FFHS
Fernfachhochschule Schweiz
Zürich | Basel | Bern | Brig

swisscom | Blockchain

PREDICTING THE ETHEREUM GAS PRICE

SUPERVISORS

KAMAL YOUSSEFI
INDUSTRY LEAD ENTEPRISE BLOCKCHAIN
SWISSCOM BLOCKCHAIN AG
kamal.youssefi@swisscom.com

URS-MARTIN KÜNZI
SUPERVISOR TERM PAPER
FFHS
urs-martin.kuenzi@ffhs.ch

AUTHOR

MARIUS GIGER
PIBS 2015
SWISSCOM AG
marius.giger@swisscom.com

ZÜRICH, 14. JANUARY 2019

ABSTRACT

The rising success of cryptocurrencies is leading to an increased adoption of high-frequency wallets especially in the financial sector. This implies the need for low transaction fees. In this paper we conduct a scientific analysis on three different algorithms for estimating gas prices in Ethereum based on the implementations of go-ethereum (naïve), Ethereum gas station express and web3. By introducing a prediction score for an estimate, we show that all three algorithms allow good gas price estimates with the gas station express algorithm outperforming the naïve as well as the web3 algorithm in terms of both confirmation time and cost savings. We come to the conclusion that many fee estimation algorithms suffer from being complicated and are making use of unverified parameters and assumptions. Our results provide an important basis for understanding and further improving fee estimation methods.

TABLE OF CONTENTS

PREFACE

This section serves as an overview over the context of the paper, its aspirations and relevance.

A. Author

At the time of writing the author of this paper is employed at "Swisscom Blockchain AG", a Swisscom venture with the goal to get Switzerland up to speed in the blockchain space. The author is part of a SCRUM team developing a digital asset custody product for financial intermediaries to send Ethereum and other crypto-currencies. The problem of estimating and minimizing transaction fees is part of the solution and caught the author's attention.

B. Audience and Boundaries

The topics of this paper are rather advanced in the vast domain of blockchains hence basic knowledge about blockchain technologies is assumed. The following sources proved to be valuable in understanding the Bitcoin ecosystem and are recommended in case of uncertainties:

- the Ethereum Whitepaper by Vitalik Buterin [1]
- the Ethereum Yellow paper by Gavin Wood [2]

This term paper focuses exclusively on fees and transactions in Ethereum and will not cover other approaches (e.g. known from Bitcoin)

I. INTRODUCTION

Blockchain-technologies such as Ethereum and other crypto currencies (e.g. Bitcoin [3] and Zcash [4] etc.) are currently entering a more mature stage. However, the user experience is still lacking in several areas. One crucial part regarding payments is the optimization of transaction fees. In many cases the basic functionality for sending transactions is covered by the concrete wallet implementations (e.g. MyEtherWallet[1]) which often make use of suboptimal naïve approaches for estimating gas prices. This is suggested by the research of Hoffreumon and Zeebroeck who show that in 2018: *"over a period of two months users paid around 114,507.16 ETH or around 47M$ in fees for no additional service"* [5].
In this paper methods to estimate the current Ethereum gas price are challenged and improvements are discussed. In more detail the paper evaluates a method for estimating the gas price based on go-ethereum's *SuggestGasPrice* method[2] and compares it to the prediction model of Ethereum gas station[3] and the web3 estimation method[4]. Since these three approaches are widely used methods for estimating gas prices in Ethereum's community.

In the context of Swisscom Blockchain AG a more advanced approach to estimate transaction fees is a valuable optimization for a product currently in development used for financial intermediaries to control funds in Ether. The algorithms and findings discussed in this paper should subsequently be implemented for different financial products of Swisscom Blockchain AG.

The rest of this paper is structured as follows: Section II provides an overview over the Ethereum Blockchain. Section III presents previous work in this area of research. Section IV introduces different fee estimation algorithms. The results are reflected in section V and section VI and VII conclude the paper and identify areas for future work.

II. ETHEREUM

In contrast to Bitcoin's whitepaper [3] which only describes the general concepts, the Ethereum whitepaper [1] and the Ethereum yellow paper [2] provide an exact specification of the Ethereum protocol and its mechanisms. Ethereum can be treated as a distributed state-machine where transactions are incrementally executed against a genesis state which results in some final state. The authors refer to the state-machine as follows: *"In Ethereum, the state is made up of objects called "accounts", with each account having a 20-byte address and state transitions being direct transfers of value and information between accounts."* [1 p. 13] which means that the state cannot only include information about account balances but also contain information such as reputations, trust arrangements or anything else that can be represented by a computer. Hence, Ethereum proclaims to be Turing-complete.

---

[1] https://myetherwallet.com [Last access: 31.11.2018]
[2] https://github.com/ethereum/go-ethereum/blob/461291882edce0ac4a28f64c4e8725b7f57cbeae/ethclient/ethclient.go#L460 [Last access: 05.11.2018]
[3] https://github.com/ethgasstation/ethgasstation-backend [Last access: 05.11.2018]
[4] https://github.com/ethereum/web3.py/blob/master/web3/gas_strategies/time_based.py [Last access: 05.01.2019]

A. *Blockchain and Mining*

The Ethereum Blockchain is made up of blocks containing a list of transactions and an identifier for the most recent state. The blocks are chained together using a cryptographic hash. Furthermore, every block holds the information about its height and timestamp as well as several other attributes. To check if a block is valid a node has to validate that the previous block exists and is valid along with checking various low-level concepts such as the timestamp, block number, difficulty, proof of work and gas limit. Additionally, a validator has to check that the state transition from the state of the previous block to the next state was executed properly such that no invalid state change occurred (e.g. reducing an account balance without increasing the opposite balance elsewhere or a double-spend attack).

Similar to Bitcoin, Ethereum uses the Nakamoto protocol to reach consensus on the order of a set of transactions which is based on *Proof of Work* (PoW). However, this could change to *Proof of Stake* (PoS) in the near future with the recent introduction of Casper [6] - a hybrid model to introduce a PoS layer on top of a PoW system. The process of reaching consensus using PoW is commonly denoted as *mining*. Wood refers to it as follows: *"Mining is the process of dedicating effort (working) to bolster one series of transactions (a block) over any other potential competitor block. It is achieved thanks to a cryptographically secure proof."* [2 p. 2].

Since the Ethereum system is decentralized all parties using it are allowed to create new blocks which means that the resulting structure of the Blockchain is a tree. To make sure that the network participants form an agreement on which path from root (genesis block) to leaf (block containing the most recent transactions) is valid, Ethereum applies a simplified version of the GHOST rule introduced by Sompolinsky and Zohar [7]. To incentivize miners Ethereum uses transaction fees and pays a reward to the creator of a block.

B. *Currency and Wallets*

Ethereum's currency unit is called Ether and is identified by the shortcut "ETH". It can be subdivided into smaller units called Wei where $1\ Ether\ = 1 * 10^{18}\ Wei$. In terms of gas prices users often refer to GigaWei (GWei) which corresponds to 1 billion Wei or 1 nanoether.

For sending Ether a user has to possess a so-called *wallet* which is a software application providing a gateway to the Ethereum system. The wallet holds the keys of one or multiple accounts and is able to create and broadcast transactions on behalf of the user. Therefore, the wallet controls access to the user's money. Unlike Bitcoin, which applies a construct called "unspent transaction output" (UTXO) [3], Ethereum is based on accounts. Every account holds information about the balance, representing the amount of Ether available to a user. When a user wants to send Ether, she has to determine how much she is willing to pay for the transaction (transaction fee) to be processed contemporary. This is done by specifying a certain gas price and a gas limit.

C. *Transaction Fees*

In order to protect the Ethereum network from abuse a concept called *gas* is used. A real-world analogy for it would be fuel. Every computational step (e.g. creating a transaction, creating contracts or executing operations on the Ethereum virtual machine) uses a certain amount of gas which is a universally agreed cost in terms of gas.

The *gas limit* determines the maximum amount of gas a user is willing to use for a transaction and has to be set during the creation of a transaction. This ensures that for instance programming errors in smart contracts do not lead to infinite loops and therefore infinite transaction costs but rather to the transaction being cancelled preemptively once the limit is reached. All unused gas

is refunded to the sender at the end of a transaction. A standard transaction has a gas limit of 21000 gas.
The *gas price* determines the amount of pay per unit of gas. By increasing this value, a user can reduce the confirmation time for a transaction. This is because miners pick unconfirmed transactions from a pool and prioritize the ones with high fees. The total cost of a transaction is hence defined as follows:

$$fee = gasLimit * gasPrice$$

Fee estimation describes the process of determining a gas price which can consecutively be used to create a new transaction. This estimation is in most cases implemented by the wallet. If the gas price estimation is done right, the transaction fees can be reduced significantly, in particular for high-frequency wallets which process thousands of transactions per day. The gas price should be high enough for the transaction to be processed contemporary and low enough to not overpay the miners. Essentially, a user has different information sources to determine the gas price:

- **History:** every transaction in the blockchain contains the fee sent.
- **Mempool:** the current pool of waiting transactions.
- **Miner strategy:** assumptions about the miners' strategies to choose new transactions from the pool.

However, these information sources are quite difficult to use. Meaning that most users do not want to download the whole blockchain for such information. Therefore, many users rely on heuristics for determining the gas price using either an Ethereum client software such as *geth*[5] or a third-party website such as Ethereum gas station[6].
The implementation of a fee estimation algorithm is not trivial since there are multiple factors accounting for the current gas price, such as:

- **State of the mempool:** The amount of unconfirmed transactions in the current mempool and their respective gas price.
- **Miner strategy:** The miner prioritization algorithm for selecting the transactions for a block.
- **System complexity:** The difficulty of simulating the estimation due to the complexity of the system.

Since Ethereum is a smart-contract enabling blockchain (an example are ERC20 based tokens[7]), the users are additionally competing for computational resources through an auction based on the gas price [5]. Thus, also the computational effort for a transaction might have an influence on the prioritization. At this point it is unclear if the computational effort (number of operations) and the gas price correlate overlinearly - meaning that transactions with a high computational overhead having to provide a higher gas price than standard transactions.
It is also worth mentioning that neither a full node nor a third-party service can guarantee having an exhaustive list of pending transactions which is due to the nature of Ethereum being a decentralized network. For this reason, the transactions are broadcasted through the network in a peer-to-peer manner. Consequently, some latency might exist.

[5] https://github.com/ethereum/go-ethereum [Last access: 05.11.2018]
[6] https://ethgasstation.info [Last access: 05.11.2018]
[7] https://theethereum.wiki/w/index.php/ERC20_Token_Standard [Last access: 05.11.2018]

The last paragraph has introduced the difficulty of predicting a good gas price because of the various parameters accounting for it. As a result, for estimating an accurate gas price the network has to be monitored by inspecting the recent history of transactions and the current state of pending transactions.

III. Related Work

In [5] Hoffreumon and van Zeebroeck analyze the nature of the auction market of Ethereum and propose a new method based on classical financial forecasting using the ARMA and GARCH model. They show that such a method outperforms naïve bidding but are not giving any estimate on how good the method performs in comparison to other estimation algorithms. They point out the lack of research studying the nature of crypto-markets and conclude that many improvements can still be made.

Huberman et al. [8] analyze the economics of Bitcoin and capture their findings in a simplified economic model that provides answers to the underlying model for the determination of fees. They point out that *"transaction fees and infrastructure level are in fact determined in an equilibrium of a congestion queueing game derived from the system's limited throughput"* [8 p. 1]. Most importantly, they show that such a system requires congestion to raise revenue and fund infrastructure.

In the context of Ethereum, estimating gas prices is often times a left out point which means that there is a certain lack of profound literature researching it. To the best knowledge of the author there has been no scientific study on the different gas price estimation algorithms implemented by the most popular wallets.

IV. Gas Price Estimation

This section provides an overview over different algorithms to estimate the gas price.

A. Data collection

The data to carry out the gas price estimation was retrieved using a go[8] application querying a previously setup Ethereum node which was allowed to fully synchronize with the network. The application makes repeated calls to the JSON RPC interface of the go-ethereum blockchain client (called *geth*) to retrieve the latest blocks.

B. naïve estimation

The naïve algorithm (see algorithm 1 in the appendix) is based on the go-ethereum *SuggestGasPrice* method[9]. It loads the past 20 blocks and retrieves all the transactions with the respective gas prices. It then extracts and orders these prices and takes the value at the 60%-percentile which means a value that is slightly higher than the median gas price of the last blocks.

Analysis

The algorithm is resistant to adversarial conditions (e.g. against users paying exorbitant fees) or from outlier data (e.g. outliers caused by programming errors in a wallet implementation) because it does not take the average fee but relies on a value close to the median. Due to the low computational overhead of the algorithm, it can be recalculated for every block. However, the algorithm has the following limitations:

[8] https://golang.org [Last access: 19.12.2018]

[9] https://github.com/ethereum/go-ethereum/blob/master/eth/gasprice/gasprice.go [Last access: 19.12.2018]

- it relies solely on historical data, meaning that it does not consider the current state of pending transactions in the mempool
- it does not measure how full a block was
- it does not track how long the transactions have been in the mempool before they were included in the block
- it does not provide a measure on how fast a transaction with a given estimate will take to be confirmed and no way of controlling this parameter
- it is not clear whether a value of $k = 20$ past blocks used for the estimation is optimal for future predictions

C. Ethereum gasstation Express Estimation

The gas station algorithm (see algorithm 2 and 3 in the appendix) is based on the Ethereum gas station express estimation method[10]. It analyzes the minimum gas prices used over the last 100 blocks and provides gas price estimates based on a percentage of these blocks accepting this price. The algorithm returns different suggestions which can be used to control the confirmation time of a newly created transaction:

- **SafeLow** (35% of blocks accepting this price): This gas price is below the average. It is intended to be both cheap and successful but will need a longer time for confirmation (usually confirms in less than 30 minutes).
- **Standard** (60% of blocks accepting this price): This gas price is above the average. It is a safe default and has a short confirmation time (usually confirms in less than 5 minutes).
- **Fast** (90% of blocks accepting this price): Transactions with this gas price should be accepted in the next few blocks (usually confirms in less than 1 minute).
- **Fastest** (maximal minimum price): Transactions with this gas price should be accepted as fast as possible. Paying more than this price is unlikely to increase transaction confirmation time. Additionally, using this value most probably overpays the miners.

Analysis

Similar to the first algorithm this algorithm is resistant to adversarial conditions. It relies on minimum gas prices accepted by a certain percentage of blocks. The algorithm gives the user a way to control the confirmation time of a newly created transaction. Due to the high computational overhead for retrieving data the algorithm should use a cache of the last 100 blocks to be recomputed for every block. The algorithm still has certain limitations:

- it relies solely on historical data, meaning that it does not consider the current state of pending transactions
- it does not track how long the transactions have been in the mempool before they were included in the block
- it does not measure how full a block was
- it is not clear whether a value of $k = 100$ past blocks is needed for future predictions

[10] https://github.com/ethgasstation/gasstation-express-oracle/blob/master/gasExpress.py [Last access: 19.12.2018]

D. Web3 Estimation

The web3 approach[11] uses data from recently mined blocks. But in contrast to the previous algorithms it allows users to determine a time span x in seconds for the transaction to be mined with a probability of p. With these two parameters the algorithm allows to return different suggestions which can be used to control the confirmation time of a newly created transaction. The following parameters are used by web3:

- **slow** ($p = 98, x = 60 * 60$): This gas price has a confirmation time of about 1 hour with a probability of 98 percent.
- **standard** ($p = 98, x = 60 * 10$): This gas price has a confirmation time of about 10 minutes with a probability of 98 percent.
- **fast** ($p = 98, x = 60$): This gas price has a confirmation time of about 1 minute with a probability of 98 percent.
- **glacial** ($p = 98, x = 60 * 60 * 24$): This gas price has a confirmation time confirmation time of about 24 hours with a probability of 98 percent.

Analysis

Similar to the first and second algorithm this algorithm is resistant to adversarial conditions by relying on low percentile gas prices. It makes use of timing assumptions using the average block time to estimate the wait time for a certain gas price. Interestingly, the algorithm groups transactions by the miner instead of the blocks possibly allowing to control malicious network participants. Using the control parameters, it gives the user a way to determine the confirmation time of a newly created transaction. Due to the high computational overhead the algorithm should use a cache of the last blocks to be recomputed for every block. The algorithm still has certain limitations:

- it relies solely on historical data, meaning that it does not consider the current state of pending transactions
- it does not track how long the transactions have been in the mempool before they were included in the block
- it does not measure how full a block was
- it is not clear whether the grouping by the miner has a positive impact

E. Prediction Score

To determine the goodness of a certain estimate a measure was introduced to evaluate the prediction. The *prediction score* was defined as follows (algorithm 5), where $0 \leq score \leq$

Algorithm 5: calculate prediction score

input : *block* := A block with transactions. *estimate* := An estimated gas price.
output: *score* := The score for *estimate* for *block*.

```
1 block.Transactions ← block.Transactions.OrderBy(tx => tx.GasPrice)
2 for i ← 0 to block.Transactions.length()-1 do
3 |   tx ← block.Transactions[i]
4 |   if tx.GasPrice > prediction then
5 |   |   score← (1.0 − (i/block.Transactions.length())) ∗ 100.0
6 |   |   return
```

Algorithm 5 - Prediction scores

[11] https://github.com/ethereum/web3.py/blob/master/web3/gas_strategies/time_based.py [Last access: 05.01.2019]

100. The prediction score is a measure that determines the percentage of transactions having a bigger gas price in a certain block. 0 means that all transactions in the block have a lower gas price than the estimate, 100 means that all the transactions have a bigger gas price. Consequently, a lower score means more chance for a transaction using an estimate to be included in this block. The score was calculated for the subsequent 10 blocks for each prediction.

V. Results

We have fully implemented the introduced algorithms in go[12] and analyzed their performance subsequently using a python script[13]. In this section we examine the estimations and the respective prediction scores and provide a comparison of the algorithms. The sampling of the algorithms was carried out over a timespan of 500 blocks. During this period for every block a gas price prediction was made using the different algorithms.

F. Estimations

The following illustration shows the predicted gas prices in GWei for the sampling period of 500 blocks. The web3 as well as the express estimation appear to be rather constant while the naïve estimation is more fluctuant.

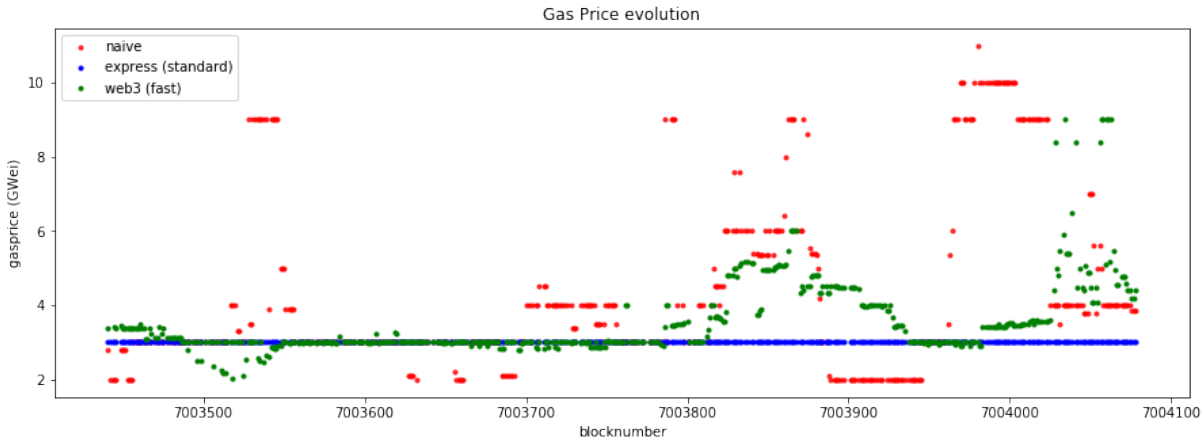

Figure 1 - gas price evolution

Using an exchange rate of 1 $ETH = 146.31\$$[14] and the following formula $fee = gasLimit * gasPrice$ the cost savings can be calculated. On average an amount of 0.0038$ using the express estimation or 0.002$ using the web3 estimation can be saved in terms of fees compared to the naïve estimation for sending a standard transaction[15]. Resulting in a cost reduction of 41.6% for the express method and a cost reduction of 19.9% for the web3 algorithm compared to the naïve algorithm.

[12] https://github.com/swisscom-blockchain/ethereum-feeestimator [Last access: 03.01.2019]
[13] https://github.com/mariusgiger/ethereum-feeestimator-analysis [Last access: 03.01.2019]
[14] https://www.coindesk.com/price/ethereum [03.01.2019 20:00]
[15] gasLimit of 21000

	naive	**express**	**web3**
average gas price	4.247490648328032	3.0	3.543043615823182
std. deviation of average price	2.31	0.0	1.03
average fee for a standard transaction	0.013$	0.009$	0.011$

Table 1 - comparison gas prices

G. Scores

To investigate the goodness of a gas price estimation the previously introduced prediction scores were calculated for every estimate for the next 10 blocks.

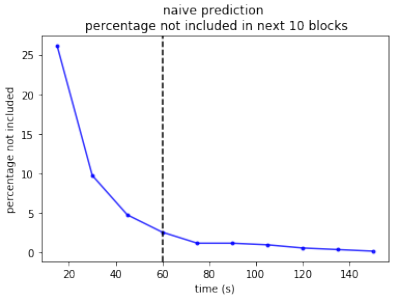

Figure 2 - naive predictions of next 10 blocks (score < 100)

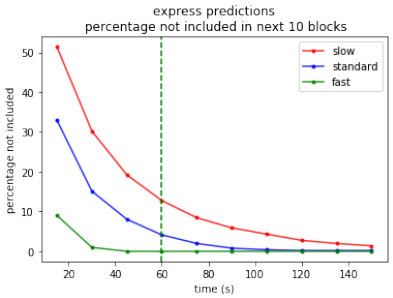

Figure 3 - express predictions of next 10 blocks (score < 100)

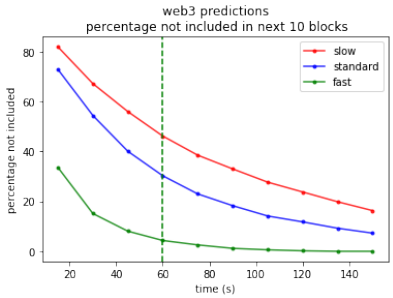

Figure 4 - web3 predictions of next 10 blocks (score < 100)

According to figure 2 the naïve estimation provides a gas price that after 75 seconds (~5 blocks[16]) would have been included with almost 100% chance. It is important to mention that this means that certain transactions with the predicted gas price would have been included but it does not indicate the amount of transactions that would have been included with this price. Thus, there is no guarantee that a particular transaction would have been included. As well for the standard gas price of the express method (fig. 3) and the fast prediction of the web3 algorithm (fig. 4) an estimate is produced that is included in 5-6 blocks. This suggests that the premises of the web3 algorithm that transactions with a fast prediction are included within 1 minute are in most cases met. Looking at the different estimations of the gas station algorithm reveals that the gas price predictions for slow, standard and fast are all being included within under 3 minutes (~10 blocks) which means that the parameters for this estimation algorithm are not very well distributed since all the estimates are included pretty fast whereas the web3 approach allows to control the confirmation times much better. Still, the gas station algorithm proved to be a little bit more efficient in terms of cost savings compared to the web3 estimation method as was indicated in the previous section.

[16] assuming an average block time of 15 seconds

When comparing the three algorithms in terms of prediction scores figure 5 shows that all algorithms produce results that are included within the next 80 seconds with a very high chance.

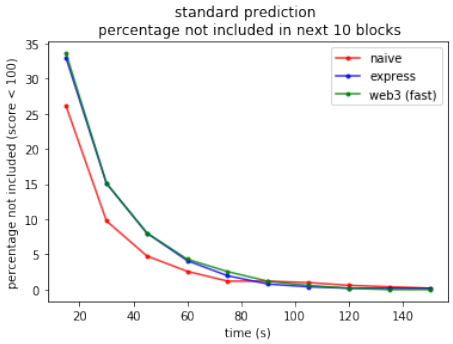

Figure 5 - comparison standard prediction (score < 100)

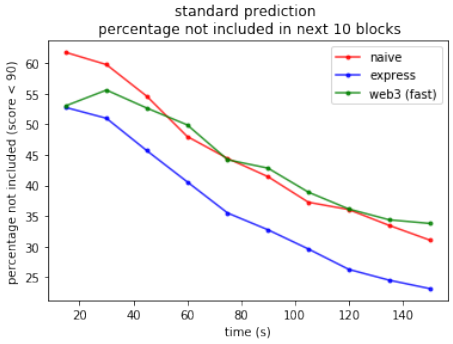

Figure 6 - comparison standard prediction (score < 90)

Figure 6 shows the percentage of blocks in which a given estimate could take more than 10% of space which indicates that the gas station express algorithm provides estimates that are included much faster with a much higher chance than both web3 and the naïve algorithm while still having the cheapest average gas price as is shown in table 1.

VI. Discussion

In this paper we have analyzed three different algorithms for estimating gas prices in Ethereum based on the algorithms of go-ethereum, Ethereum gas station express and web3. The evaluation of the data presented in this work leads to the observation that all three algorithms allow good gas price estimates with the gas station express algorithm outperforming the naïve as well as the web3 algorithm in terms of both confirmation time and cost savings.

From the short review of the algorithms in section IV, the following key findings emerge:

- **Complexity**: Many of the existing algorithms are rather complicated and not formally documented. They make certain assumptions such as the grouping by miner of the web3 algorithm or the structure of the hashpower table of the gas station express algorithm that are neither explained nor publicly tested. Furthermore, they make use of various parameters such as the number of inspected blocks or the percentile for gas prices that were in the best case retrieved via trial and error.
- **Initial data/Hybrid model**: Algorithms such as the gas station algorithm or the web3 algorithm need some time to warm up. For a productive scenario the use of a hybrid model falling back on naïve estimation in case not enough data is available would be suited best.
- **Gas price history**: The results demonstrate that looking at the history of recent blocks, their transactions and the associated gas prices provides a good measure for estimating a gas price. However, including the information about the current state of the mempool at time of prediction could most probably further improve the estimates.

When comparing our results to those of Hoffreumon and Zeebroeck [5], who showed that users paid 47M$ in fees for no additional service within a period of two months in 2018, it must be pointed out that the extremely small savings in terms of money by using the express or web3 estimation algorithm instead of the naïve algorithm suggest that many users don't even use a naïve estimation for determining the gas price but instead make use of a much higher gas price. We speculate that this might be explained due to the fact that *gas, gas limit and gas price* are

rather abstract concepts and difficult to understand for non-technical users. Therefore, the ease of use of the gas price estimation is of primary importance in a wallet application.

One concern about the findings of the prediction scores is that the score does not indicate if a particular transaction would be included, only that certain transactions are included with a gas price less or equal to the estimated gas price. Future research should therefore further develop the concept of scores to include a probability that a certain transaction for a given estimate will be included in a block. However, this requires that the mempool is monitored at every prediction. Another limitation of this paper involves the relatively small sample size of 500 blocks which made the differences between the algorithms clear but may exclude special cases such as the behavior in times of heavy network congestion.
Although the algorithms analyzed in this paper are widely used, they remain not the only ones in the vast space of cryptocurrencies. It will be important that future work investigates other algorithms such as the standard gas station algorithm[17] which has high similarities to the express algorithm but implements some more subtleties or fee estimation algorithms from other cryptocurrencies such as Bitcoin and compares it to this work.
As also recommended above, future investigations should include the use of the current state of pending transactions (mempool) to further improve the estimation results. However, this will include another layer of complexity, which implicates a profound implementation of the basic fee estimation algorithm beforehand. Additionally, other parameters such as the selection strategy of miners should be evaluated. Most importantly for non-standard transactions that cause a computational overhead.

VII. Conclusion

In this paper we have analyzed fee estimation algorithms based on go-ethereum, gas station express and web3. We have evaluated their performance with a newly introduced prediction score and compared their results. We show that the web3 algorithm provides very good control parameters and that the gas station algorithm makes slightly better predictions on average in terms of both confirmation time and cost savings compared to the naïve and the web3 estimation method. We point out that many algorithms suffer from being very complicated which makes an extensive analysis difficult.
This work only scratches on the surface of fee estimation. Despite the limitations the results are valuable in understanding the properties of fee estimation algorithms and provide a profound overview on three go-to choices for fee estimation. We hope to have shed some light into the fee estimation of Ethereum and hope to see our findings reflected in the next generation of fee estimation algorithms.

[17] https://github.com/ethgasstation/ethgasstation-backend [Last access: 03.01.2019]

APPENDIX

The source code created for this paper can be found in the following repositories:

- Algorithms: https://github.com/swisscom-blockchain/ethereum-feeestimator [Access on demand: Marius.Giger@Swisscom.com]
- Analysis: https://github.com/mariusgiger/ethereum-feeestimator-analysis

A. *Naïve Algorithm*

Algorithm 1: naive fee estimation

input : The last k := 20 blocks *Blocks* with transactions
output: The estimated gasPrice *gp*

```
1  Retrieve the last k blocks including their transactions.;
2  gasPrices ← [];
3  for i ← 0 to k do
4      transactions ← Blocks[k].Transactions;
5      for j ← 0 to transactions.length() do
6          gasPrices.Add(transactions[j].gasPrice);
7  Sort(gasPrices);
8  percentile← 60;
9  index← (gasPrices.length()-1)*percentile / 100;
10 gp← gasPrices [index];
```

Algorithm 1 - Naive estimation

B. *Gasstation Express Algorithm*

Algorithm 2: Ethereum gasstation estimation

input : The last k := 200 *blocks* with transactions
output: The estimated gas price $safeLowPrice$, $standardPrice$, $fastPrice$, $fastestPrice$

```
1  Retrieve the last k blocks including their transactions.
2  predictionTable ← MakePredictionTable(blocks)
3
4  safeLow ←predictionTable.Where(p => p.HashPowerAccepting > 35)
5  standard ←predictionTable.Where(p => p.HashPowerAccepting > 60)
6  fast ←predictionTable.Where(p => p.HashPowerAccepting > 90)
7  hpaMax ←Max(predictionTable.Select(p => p.HashPowerAccepting))
8  fastest ←predictionTable.Where(p => p.HashPowerAccepting == hpaMax)
9
10 safeLowPrice ←Min(safeLow) / 10
11 standardPrice ←Min(standard) / 10
12 fastPrice ←Min(fast) / 10
13 fastestPrice ←Min(fastest) / 10
```

Algorithm 2 - Gas station express estimation

Algorithm 3: MakePredictionTable() for Ethereum gasstation estimation

input : A number of *blocks* with transactions
output: The *predictionTable* for estimating gas prices

```
minGasPrices ← []
Extract the minimum gas price of every block.
foreach block ∈ blocks do
    txs ← block.Select(b => b.Transactions)
    minGasPrice ← txs.OrderBy(t => t.GasPrice).First()
    minGasPrices.Add(minGasPrice)
Split the gas prices into buckets of 10 Gwei.
gpGroups ←minGasPrices.GroupBy(g => RoundToTenGwei(g))
gpGroups ←gpGroups.OrderBy(g => g.Key)
gpGroupCounts ←gpGroups.Select(g => g.Length())
cumulativeSums ←CumulativeSum(gpGroupCounts)
totalBlocks ←Sum(gpGroupCounts)

Calculate hashtable.
hashpower ← []
for i ← 0 to cumulativeSums.Length() do
    hashPct ←cumulativeSums [i] / totalBlocks * 100
    hashpower.Add(HashPct : hashPct, Price : gpGroups[i].Key, Count :
     gpGroupCounts[i])
predictionTable ← []
for i ← 0 i < 10; i = i + 1 do
    predictionTable.Add(Price: i)
for i ← 0 i < 1010; i = i + 10 do
    predictionTable.Add(Price: i)
foreach val ∈ predictionTable do
    Get the hashpower accpeting the gas price.
    prices ← hashpower.Select(g => g.Price)
    hpas ←
     hashpower.Where(g => g.Price >= val.Price).Select(g => g.HashPcts)

    if val.Price > Max(prices) then
        val.HashPowerAccepting ← 100
    else if val.Price < Min(prices) then
        val.HashPowerAccepting ← 0
    else
        val.HashPowerAccepting ← Max(hpas)
← predictionTable
```

Algorithm 3 - gas station express estimation: make prediction table

C. *Web3 Algorithm*

Algorithm 4: web3j fee estimation

input : $maxWaitSeconds := 60$ Desired maxiumum number of seconds a transaction should take to mine. $sampleSize := 120$ Number of recent blocks to sample. $p := 98$ Desired probability that the transaction will be mined within $maxWaitSeconds$.

output: $gp :=$ The estimated gasPrice

```
avgBlockTime ← GetAvgBlockTime(sampleSize)
waitBlocks ← math.Ceil(maxWaitSeconds / avgBlockTime)
Retrieve the last k := sampleSize blocks including their transactions.
txs ← GetTxsForBlocks(sampleSize)
groupedByMiner ← txs.GroupBy(tx => tx.Miner)
minerData ← []
foreach group ∈ groupedByMiner do
    miner ← group.Key
    blocks ←GetBlockCount(group)
    gasPrices ←group.Select(tx => tx.GasPrice)
    pricePercentile ← gasPrices[(gasPrices.length() − 1) ∗ 20/100]]
    minGasPrice ←Min(gasPrices)
    minerData.Add(Miner: miner, Blocks: blocks, GasPrice: minGasPrice,
      LowPercentileGasPrice: pricePercentile)
minerData ←minerData.OrderBy(m => m.LowPercentileGasPrice)
probabilities ← []
foreach idx, miner ∈ minerData do
    Compute probability that a tx will be accepted at a gasprice by the miner.
    blocksAccepting ← minerData[idx:].Select(m => m.Blocks).Sum()
    invProbPerBlock ← (sampleSize − blocksAccepting)/sampleSize
    probabilityAccepted ← 1−math.Pow(invProbPerBlock, waitBlocks)
    probabilities.Add(Probability: probabilityAccepted, GasPrice:
      miner.LowPercentileGasPrice)
first ←probabilities [0]
last ←probabilities [probabilities.length()-1]
if p >= first.Probability then
    gp ← first.GasPrice
    return
else if p <= last.Probability then
    gp ← last.GasPrice
    return
for i ← 0 to probabilities.length()-1 do
    left ← probabilities[i]
    right ← probabilities[i + 1]
    if p < right.Probability then
        continue
    adjProb ← p − right.Probability
    windowSize ← left.Probability − right.Probability
    position ← adjProb/windowSize
    gasWindowSize ← left.GasPrice − right.GasPrice
    gp ←math.Ceil(right.GasPrice + gasWindowSize ∗ position)
    return
gp← -1
```

Algorithm 4 - web3 estimation

REFERENCES

[1] V. Buterin, “A Next-Generation Smart Contract and Decentralized Application Platform,” 2014.

[2] G. Wood, “Ethereum: A secure decentralised generalised transaction ledger.,” 2014.

[3] S. Nakamoto, “Bitcoin: A peer-to-peer electronic cash system.,” 2008.

[4] D. Hopwood, S. Bowe, T. Hornby, and N. Wilcox, “Zcash Protocol Specification,” pp. 1–53, 2017.

[5] C. Hoffreumon and N. Van Zeebroeck, “Forecasting short-term transaction fees on a smart contracts platform,” 2018.

[6] V. Buterin and V. Griffith, “Casper the Friendly Finality Gadget,” pp. 1–10, 2017.

[7] Y. Sompolinsky and A. Zohar, “Secure High-Rate Transaction Processing in Bitcoin,” in *International Conference on Financial Cryptography and Data Security*, 2015, pp. 507–527.

[8] G. Huberman, J. D. Leshno, and C. C. Moallemi, *Monopoly without a monopolist: An economic analysis of the bitcoin payment system.* 2017.