On Space-Scarce Economy In Blockchain Systems

Alexander Chepurnoy, Vasily Kharin, and Dmitry Meshkov

No Institute Given

Abstract. In this paper we study space-scarce economy in massively replicated open blockchain systems. In these systems, such as Bitcoin, memory to hold a current state snapshot needed to validate transactions becomes the most scarce resource eventually. The issue is even more critical for blockchain systems used to store data (votes, certificates, logs etc.). Uncontrolled state size growth could lead to security issues, such as denial-of-service attacks. Only technical solutions, not economic, have been proposed to tackle this problem to the moment. In contrast, we propose to add a new component to a transaction fee scheme based on how much additional space will be needed for new objects created in result of transaction processing and for how long they will live in the state. We provide three possible options towards implementing the new fee component, namely *prepaid outputs*, *postpaid outputs* and *scheduled payments*. We provide an analysis of the model with respect to all the three options. We show that the state growth could be bounded by a fee factor, miners are getting additional stable rewards and lost coins are being taken back into circulation eventually.

1 Introduction

Bitcoin [1] 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 scheme is relied on mining process. If majority of miners are honest, then Bitcoin meets its security goals as formal analysis [2] shows. For 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. A transaction fee is set by a user during transaction creation. Transaction fees are useful for an existing cryptocurrency economy for two reasons:

1. *Incentivization of miners.* A rational Bitcoin 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 [3,4]. As shown in [4], even in absence of block size limit, Bitcoin fee market is healthy and the miners surplus is maximized at a finite quantity of block space. Thus the miner is incentivized to produce a block of a limited size.

This means that only a subset of transactions which provides enough value to a miner will be included in a block. A paper [4] provides a procedure to calculate transaction fee based on block propagation time.

2. *Limit resources usage and prevent spam.* Besides of network utilization, transaction processing requires a miner to spend some computational resources. For most of the cryptocurrencies, a transactional language is limited (with Bitcoin Script [5] being one of the most limited), thus a number of CPU cycles needed to process a transaction is strictly bounded and corresponding computational costs are not directly considered. In contrast, in cryptocurrencies supporting smart contract languages, such as [6,7,8], transaction processing may require a lot of computations, and computational costs are included in transaction fee. This cost is specific to concrete transactional language and is out of scope of this paper.

A transaction in Bitcoin fully spends outputs from previous transactions, and also creates new outputs of user-defined values. A notable and the only exception is a coinbase transaction of a block which creates fixed amount of money out of thin air and also claims transaction fees without referring to any outputs (a fee for a non-coinbase transaction is sum of claimed outputs values minus sum of values for created outputs). A node is checking a transaction in Bitcoin by using a set of unspent outputs. In other cryptocurrencies a representation of a *state* needed to validate and process an arbitrary transaction could be different (for example, in Ethereum [9] 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 random-access memory. 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 succesful denial-of-service attacks against the nodes[10]. Similarly, in 2013 a denial-of-service attack against serialized transactions residing in a secondary storage (HDD or SSD) was discovered in Bitcoin[11].

The main purpose of this paper is to consider a new mandatory component in a transaction fee scheme reflecting state growth. In all known cryptocurrencies of today, an element of the state once created lives possibly forever without paying anything for that. This leads to continuously increasing state (we point to Bitcoin unspent transaction outputs (UTXO) set size as an example [12]). Moreover, state may grow fast during spam attack, for example, 15 million outputs were quickly put into UTXO set during spam attacks against Bitcoin in July 2015 [13], and most of these outputs are not spent yet. The paper [14] 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 big capacity), while other nodes are checking proofs of state transformations generated by miners, and size of a proof (in average and also in a worst case) is about $log(S)$ in regards with a state size S. Nevertheless, big state could lead to centralization of mining or SPV mining [15], and these concerns

should be addressed. Also, there is an increasing demand to use a blockchain as a data storage, and storing permanently objects in the state without a cleaning procedure is not a viable option.

We propose an economic solution to the problem of unreasonable state growth (such as spam attacks, or objects not being using anymore but still living in the blockchain). The solution is a new mandatory fee component. We state that a user should pay fee for both the additional space needed to store objects created by a transaction, and also for lifetime of new bytes. This model is usual for cloud storage services where users pay for gigabytes of data per month. We provide a possibility for miners to control their storage requirements by changing a fee factor. Later in this paper we will refer to this new fee component as to a *space-time fee*.

Proposed fee regime is promoting money circulation in the blockchain economy. The limited lifetime of a state element also leads to lost coins being taken back into circulation (supposedly by miners).

Summarizing, we study an economy where quick-access storage of a node in a massively-replicated system becomes the most scarce system resource eventually. Thus we call such an economy a *space-scarce economy*.

1.1 Assumptions

Here we provide assumptions our model is based on:

- all the fees for a block are going to a single miner like in Bitcoin. There are proposals to share the rewards for a block within a group of miners, for example in [16,17], and they are out of scope of the paper.
- a state is a set of unspent outputs. An output is not modifiable so can be only created and then spent at whole.
- an output is protected by a spending condition which is defined as a logical formula. Predicates in the formula can refer to properties of a blockchain (for example, its current height available via variable $Height$), spending transaction tx and the output out itself. We assume that it is possible to compare two scripts, and also it is possible to determine whether the spending transaction contains an output with a given property. For example, $tx.has\_output(script = out.script)$ evaluates to true if the spending transaction contains an output with the same script as the output has. Note that Bitcoin Script is too limited to support scripts comparison as well as using the spending transaction and the output to spend in a spending condition.
- for simplicity, we assume that a block is of a finite size but all the transactions a miner has at a moment of block generation can be packed into it, if otherwise is not stated explicitly.
- time is measured via *height* which is a number of blocks since an initial block (a genesis block) till a block of interest.
- all anyone-can-spend outputs are collected by miners immediately as they appear.

– 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.

1.2 Structure of the Paper

The paper is organized as follows. A design of our new fee component is provided in Section **??**. The model then is analyzed in Section **??**. In Section **??** we observe related work, and in Section **??** we shape a plan for further research.

References

1. S. Nakamoto, Bitcoin: A peer-to-peer electronic cash system.
2. J. Garay, A. Kiayias, N. Leonardos, The bitcoin backbone protocol: Analysis and applications, in: Annual International Conference on the Theory and Applications of Cryptographic Techniques, Springer, 2015, pp. 281–310.
3. G. Andresen, Back-of-the-envelope calculations for marginal cost of transactions. URL `https://gist.github.com/gavinandresen/5044482`
4. P. R. Rizun, A transaction fee market exists without a block size limit.
5. bitcoin Wiki, Bitcoin script. URL `https://en.bitcoin.it/wiki/Script`
6. P. L. Seijas, S. Thompson, D. McAdams, Scripting smart contracts for distributed ledger technology, in: International Conference on Financial Cryptography and Data Security, 2017.
7. L. Goodmani, Michelson: the language of smart contracts in tezos. URL `https://tezos.com/pages/tech.html`
8. Solidity. URL `https://solidity.readthedocs.io`
9. G. Wood, Ethereum: A secure decentralised generalised transaction ledger, Ethereum Project Yellow Paper. URL `https://ethereum.github.io/yellowpaper/paper.pdf`
10. The ethereum network is currently undergoing a dos attack. URL `https://blog.ethereum.org/2016/09/22/ethereum-network-currently-undergoing-dos-attack/`
11. M. Vasek, M. Thornton, T. Moore, Empirical analysis of denial-of-service attacks in the bitcoin ecosystem, in: International Conference on Financial Cryptography and Data Security, Springer, 2014, pp. 57–71.
12. Blockchain.info, Number of unspent transaction outputs. URL `https://blockchain.info/charts/utxo-count?timespan=all`
13. Bitcoin_Wiki, July 2015 flood attack. URL `https://en.bitcoin.it/wiki/July_2015_flood_attack`
14. L. Reyzin, D. Meshkov, A. Chepurnoy, S. Ivanov, Improving authenticated dynamic dictionaries, with applications to cryptocurrencies, in: International Conference on Financial Cryptography and Data Security, 2017.
15. Spv mining. URL `https://bitcoin.org/en/alert/2015-07-04-spv-mining`

16. I. Eyal, A. E. Gencer, E. G. Sirer, R. Van Renesse, Bitcoin-ng: A scalable blockchain protocol, in: 13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16), USENIX Association, 2016, pp. 45–59.
17. E. K. Kogias, P. Jovanovic, N. Gailly, I. Khoffi, L. Gasser, B. Ford, Enhancing bitcoin security and performance with strong consistency via collective signing, in: 25th USENIX Security Symposium (USENIX Security 16), USENIX Association, 2016, pp. 279–296.