIP-1559 introduced a base fee which is the minimum amount of gas required by the network to consider the transaction in the next block. The protocol also added a priority fee which is the reward the miner receives upon adding a transaction to a block. Finally, the protocol introduced the max fee which is the maximum amount of gas the user is willing to pay to include a transaction in the next block. EIP-1559 protocol updates the base fee with respect to current block utilization by either increasing the base fee if the block capacity exceeds a predefined threshold or decreasing it otherwise [9].

The motivation behind our algorithm comes from the fact that the protocol maintains some level of stability by increasing and decreasing the base fee. Accordingly, EIP-1559 aims

for rapid spikes in gas prices and allows oracles to improve gas price predictability.

In this chapter, we revisit the question of gas price prediction mechanisms. We propose a new gas price prediction algorithm for EIP-1559 compatible transactions. We formulate the algorithm based on two main variables; Ethereum's latest base fee and the percentage of the last block utilization.

We collect, process, and analyze a dataset of 597551 Ethereum blocks gathered between January 2022 and May 2022. We test the performance of our algorithm and then we compare the probability of acceptance for each block utilization with the results of two publicly available oracle APIs (Anyblock [2] and Blocknative [7])). We select Anyblock and Blocknative since they belong to the same category of oracles that predicts gas price and associate the prediction with a probability of acceptance in the next available block. Our results show that our algorithm outperforms both Anyblock and Blocknative especially in times of high blockchain congestion.

We organize the rest of this chapter as follows. In section 2, we provide an overview of different types of prediction mechanisms utilized by publicly available gas price oracles. In section 3, we describe our data collection and processing methods and then we devise a new prediction algorithm based on block utilization. Section 4 presents the accuracy of the algorithm in terms of probability of acceptance and a comparative evaluation with Anyblock and Blocknative oracle. Finally, we conclude in section 5.

Related Work

Although gas price prediction is a critical technology for Dapp operations, until recently, not much work has been done to understand and improve gas prediction mechanisms.

Pierro and Rocha considered the relationship between blockchain factors, such as network hash rate and the price of Ethereum in USD, and the price predictions by ETHGasStation [27]. They used Granger causality and discovered that the changes in the

number of pending transactions and the number of miners have a statistically significant influence on changes in predicted gas prices.

Other efforts aimed to provide better gas prediction models. Liu et al. developed a regression XGBoost model to predict the gas price cutoff for transaction acceptance into the next block [25]. Their approach limited transaction overpricing and showed that 74.9% of transactions could save on gas fees. Werner et al. noticed a seasonality to gas prices of accepted transactions and developed a model based on Gated Recurrent Units for gas price prediction [32]. Their model outperformed the prediction mechanism in Geth to show cost savings of over 50% and an inclusion delay of 1.3 blocks. Carl and Ewerhart used a seasonal ARIMA model to predict median threshold gas price [10]. Finally, Singh et al. showed that a Random Forrest model also led to more accurate predictions than those offered by ETHGasStation [28]. Below we discuss the prediction mechanisms for four publicly available gas price oracles:

EthGasStation [17] examines the gas prices submitted by transactions over the latest 200 blocks and predicts gas prices based on the minimum gas price accepted in a percentage of blocks. The oracle provides gas price prediction for 3 different tiers: standard, fast, and instant. The standard, fast, and instant tiers refer to a transaction expected to be accepted in a block in the next 5 minutes, 2 minutes, and 30 seconds respectively.

Etherchain makes a prediction on the gas price that should be paid for a transaction to be confirmed within a certain number of blocks. We could not find information about Etherchain's prediction mechanism. However, we compared the max fee recommended by Etherchain with respect to the latest base fee retrieved by the same tool and we concluded the following:

$$predicted\_max\_fee = base\_fee \times 2.03$$

Etherchain provides gas price prediction and priority fee prediction for 3 different tiers: Standard, fast, and instant with transaction acceptance delay similar to EthgasStation.

Anyblock's gas price API estimates the priority fee and the maximum fee for the five tiers: slow, standard, fast, faster, and instant. Anyblock also provides the probability of acceptance in each tier. Depending on the probability of acceptance, Anyblock takes into account the three cheapest EIP-1559 transactions in each of the latest 20 blocks. The estimates are based on the percentiles of the cheapest priority fees in the latest 20 blocks. The priority fee is predicted by Anyblock for each tier whereas the max fee is not estimated but rather statically calculated from the current base fee using the following formula:

$$predicted\_max\_fee = (base\_fee \times 2) + priority\_fee$$

Anyblock formed a relationship between the transaction's gas price percentile and the probability of acceptance by examining more than 100,000 blocks from Ethereum's mainnet. Anyblock provides five different tiers representing different probabilities of acceptance based on the percentile of transactions gas price. For example, the slow tier ensures that transactions submitted at the recommended price are at the 35_{th} percentile with respect to other transactions in the pool and have a 30% probability of acceptance. Further, the standard tier ensures that transactions submitted at the recommended price are at the 60_{th} percentile with respect to other transactions in the pool and have a 60% probability of acceptance. On the other hand, the fast tier ensures that transactions submitted at the recommended price are at the 90_{th} percentile with respect to other transactions in the pool and have a 90% probability of acceptance. Finally, the instant tier, which is the fastest tier offered by Anyblock, ensures that transactions submitted at the recommended price are at the 100_{th} percentile with respect to other transactions in the pool and have a 99% probability of acceptance.

Blocknative states that their tool inspects all pending Ethereum transactions in the waiting pool and predicts the minimum gas price required to confirm the transaction in the next block. The prediction mechanism uses machine learning to train a prediction model using a quantile regression technique. After experimenting with several prediction models,

Blocknative's research team concluded that quantile regression can accurately predict the gas price for the next block gas price. Further, the quantile regression model requires a quantile parameter that will reflect the probability of acceptance of gas price prediction. Blocknative refers to the probability of acceptance as "confidence" and they separate it into 5 different tiers: Confidence 99, Confidence 95, Confidence 90, Confidence 80, and Confidence 70 which refers to the probability of being included in the next block respectively.

Prediction Algorithm

In the following section, we discuss the first two parts of our prediction algorithm. We discuss our data-gathering and processing techniques. Then, we show the steps that we followed to build the utilization lookup that we eventually use to predict the max gas price

Data Processing and Classification

We stored 597,551 Ethereum blocks from our local Geth node between January 2022 and May 2022. We parsed and stored the blockchain information in a local database for persistence and data analysis. The yardstick algorithm predicts the max gas fee based on two main variables, base fee, and block utilization. Therefore, we processed blocks between September 2021 and April 2022 to ensure all blocks were created after EIP-1559 when Ethereum introduced the notion of base and max fees. We calculate the utilization of each block as the ratio of the amount of gas spent with respect to the maximum block gas capacity. Then, we group the blocks based on their utilization. For example, all blocks with 90% utilization are grouped together. Further, from each block, we select the base fee, a transaction that paid the highest gas price, and the one that paid the lowest. Now, we have a dictionary with each minimum and maximum gas price spent with respect to each block utilization. Then, we calculate the uplift between the base fee and the max fee paid and the uplift between the base fee and the min fee paid. Figure **??**, demonstrates the steps

we followed to process and modify Ethereum blockchain data.

Algorithm 3.1: Data Processing and Classification

1: **for** *block in blocks* **do**
2: block utilization = Get_Block_Utilization(block number)
3: base fee = Get_Block_Base_Fee(block number)
4: max gas price = max(gas prices paid in block)
5: max uplift = max gas price/base fee
6: min gas price = min(gas prices paid in block)
7: min uplift = min gas price/base fee
8: add the object to its corresponding utilization group
9: **end for**

Utilization Lookup

After categorizing the min and max uplift by their respective block utilization, in Algorithm 3.2, we calculate the mean of all the max and min uplifts grouped by block utilization.

Algorithm 3.2: Uplift-Utilization Lookup

1: **for** *utilization in utilization groups* **do**
2: max uplift = mean(all max uplifts)
3: min uplift = mean(all min uplifts)
4: **end for**

Evaluation

In the following section, we verify our algorithm by finding the probability of acceptance of transactions that consulted with our oracle. Then, we compare our probability of acceptance results to those of Anyblock and Blocknative. We chose these two oracles due to their similar prediction nature that focuses on the probability of acceptance.

Probability of Acceptance

We calculate the block utilization at the time of prediction. Then, we refer to the lookup table to determine the maximum and minimum uplift. Afterward, We calculate the maximum gas fee by multiplying the maximum uplift by the base fee of the latest block. Then, We scan the waiting pool and check for all transactions that match our predicted maximum gas price. We check whether the transaction was accepted in the next block or not for each matched transaction. Then, we record the block utilization and the ratio of accepted transactions with respect to the number of matched transactions in each block where our predicted price matches some transactions in the waiting pool. Finally, we calculate the probability of acceptance as the mean of all recorded ratios.

Figure 3.1 and Figure 3.2 show the recommended maximum and minimum uplift with respect to the block utilization. The x-axis shows the block utilization, while the y-axis shows the different maximum uplifts. We notice that as block utilization increases, transactions require a larger amount of gas to enter the blockchain. Hence, the increase in uplift allows a higher maximum gas fee and accordingly increases the probability of acceptance. In Algorithm 3.3, we test our algorithm by calculating the probability of acceptance for the transactions that match our prediction. We group the prediction instances with respect to the block's utilization at the time of prediction. Accordingly, we were able to calculate

the number of transactions that consulted our oracle and then we check whether those transactions made it into the next block or not.

In Figure 3.3, we present the probability of acceptance for transactions consulting our minimum and maximum gas price prediction. The x-axis shows the block utilization and the y-axis shows the probability of acceptance. The light red dots show the probability of acceptance of transactions for transactions using our recommended max gas fee while the faded grey dots represent the transactions that used our minimum gas price.

The red and grey are fitted lines on the shortest distance metric that represent an increasing and decreasing trend respectively. We notice that as the block utilization increases, our maximum predicted price yields a higher probability of acceptance whereas our minimum

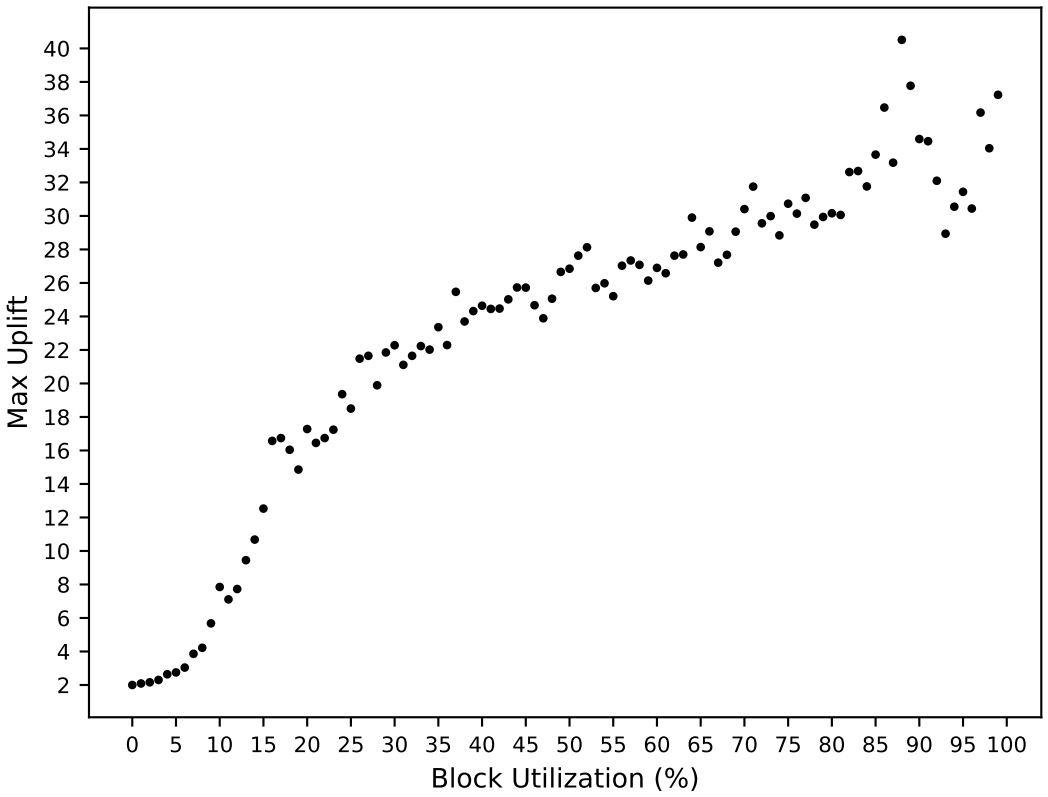

Figure 3.1: Block Utilization vs Maximum Uplift.

predicted price is lower.

Comparative Analysis with Anyblock and Blocknative

In this section, we perform a comparative analysis between Yardstick, Anyblock, and Blocknative's probability of acceptance. To conduct the analysis, we applied Algorithm 3.3 on Anyblock and Blocknative which resulted in the probability of acceptance for all block utilization.

Probability of Acceptance In Figure 3.4, we present the probability of acceptance for transactions consulting with Yardstick, Anyblock, and Blocknative. We present the block

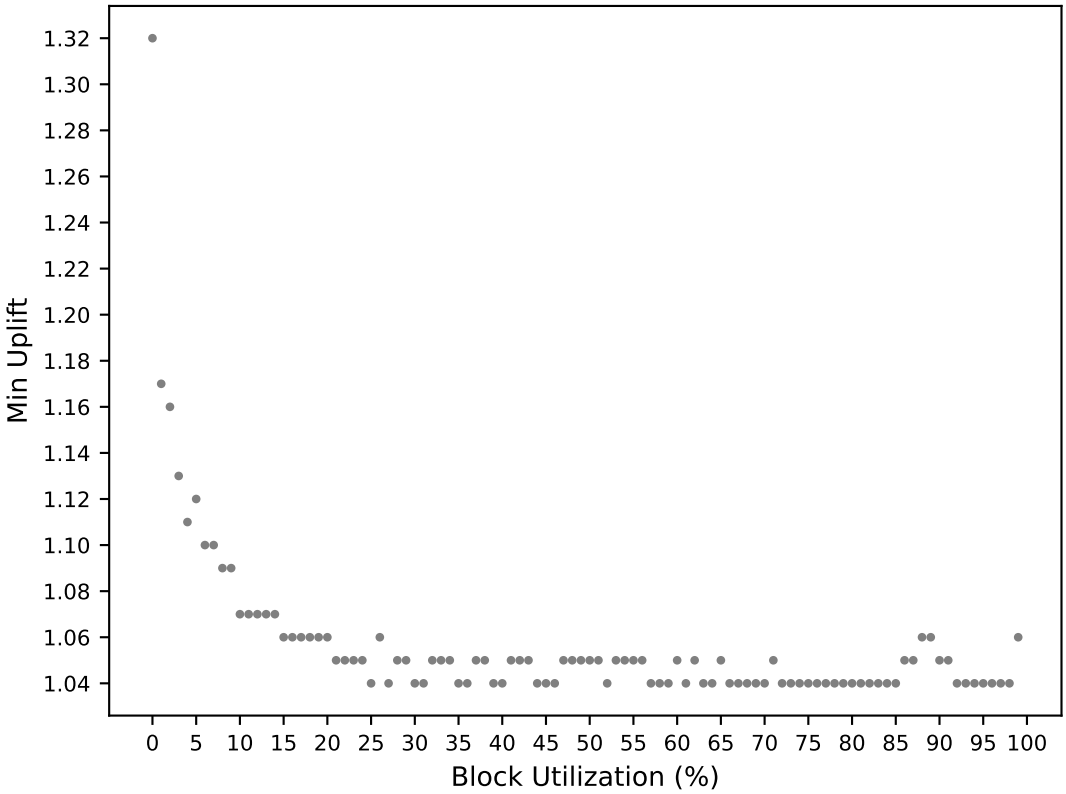

Figure 3.2: Block Utilization vs Minimum Uplift.

Algorithm 3.3: Probability of Acceptance

1: **for** *block in blocks* **do**
2: block utilization = Get_Block_Utilization(block number)
3: base fee = Get_Block_Base_Fee(block number)
4: max uplift = Get_Max_Uplift_From_Utilization_Lookup(block utilization)
5: min uplift = Get_Min_Uplift_From_Utilization_Lookup(block utilization)
6: max prediction = base fee x max uplift
7: min prediction = base fee x min uplift
8: get transactions that match our max and min predictions from the waiting pool
9: max probability of acceptance = number of matched transactions/number transactions accepted in next block
10: min probability of acceptance = number of matched transactions/number transactions accepted in next block
11: add max and min probabilities to corresponding utilization group
12: **end for**
13: **for** *utilization in utilization groups* **do**
14: max probability of acceptance = mean(all max probability of acceptance)
15: min probability of acceptance = mean(all min probability of acceptance)
16: **end for**

utilization on the x-axis and the probability of acceptance on the y-axis. Yardstick's probability of acceptance outperforms Blocknative's on every block utilization. We can see that both oracles improve with higher block utilization however Yardstick's probability of acceptance surpasses Blocknative's. On the other hand, we observe that transactions consulting Anyblock in times of low block utilization have a higher chance of acceptance than those consulting oracle during high block utilization. Hence, the blue fitted line in

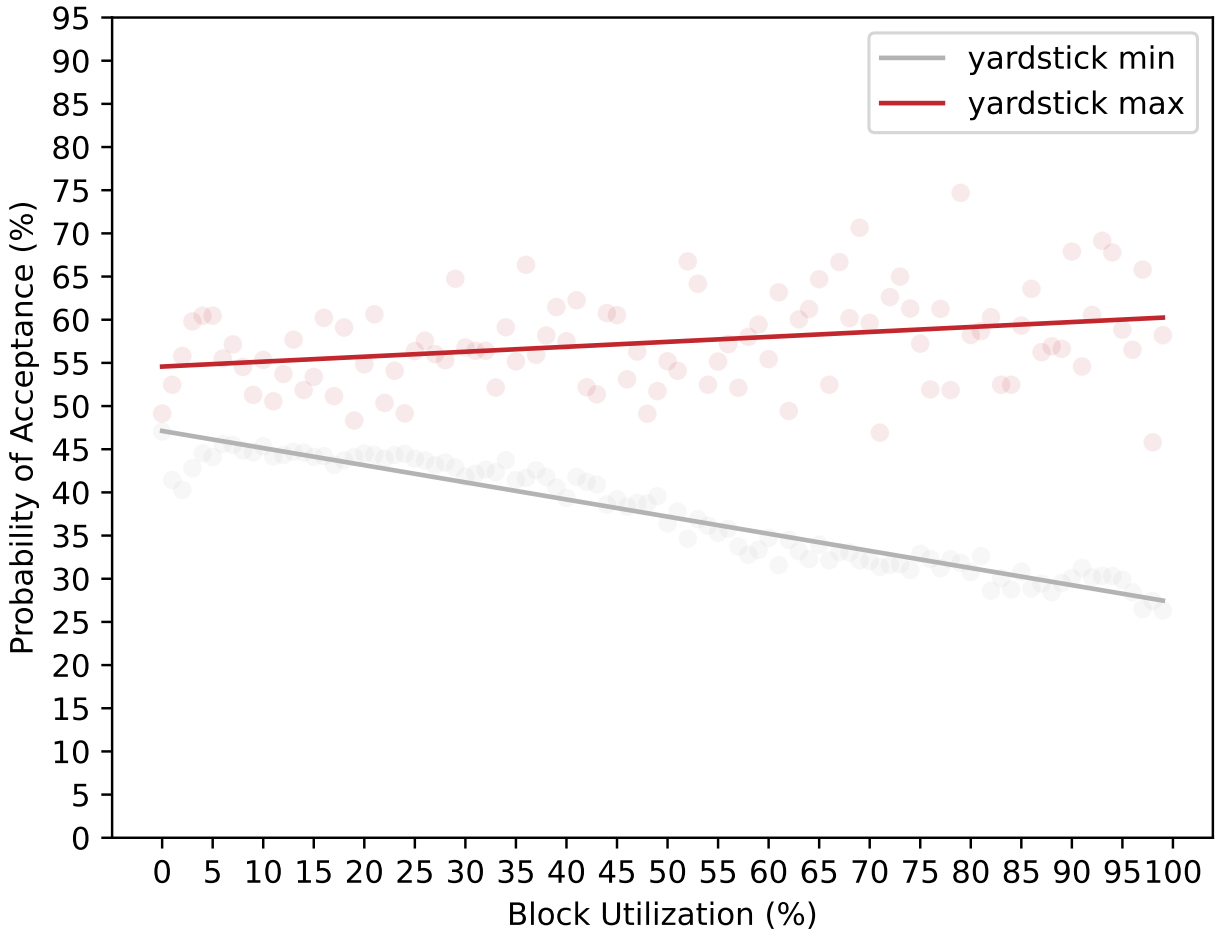

Figure 3.3: Yardstick Probability of Acceptance.

Figure 3.4 shows a negative slope whereas the red (Yardstick) and green (Blocknative) show an increasing trend. We also noticed that although Anyblock's probability of acceptance decreases as block utilization increases, it outperforms Blocknative for all utilization and Yardstick for utilization below 50%. Finally, Yardstick shows an improved performance in times of high blockchain traffic and thus providing a higher probability of acceptance for transaction consulting with our oracles.

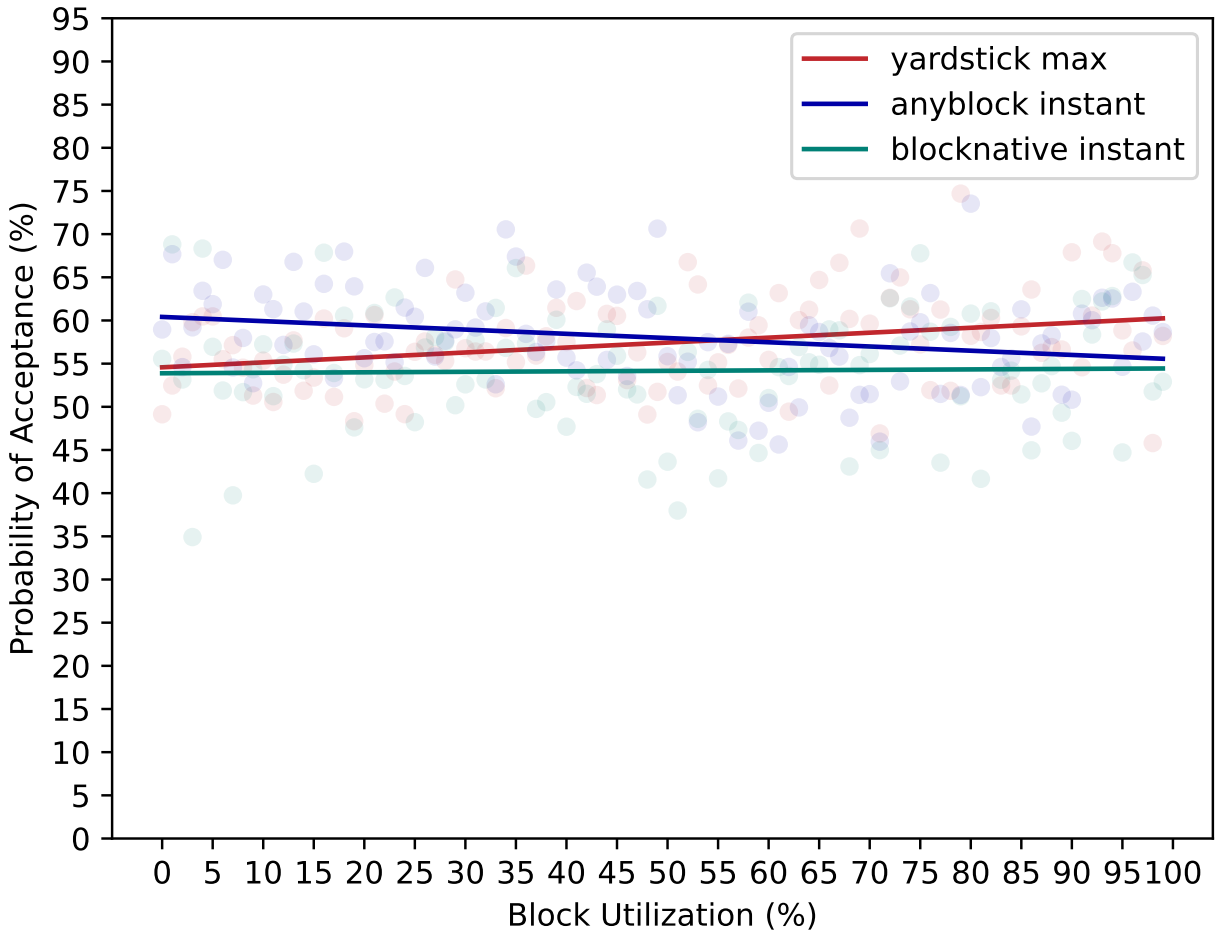

Figure 3.4: Yardstick Maximum Prediction vs Anyblock and Blocknative's instant tiers.

Time Wasted In Figure 3.5, we present the time wasted for transactions consulting with Yardstick, Anyblock, and Blocknative.

On the x-axis, we present the oracles and on the y-axis, we show the distribution of time wasted for each prediction instance measured in seconds. We also present and compare the means of each distribution. We conclude that Yardstick wastes 29 seconds on average when the prediction misses the target. On the other hand, Blocknative and Anyblock waste 32 and 43 seconds on average respectively. We observe a slight improvement in Yardstick's performance when the oracle underprices a transaction.

Money Wasted In Figure 3.6, we present the money wasted for transactions consulting with Yardstick, Anyblock, and Blocknative.

On the x-axis, we present the oracles and on the y-axis, we show the distribution of money wasted for each prediction instance measured in Gwei. We also present and compare the means of each distribution. We conclude that Yardstick wastes 40 Gwei on average when the prediction overprices a transaction. On the other hand, Blocknative and Anyblock waste 43 and 51 Gwei on average respectively. We observe a slight improvement in Yardstick's performance when the oracle overprices a transaction.

Statistical Significance

In this section, we report and discuss the results of our statistical significance testing in terms of t-test and p-value.

In Table 3.1, we present the t-value and the p-value of the statistical analysis conducted on Yardstick max prediction and Anyblock's instant tier. We conducted the analysis on three different metrics; Probability of Acceptance, Time wasted, and money wasted. The two-tailed test for the probability of acceptance yielded a t-value of -0.48 and a p-value of

$0.62 > alpha(= 0.05)$ and hence we fail to reject the null hypothesis. Therefore, the test yielded that there is no consequential relationship between the Yardstick and Anyblock's probability of acceptance. For the time and money wasted metrics, the difference in mean of our sample data is -5.61 and -8.74 respectively. Our p-values are 0.023 and 0.045 which is smaller than $alpha(= 0.05)$. Therefore, we can reject the null hypothesis of no difference and say with a high degree of confidence that the true difference in means is not equal to zero.

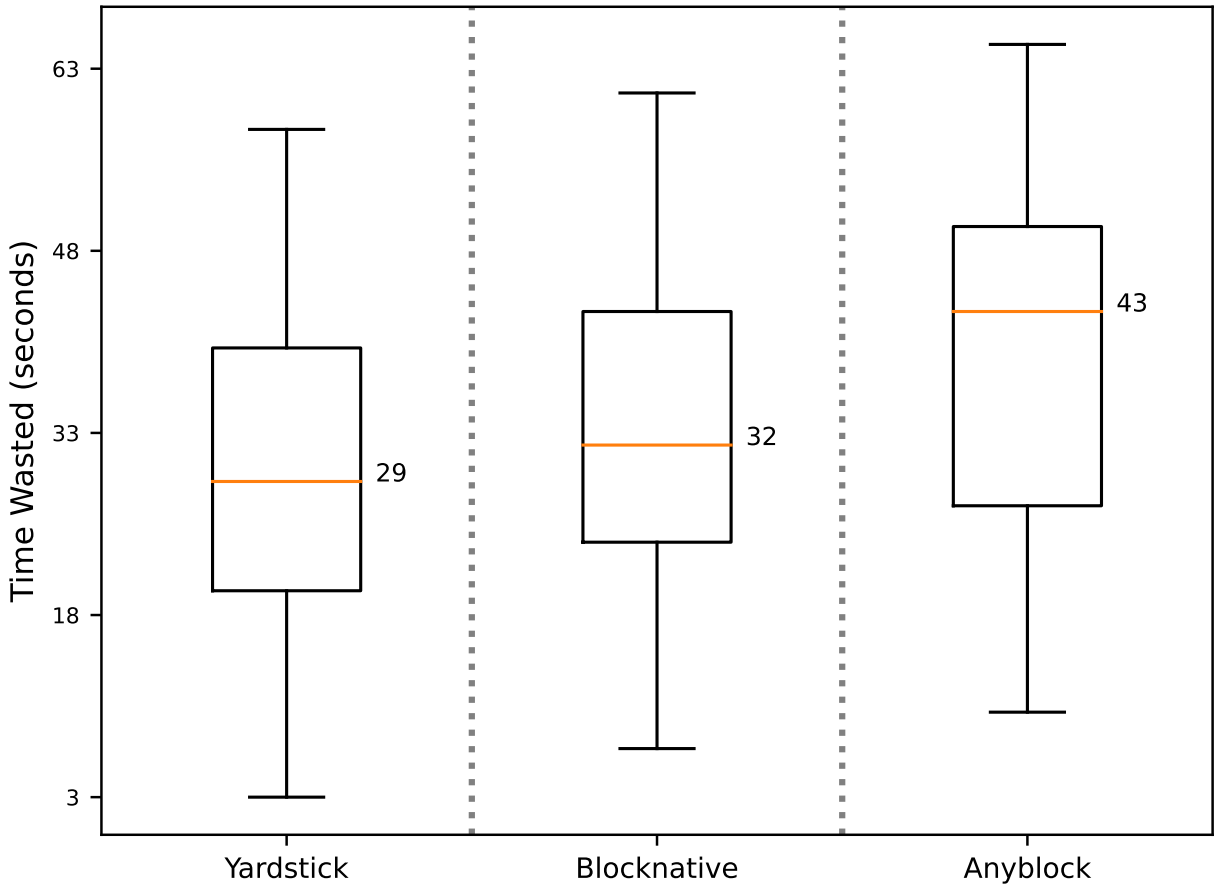

Figure 3.5: Time Wasted: Yardstick Maximum Prediction vs Anyblock and Blocknative's instant tiers.

In Table 3.2, we present the t-value and the p-value of the statistical analysis conducted on Yardstick max prediction and Blocknative's instant tier. We conducted the analysis on three different metrics; Probability of Acceptance, Time wasted, and money wasted. The two-tailed test for the probability of acceptance yielded a t-value of 2.437 and a p-value of $0.015 < alpha(= 0.05)$ and hence we reject the null hypothesis. Therefore, there is a definite, consequential relationship between Yardstick and Blocknative's probability of acceptance. For the time and money wasted metrics, the difference in mean of our sample

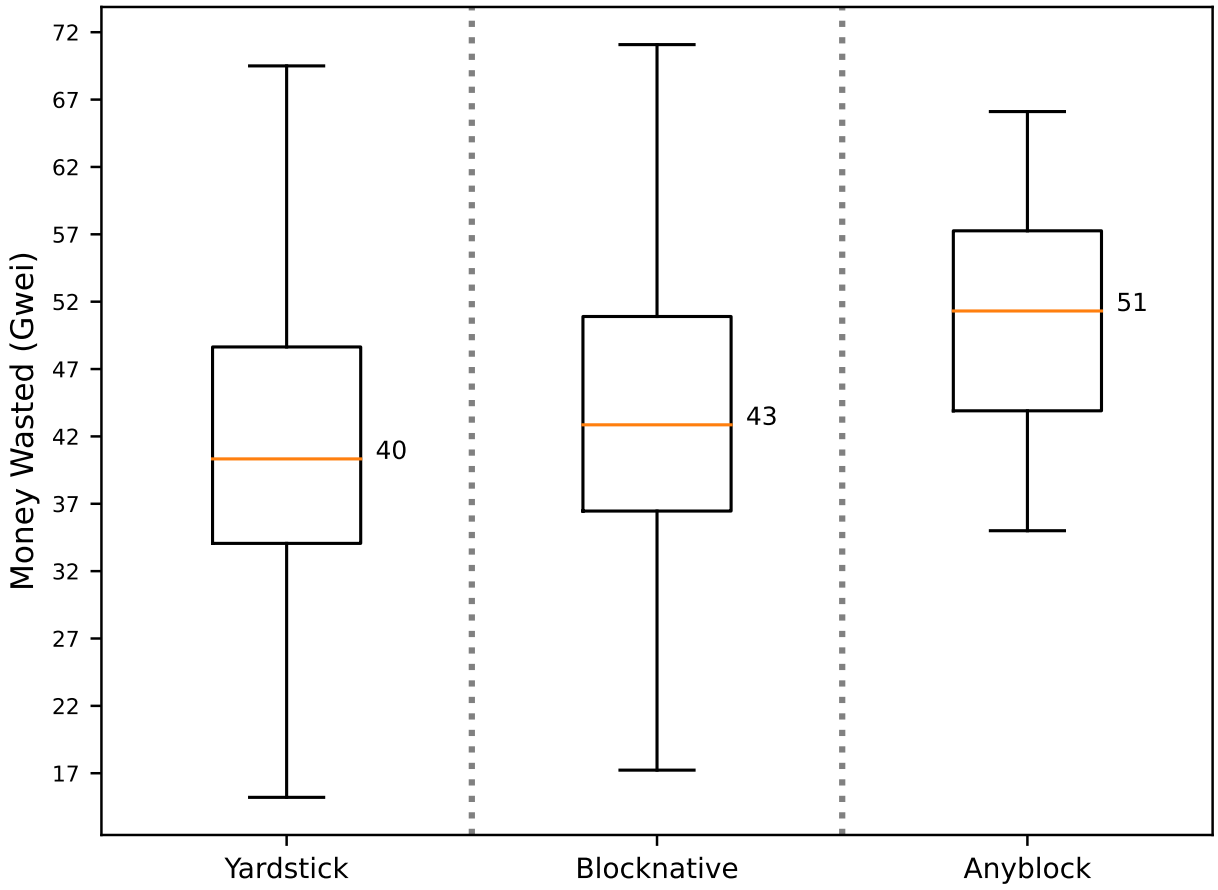

Figure 3.6: Yardstick Maximum Prediction vs Anyblock and Blocknative's instant tiers.

data is -11.05 and -7.95 respectively. Our p-values are 0.029 and 0.031 which is smaller than $alpha(= 0.05)$. Therefore, we can reject the null hypothesis of no difference and say with a high degree of confidence that the true difference in means is not equal to zero. This means that we need to collect more data for our model to be more statistically significant.

Metric	t-test	p-value
Probability of Acceptance	-0.48	0.62
Time Wasted	-5.61	0.023
Money Wasted	-8.74	0.045

Table 3.1: Statistical Significance between Yardstick and Anyblock Instant Tier

Metric	t-test	p-value
Probability of Acceptance	2.437	0.015
Time Wasted	-11.05	0.029
Money Wasted	-7.95	0.031

Table 3.2: Statistical Significance between Yardstick and Blocknative Instant Tier

Conclusions

In this chapter, we introduced Yardstick; a new gas prediction mechanism for EIP-1559-compatible gas price oracle. Yardstick gas price prediction is determined by two factors:

the latest block utilization and base fee. We built a dictionary that serves as a lookup table mapping each utilization to an estimated base fee multiplier. a.k.a uplift. Similar to Anyblock and Blocknative, our mechanism provides a probability of acceptance for each predicted value allowing us to perform a comparative analysis between the oracles. Finally, Yardstick yielded a higher probability of acceptance with a lower time and money wasted compared to Anyblock and Blocknative, especially in times of high block utilization.

CONCLUSION

In this dissertation, we expanded on the previous work done on oracles' performance metrics and added the metrics of time and money wasted. The performance metrics allow us to quantify the overpricing and underpricing in terms of seconds and Gwei wasted respectively. We also applied the metrics to 5 publicly available oracles (Etherchain, EthGasStation, Web3.py, Anyblock, and Blocknative) and provided a comparative analysis that will help users make informed decisions and save time and money. We also applied our metrics on oracles before and after the release of EIP-1559 and we concluded that gas prediction post-EIP-1559 requires further investigation. Accordingly, we developed a new mechanism to predict gas prices based on block utilization and base fee. Further, we performed a comparative analysis between Yardstick, our oracle, and two publicly available oracles that follow the probability of acceptance approach. We notice that Yardstick scored a higher probability of acceptance than Anyblock and Blocknative when block utilization exceeds 50%. Furthermore, when Yardstick underprices a transaction, the oracle wastes 29 seconds on average compared to 32 and 43 seconds wasted on average by Blocknative and Anyblock respectively. Lastly, we calculated the money wasted for transactions overpriced by Yardstick and noticed that Yardstick wastes an average of 40 Gwei per overpricing transaction. Recall, that Blocknative and Anyblock waste an average of 43 and 51 Gwei respectively when the oracle overprices a transaction. Hence, Yardstick outperforms Blocknative and Anyblock in terms of probability of acceptance, time wasted, and money wasted. Finally, our solution can be further investigated and applied in other Blockchain ecosystems. In fact, our prediction algorithm highly correlates to the block utilization and hence we can develop new lookup tables for other Blockchains such as Bitcoin, Solana, and Cardano.

REFERENCES CITED

[1] Giuseppe Antonio Pierro, Henrique Rocha, Roberto Tonelli, and Stéphane Ducasse. Are the Gas Prices Oracle Reliable? A Case Study using the EthGasStation. In *IEEE Workshop on Blockchain Oriented Software Engineering (IWBOSE)*, February 2020.

[2] Anyblock. API endpoint. `https://api.anyblock.tools/ethereum/latest-minimum-gasprice/`, Accessed April 2021.

[3] Anyblock. Gas Price Prediction. `https://www.anyblockanalytics.com/use-cases/gas-price-predictions/`, Accessed April 2021.

[4] Tim Beiko. London mainnet announcement, July 2021.

[5] Blockchair. Transaction Volume (ETH). `https://blockchair.com/ethereum/charts/transaction-volume-eth`, April 2021.

[6] Blocknative. Introducing blocknative gas platform: A fresh take on ethereum gas price estimation. `https://www.blocknative.com/blog/introducing-gas-platform`, Accessed April 2021.

[7] Blocknative. API endpoint. `https://www.blocknative.com/gas-estimator?gasType=ethereum`, Accessed April 2022.

[8] Vitalik Buterin. Might EIP-1559 run the risk of over-stressing nodes and miners during periods of high usage?, January 2021.

[9] Vitalik Buterin, Eric Conner, Rick Dudley, Matthew Slipper, Ian Norden, and Abdelhamid Bakhta. EIP-1559: Fee market change for eth 1.0 chain, April 2019.

[10] David Carl and Christian Ewerhart. Ethereum gas price statistics. Technical Report Working Paper No. 373, Social Science Research Network, December 2020.

[11] José Eduardo de Azevedo Sousa, Vinícius Oliveira, Júlia Valadares Glauber Dias Gonçalves, Saulo Moraes Villela, Heder Soares Bernardino, and Alex Borges Vieira. An analysis of the fees and pending time correlation in Ethereum. *International Journal of Network Management*, July 2020.

[12] Etherchain. API endpoint. `https://etherchain.org/api/gasPriceOracle`, Accessed April 2021.

[13] Etherchain. Gas Price Oracle. `https://etherchain.org/tools/gasPriceOracle`, Accessed April 2021.

[14] Etherscan. Ethereum developer apis. `https://etherscan.io/apis#stats`, Accessed April 2021.

[15] ETHGasStation. gasstation-express. https://github.com/ethgasstation/gasstation-express-oracle, March 2020.

[16] ETHGasStation. API endpoint. https://ethgasstation.info/api/ethgasAPI.json?api-key=XXAPI_Key_HereXXX, Accessed April 2021.

[17] ETHGasStation. Gas price. https://docs.ethgasstation.info/gas-price, Accessed April 2021.

[18] William Foxley. Ethereum Transaction Fees Hit Record Highs as Ether, DeFi Coins Soar. https://www.coindesk.com/ethereum-transaction-fees-hit-record-highs-as-ether-defi-coins-soar, February 2021.

[19] Go Ethereum. eth namespace. https://geth.ethereum.org/docs/rpc/ns-eth, Accessed April 2021.

[20] Go Ethereum. JSON-RPC Server. https://geth.ethereum.org/docs/rpc/server, Accessed April 2021.

[21] Go Ethereum. txpool namespace. https://geth.ethereum.org/docs/rpc/ns-txpool, Accessed April 2021.

[22] Uri Klarman, Soumya Basu, Aleksandar Kuzmanovic, and Emin Gun Sire. bloXroute: A Scalable Trustless Blockchain Distribution Network. https://bloxroute.com/wp-content/uploads/2019/11/bloXrouteWhitepaper.pdf, November 2019.

[23] Arnaud Laurent, Luce Brotcorne, and Bernard Fortz. Transaction fees optimization in the ethereum blockchain. *Blockchain: Research and Applications*, 3(3), 2022.

[24] Stefanos Leonardos, Barnabé Monnot, Daniël Reijsbergen, Stratis Skoulakis, and Georgios Piliouras. Dynamical analysis of the EIP-1559 ethereum fee market, February 2021.

[25] Fangxiao Liu, Xingya Wang, Zixin Li, Jiehui Xu, and Yubin Gao. Effective GasPrice Prediction for Carrying Out Economical Ethereum Transaction. In *Dependable Systems and Their Applications (DSA)*, January 2020.

[26] Yulin Liu, Yuxuan Lu, Kartik Nayak, Fan Zhang, Luyao Zhang, and Yinhong Zhao. Empirical analysis of EIP-1559: Transaction fees, waiting time, and consensus security. January 2022.

[27] Giuseppe Antonio Pierro and Henrique Rocha. The Influence Factors on Ethereum Transaction Fees. In *Workshop on Emerging Trends in Software Engineering for Blockchain*, May 2019.

[28] Harsh Jot Singh and Abdelhakim Senhaji Hafid. Prediction of Transaction Confirmation Time in Ethereum Blockchain Using Machine Learning. In *Blockchain and Applications*, January 2020.

[29] Greg Thomson. Bitcoin and Ethereum slow down as transaction values and fees plunge 70%. https://cointelegraph.com/news/bitcoin-and-ethereum-slow-down-as-transaction-values-and-fees-plunge-70, March 2021.

[30] Web3.py. Gas Price API. https://web3py.readthedocs.io/en/stable/gas_price.html, Accessed April 2021.

[31] Ingo Weber, Vincent Gramoli, Alex Ponomarev, Mark Staples, Ralph Holz, An Binh Tran, and Paul Rimba. On Availability for Blockchain-Based Systems. In *IEEE Symposium on Reliable Distributed Systems (SRDS)*, September 2017.

[32] Sam M. Werner, Paul J. Pritz, and Daniel Perez. Step on the Gas? A Better Approach for Recommending the Ethereum Gas Price. In *Mathematical Research for Blockchain Economy*, October 2020.

[33] Zhao Yinhong and Nayak Kartik. EIP-1559 in retrospect. https://decentralizedthoughts.github.io/2022-03-10-eip1559/, March 2022.