A Systematic Approach To Cryptocurrency Fees

Alexander Chepurnoy[1,2], Vasily Kharin[3], Dmitry Meshkov[1,2]

[1] Ergo Platform
[2] IOHK Research
{alex.chepurnoy, dmitry.meshkov}@iohk.io
[3] Helmholtz Institute Jena, Froebelstieg 3, 07743, Jena, Germany
v.kharin@protonmail.com

Abstract. This paper is devoted to the study of transaction fees in massively replicated open blockchain systems. In such systems, like Bitcoin, a snapshot of current state required for the validation of transactions is being held in the memory, which eventually becomes a scarce resource. Uncontrolled state growth can lead to security issues. We propose a modification of a transaction fee scheme based on how much additional space will be needed for the objects created as a result of transaction processing and for how long will they live in the state. We also work out the way to combine fees charged for different resources spent (bandwidth, random-access state memory, processor cycles) in a composite fee and demonstrate consistency of the approach by analyzing the statistics from Ethereum network. We show a possible implementation for state-related fee in a form of regular payments to miners.

1 Introduction

Bitcoin [16] was introduced in 2008 by S. Nakamoto as a purely peer-to-peer version of electronic cash with a ledger written into blockchain data structure securely replicated by each network node. Security of the cryptocurrency relies on its mining process. If majority of miners are honest, then Bitcoin meets its security goals as formal analysis [10] shows. For the work done a miner is claiming a reward which consists of two parts. First, some constant number of bitcoins are created out of thin air according to a predefined and hard-coded token emission schedule. Second, a miner claims fees for all the transactions included into the block.

As shown in [7] constant block rewards is an important part of the Bitcoin protocol. Once a predetermined number of coins will enter a circulation and miners will be rewarded by transaction fees only, their rational behavior could be different from the default mining strategy. It is still an open question whether Bitcoin will meet its security goals in such circumstances, but at least number of orphaned blocks will increase making Bitcoin less friendly for regular users.

A transaction fee, which is set by a user during transaction creation, is useful to limit miners resource usage and prevent spam. In most cases a user pays a fee proportional to transaction size, limiting miners *network* utilization. A

rational miner does not include all the valid transactions into blocks as, due to the increased chances of orphaning a block, the cost of adding transactions to a block could not be ignored [3,18]. As shown in [18], even in the absence of block size limit Bitcoin fee market is healthy and the miner's surplus is maximized at a finite size of a block. Thus miners are incentivized to produce blocks of a limited size, so only transactions providing enough value to a miner will be included in a block. The paper [18] provides a procedure to estimate transaction fee based on block propagation time.

Besides network utilization, transaction processing requires a miner to spend some *computational* resources. In Bitcoin the transactional language[4] is very limited, and a number of CPU cycles needed to process a transaction is strictly bounded, and corresponding computational costs are not included in the fee. In contrast, in cryptocurrencies supporting smart contract languages, such as Solidity [1] and Michelson [13], transaction processing may require a lot of computations, and corresponding costs are included in transaction fee. Analysis of this fee component is done for concrete systems in [8,14], and is out of scope of this paper.

In this work we address a problem of miners *storage* resources utilization. A regular transaction in Bitcoin fully spends outputs from previous transactions, and also creates new outputs of user-defined values and protecting scripts. A node checks a transaction in Bitcoin by using a set of unspent outputs (UTXO). In other cryptocurrencies a representation of a *state* needed to validate and process an arbitrary transaction could be different (for example, in Ethereum [22] such structure is called the *world state* and fixed by the protocol). To process a transaction quickly, the state (or most accessed part of it) should reside in expensive random-access memory (RAM). Once it becomes too big to fit into RAM an attacker can perform denial-of-service attacks against cryptocurrency nodes. For example, during attacks on Ethereum in Autumn, 2016, an attacker added about 18 million accounts to the state (whose size was less than 1 million accounts before the attack) and then performed successful denial-of-service attacks against the nodes [20]. Similarly, in 2013 a denial-of-service attack against serialized transactions residing in a secondary storage (HDD or SSD) was discovered in Bitcoin [19].

In all the cryptocurrencies we are aware of, an element of the state once created lives possibly forever without paying anything for that. This leads to perpetually increasing state (e.g. the Bitcoin UTXO size [6]). Moreover, state may grow fast during spam attacks, for example, 15 million outputs were quickly put into the UTXO set during spam attacks against Bitcoin in July, 2015 [5], and most of these outputs are not spent yet. The paper [17] is proposing a technical solution for non-mining nodes where only miners hold the full state (assuming that they can invest money in random-access memory of sufficiently large capacity), while other nodes are checking proofs of state transformations generated by miners, and a size of a proof (in average and also in a worst case) is about $log(|s|)$, where $|s|$ is a state size. Nevertheless, big state could lead to centralization of mining or SPV mining [2], and these concerns should be addressed.

The question of internalizing the costs of state load was raised in [15], but to the best of our knowledge there has not been any practical solution proposed yet. Also, there is an increasing demand to use a blockchain as a data provider, and permanently storing objects in the state without a cleaning procedure in such a case is not a viable option.

1.1 Our contribution

In this paper we propose an economic solution to the problem of unreasonable state growth (such as spam attacks, or objects not being used anymore but still living in the validation state). The solution is a new mandatory fee component. A user should pay a fee based on both the additional space needed to store objects created by a transaction, and also for lifetime of the new bytes. This model is typical for cloud storage services where users pay for gigabytes of data per month.

We also consider an approach to combine fees charged for different resources consumed by a transaction: bandwidth, random-access memory to hold state, and processor cycles to process computations prescribed by the transaction. We propose to charge only for a resource which is consumed most of all, so we can talk about storage-oriented, network-oriented or computation-oriented transactions. We provide an evaluation of Ethereum usage data which shows that it is possible for this cryptocurrency to determine transaction type.

We propose a convenient way to charge for state memory consumption (considering output lifetime also). Our approach is convenient for users who do not know for how long they would like to store their outputs in the system. The approach is called "scheduled payments", as we propose to charge periodically for the bytes of memory consumed.

1.2 Structure of the paper

The paper is organized as follows. We put assumptions behind our model and its analysis into Section 2. Then we provide an algorithm for a composite fee assignment in Section 3. We propose a possible approach to charge for state memory consumption in Section 4. We show results of Ethereum data evaluation in Section 5. We conclude with Section 6.

2 Preliminaries

We shape our model with the following assumptions:

– without loss of generality, we assume throughout the paper that a transaction creates new objects called outputs and spends outputs from previous transactions. Thus the state needed for transaction validation is about a set of outputs yet unspent. The size of the state then is the sum of sizes of all the unspent outputs.

– we assume that a transaction does not change size of the state significantly
– for an implementation, we assume that it is always profitable for a miner to collect fees from unspent outputs.
– we are considering minimal mandatory fees in the paper. All the nodes are checking that a fee paid by a transaction is not less than a minimum and rejecting the whole block if it contains a transaction violating fee rules. Thus a fee regime is considered as a part of consensus protocol in our work. A user can pay more than the minimum to have a higher priority for a transaction of interest to be included into a block.

3 An algorithm for the fee assignment

As mentioned in the introduction, we develop a fee regime with two goals in mind, namely incentivization of miners and spam prevention. In this chapter we reason about the guiding principles for the fee assignment and end up with an example of a practical fee assignment rule.

The evolution of blockchain networks has demonstrated the main resources being used. First and the most important so far, the memory of network nodes is limited resource. Blocks in the blockchain after processing are stored in a secondary storage, where a cost of a storage unit is low. In contrast, to validate a transaction, some state is needed (for example, unspent outputs set in Bitcoin is used to validate a transaction), and this state should reside in expensive random-access memory.

Second, it is obvious, especially with the development of smart contracts, that a cost to process a transaction can be more than just a storage cost: transactions can contain relatively complicated scripts which are meant to be executed by all the nodes in the network. The most famous example is the Ethereum network implementing the concept of a "world computer" [22].

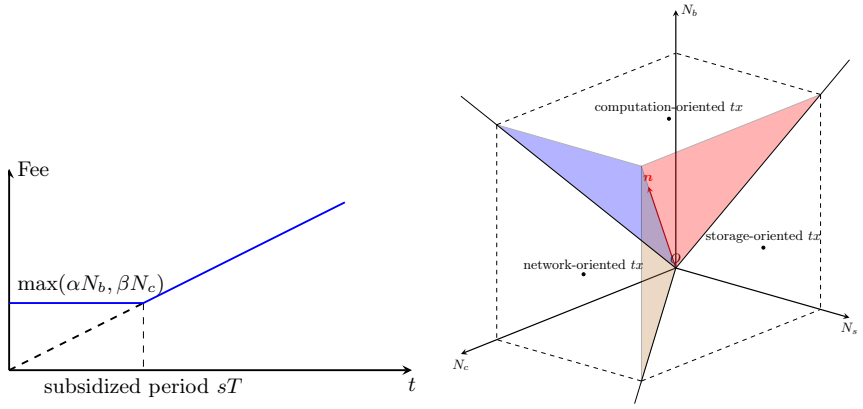
Third, there is the network load created by every transaction. If an output is created in one block and spent right in the next one, it provides almost zero overhead in terms of validation state size, but creates the network load needed for synchronization.

A transaction fee should incorporate all the three components stated above. As shown in [8], assigning the fee to the storage as if it was execution of some code can lead to significant disbalance for rich enough scripting language (for example, for the data being written with an opcode other than conventional storage one). Thus, we propose to charge for a component which demands more resources. That is, storage-oriented transactions should be charged for state memory consumption, the computation-oriented transactions should be charged for script execution, and all the others by the network load. This can be formalized as follows:

$$\mathrm{Fee}(tx) = \max\left(\alpha \cdot N_b(tx), \beta \cdot N_c(tx), S(state) \cdot \sum_i (B_i \cdot L_i)\right). \qquad (1)$$

Here α and β are the pricing coefficients, $N_b(tx)$ is transaction size which defines the network load, $N_c(tx)$ is the estimation of the computational cost of transaction, $S(state)$ is the cost of the storing one byte in the state for the unit of time (a block), L_i is the time for which the output i is being stored in the state, and B_i is its size in bytes.

Since the time for the data to reside in the state is usually unknown, the third argument of $max()$ in Eq. (1) cannot be deduced directly at transaction submission time. For this purpose we introduce a proposal for scheduled payments later in Section 4. The third argument in Eq. (1) becomes dominant with time. Starting at a moment sT since transaction when it happens, the fee is increasing at a constant rate (see Fig. 1a). The possible implementation of this algorithm is described in Section 4.



(a) Transaction cost as a function of the output existence time.

(b) Space of transactions split by Eq. (1) into the subregions of the dominant fees.

Fig. 1: Fee differentiation by resource consumption

The obvious questions here are as follows. What are guiding principles for choosing α, β and $S(\cdot)$? There is also the question of estimating $N_c(tx)$, which can be solved only by executing the script for Turing–complete languages. It is known as the worst case execution time problem [21]. We leave the latter question beyond the scope of the paper, and answer the former one below.

3.1 Choice of the relative values of α, β, $S(state)$

Assume for now that for every transaction we know for how long its outputs will be stored in the state. We will overcome this difficulty later. Based on Eq. (1), we come up with the notion of space of transactions, which is three–dimensional in our case — every transaction is defined by three numbers: $N_b(tx)$, $N_c(tx)$,

$N_s(state, tx) = \sum_i (B_i \cdot L_i)$. Eq. (1) is splitting this space into three regions: network–oriented transactions, space–oriented transactions, and computation–oriented transactions (see Figure 1b). All the splitting is governed by the direction of vector $\boldsymbol{n}$ which defines the line $\alpha N_b = \beta N_c = N_s$. Varying the coefficients α and β, one can change the direction of $\boldsymbol{n}$ adjusting the formal fee prescription to the sensible values.

3.2 Choice of $S(state)$

The simplest way of assigning the $S(state)$ value is by making it constant. However, this does not fully solve the problem of limiting the state size. What is being controlled in this case, that is the rate at which the data is being submitted, but not the state size itself. One could also manually define the maximal size of the state for the network. This solution, in turn, has its own caveats. For example, once the state is kept (almost) full by the participants, it can be (almost) impossible to submit the transaction increasing the state size. The time till it becomes possible is hardly predictable.

Preferable properties of the current state size could be formulated as follows: it should be predictable, stable, and below some externally given value (an upper bound on state size, being unique for the whole network).

Another natural question arising is whether the rigid state size restriction is necessary? It is easy to imagine the situation where the formal possibility of exceeding the state is still present, but hardly ever being used. For example, if one wants to constrain the state size to 10MB, the possible solution is to set normal price for submitting data to store if the state size after submission is below 10MB, but some astronomical price for the luxury of storage above 10MB. So, formally it will be possible, but in fact, hardly ever used, with every usage bringing significant profit to miners. The generalization of this idea is to form the explicit dependence of price on the state load (it will referred to as "pricing curve"). A good pricing curve should provide at least one stable equilibrium of the state size; the minimal dependence on initial conditions (if possible), and high rewards for miners. The latter could serve as good optimization parameter. Extreme cases are zero price with huge data submission and miners get nothing; and infinite price with zero data submission and miners get nothing. As usual, a maximal outcome is in between. The pricing policies described above are two particular cases of pricing curve (see Figure 2). That is, we assume that the price of data storage in the state $S(state)$ varies with the current state load x.

Note that the pricing curve is defined by a small number of parameters and to be the same for all the network. To impose an upper bound on the state size, one can choose the pricing curve formally going to infinity at some finite state size. The rigid boundary can be provided by divergence higher than $1/(x_{max} - x)$. One can also try to estimate the optimal state size for a given differentiable pricing curve. The data submission rate $N(S(x))$ is fully defined by the current storage price $S(x)$. Rewards rate obtained by the miners for stable state size at price S is given by $y = S \cdot N(S)$. An example is provided in Figure 3. First, it provides a possible method of measuring explicit form of the function $N(S)$ in

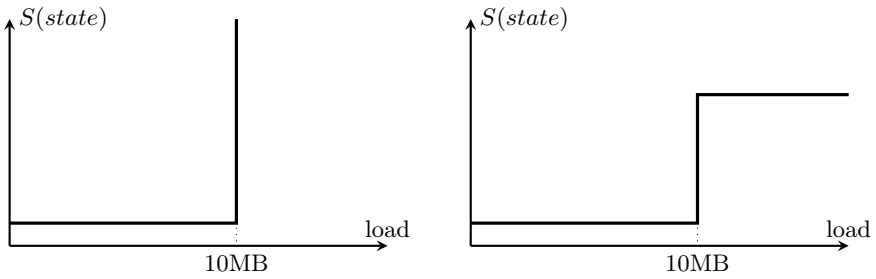


Fig. 2: Examples of pricing curves: rigid state size restriction (left) and overflow fees (right, see text). The value of 10MB is taken arbitrarily.

the model: one has to set up the price, and observe the static rewards. Second, one may wonder about the price S^*, optimal for the miners in terms of rewards. Obviously, it satisfies $N(S^*) + S^* \cdot N'(S^*) = 0$, where prime is derivative with respect to price. As usual, the optimal price here does not depend on the pricing policy, but rather the implicit property of the network. Having the price varying freely can be considered beneficial both for miners and for network as a whole, since it allows the first ones to optimize signing strategy, and herewith the state size is automatically adjusted to the relatively predictable level $S^{-1} \cdot S^*$.

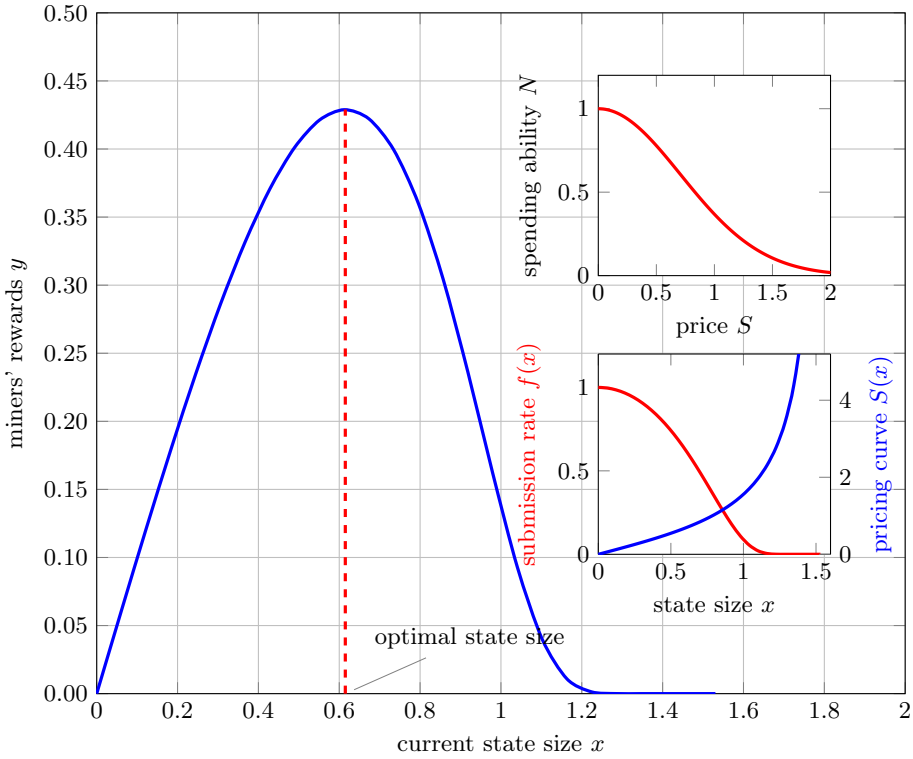


Fig. 3: Example of the rewards curve.

4 Scheduled Payments

In this section we propose a concrete way to charge for state bytes consumed (or released). There are few possible options for that. A user may, for example, specify lifetime for a coin during its creation and pay in advance, this is not very convenient for him though. Another option is to charge when the coin is being spent, or allow to spend a coin (by anyone, presumably, a miner) when its value is overweighted by the state component of the fee. As a drawback, if coin is associated with a big value, it could live for very long time, maybe without a reason.

We propose more convenient method of charging; we name it *scheduled payments*. In this scheme a user must set special predefined script for a coin (otherwise a transaction and also a block containing it are invalid), which contains a user-specific logic (we call it a *regular script*) and a spending condition which allows anyone (presumably, a miner) to create a transaction claiming this output, necessarily creating a coin with the same guarding condition and a value not less than original minus the state fee. These two parts (regular script and a fee charging condition) are connected by using the $\vee$ conjecture. We assume

that α and β are fixed. We also assume that subsidized period sT is to be stored along with the coin by each validating node. Then a guarding script for the coin would be like:

$$\begin{aligned}&(regular\_script)\vee\\&(height > (out.height + sT) \wedge (out.value \leq S_c \cdot B \cdot sT \vee\\&\quad tx.has\_output(value = out.value - S_c \cdot B \cdot sT, script = out.script))),\end{aligned} \tag{2}$$

where *height* is a height of a block which contains a spending transaction; *out.height* is a height when the output was created; *out.height* and *out.script* contain value and spending script of the output, respectively; $tx.has\_output()$ checks whether a spending transaction has an output with conditions given as the predicate arguments, and S_c is the value of $S(state)$ when the coin created. As in Section 3, constant B is the output size.

5 Evaluation

In this section we experimentally study what could be the real-world ratio between the pricing coefficients $\alpha, \beta, S(state)$. To extract the realistic values, and to verify validity of described transaction classification, we use data from the Ethereum network. We consider Ethereum as a good example, since all the three fee components are presented in this cryptocurrency. The network load parameter $N_b(tx)$ is simply a transaction size; the state load $\Delta(tx)$ can be deduced from the blockchain by extracting SSTORE and CREATE operations from the transaction tx [4]. To determine the computational load $N_c(tx)$, we count Ethereum gas consumed by transaction processing minus its storage cost and so-called base cost, which is proportional to the transaction size.

The results of processing first $2{\cdot}10^6$ blocks in Ethereum network are presented in Figure 4. Each point corresponds to a transaction. One can notice that parts of the distribution in Figure 4 extend along the coordinate axis; these are the transactions which can be unambiguously distinguished by their type of the resource consumption. Their presence confirms our expectations on the nature of resource consumption, and serves as a justification of the proposed classification scheme. The space of transactions is split into three parts by the aforementioned vector $\mathbf{n}$ with the endpoint at the first momentum of the transactions with at least 2 non-zero components.

Another parameter of interest is a storage object lifetime. Associate it directly with smart contract data lifetime is weakly relevant to our scheme as the users are not incentivized to remove data from the state earlier rather than later. Thus we consider the delay between the data submission and a first request to be the reliable parameter reflecting the needs of the users. Analysis of Ethereum blockchain shows that in a lot of cases data stored in the world state is touched by other transactions in the same block or few blocks after creation. We filter

[4] Information on these operations can be found in the Ethereum Yellow Paper [22].

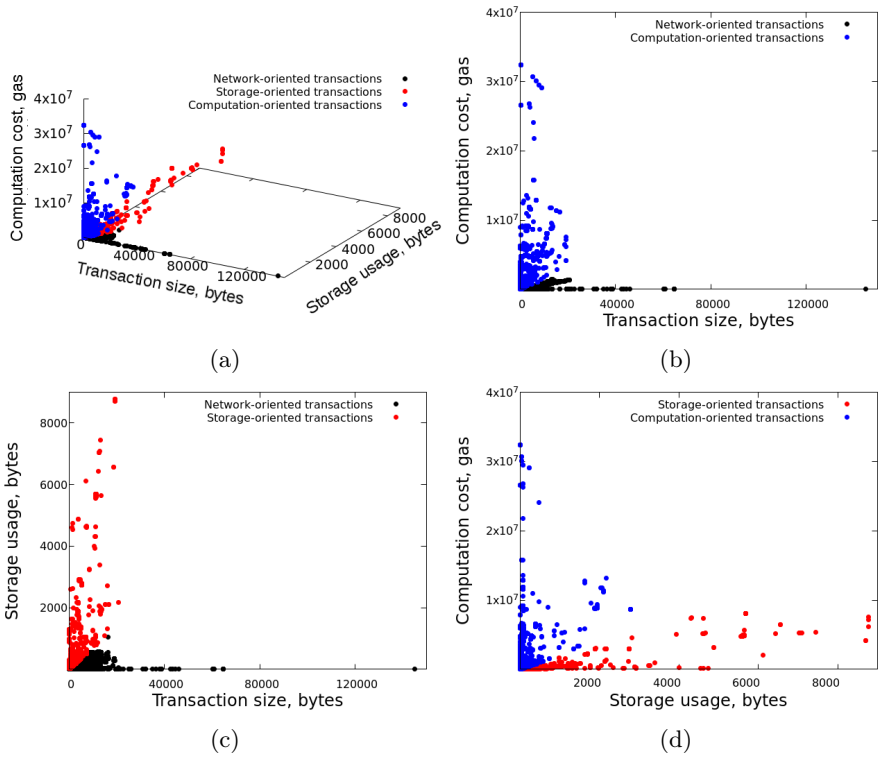


Fig. 4: Ethereum transactions differentiation by resource consumption

out such cases as they do not show using blockchain as a storage. Excluding such short-lived data from our analysis we estimate that average lifetime of a data object in Ethereum is 23,731 blocks (or about 4 days considering 15 seconds average delay between blocks).

This gives the following estimation on ratios between the pricing coefficients for the expected state size:

$$\begin{aligned} \frac{\alpha}{\beta} &\approx 7.7 \cdot 10^{-3} \\ \frac{\alpha}{S} &\approx 6.7 \cdot 10^{-4}, \end{aligned} \tag{3}$$

where S is the cost of the storage of byte of output in the state for one block, which does not depend on state size in Ethereum. The estimations are quite approximate, while changing them does not affect fees for most of transactions unambiguously attributed by a concrete type of the resource consumption.

6 Concluding remarks

Blockchain technology relies on miners, that safeguard the integrity of the blockchain in exchange for a revenue, that usually consist of two parts: block reward and transaction fees. Transaction fees are useful to limit miners resource usage and prevent spam.

While in most of cryptocurrencies a transaction fee is addressed as an atomic concept, in this paper we have shown that it is reasonable to introduce the three components of a fee associated with resources utilized: network, computation or storage. The analysis of Ethereum blockchain shows that transactions in such a three-dimensional space are distributed close to one of the 3 axes, allowing us to unambiguously classify transactions by consumed resource.

Storage part of the fee is already discussed in literature as a necessary tool to limit miners storage consumption [15,17]. This fee component is required to make the state size more predictable, but its implementation is challenging since transaction lifetime is not known at the time when the transaction is created. In the paper we have described the concrete method to charge for state bytes consumed that can be fully implemented on the script level.

Besides limiting the size of the state, storage fee provides valuable side effects. In particular, it provides a way to return coins with lost keys into circulation. Although necessity of coin recirculation is still an open question, it has been widely discussed in literature (e.g. [12,11]) in connection with the prevention of the deflation, which may eventually occur in cryptocurrencies with fixed supply. Enforced coin recirculation has been implemented in some cryptocurrencies [9].

Another important side effect of the storage fee is that it provides additional rewards for miners. Even when all coins are emitted and fixed block reward goes to zero, storage fee will provide stable rewards for miners, which do not depend on user transactions included into block. This will make destructive mining strategies described in [7] less profitable.

With these factors taken into account, the ready-to-implement system is provided, which is believed to solve the problem of uncontrollable state growth, provide some valuable side effects by the same means, while preserving currently existing methods for transaction fees and code execution costs.

References

1. Solidity language, `https://solidity.readthedocs.io`
2. SPV mining, `https://bitcoin.org/en/alert/2015-07-04-spv-mining`
3. Andresen, G.: Back-of-the-envelope calculations for marginal cost of transactions (2013), `https://gist.github.com/gavinandresen/5044482`
4. Bitcoin Wiki: Bitcoin script, `https://en.bitcoin.it/wiki/Script`
5. Bitcoin Wiki: July 2015 flood attack, `https://en.bitcoin.it/wiki/July_2015_flood_attack`
6. Blockchain.info: Number of unspent transaction outputs, `https://blockchain.info/charts/utxo-count?timespan=all`

7. Carlsten, M., Kalodner, H., Weinberg, S.M., Narayanan, A.: On the instability of bitcoin without the block reward. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. pp. 154–167. ACM (2016)
8. Earls, J.: The Economics of Gas Models. Conference talk. (2017), http://earlz.net/view/2017/10/02/1550/economics-of-fees-and-gas, CESC 2017 — Crypto Economics Security Conference, Berkeley, USA
9. Friedenbach, M.: Freicoin, http://freico.in
10. Garay, J., Kiayias, A., Leonardos, N.: The bitcoin backbone protocol: Analysis and applications. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 281–310. Springer (2015)
11. Gjermundrød, H., Chalkias, K., Dionysiou, I.: Going beyond the coinbase transaction fee: Alternative reward schemes for miners in blockchain systems. In: Proceedings of the 20th Pan-Hellenic Conference on Informatics. p. 35. ACM (2016)
12. Gjermundrød, H., Dionysiou, I.: Recirculating lost coins in cryptocurrency systems. In: International Conference on Business Information Systems. pp. 229–240. Springer (2014)
13. Goodmani, L.: Michelson: the language of smart contracts in tezos, https://www.tezos.com/static/papers/language.pdf
14. Luu, L., Teutsch, J., Kulkarni, R., Saxena, P.: Demystifying incentives in the consensus computer. In: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security. pp. 706–719. ACM (2015)
15. Möser, M., Böhme, R.: Trends, Tips, Tolls: A Longitudinal Study of Bitcoin Transaction Fees, pp. 19–33. Springer Berlin Heidelberg, Berlin, Heidelberg (2015), https://doi.org/10.1007/978-3-662-48051-9_2
16. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system (2008)
17. Reyzin, L., Meshkov, D., Chepurnoy, A., Ivanov, S.: Improving authenticated dynamic dictionaries, with applications to cryptocurrencies. In: International Conference on Financial Cryptography and Data Security (2017)
18. Rizun, P.R.: A transaction fee market exists without a block size limit (2015)
19. Vasek, M., Thornton, M., Moore, T.: Empirical analysis of denial-of-service attacks in the bitcoin ecosystem. In: International Conference on Financial Cryptography and Data Security. pp. 57–71. Springer (2014)
20. Wilcke, J.: The Ethereum network is currently undergoing a DoS attack, https://blog.ethereum.org/2016/09/22/ethereum-network-currently-undergoing-dos-attack/
21. Wilhelm, R., Engblom, J., Ermedahl, A., Holsti, N., Thesing, S., Whalley, D., Bernat, G., Ferdinand, C., Heckmann, R., Mitra, T., Mueller, F., Puaut, I., Puschner, P., Staschulat, J., Stenström, P.: The worst-case execution-time problem—overview of methods and survey of tools. ACM Trans. Embed. Comput. Syst. 7(3), 36:1–36:53 (May 2008), http://doi.acm.org/10.1145/1347375.1347389
22. Wood, G.: Ethereum: A secure decentralised generalised transaction ledger. Ethereum Project Yellow Paper (2014), https://ethereum.github.io/yellowpaper/paper.pdf