Two Ways Gas Price Oracles Miss The Mark

Kemal Turksonmez, Marcin Furtak, Mike P. Wittie, David L. Millman
Gianforte School of Computing
Montana State University
kemalturksonmez@gmail.com, marcin_furtak@interia.pl, mike.wittie@montana.edu, david.millman@montana.edu

***Abstract*—Blockchain transactions compete for limited space in blockchain blocks. Miners prefer to include transactions with higher fees into new blocks. In the context of Ethereum, gas price price oracles predict fees such that transactions submitted at those fees make it into a block within a target delay. In practice, however, oracles are not accurate, which makes it difficult for distributed applications to operate predictable services in terms of price and performance.**

To understand oracle performance we define a new gas prediction accuracy metric. We demonstrate that oracles underprice transactions, causing them to miss the delay target, as well as overprice transactions, causing them to meet the delay target, but at a higher than necessary cost. We provide comparative analysis of five gas price oracles showing their relative accuracy, transaction accept rates, price stability, and discuss factors that influence oracle accuracy. We observe that the ETHGasStation oracle produces the most accurate and stable price predictions. For users that prefer to run their own oracle Web3.py provides comparable performance.

***Keywords*—blockchain, transaction fee, transaction delay, gas price oracles**

I. Introduction

Blockchain throughput, in terms of transactions per second, is fundamentally limited by the time it takes to disseminate and validate new blocks [1]. To ensure fair sharing of blockchain resources and provide incentives to miners to validate transactions, the inclusion of a transaction into a block requires a fee. These fees, however, are volatile as a result of changes in transaction volume [2] and cryptocurrency price [3], [4].

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 new blocks to maximize block rewards [5].

To help blockchain users submit transactions that enter the blockchain within some expected time and at a reasonable price, gas price oracles provide gas price predictions based on historical blockchain data. Unfortunately, oracle predictions are less accurate than advertised [6], which makes it difficult for distributed applications (Dapps) to operate predictable services both in terms of price and performance.

Currently the accuracy of the gas price oracles is not well understood. Pierro et al. have analyzed the accuracy of the ETHGasStation oracle and found that transactions submitted with predicted gas prices fail be included in Ethereum block within the expected delay [6]. Their work however 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 [7]–[9]. As a result, blockchain users do not know which publicly available oracle to rely on and how much error to expect.

In this paper we revisit the question of gas price oracle accuracy. We propose a new gas prediction accuracy metric that allows us to consider two types of gas price misprediction. First, when oracles provide a price that is too low, which leads to longer than expected transaction delay. Second, when oracles provide a price that is too high, which leads to overpayment.

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. Based on this dataset we compare the accuracy of three publicly available oracle APIs (ETHGasStation [10], Etherchain [11], Anyblock [12]) and two standalone CLI tools (Web3.py [13] and Geth [14]). Our results show that the ETHGasStation oracle produces the most accurate and stable price predictions. For users that prefer to run their own oracle Web3.py provides comparable performance.

The rest of this paper is organized as follows. In Section II we discuss the background on miner transaction pool mechanics and existing gas price oracles. Section III details the related work in gas price prediction and measurement. In Section IV we describe our data collection process. Section V presents the results of our analysis of oracle accuracy. Finally, we conclude in Section VI.

II. Background

A. Miner Transaction Pool Mechanics

Ethereum transactions enter the blockchain via miner nodes, who gossip the transaction among themselves. In some simplification, when a miner receives a transaction it validates the transaction and moves it into the executable (pending) transaction pool. The transactions in that pool are sorted by gas price. The miner will try to include high value transactions in its blocks, but may also remove transactions because they have been included in blocks by other miners, or because they are replaced in the pool by higher valued transactions. Thus, at any given point, there is a market-drive price cutoff for transactions to make it into a block. There also is a second price cutoff for transaction to remain in the pool and stand a chance for inclusion is subsequent blocks. Because the demand for space in blocks fluctuates with the volume of submitted transactions, it is easier to think of these price cutoffs in terms of the percentile of a transactions gas price in the pending transaction pool.

B. 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*, specifically `instant`/`fastest`, `fast`, `average`/`standard`/`normal`, and `slow`/`safe low` depending on an oracle's nomenclature. ETHGasStation [15], Etherchain [16], and Web3.py [13] specify their tiers in terms of expected transaction acceptance delay (TAD). The delay target for transactions submitted at the `instant` price point is the delay of one, or two blocks, or under 30 sec. The `fast` tier generally means within 2 min, `average` 5 min, and `slow` 30 min. Anyblock [17] and Geth [14] on the other hand, specify their 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 99th percentile, `fast` at the 90th percentile, `average` at the 60th percentile, and `slow` at the 35th percentile.

ETHGasStation and Etherchain use a Poisson Regression Statistical model to predict transactions based on gas prices in the past 200 blocks, updated every 100 blocks, or 25 min [6]. Anyblock simply reports the prices of specific gas price percentiles of accepted transactions in the past 200 blocks, without using Poisson Regression to tie them to acceptance delays [17].

Besides the above oracles accessible through REST APIs, we also investigate standalone tools. The Geth [14] implementation of an Ethereum miner offers a CLI to estimate gas price based on percentile and the number of observed blocks [18]. Web3.py [13] is a library that connects to a Geth node and creates a gas price prediction based on expected delay, the number of observed blocks, and whether more recent transactions should be weighted more heavily.

III. Related Work

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

Weber et al. were the first to analyze the relationship between gas price and transaction acceptance delay [19]. 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 [5]. These results may be understood in the context of transaction pool mechanics, where miners prefer higher priced transaction for inclusion in blocks, but do not follow strict priority.

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 [20]. They used Granger causality and discovered that the changes in the number of pending transactions and in the number of miners have statistically significant influence on changes in predicted gas prices. Lacking a metric for oracle accuracy, however, their work did not investigate the Granger causality of environmental factors on the accuracy of oracle predictions. Pierro et al. extended that work to show that ETHGasStation misses the transaction acceptance delay more often than the advertised 2% of the time [6]. They also showed that gas prediction accuracy improves with more frequent prediction based on more up-to-date data.

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 [7]. Their approach limited transaction overpricing and showed that 74.9% of transaction 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 [8]. Their model outperformed the prediction mechanism in Geth to show cost savings of over 50% and inclusion delay of 1.3 blocks. Carl and Ewerhart used a seasonal ARIMA model to predict median threshold gas price [9]. Finally, Singh et al. showed that a Random Forrest model also led to more accurate predictions than those offered by ETHGasStation [21].

In this paper, however, we focus on the accuracy of publicly gas price oracles available to Dapps. At the same time, our accuracy metrics may be applied to new prediction mechanisms to compare them against the status quo.

IV. Data Collection

To help interpret the results we briefly describe our data collection process and data structure.

A. 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 [22]–[24]. 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 I.

B. Oracle Data

We collected data from five different oracles: ETHGasStation, Etherchain, Anyblock, Web3.py, and Geth. We access ETHGasStation, Etherchain, Anyblock through their REST APIs [10]–[12] and parse the gas price for each tier. To access Web3.py and Geth gas prediction CLIs [18] we deployed a local Geth node. We query each endpoint when the Geth node

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	The list of transaction hashes in the block
`Transaction`	`hash`	Geth API	The hash of the transaction
	`sub_ts`	Geth API	The time the transaction was submitted
	`price`	Geth API	The gas price at which the transaction was submitted
`Oracle`	`name`	Oracle API/CLI	The name of the oracle
	`tier`	Oracle API/CLI	The name of a price prediction tier
	`price`	Oracle API/CLI	Price predicted by the oracle for the tier
	`ts`	Query script	Time at which we recorded the prediction
`Environment`	`net_hash_rate`	Etherscan	The combined hash rate of Ethereum miners
	`eth_usd`	Etherscan	The price of Ethereum in USD
	`ts`	Query script	The time at which queries for the data

TABLE I: Database schema of the collected data.

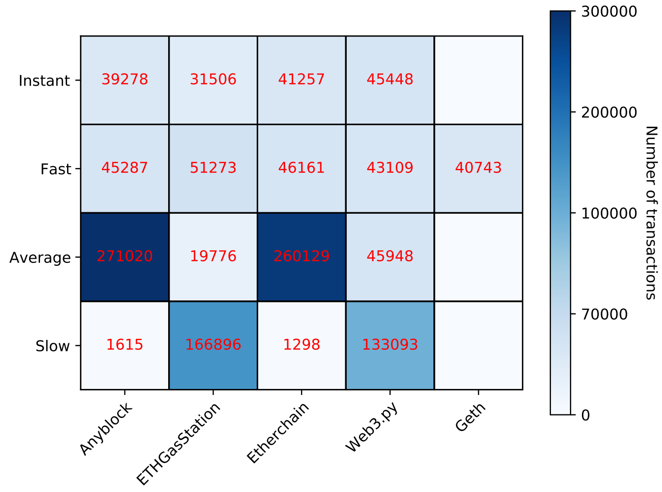


Fig. 1: Transactions that match gas price prediction for each oracle and tier.

detects a new block (around 13 sec on average) and record the data in the `Oracle` table as shown in Table I.

Web3.py and Geth only permit a price query with respect to one set of parameters at a time. To enable queries to Web3.py for all four tiers with the block interval, we modified and rebuilt Web3.py's `generateGasPrice` function. By isolating and combining instances where each category prediction was repeating the same calculations, we were able to cut the prediction runtime to a quarter of the original time. This modification allowed us to query Web3.py four times during each 13 sec data collection period. Unfortunately, we were unable to do the same for the Geth CLI and so we configured Geth to only report prices at the `fast` tier. We configured Geth parameters to report on prices at the 90th percentile corresponding to Anyblock's `fast` tier.

C. *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 [25]. We record the data in the `Environment` table in Table I.

V. Evaluation

A. *Number of Transactions*

To illustrate the integration of data in the `Oracle` and `Transaction` tables, we compute the relative transactions counts using price predictions from the different oracles and tiers. Of course it is not possible to know whether a transaction submitted at a price predicted by an oracle was actually submitted after consulting the oracle, but we follow the assumption of Pierro et al. [6] that it might as well have been.

Figure 1 shows the number of transactions submitted at the price predicted by each oracle and tier. 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. So for example, 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.pred_ts+13 s
```

where 13 sec is the block interval. Note that as mentioned in Section IV-B we do not have gas price prediction for Geth other than for the `fast` tier.

In general, we observe that the `instant` and `fast` tiers are similarly popular across oracles, in that there are comparable numbers of transactions submitted at the gas price predicted by all oracles in those tiers. We also observe that the `average` tiers for Anyblock and Etherchain and the `slow` tiers for ETHGasStation and Web3.py are the most popular. This result indicates that blockchain users tend to accept eventual transaction acceptance for lower price.

The data we collected does not corroborate the results presented by Pierro et al., who claim that the `fast` and `average` tiers of ETHGasStation are used by users only 6.3% of the time [6]. We observe that for ETHGasStation transactions use the `slow` tier most frequently, but that the remaining tiers are equally likely. Specifically, we observe that the `fast` and `average` were used by 23.8% of transactions. We suspect that this difference between our results and those of Pierro et al. stem from data being collected at different observation periods,

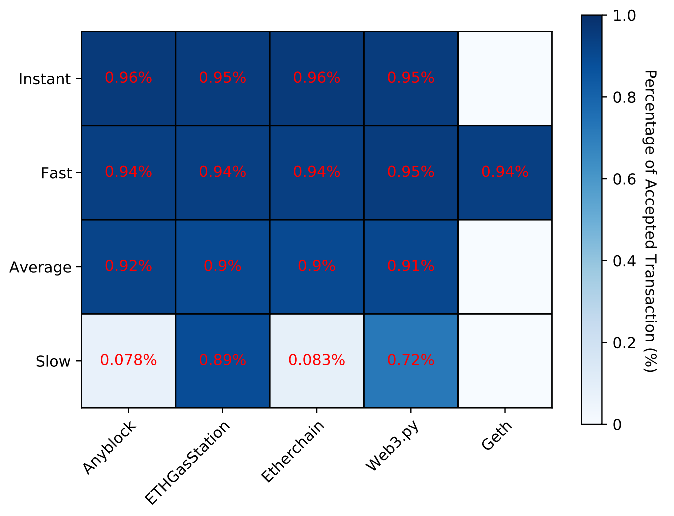


Fig. 2: Percentage of transactions eventually accepted for each oracle and tier.

as transaction fees have increased in 2021 likely leading more users to opt for lower gas price tiers.

B. Transaction Acceptance Rates

We also investigate what percentage of transactions submitted under the price of each oracle and tier eventually make it into a block. Figure 2 shows the percentage of transactions submitted at the price of each oracle and tier that are eventually accepted into a block. The x-axis lists the oracles, while the y-axis the prediction tiers.

We observe that transaction in higher priced tiers are more likely to be accepted into the blockchain. The difference between acceptance rates is small for the `instant`, `fast`, and `average` tiers, but is markedly lower for the `slow` tier. Notably, ETHGasStation and Web3.py perform significantly better at the `slow` tier than Anyblock and Etherchain.

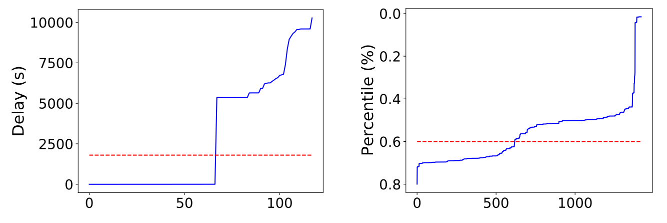
These results indicate that ETHGasStation has the highest transaction acceptance rates across all prediction tiers.

C. 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, or the percentile of accepted transaction price, e.g. Anyblock. Depending on the type of oracle we consider, we can measure its accuracy in terms of transaction acceptance delay (TAD), or accepted transaction percentile (ATP).

We measure TAD as the difference between transaction submission time and the time of the block that includes 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
```



(a) ETHGasStation `slow` TAD vs. transaction count sorted.

(b) Anyblock `average` ATP vs. transaction count sorted.

Fig. 3: Examples of TAD and ATP distributions.

The ATP query is a bit more complex, 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 own block. We base the ATP calculation on 200 blocks, since that's the time horizon of input data to both ETHGasStation and Anyblock [17], [20], [26].

Figure 3 shows the TAD for transactions in ETHGasStation's `slow` tier and the ATP for transactions in Anyblock's `average` tier. The y-axis in Figure 3a marks TAD in seconds. The y-axis (reversed) in Figure 3b marks the ATP in percent. The x-axis in both graphs marks the transactions accepted at the price of each oracle/tier sorted by TAD/ATP values. The solid blue lines show the TAD and ATP values, while the dashed red lines show the target TAD and ATP values for the tiers.

We observe that TAD/ATP values fall both above and below the target tier value. When the TAD/ATP line lies above the target line the oracle-proposed gas price is too low and transactions take longer than target to make it into a block. This is the case of a gas oracle *underpricing* its prediction. When the TAD/ATP 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.

Based on the examples of underpricing and overpricing predictions illustrated in Figure 3 we propose to compute oracle accuracy as follows.

1) First, we focus on comparing the number and types of mis-predictions. 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. [6], [20] noted that gas price oracles report a margin of error of 2%. We were not able to verify that number in oracle documentations. As such, we consider any transaction whose TAD differs from the target as suffering from under- or overpricing.

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

$$\widehat{D}_{o,t} = \frac{D_{o,t} - \min\limits_{o' \in O, t' \in T} D_{o',t'}}{\max\limits_{o' \in O, t' \in T} D_{o',t'} - \min\limits_{o' \in O, t' \in T} D_{o',t'}}$$

Similarly, for target transaction delay:

$$\widehat{r}_{o,t} = \frac{r_{o,t} - \min\limits_{o' \in O, t' \in T} D_{o',t'}}{\max\limits_{o' \in O, t' \in T} D_{o',t'} - \min\limits_{o' \in O, t' \in T} D_{o',t'}}$$

3) 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.

4) 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 allows us to understand how badly oracles miss their tier TAD/ATP targets during our observation period. The inaccuracy values increase when more transactions miss tier TAD/ATP, or when fewer do so, but more significantly.

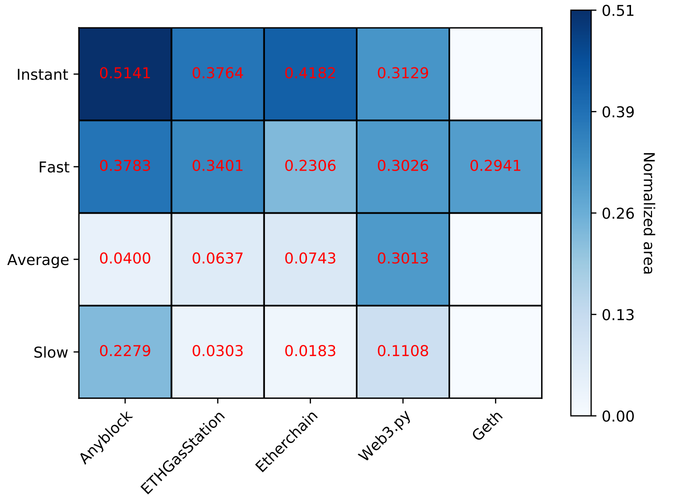
5) To compute oracle inaccuracy based on ATP, we simply replace TAD with ATP in the above steps.

Figure 4 shows the underpricing and overpricing inaccuracy for the different oracles and tiers. 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 4a and $i^+_{o,t}$ in Figure 4b.

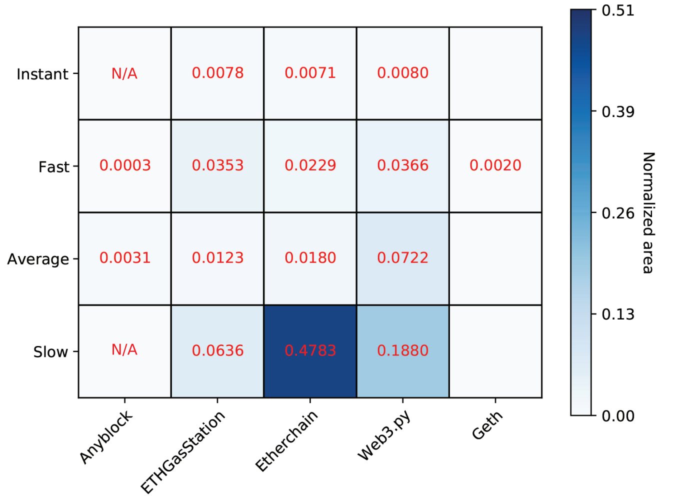
We observe in Figure 4a that the `instant` and `fast` tiers show the greatest degree of underpricing inaccuracy as they miss the target TAD and ATP most often and to the greatest extent. While the `average` and `slow` predictions generally make their TAD and ATP targets more often, the notable exceptions are Anyblock `slow` and Web3.py `average`.

We also observe in Figure 4b that, comparatively, overpricing is less of an issue with the exception being the Etherchain `slow` tier, where transactions could have made their target TAD at a lower price. Notable is the absence of overpricing for Anyblock `instant` and `slow` tiers for the observed transactions as shown by the N/A values.

These results indicate that ETHGasStation offers the best accuracy both in terms of avoiding underpricing and overpricing of transactions.



(a) Underpricing inaccuracy $i^-_{o,t}$.



(b) Overpricing inaccuracy $i^+_{o,t}$.

Fig. 4: Oracle inaccuracy.

D. *Gas Prediction Stability*

We also wanted to investigate the stability of gas price predictions over time. As discussed in Section II-A miners select the most highly valued transactions in their pools for inclusion into the next mined block. As such a transaction's probability for being accepted into a block depends on the percentile of its gas price with respect to other transactions in the pool. The question we want to answer is how does a transaction's price percentile in a pool changes as new transactions enter the pool and others are accepted into a block.

Figure 5 shows the gas price percentiles of Anyblock predictions in different tiers. The y-axis shows the price percentiles, while the x-axis shows the time elapsed since the initial prediction. We observe that as time elapses the initial percentile of a prediction changes. The change makes it more or less likely for the transaction to be accepted. Eventually around 92% (Figure 1 times Figure 2) of Anyblock transactions get accepted when the the percentile of their price gets high enough before the transaction is kicked out of the pool (percentile drops). Notice the rise of all percentiles over

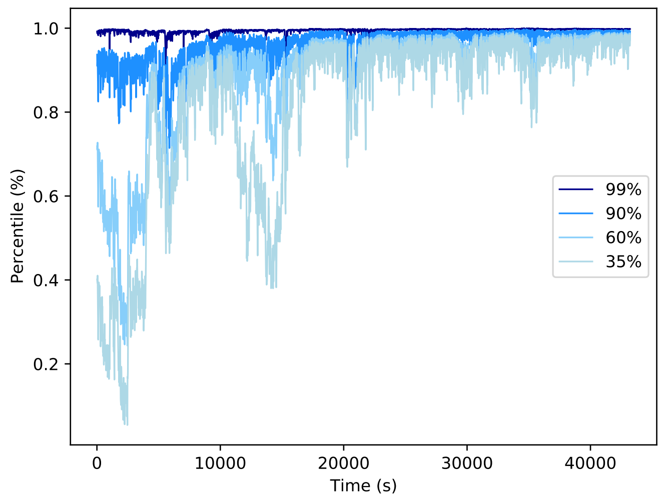


Fig. 5: Gas price percentiles in transaction pool over time.

	5%	10%	20%	30%
Transaction pool 0.35%	0:01:25	0:04:47	0:16:30	0:43:04
Transaction pool 0.60%	0:02:07	0:06:30	0:19:59	0:52:18
Transaction pool 0.90%	0:06:39	1:06:47	2:43:31	4:04:18
Transaction pool 0.99%	1:28:43	3:14:46	6:29:29	8:32:29

TABLE II: Average amount of time for a price percentile to change by a certain percentage in the transaction pool.

the observation period; even low priced transactions may be accepted into new blocks as new transactions are submitted at a lower price. This result shows that as new transactions enter the pool and others make the move from the pool into blocks the percentile of a prediction varies significantly over time.

Table II shows the time as hours:minutes:seconds of how long it takes for a transaction percentile in a miner pool (rows) to change by a number of percent (columns). To calculate these values, we averaged how long it takes for 500 different transaction percentiles to change by some percentage in the positive or negative direction as illustrated in Figure 5. We observe that lower price percentiles are less stable, which leads low price transactions to be removed at times from some miner pools, increasing their acceptance delay, or necessitating their resubmission.

Given the level of stability of transaction price percentiles in transaction pools, we wanted to compare it to the stability of predicted gas prices. Table III shows the time as hours:minutes:seconds of how long it takes for the percentile of an oracle/tier prediction price (rows) to change by a number of percent (columns). To calculate these values, we averaged how long it takes for 500 different predictions for each oracle and tier to change by some percentage in the positive or negative direction.

We observe that the ETHGasStation prices are the least stable (have the shortest durations for each percentage/tier), which means that the oracle adjusts them most frequently. Based on the results of Pierro et al. [6] showing that more frequent prediction lead to greater oracle accuracy, we believe that the greater predicted price volatility of ETHGasStation allows it to track percentile fluctuation in transaction pools

	5%	10%	20%	30%
Anyblock Slow	51:46:26	51:46:26	51:46:26	51:46:26
Anyblock Average	0:49:14	2:18:47	9:37:09	10:56:06
Anyblock Fast	0:46:41	1:21:21	6:31:36	15:54:56
Anyblock Instant	0:27:58	0:36:33	4:19:12	9:34:05
ETHGasStation Slow	0:10:25	0:12:50	0:17:10	0:17:22
ETHGasStation Average	0:18:00	0:32:47	1:15:24	1:17:49
ETHGasStation Fast	0:05:27	0:08:38	0:12:06	0:16:29
ETHGasStation Instant	0:07:15	0:11:42	0:13:48	0:16:15
Etherchain Slow	1:57:35	1:57:35	1:57:35	1:57:35
Etherchain Average	0:19:12	0:40:45	1:38:29	1:41:31
Etherchain Fast	0:18:26	0:44:35	4:50:17	15:44:28
Etherchain Instant	0:13:36	0:26:30	3:02:48	9:00:21
Web3.py Slow	0:40:49	1:45:36	4:26:39	14:52:12
Web3.py Average	0:56:08	1:32:37	10:59:57	21:11:10
Web3.py Fast	0:45:29	1:26:41	9:10:59	16:25:44
Web3.py Instant	0:37:38	1:37:43	7:36:12	16:45:15
Geth Fast	0:50:22	1:21:54	6:37:11	15:59:23

TABLE III: Average amount of time for each prediction and tier to change by a certain percentage.

more readily, and thus provide the most accurate predictions among the oracles. We also observe that the predictions for Anyblock and Etherchain `slow` change very rarely, which results in the same time for them to change by 5% through 30%. These tiers suffer from significant overpricing and underpricing respectively and could benefit from more dynamic adjustment of predicted price.

E. Factors Affecting Gas Prediction Accuracy

Finally, we wanted to extend the analysis of Pierro and Rocha to understand the factors that affect oracle accuracy. Pierro and Rocha computed Granger causality to detect the influence of environmental factors on gas price predictions [20]. We use a similar approach to understand the impact of these factors on the *accuracy* of gas price predictions.

Granger causality is a statistical test to determine whether one time series is useful in forecasting another. The Granger causality test may be used with statistically stationary time series, where statistical properties, such as mean, variance, etc. are constant over time. Pierro and Rocha use the Augmented Dickey-Fuller (ADF) to show that factors such as `transaction_count`, `net_hash_rate`, and `eth_usd` are stationary after first differentiation.

To confirm factor data stationarity, we first compute `transaction_count` based on the `Transaction` table in Table I for each 13 sec period. We also average `net_hash_rate` and `eth_usd` from the `Environment` table for each 13 sec period. We then perform the ADF test on these series and confirm their stationarity in our data. We also confirm that the underpriced and overpriced inaccuracy is also stationary after first differentiation.

With this verification in hand, we calculate Granger causality between the environmental factors and inaccuracy metrics and present the results in Tables IV and V. The left column shows the oracle and tier combination, while the top row shows the factors. We present statistically significant Pearson correlations in bold. For rows without entries we did not have sufficient data to calculate

	Tx. Count	Net. Hash Rate	ETH/USD
Anyblock Slow			
Anyblock Average	7.87E-01	2.32E-01	4.20E-01
Anyblock Fast	**2.42E-02**	3.24E-01	8.20E-02
Anyblock Instant	5.65E-01	**2.26E-02**	3.25E-01
ETHGasStation Slow			
ETHGasStation Average			
ETHGasStation Fast			
ETHGasStation Instant	7.32E-01	**8.76E-03**	1.02E-01
Etherchain Slow			
Etherchain Average			
Etherchain Fast			
Etherchain Instant	**2.95E-02**	2.29E-01	**3.16E-03**
Web3.py Slow			
Web3.py Average			
Web3.py Fast			
Web3.py Instant	4.80E-01	**1.27E-02**	5.51E-02
Geth Fast	**5.49E-04**	**4.10E-02**	1.83E-01

TABLE IV: Factors do not cause underpriced predictions.

	Tx. Count	Net. Hash Rate	ETH/USD
Anyblock Slow			
Anyblock Average	**2.22E-02**	**6.90E-03**	6.56E-01
Anyblock Fast	**8.84E-05**	2.12E-01	**1.03E-03**
Anyblock Instant			
ETHGasStation Slow	8.38E-02	2.78E-01	**2.12E-04**
ETHGasStation Average	4.48E-01	8.56E-02	2.21E-01
ETHGasStation Fast	1.59E-01	4.11E-01	2.01E-01
ETHGasStation Instant	**1.99E-02**	4.89E-01	**3.92E-02**
Etherchain Slow	7.70E-01	**2.66E-02**	1.77E-01
Etherchain Average	6.07E-01	4.88E-01	5.68E-01
Etherchain Fast	**1.18E-02**	6.19E-02	3.59E-01
Etherchain Instant	1.06E-01	8.71E-01	**3.45E-02**
Web3.py Slow	6.97E-01	2.18E-01	7.25E-02
Web3.py Average	2.22E-01	2.09E-01	5.69E-02
Web3.py Fast	6.03E-01	3.32E-01	1.90E-01
Web3.py Instant	3.93E-02	5.92E-01	**1.32E-02**
Geth Fast	**3.04E-03**	4.41E-01	3.32E-01

TABLE V: Factors does not cause overpriced predictions.

Granger causality – the `statsmodels.tsa.stattools.grangercausalitytests` returns `InfeasibleTestError` due to too many constant values in a series [27].

In general we observe that oracle accuracy tends to be most sensitive to the count of transactions in transaction pools, though the accuracy of most oracles and tiers cannot be predicted from the transaction counts. Interestingly, the accuracy of most of the oracle and tier combinations cannot be predicted from the price of Ethereum in USD, meaning that their accuracy is not affected by the changes in the value of cryptocurrency. Nevertheless the accuracy of some oracle and tier combinations do exhibit sensitivity to the price of Ethereum.

VI. Conclusions

In this paper we analyzed and compared the performance of Ethereum gas price oracles in terms of their accuracy, transaction accept rates, price stability, and sensitivity of prediction accuracy to environmental factors. Our analysis was based on a new oracle accuracy metric that captures both underpricing and overpricing of transactions. We observed that ETHGasStation produces the most accurate and stable price predictions. For users that prefer to run their own oracle Web3.py provides comparable performance. At the same time, the accuracy and transaction acceptance rates of existing oracles leave much to be desired and improved gas prediction mechanisms are needed for Dapps to provide predictable operational costs and performance.

References

[1] U. Klarman, S. Basu, A. Kuzmanovic, and E. G. Sire, "bloXroute: A Scalable Trustless Blockchain Distribution Network." https://bloxroute.com/wp-content/uploads/2019/11/bloXrouteWhitepaper.pdf, Nov. 2019.

[2] Blockchair, "Transaction Volume (ETH)." https://blockchair.com/ethereum/charts/transaction-volume-eth, Apr. 2021.

[3] W. 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, Feb. 2021.

[4] G. 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, Mar. 2021.

[5] J. E. de Azevedo Sousa, V. Oliveira, J. V. G. D. Gonçalves, S. M. Villela, H. S. Bernardino, and A. B. Vieira, "An analysis of the fees and pending time correlation in Ethereum," *International Journal of Network Management*, July 2020.

[6] G. Antonio Pierro, H. Rocha, R. Tonelli, and S. Ducasse, "Are the Gas Prices Oracle Reliable? A Case Study using the EthGasStation," in *IEEE Workshop on Blockchain Oriented Software Engineering (IWBOSE)*, Feb. 2020.

[7] F. Liu, X. Wang, Z. Li, J. Xu, and Y. Gao, "Effective GasPrice Prediction for Carrying Out Economical Ethereum Transaction," in *Dependable Systems and Their Applications (DSA)*, pp. 329–334, Jan. 2020.

[8] S. M. Werner, P. J. Pritz, and D. Perez, "Step on the Gas? A Better Approach for Recommending the Ethereum Gas Price," in *Mathematical Research for Blockchain Economy*, pp. 161–177, Oct. 2020.

[9] D. Carl and C. Ewerhart, "Ethereum gas price statistics," Tech. Rep. Working Paper No. 373, Social Science Research Network, Dec. 2020.

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

[11] Etherchain, "API endpoint." https://etherchain.org/api/gasPriceOracle, Accessed Apr. 2021.

[12] Anyblock, "API endpoint." https://api.anyblock.tools/ethereum/latest-minimum-gasprice/, Accessed Apr. 2021.

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

[14] "Go Ethereum." https://geth.ethereum.org/, Accessed Apr. 2021.

[15] ETHGasStation, "Gas price." https://docs.ethgasstation.info/gas-price, Accessed Apr. 2021.

[16] Etherchain, "Gas Price Oracle." https://etherchain.org/tools/gasPriceOracle, Accessed Apr. 2021.

[17] Anyblock, "Gas Price Prediction." https://www.anyblockanalytics.com/use-cases/gas-price-predictions/, Accessed Apr. 2021.

[18] Go Ethereum, "Command-line options." https://geth.ethereum.org/docs/interface/command-line-options, Accessed Apr. 2021.

[19] I. Weber, V. Gramoli, A. Ponomarev, M. Staples, R. Holz, A. B. Tran, and P. Rimba, "On Availability for Blockchain-Based Systems," in *IEEE Symposium on Reliable Distributed Systems (SRDS)*, Sept. 2017.

[20] G. A. Pierro and H. Rocha, "The Influence Factors on Ethereum Transaction Fees," in *Workshop on Emerging Trends in Software Engineering for Blockchain*, May 2019.

[21] H. J. Singh and A. S. Hafid, "Prediction of Transaction Confirmation Time in Ethereum Blockchain Using Machine Learning," in *Blockchain and Applications*, pp. 126–133, Jan. 2020.

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

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

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

[25] Etherscan, "Ethereum Developer APIs." https://etherscan.io/apis#stats, Accessed Apr. 2021.

[26] ETHGasStation, "gasstation-express." https://github.com/ethgasstation/gasstation-express-oracle, Mar. 2020.

[27] statsmodels, "grangercausalitytests." https://www.statsmodels.org/stable/generated/statsmodels.tsa.stattools.grangercausalitytests.html, Accessed Apr. 2021.