On Space-Scarce Economy In Blockchain Systems

No Author Given

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. Alex notes:write abt fee adjustment rule 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. Alex notes:check this

1 Introduction

Bitcoin [**?**] 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 [**?**] 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 [**?**,**?**]. As shown in [**?**], 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 [?] 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 [?] 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 [?,?,?], 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 [?] 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[?]. Similarly, in 2013 a denial-of-service attack against serialized transactions residing in a secondary storage (HDD or SSD) was discovered in Bitcoin[?].

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 [?]). 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 [?], and most of these outputs are not spent yet. The paper [?] 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 [?], 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.

1.1 Our contributions

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*.

Alex notes:write abt fee adjustment rule

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*.

1.2 Structure of the paper

Alex notes:todo: write

2 Preliminaries

In order to simplify analysis, we fix following assumptions:

3 Algorithm of the fee assignment

In spite of the problems and the possible solution outlined in the introduction, the essential element of the space-scarce economy is the method of assigning the fee to every transaction in a sensible way. In this chapter we reason about the guiding principles for the fee assignment, and end up with the example of a practical fee assignment rule.

The evolution of the blockchain networks has demonstrated the main obstacles for the system to be truly distributed in the real world. First and the most important so far, the storage capacity of the nodes is limited. In particular, it becomes more costly with time to run a node on a desktop hardware in a system where the full blockchain is needed for the operation V:more details. Elegant

solutions have already been found for this problems in recent years V:citations, consisting in practical unnecessity of storing the blockchain itself, but rather parts of header chain, and the state described above. However, the part of this problem remains, as a possibility of flooding the state with a long list of UTXO, which are never spent. This also has a side economical effects, since coins from lost (accidentally or intentionally) UTXO's are never returned to the circulation, and it becomes important if the total amount of coins is unchanged.

Second, it becomes obvious, especially with the development of smart contracts, that a transaction "cost" for the node can be more than just a storage: transactions can contain relatively complicated scripts which are meant to be executed by the nodes, providing additional overhead. The extreme and most famous example is the Ethereum network with the concept of the "World Computer".

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 storage, but creates the network load needed for synchronization.

Alex notes:refer here to a fee as a security measure concepts in the intro?

As stated in the intro, a transaction fee is an anti-overload tool. Thus a transaction fee should incorporate all the three of components above. We propose to charge for a component which processing demands more resources. That is, storage-oriented transactions should be charged for storage consumption, the computation-oriented transactions should be charged for script execution, and all the others by the network load. The last sentence can be formalized in its simplest form as follows:

$$\mathrm{Fee}(tx) = \max(\alpha \cdot N_b(tx), \beta \cdot N_c(tx), S(state, tx) \cdot L[tx])\,. \tag{1}$$

Here α and β are the pricing coefficients, N_b is the number of bytes in the transaction, which defines the network load, $N_c(tx)$ is the estimate of the computational cost of transaction, $S(state, tx)$ is the cost of the storage of outputs in the state for the unit of time, $L([tx])$ is the time which transaction is being stored in the state. V:Oops. Do we mean now that if the transaction frees the space in the state, the corresponding output can be stored forever? If yes, it cancels vast part of the reasoning: I create tx_0 with two outputs; in the next block tx_1 merging them, and store the output of tx_1 for free. Alex notes:Initially it was $\gamma \cdot S(state) \cdot (\sum L([coin_{created}]) - \sum L([coin_{spent}]))$, so the fee could be negative, but newly created coins are living for finite time nevertheless The obvious problems here are: What are the guiding principles for choosing α, β and S? How to arrange the charging for state spacetime consumption, if transaction contains no information on when the outputs will be spent? We address these problems in the following sections. There is also the problem of estimating $N_c(tx)$, which is extremely hard V:or unsolvable? V:How hard, does anybody know the specific name of this problem? for the Turing–complete languages, however solvable with the restrictions on the language V:citations, which restrictions, or stick to one existing. We leave it beyond the scope of the paper.

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

Assume for now, that for every transaction we know the time the outputs will be stored in the state $L[tx]$. 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: $N_b(tx)$, $N_c(tx)$, $N_s(tx) = S(tx)L[tx]$. Eq. (1) provides a prescription of splitting this space into three regions: network–oriented transactions, space–oriented transactions, and computation–oriented transactions (see Fig. 1). All the splitting is governed by the direction of vector $\boldsymbol{n}$ definig 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. V:It would be good and convincing to get some actual data from the existing network.

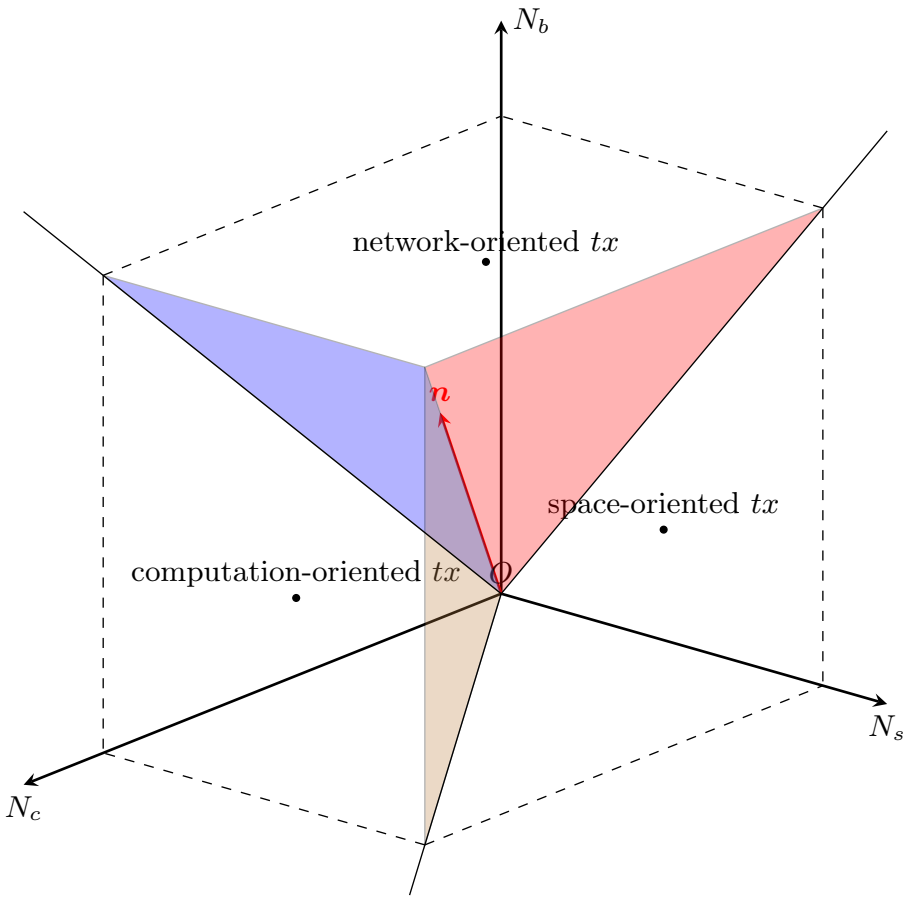

Fig. 1. Space of transactions splitted by Eq. (1) into the subregions of the dominant fees.

3.2 Choice of $S(tx)$

Alex notes:Why S(tx), not S(state, tx)?

The simplest way of assigning the value of $S(tx)$ is by making it proportional to the amount of data to be written into the state. That is, specifying the price of storage of 1 byte of data per one day. However, as it is shown in the Appendix, this does not fully solve the problem of limiting the state size. What is being controlled in this case, is the rate at which the data is being submitted, but not the state size itself. One could also manually define the maximal volume 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 (upper restriction 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"). The good pricing curve must 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 – huge data submission – miners get nothing; and infinite price – zero data submission – miners get nothing. As usual, the maximal outcome is between. The pricing policies described above are two particular cases of pricing curve (see Fig. 2). The values of $S(tx)$ and $p(x)$ are connected via

Alex notes:definitions of y and $p()$ are missed

$$S(tx) = \int_{x}^{x+\Delta(tx)} p(y)\, dy\,, \tag{2}$$

where x is the current state load, $\Delta(tx)$ is the change of the state size by transaction tx. V:How to order transactions in the block? Is it Ok to use the difference? What if it is negative?.

Note that the pricing curve is defined by a small number of parameters and to be the same for all the network. To impose the upper restriction on the state size one can choose the pricing curve formally going to infinity at some finite state size. One can also try to estimate the optimal state size for a given differentiable pricing curve. In the continual with the assumptions described in the appendix, the data submission rate $N(p(x))$ is fully defined by the current storage price

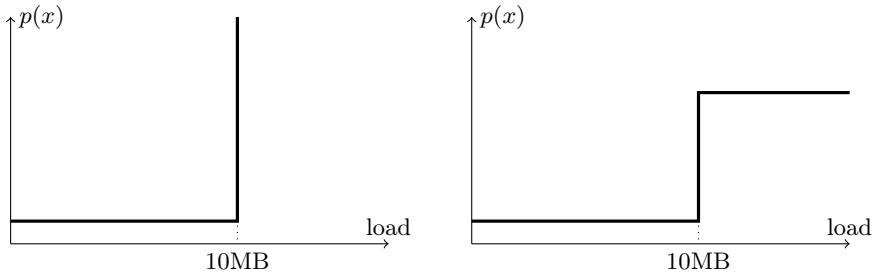

Fig. 2. Examples of pricing curves: rigid state size restriction (left) and overflow fees (right, see text). The value of 10MB is taken arbitrarily.

$p(x)$. Rewards rate obtained by the miners for stable state size at price p is given by $y = pN(p)$. Example is provided at Fig. 3 First, it provides a possible method of measuring explicit form of the function $N(p)$ in the model: one has to set up the price, and observe the static rewards. Second, one may wonder about the price p^*, optimal for the miners in terms of rewards. Obviously, it satisfies $N(p^*) + p^*N'(p^*) = 0$, 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 with this given the state size is automatically adjusted to the relatively predictable level $p^{-1}(p^*)$.

3.3 Example of pricing curve

From what was said above, the conclusion is that two values are particularly important for the model: maximal state size, and optimal price. In order to preserve the limitation on the stored amount of data, one can let the storage price go to infinity faster than $1/x$ moving closer to the limiting size. On the other side, there may be some intermediate "preferred" size of the state x_d. This, in turn, can be matched to the optimal price. One may also wish to introduce the lower bound on the storage price p_{min}. Taking these values it is easy to construct the function $p(x)$ with the desired properties. As an example,

$$p(x) = p_{min} + (p^* - p_{min})\frac{x}{x_d}\frac{x_{max} - x_d}{x_{max} - x}\,, \tag{3}$$

Another question is how one should deal with the chunks of data comparable to the state size. The logical way is to treat $p(x)$ as the differential property (which it actually is). So if the current state size is x_0, and the participant is willing to submit the data block of size L for unit time, he is charged by the amount of $\int_{x_0}^{x_0+L} p(x)\,dx$. For the sample pricing curve it is

$$P = L\,(p^* - \eta\Delta p) - \eta(\eta - 1)\Delta p\, x_d \log\left(1 - \frac{L}{x_{max} - x_0}\right)\,, \tag{4}$$

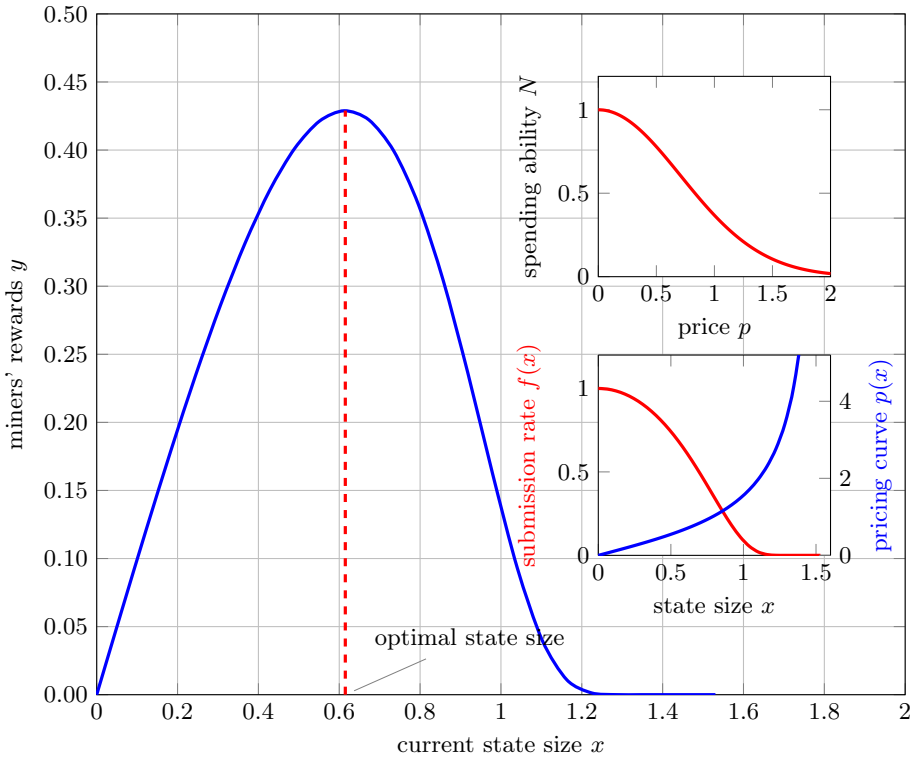

Fig. 3. Rewards curve.

where $\eta = x_{max}/x_d, \Delta p = p^* - p_{min}$. Note that the logarithmic divergence guards the upper size limit.

3.4 Unknown storage time

V:Here — Dima's idea as we discussed it in Moscow. Q: postpaid, or pay-as-you-go?

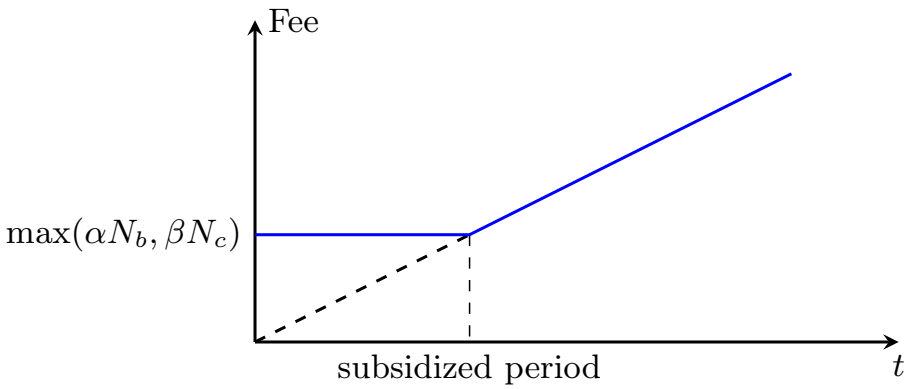

Fig. 4. Transaction cost as a function of output existence time.

4 Scheduled Payments

If the user do not move the coins before this, then anyone (presumably, a miner) can create a transaction claiming this output, returning all the coins back to the owner except of a space-time fee $K \cdot B \cdot D_s$. A spending script for the output would be like:

$$\begin{aligned}&(regular\_script) \vee \\ &(Height > (out.height + D_s) \wedge (out.value \leq K \cdot B \cdot D_s \vee \\ &\quad tx.has\_output(value = out.value - K \cdot B \cdot D_s, script = out.script)))\end{aligned} \tag{5}$$

In this model D_s is another protocol parameter which may be fixed by a protocol design or changed via miners voting like K.

5 Evaluation

A State size dynamics

For the primary analysis assume that participants act honestly: they submit data if they need to do so and it is affordable; they do not in the opposite

case. For the sake of simplicity, suppose that the time of storage of data block is fixed, and equal to T. In order to reduce the discrete stochastic model to continuous deterministic one, assume that the number of participants is large, and the typical size of the submitted data chunk is much less than characteristic pricing curve variation scale. The amount of data submitted to the state is changing continuously. At every given moment of time the users submit the data at some rate (say, MB/s) f, which is defined by the current price (participants submit more when cheap, and less when expensive). The current price is fully determined by the current state load x. After time interval T data is erased from the state. The data is written into the state at rate $f(x(t))$, and erased at rate $f(x(t-T))$. Under these assumptions, the evolution of the state load is defined by the following equation:

$$\frac{dx}{dt} = f(x(t)) - f(x(t-T))\,. \tag{6}$$

If one measures time in the data block TTL T, the equation takes the form

$$\dot{x} = f(x(t)) - f(x(t-1))\,. \tag{7}$$

The equations of this type are called delay differential equations (DDE), and have been studied widely for the vast amount of control problems.

Now few words about the function f, and how to convert it to the miner's income. What participants know is the pricing curve $p(x)$. For every value P of this function there is amount of people $N(P)$ who want to submit data and can afford it at time interval T. The function $N(P)$ is non-increasing, and going to zero for sufficiently large P (every participant has maximal price he is ready to pay, and will also try to submit something if current price is lower; for sufficiently large price no one is ready to pay). In these notations $f(x) = N(p(x))$ (see inset on Fig. 3), and the profit rate which miners get from current submissions is $y(t) = p(x(t))f(x(t))$.

A.1 Constant storage price

A.2 State-dependent storage price

Alex notes:to remove totally?

B 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 [**?**,**?**], 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.

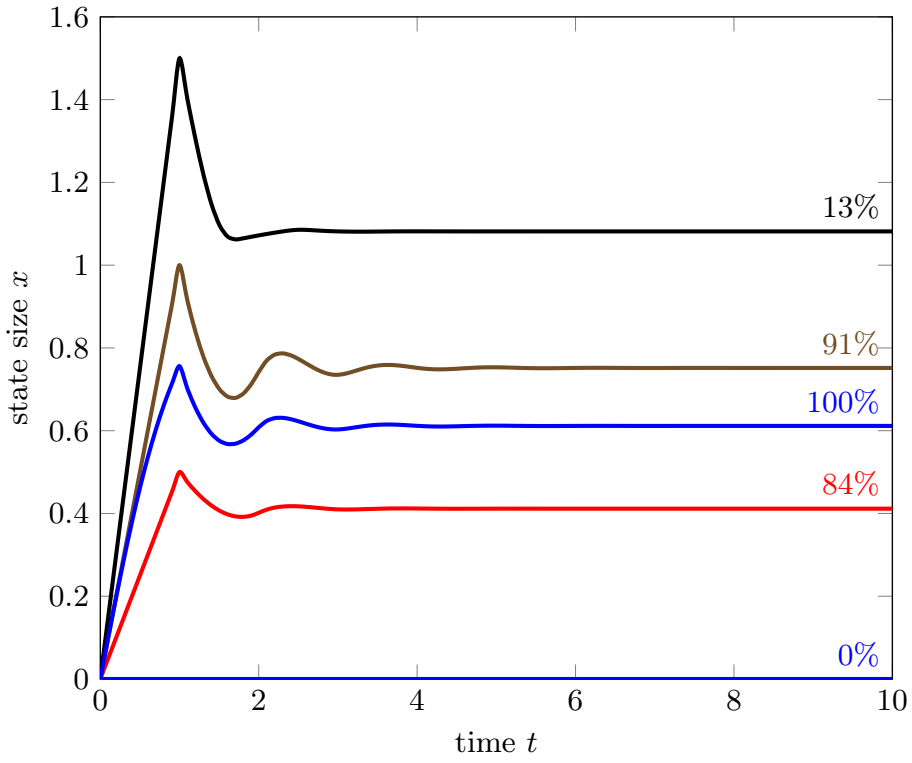

Fig. 5. Dynamics of the state size for various initial rates. The numbers above the curves are the miners reward rates with respect to maximal possible stationary rewards.

- 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.