The Gas Triangle and Its Challenges to the Development of Blockchain-Powered Applications

Gustavo A. Oliva
gustavo@cs.queensu.ca
SAIL Lab - Queen's University
Kingston, Ontario, Canada

Ahmed E. Hassan
ahmed@cs.queensu.ca
SAIL Lab - Queen's University
Kingston, Ontario, Canada

ABSTRACT

Ethereum is the most popular blockchain platform for the development of blockchain-powered applications (a.k.a, ĐApps). Developing a ĐApp involves translating requests captured in the frontend of an application into contract transactions. However, transactions need to be payed for. Ethereum employs the gas system to charge transaction fees. The gas system has three key components, namely gas price, gas usage, and gas limit. We refer to these components and their interplay as the gas triangle. In this paper, we claim that the inherently complex gas triangle should *not* be exposed to end-users. We conduct two studies that provide empirical evidence to support our claim. In light of our results, we provide a list of recommendations to novice end-users. We conclude the paper with a list of research challenges that need to be tackled in order to support the development of next-generation ĐApps that completely hide the gas triangle from end-users.

CCS CONCEPTS

• **Human-centered computing** → **Interaction paradigms**; • **Computer systems organization** → **Distributed architectures**.

KEYWORDS

Ethereum, Decentralized Applications, gas system, cost-effective

ACM Reference Format:
Gustavo A. Oliva and Ahmed E. Hassan. 2021. The Gas Triangle and Its Challenges to the Development of Blockchain-Powered Applications. In *Proceedings of the 29th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '21), August 23–28, 2021, Athens, Greece.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3468264.3473130

1 INTRODUCTION

Since the introduction of Bitcoin, several other blockchain platforms have been developed. Some of these new blockchains are called *programmable blockchains*. In addition to supporting cryptocurrency transactions, programmable blockchain also host and execute *smart contracts*. Smart contracts are simply stateful, general-purpose computer programs. The functions defined in these smart contracts are invoked via transactions known as *contract transactions*. Most importantly, programmable blockchains enable the development of blockchain-powered applications. Currently, the most popular programmable blockchain is Ethereum [3]. The cryptocurrency in Ethereum is called Ether (ETH). In the Ethereum community, blockchain-powered applications are known as *Decentralized Applications*, or simply, ĐApps. Transactions in blockchain platforms commonly need to be paid for. Ethereum uses the gas system to charge transactions fees.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
ESEC/FSE '21, August 23–28, 2021, Athens, Greece
© 2021 Association for Computing Machinery.
ACM ISBN 978-1-4503-8562-6/21/08…$15.00
https://doi.org/10.1145/3468264.3473130

2 THE GAS TRIANGLE

The payment system of Ethereum is complex. As opposed to typical cloud services in which users choose a compute capacity and pay by the hour (*pay-as-you-go*), transaction fees in Ethereum do not have a fixed value. Instead, transaction fees are calculated using a *gas system*. This gas system has three key components: *gas usage*, *gas limit*, and *gas price*. We refer to these components and their interplay as the *gas triangle* (Figure 1). In the following, we describe each key component of the gas triangle:

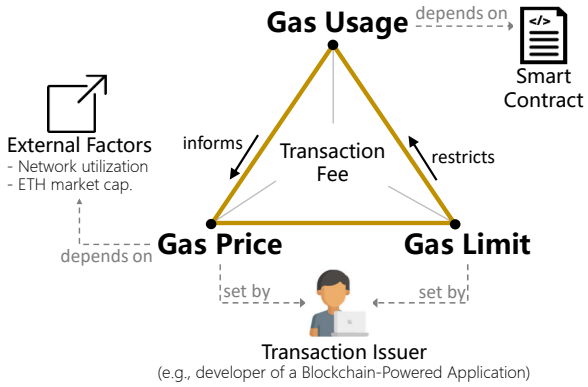


Figure 1: The gas triangle.

• **Gas Usage.** Gas is a measurement unit for computational work in Ethereum. A cryptocurrency transaction burns a fixed number of gas units. Conversely, the amount of gas units that are burnt by a contract transaction depends on the *number* and *type* of bytecode instructions that are executed during runtime in order to fulfil such a transaction. From a practical perspective, the gas usage of a contract transaction depends essentially on which contract function is being invoked, the values of the input parameters, and the state of the contract (i.e., attribute values).

• **Gas Price.** The gas price is a transaction parameter that needs to be set by the transaction issuer for every transaction. The gas price corresponds to the amount of cryptocurrency (Ether) that one

is willing to pay for each unit of burnt gas. Assume that a contract transaction burns 30,000 gas units. If a transaction issuer sets a gas price of 2 GWEI (10^{-9} ETH), then the final transaction fee is going to be 30,000 * 2 GWEI = 60,000 GWEI. In other words, the transaction fee corresponds to *gas usage* × *gas price*. Miners, who are the entities that effectively select and process transactions, get rewarded by transaction fees. Hence, the higher one sets the gas price of a transaction, the more eager miners will be to pick such a transaction, ultimately resulting in a faster transaction processing time. Gas price also largely depends on *external factors*, such as the utilization level of the blockchain network (e.g., when the network is highly congested, prices tend to increase) and the ETH market capitalization (e.g., as proxified by the ETH to USD exchange rate).

• **Gas Limit.** The gas limit is another transaction parameter that needs to be set by transaction issuers for every transaction. This parameter serves as a limiter for the transaction fee. For instance, assume that a smart contract function executes a `for` loop with 10 iterations. Now also assume that such a function has a bug in it, which sometimes causes the `for` loop to be run with 1 million iterations. If a contract transaction triggers this buggy behavior, then the transaction issuer would need to pay a massively high transaction fee due to the abnormally high gas usage. The gas limit ensures that a smart contract transaction can burn at most x units of gas. In other words, the transaction fee is at most *gas limit* × *gas price*. If a transaction needs more gas than the specified limit, then such a transaction fails with an out-of-gas error and gets rolled back (but the transaction issuer still pays the fee, since miners had to process the transaction).

Figure 1 illustrates the interplay between the components of the gas triangle. The gas limit *restricts* the gas usage. The gas usage, in turn, *informs* the gas price. Moreover, as we discussed above, some components of the gas triangle have dependencies to other elements. The gas usage depends on the smart contract being invoked. The gas price depends on external factors (e.g., network congestion). These dependencies influence the gas components themselves, which, in turn, influence the interplay again. Hence, the gas triangle renders the Ethereum transaction payment system complex.

3 ĐAPPS AND THE GAS TRIANGLE

Nowadays, most of the ĐApps expose the complexity of the gas triangle to end-users. Developers of these ĐApps commonly instruct end-users to install and use a wallet. A wallet is an application (typically a web browser extension) that enables end-users to send transactions to a blockchain without having to set up their own node. In the context of ĐApps, a wallet acts as a mediator, sitting between the end-user and the smart contract(s) of the ĐApp. More specifically, when an end-user triggers a functionality in the frontend of the ĐApp, the wallet pops up and requests the end-user to fill in the gas parameters of the transaction. Wallets typically offer suggested values for gas price and gas limit. Developers of ĐApps can also override these suggestions that are shown to end-users. Metamask and MyEtherWallet are two popular Ethereum wallets.

The complexity of the gas triangle and its intricacies should *not* be pushed to end-users (especially novice ones) and wallets. For example, if end-users override the wallet-suggested gas limit with an improper value (i.e., one that is too low), the transaction will fail with an out-of-gas error. Such an error might be a big enough reason for the end-user to stop using the ĐApp. In addition, if end-users set an improper gas price, they might also get frustrated. If the gas price is set too low, transactions will possibly take days to finish, compromising the end-user experience and yielding frustration. If the gas price is set too high, the end-user might get the feeling that using ĐApps is too costly. Finally, and more generally, exposing the gas triangle to end-users implies that they have to be acquainted with the transaction processing dynamics in a blockchain. That is, the gas triangle yields a substantial, unnecessary cognitive overhead on end-users, which only hampers the wide and successful adoption of otherwise potentially transformative applications.

> *The gas triangle and its inherent complexity should *not* be exposed to end-users of ĐApps.*

4 EMPIRICAL EVIDENCE

The previous section warns against exposing the gas triangle to end-users. In this section, we provide supporting empirical evidence to our claim by investigating how end-users set up the gas parameters (i.e., gas price and gas limit) of transactions.

4.1 Gas Price Study

In this section, we show that the manner in which novice blockchain end-users set the gas price of transactions is suboptimal compared to that of experienced end-users. More specifically, we leverage the ledger nature of a blockchain and conduct a historical comparison of the gas prices set by novice and experienced end-users. In the following, we detail our data collection approach, our gas price comparison approach, and our results.

Data Collection. Our data collection steps are as follows:

(1) Analysis period. We study the *Byzantium* era of Ethereum, which started on Oct 16th 2017 and lasted until Feb 28th 2019. Byzantium was the first Ethereum version to have substantial transactional activity, thus attracting several new users to the platform.

(2) Data Source. We collect data from the public Ethereum dataset on Google BigQuery (https://bit.ly/3ePGxiF). This dataset is an easy-to-consume dump of the historical data stored in the Ethereum blockchain.

(3) Collected pieces of data. We collect transaction metadata, which includes the transaction issuer id, the gas price, and the transaction *nonce*. The transaction *nonce* value records the number of transactions previously sent by the transaction issuer. This parameter exists to preserve transaction ordering and we use it as proxy for user experience. More specifically, for a given transaction t, we say that the issuer of t is a *novice* if the transaction's *nonce* ≤ 4 (i.e., t is one of the initial five transactions submitted by the issuer). In turn, we consider the issuer of t to be *experienced* when *nonce* ≥ 49 (i.e., one becomes experienced from the submission of their 50th transaction). We exclude the nonce interval [5,48] to avoid introducing noise in our data.

(4) Data filtering. We apply the following data filtering steps in order to enable a fair comparison between prices of novices and experienced end-users:

• Controlling for the studied contract functions. Different smart contracts provide different functionalities. As a consequence, contracts likely have different user bases. Contracts are also created at different points in time, where the impact of external factors

can be substantially different. As such, it does not make sense to compare gas prices sent to different smart contract functions. Therefore, we control for smart contract functions when comparing the prices of novice and experienced users. During Byzantium, 11,622 functions received 90% of the contract transactions. These functions comprise our initial set of studied functions.

• Exclusion of unsuitable functions. It is unfeasible to compare the prices of novice and experienced users for functions with a very small userbase. Hence, we exclude functions whose ratio of *unique* transaction issuers is below 1%. This filter also ensures that we select functions that are designed to be executed by end-users and not smart contract developers themselves (see Section 6). We also exclude functions that only have either novice or experienced end-users. Lastly, we exclude functions that received less than 10 transactions by either novice or experienced end-users.

• Exclusion of highly engaged end-users. If an end-user sends an unusually high number of transactions to a certain function, then we assume that such engagement is likely to influence his/her pricing behavior. Therefore, for every studied function, we determine the number of transactions sent by each end-user, calculate the 95th percentile of this distribution, and remove the end-users above such percentile. By doing it, we avoid having gas prices of experienced end-users being biased towards a very small set of core users.

Approach for gas price comparison. Our hypothesis is that novices tend to choose wallet-suggested gas prices for their transactions more often than experienced end-users. We expect that experienced users will "tweak" wallet-suggested prices or even rely on other sources of information to make more informed gas price choices. More concretely, we expect that the gas prices chosen by novices should be more repetitive (less surprising) than those chosen by experienced end-users. This notion of surprisingness is captured by a measure known as *entropy*. The entropy of a random variable can be interpreted as the average level of "surprise" inherent in the variable's possible outcomes.

We proceed as follows. First, we select the transactions submitted by experienced users, group these transactions according to their targeted function, and calculate the Normalized Shannon Entropy (NSE) over the gas prices of each function f (entropyExperienced(f)). When all the values of a given distribution X are distinct (i.e., they all have the same probability of occurring), maximum NSE is achieved (one). Conversely, if all the elements of X are identical, minimum NSE is achieved (zero). We then proceed analogously for transactions submitted by novices (entropyNovice(f)). Finally, for each studied function f, we calculate the following ratio:

$$\text{entropyRatio}(f) = \frac{\text{entropyExperienced}(f)}{\text{entropyNovice}(f)}$$

An entropyRatio(f) above 1 indicates that the gas prices set by experienced users is less repetitive (i.e., higher entropy) than those set by novice users for a given function f.

Result 1) Experienced end-users interact with the blockchain in a more cost-effective manner compared to novices. Figure 2a depicts the entropyRatio(f) for all studied functions. Analysis of the figure indicates that the gas price entropy of experienced end-users is indeed typically higher than that of novices. We also perform a two-tailed Wilcoxon signed-rank test ($\alpha = 0.05$) to compare the gas price entropy of novice and experienced end-users. The result indicates that the difference is statistically significant (p-value < 2.2e-16). In order to understand the practical significance of this difference, we compute the Cliff's Delta effect-size score. We obtained a *medium* (0.33) effect size [4].

In other words, experienced end-users are more selective in choosing the gas prices of their transactions. Given the symbiosis between the components of the gas triangle, choosing gas prices more selectively optimizes the balance between transaction cost and transaction processing speed. As a result, experienced users interact with the blockchain in a more *cost-effective* manner.

4.2 Gas Limit Study

A transaction runs out of gas when the gas limit is insufficient to process the whole transaction. In this section, we conduct a study in which we show that the chances that a contract transaction will run out of gas varies between novice and experienced users. In the following, we detail our data collection approach, our gas limit comparison approach, and our results.

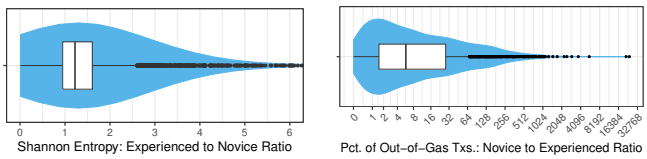
Data Collection. We use the same data as that of the previous study, including the filtering steps. The only exception is the filtering step (c), which is only applicable in the gas price study.

Approach for gas limit comparison. We select the transactions submitted by novices, group these transactions according to the targeted function, and calculate the percentage of out-of-gas transactions of each function f (percentageOutOfGasNovice(f)). Next, we proceed analogously for experienced users. Finally, for each function f, we calculate the following ratio:

$$\text{outOfGasRatio}(f) = \frac{\text{percentageOutOfGasNovice}(f)}{\text{percentageOutOfGasExperienced}(f)}$$

An outOfGasRatio(f) above 1 indicates that novice users run into out-of-gas errors more frequently than experienced users for a given function f.

Result 2) Novice end-users run into out-of-gas errors more frequently than experienced users. Figure 2b depicts the distribution of outOfGasRatio(f) for all studied functions. We note that, for the vast majority of functions (82.1%), the outOfGasRatio(f) is higher than one. The median of the distribution is also high (5.8), which indicates that novices run into out-of-gas errors remarkably more frequently than experienced users for several functions. To better understand the difference between the percentages of out-of-gas transactions issued by novice and experienced end-users, we perform a two-tailed Wilcoxon signed-rank test ($\alpha = 0.05$) and compute the Cliff's Delta effect size. The results indicate that the difference between the distributions is statistically significant and that the effect size is *large* (0.53).



(a) entropyRatio(f) **for all studied functions f (trimmed at 6).** **(b)** outOfGasRatio(f) **for all studied functions f (log1p scale)**

Figure 2: Comparison between novice and experienced end-users with relation to the setup of gas parameters.

5 RECOMMENDATIONS TO NOVICE USERS

Our results reiterate the idea that exposing the gas triangle to end-users should be avoided. However, we acknowledge that this is the current ĐApp development paradigm and that at least 2,800 ĐApps have already been deployed in Ethereum [5] (at the time of writing this paper). Therefore, in order to help out novice (and prospective) end-users of Ethereum ĐApps, we derive the following list of practical recommendations:

• **ĐApp developers should provide didactic information regarding the gas triangle.** In particular, ĐApp developers should make their end-users aware that (i) transactions need to be paid for, (ii) the final transaction fee is a function of gas usage and gas price, (iii) only the latter in under their control, and (iv) higher gas price frequently means faster processing time. We encourage ĐApp developers to provide a "Getting started" section in the documentation of their application, which should target mainly novices.

• **Novice end-users should beware of gas price pitfalls.** We highlight two gas price pitfalls. First, *end-users should beware of wallet suggested gas prices.* Modern wallets typically suggest gas prices for different transaction processing times. However, these suggested prices can be manipulated. For instance, ĐApp developers can overwrite gas price suggestions in Metamask [1]. More generally, we encourage novices to make informed gas price choices instead of blindly accepting the wallet's suggested price. This can be done by leveraging services such as the EthGasStation (https://ethgasstation.info). Second, *end-users should know that there are no constraints for gas prices in Ethereum* (i.e., as long as there is enough account balance, the transaction can be submitted). Recently, a mistake made by a developer of a large crypto exchange resulted in a transaction being submitted with an abnormally high gas price, which resulted in a transaction fee of approximately US$80,000 [2]. Therefore end-users should watch out for typos when manually inputting a gas price, as transactions cannot be rolled back in Ethereum.

• **Novice end-users should rarely change the wallet-suggested gas limit for a transaction.** Given the current advances in wallet development, novice end-users can somewhat safely rely on the gas limit suggested by wallets. Novices are thus better off not changing the gas limit manually. As a precaution measure, we advise novices to set a low gas price when issuing their first transaction towards a given contract function (e.g., to mitigate monetary losses in case of an out-of-gas error).

6 RETHINKING THE DESIGN OF ĐAPPS

The exposure of the gas triangle to end-users only hampers a wide and successful adoption of otherwise potentially transformative and disruptive blockchain-powered applications.

The *next-generation* of ĐApps should completely hide the gas triangle and its intricacies from end-users. We advocate that the burden of setting appropriate gas parameters should be left entirely to the developers of ĐApps. In other words, for all practical purposes, end-users should have no awareness that a blockchain platform is the backend of an application. We highlight that such a paradigm change does not hinder blockchain transparency (e.g., auditability), since platforms such as Ethereum record the gas usage, gas limit, and gas price associated with every processed transaction.

Despite the clear need for a paradigm change, developing next-generation ĐApps is far from trivial [1]. When engineering a next-generation ĐApp, developers need to translate requests captured in the frontend of an application into one or more contract transactions. A key challenge faced by developers of these applications is ensuring that gas prices are set in such a way that the balance between transaction processing times (which influence the end-user final experience) and transaction costs (which derive from transaction fees) is optimized. That is, the challenge is to develop *cost-effective* ĐApps in face of the complexity of the gas triangle [6].

In a nutshell, next-generation ĐApps should be preferred over the current development paradigm where the gas triangle is exposed to end-users. Yet, developing cost-effective next-generation ĐApps is remarkably challenging. We foresee the need for software engineering research in the following topics:

• **Assessment of state-of-the-practice prediction models for transaction processing times.** Developers of next-generation ĐApps should strive to set gas prices as low as possible without compromising transaction processing times. Currently, wallets and services such as EthGasStation provide estimates of processing time. Future research should evaluate the accuracy of these prediction models (e.g., in face of drastic changes in external factors (Figure 1)) and also investigate whether more accurate models can be designed.

• **Assessment of the trade-offs between gas optimizations and code readability.** As depicted in the gas triangle (Figure 1), gas usage influences the transaction fee. Developers of next-generation ĐApps can reduce costs derived from transaction fees by implementing gas-optimized smart contracts. In a recent survey with developers, Zou et al. [7] observed that gas optimizations (e.g., the use of assembly constructs) are not trivial to implement and hampers code readability. Further research in gas usage estimation, gas optimization, and the trade-off between gas optimization and code readability should be carried out.

• **Verification of next-generation ĐApps.** As we mentioned before, next-generation ĐApps translate requests captured in the frontend into one or more contract transactions. Hence, such a translation needs to be verified (in addition to testing the smart contract itself). In particular, verification should ensure that the gas parameters of these transactions are set adequately (e.g., gas price is within a pre-specified range).

REFERENCES

[1] Cryptokitties Team. 2017. CryptoKitties birthing fees increases in order to accommodate demand. https://bit.ly/3wVzvPK. (2017). [Online; accessed 01-December-2019].
[2] Kaitlyn Kubrick. 2020. Mystery of $80,000 transaction fee resolved in Ethereum. https://bit.ly/3kLUp1m. (2020). [Online; accessed 15-March-2020].
[3] Gustavo A. Oliva, Ahmed E. Hassan, and Zhen Ming (Jack) Jiang. 2019. An Exploratory Study of Smart Contracts in the Ethereum Blockchain Platform. *Empirical Software Engineering* (2019).
[4] J. Romano, J.D. Kromrey, J. Coraggio, and J. Skowronek. 2006. Appropriate statistics for ordinal level data. In *Annual meeting of the Florida Association of Institutional Research*. 1–3.
[5] State of the DApps. 2020. State of the DApps - Blockchain Apps for Ethereum. https://bit.ly/36YYUh4. (2020). [Online; accessed 02-May-2020].
[6] Abdullah A. Zarir, Gustavo A. Oliva, Zhen M. (Jack) Jiang, and Ahmed E. Hassan. 2021. Developing Cost-Effective Blockchain-Powered Applications: A Case Study of the Gas Usage of Smart Contract Transactions in the Ethereum Blockchain Platform. *ACM Transactions on Software Engineering and Methodology* (2021).
[7] W. Zou, D. Lo, P. S. Kochhar, X. D. Le, X. Xia, Y. Feng, Z.Chen, and B. Xu. 2019. Smart Contract Development: Challenges and Opportunities. *IEEE Transactions on Software Engineering* (2019).