Saving proof-of-work by hierarchical block structure: Bitcoin 2.0?

Qui Somnium
email: qui.somnium@tuta.io

April 7, 2022

To my wonderful wife, our children and yours as well

Abstract

We argue that the current *Proof of Work* based consensus algorithm of the Bitcoin network suffers from a fundamental economic discrepancy between the real-world transaction costs incurred by miners and the wealth that is being transacted. Put simply, whether one transacts 1 satoshi or 1 bitcoin, the same amount of electricity is needed when including this transaction into a block. The notorious Bitcoin blockchain problems such as its high energy usage per transaction or its scalability issues are, either partially or fully, mere consequences of this fundamental economic inconsistency. We propose making the computational cost of securing the transactions proportional to the wealth being transfered, at least temporarily.

First, we present a simple incentive based model of Bitcoin's security. Then, guided by this model, we augment each transaction by two parameters, one controlling the time spent securing this transaction and the second determining the fraction of the network used to accomplish this. The current Bitcoin transactions are naturally embedded into this parametrized space. Then we introduce a sequence of hierarchical block structures (HBSs) containing these parametrized transactions. The first of those HBSs exploits only a single degree of freedom of the extended transaction, namely the time investment, but it allows already for transactions with a variable level of trust together with aligned network fees and energy usage. In principle, the last HBS should scale to tens of thousands timely transactions per second while preserving what the previous HBSs achieved.

We also propose a simple homotopy based transition mechanism which enables us to relatively safely and continuously introduce new HBSs into the existing blockchain.

Our approach is constructive and as rigorous as possible and we attempt to analyze all aspects of these developments, al least at a conceptual level. The process is supported by evaluation on recent transaction data.

1 Introduction

Arguably, in 2008 a new era of finance began. In a groundbreaking paper [16], Satoshi Nakamoto described a pure peer-to-peer payment system of Bitcoin. He presented an elegant and rather simple way how to prevent double spending in decentralized distributed networks without a trusted party using a *Proof of Work* (PoW) concept from [1].

The Bitcoin payment system, despite being extremely well thought out and elegant, could not grow up, at least yet, to the very expectations of its creators.

Let us read Satoshi Nakamoto's statement from Section 1 of [16], arguing against financial institution mediation: *"... The cost of mediation increases transaction costs, limiting the minimum practical transaction size and cutting off the possibility for small casual transactions,... "*.

Unfortunately, looking at the current transaction costs on the main Bitcoin network or its available transaction throughput, it is obvious that Bitcoin suffers from the same symptoms but caused by a different illness. Let us paraphrase the Bitcoin scalability problem in a sentence similar to Satoshi Nakamoto's one: *The costs of requiring that all transactions should be equal and all of them seen by all the nodes in the network increases transaction costs, limiting the transaction throughput and cutting off the possibility for small casual transactions.*

The limited transaction throughput is arguably the primary technological hurdle of Bitcoin's use as means of payment. And the high transaction costs are the main economical hurdle. But as indicated above, these are mere effects of uniformity and inflexibility of Bitcoin transactions, which do not respect economic realities and consequently cannot map to many real world situations. Put simply, if we require that each transaction is seen, signed and stored by everybody, the system has to be costly, slow and almost impossible to scale.

This paper is about, to our best knowledge, an innovative path of possible practical extensions of Bitcoin's PoW based consensus algorithm and its protocol, starting form the definition of its transaction. We need to relax and expand the notion of a Bitcoin transaction and also of its block and network structure, all those aspects, in order to resolve its scalability issues.

In Section 2 we first discuss the concepts of trust and risk, focusing on their multi-scale nature, which is in strong contrast to flat security guaranties and costs of the current Bitcoin implementation. This fundamental discrepancy essentially locks the Bitcoin development in a suboptimal state and prevents further innovation. In the followings sections we gradually annihilate this inconsistency, while preserving Bitcoin's simplicity to a maximal possible degree. Importantly, transactions would still be saved in an adapted multi-layered blockchain and secured by the mainnet and not a distinctively separate second layer technology such as *Lightning Network* [2].

In preparatory Section 3, we discuss security, fees and electric power consumption of Bitcoin transactions and the alignment of these three aspects. We conclude the section by definition of a 2-parameter family of extended transactions. In Section 4 we propose a sequence of hierarchical block structures (HBSs) containing those parametrized transactions. Starting from the first HBS pre-

sented in Section 4.1, users can choose a level of security which they require to transact and this in a transparent and simple way to grasp. And the costs will be proportional to this level of trust. Sections 4.2 and 4.3 are devoted to the scaling of on-chain transaction capacity. Some supporting material is presented in Appendix A and cited when needed. Particularly, alternative consensus algorithms are shortly discussed in Section A.3. For reproducibility of results, almost all algorithms used in this publication are published on github [19] and for completeness also presented in Appendix B.

Finally, let us note that while the ideas that are presented in this paper are certainly widely applicable outside of Bitcoin's realm, their possible applications in different contexts are left for the reader.

2 Rigidity of Bitcoin's trust model

2.1 Distributed trust model avoids systemic risks

Bitcoin allows for direct peer-to-peer online payments without an all-knowing trusted party(ies). In essence, the trust is distributed in the network among its participants, particularly among its mining nodes. The information exchange among the participants is governed by Bitcoin's protocol and its rules. Due to its distributed nature, Bitcoin avoids all kinds of systemic risks, e.g. it cannot be easily globally censored and it does not have a single point of failure and it is permissionless, it is anyone can join and leave network at any time.

2.2 At individual level risks are countless

All sorts of other risks remain. For example, the Bitcoin offers irreversible transactions. This is an advantage for sellers, not so much for buyers. Satoshi Nakamoto, being aware of it, proposed routine escrow mechanisms to protect buyers [16]. But an escrow agent (e.g. implemented via multisig) is a trusted third party. Or imagine that you buy a property and pay in bitcoins. The ownership of the property and/or the coins can be disputed by either party or even a third party at any moment, i.e. the parties are exposed to legal risk and the judiciary might become a trusted party. Further, we all are susceptible to 5$ wrench attacks. Last but not least, humans are fallible and we often loose access to private keys.

The above mentioned risks are mostly local, either transaction specific risks having to do with the fact that their underlying contracts relate to values in physical world or any other risks at individual level having to do with our sheer existence in real world and our own fallibility.

Altogether, as the scale goes from global to local, both trust and risks become concentrated when making transactions. As individuals, we have to be and are prepared to face a myriad of foreseen and unforeseen circumstances. Based on risks we face and our abilities to mitigate them, trust we have in our

counterparty and taking into account transaction costs and rewards, we either decide to transact or not.

2.3 Relaxing trust model of consensus algorithm?

The main recurrent costs for bitcoin miners are electric energy bills, see Appendix A.5. Since the hashing time is linearly proportional to size of data being hashed, see Algorithm 4, it seems natural to compute fees based on transaction size, as it is currently done. And indeed, purely from the miners' perspective, whether a transaction moves 100 BTC or 1 Satoshi, it does not matter if their bit size is the same. Is this not a problem? Should not moving a fortune cost more then paying few cents?

In our opinion, it clearly should. Otherwise, small payments will become clearly impossible as Bitcoin transitions from issuance system where miners are paid mainly by rewards to a system based on transactions fees, and this regardless whether they could be included into a block given the size constraints or not. The second but not secondary issue is that currently those transacting smaller amounts actually pay fees for security of those transacting bigger ones, promoting a rising inequality.

We think that this discrepancy between transaction fees and wealth being transfered is probably the most fundamental problem, stopping the Bitcoin network from becoming a general payment system as envisioned by Satoshi Nakamoto. So, how to cut the Gordian knot?

A solution becomes almost evident, if one aims for a *transaction fee per byte* ϕ proportional to the wealth being transfered v. Then, at least conceptually, arbitrarily small payments become possible. And it clearly makes system at once more honest. A direct consequence of this aim is that the security of transactions has to be proportional to the wealth being transfered as well, at least temporarily. Otherwise a difficulty would arise how to pay the miners for their work.

And indeed, from the user risk/reward perspective described in the previous section is Bitcoin's current consensus protocol rather too coarse, idealistic and rigid. It simply does not allow a user to choose an appropriate costs/risk ratio. The transactions are public and every full node has to be aware of all the transactions ever included in the blockchain. Either a user accepts this absolute security and pays all the mining nodes for their work the corresponding price and bears the consequences (e.g. a long confirmation time) or he opts to not transact. The second option is a rational choice in many practical circumstances. Again, a successful general online payment system has to allow for inexpensive quick micro payments.

In the next sections, we will include *level of trust in the payment system* as one of the risks and *network fee* as one of the costs in individual's decision space, with some safe defaults which are keeping wealth being transfered and safety proportional.

3 Extending transaction space

The Bitcoin system can be seen as a *state-transition machine* [4]. Its *state* s consists essentially of all ownership records of all existing coins and it can be changed only according to the Bitcoin protocol by applying the corresponding *state transition function* τ_B. This function is a very handy abstraction, which allows one to effectively hide whole the Bitcoin protocol complexity under one symbol. It takes old state s_o, a transaction t and evaluates as

$$\tau_B(s_o, t) = \begin{cases} s_n & \text{if } t \text{ is compliant,} \\ (s_o, e) & \text{if } t \text{ is not compliant,} \end{cases} \tag{1}$$

where s_n is the new state and e is a returned error in the case of noncompliant t. Any transaction t is essentially a set of instructions created by the owner of unspent bitcoins, called *Unspent Transaction Outputs* (UTXOs), how to allocate those coins. As the name clearly suggests, UTXOs are outputs of previous transactions that have not yet been inputs of any already existing valid transaction. Each full node computes its UTXO set independently by validating all transactions in the blockchain. It recurrently applies Equation (1) starting from bitcoins created as mining rewards in a *coinbase* transaction. Consequently, UTXOs are a derived representation of a current state s which is actually recorded in the blocks.

The main goal of this section is to define a concept of an extended Bitcoin transaction. Before we actually do so in Section 3.3, we describe a simple incentive based economic model of Bitcoin transactions, focusing first on the aspect of security in Section 3.1 and then discussing transaction fees in Section 3.2.

3.1 Transaction security

As explained in Appendix A.6, the long term investments of miners to acquire highly specialized non-repurposable hashing hardware can be seen as their main incentive to act honestly and keep the Bitcoin network running smoothly [13]. Any loss of trust would negatively impact the BTC market prize, decreasing the future returns of the miners.

Consequently, the security of Bitcoin transactions is proportional to *miner commitment* $c[\ddot{\text{B}}]$, which in turn is proportional to the total hashrate of the network h_B in H/s. Under the assumption of linearity we can stipulate that:[1]

$$c[\ddot{\text{B}}] := \gamma[\ddot{\text{B}}\ \text{s/H}] \cdot h_B[\text{H/s}], \tag{2}$$

where $\gamma > 0$ is for simplicity assumed being a real constant. Since we do not know its value, the equation (2) is only qualitatively interesting. E.g. the dimension $\ddot{\text{B}}s/H$ of γ tells us that the miners are committed, if their BTC gains per one hash are high and the "time horizon" of value extraction is long.

[1] Occam's razor.

From now on, the square brackets in formulas are exclusively used to present dimensions of variables, with an exception of a closed interval.

The miners have to find a nonce for their block such that its hash is smaller than a predefined target determined by the *difficulty* Δ [7]. The difficulty is adjusted every 2016 blocks, i.e. approximately every 2 weeks, with the goal to achieve average block time of 10 minutes. Time, the difficulty Δ and the total Bitcoin network hashrate h_B fulfill this equation

$$h_B \left[\frac{\mathrm{H}}{\mathrm{s}}\right] = \frac{1}{65535}\frac{2^{48}\Delta[\mathrm{H}]}{600\mathrm{s}} \approx \frac{2^{32}\Delta[\mathrm{H}]}{600\mathrm{s}}. \tag{3}$$

The difficulty Δ is usually presented as being dimensionless quantity, but it actually represents the number of hashes computed by all Bitcoin miners in one block time, on average, divided by 2^{32}.

Just another ingredient of the "security soup" is the realization that *hashing time* t_h is linearly proportional to bitsize $b(m)$ of a hashed message m, see Algorithm 4. We can write:

$$t_h(m)[\mathrm{s}] := \eta \left[\frac{\mathrm{s}}{\mathrm{b}}\right] \cdot b(m)[\mathrm{b}], \tag{4}$$

where variable η represents the time necessary to sign one bit of information. Let us call it *time investment*. In the case of signing by all Bitcoin miners, the corresponding η_B can be estimated easily from block size $b(\mathcal{B})$ in bits:

$$\hat{\eta}_B \left[\frac{\mathrm{s}}{\mathrm{b}}\right] = \frac{600\mathrm{s}}{\overline{b(\mathcal{B})}[\mathrm{b}]}, \tag{5}$$

where $\overline{b(\mathcal{B})}$ depicts the average block size and subscript B means we consider the whole Bitcoin network. From now on, $\overline{x}$ will depict the sample mean value of x and $\hat{x}$ a general estimator of x. From block size limit of e.g. 1MB, we obtain a lower bound on η_B of approximately 7.15×10^{-5}.

We propose the following simple incentive based Bitcoin security model:

Definition 1 *A security $s(t)$ of transaction t with output value $v(t)$ is defined as*

$$s(t)\left[\frac{H}{\ddot{B}}\right] := \frac{h(t)[H/s] \cdot t_h(t)[s]}{v(t)[\ddot{B}]}, \tag{6}$$

where $h(t)[H/s]$ is the hashrate used to sign t in time $t_h(t)$.

This definition is intuitively very easy to understand. The security is proportional to the hashing resources used and to the time they are employed to compute the transaction signature. And it is inversely proportional to the value transfered since it is the eventual gain from the double spending attack targeting this transaction.

Using (4) we obtain for the security s_B of current Bitcoin transactions:

$$s_B(t) = \frac{h_B \cdot \eta_B \cdot b(t)}{v(t)}. \tag{7}$$

which leads to an average per block $\mathcal{B}$ security

$$\overline{s_B(t)}|_{t\in\mathcal{B}} = \frac{h_B \cdot \eta_B}{|\mathcal{B}|} \sum_{t\in\mathcal{B}} \frac{b(t)}{v(t)}, \tag{8}$$

where $|\mathcal{B}|$ represents the cardinality of set $\mathcal{B}$, i.e. here the number of transactions in $\mathcal{B}$.

Definition 1 describes the security from the point of view of individual transactions. Currently, transactions with diverse output values v are included into the same block. Following the developments, security per block $\mathcal{B}$ can be consistently defined as:

$$s_B(\mathcal{B}) := \frac{h_B \cdot t_h(\mathcal{B})}{v(\mathcal{B})} = \frac{h_B \cdot \eta_B \cdot b(\mathcal{B})}{v(\mathcal{B})} \overset{(5)}{\approx} \frac{h_B \cdot 600s}{v(\mathcal{B})} \cdot \frac{b(\mathcal{B})}{\overline{b(\mathcal{B})}} \overset{(3)}{\approx} \frac{2^{32}\Delta}{v(\mathcal{B})}, \tag{9}$$

which sheds more light on relation between security and difficulty. The numerator of (9) is an equivalent of difficulty, up to multiplication by a constant. When the total output value $v(\mathcal{B})$ per block $\mathcal{B}$ rises in time, difficulty Δ has to rise at least proportionally to preserve security. Based on the alignment of (9) with (6), we can define the transaction difficulty $\delta(t)$ of a transaction t accordingly:

$$\delta(t)[\mathrm{H}] := h(t) \cdot t_h(t) \overset{(4)}{=} h(t) \cdot \eta(t) \cdot b(t) \tag{10}$$

The current security per block $s_B(\mathcal{B})$ can be estimated as

$$\hat{s}_B(\mathcal{B}) = \frac{2^{32}\Delta}{\overline{v(\mathcal{B})}}, \tag{11}$$

where average output value per block $\overline{v(\mathcal{B})}$ can be computed from real blockchain data, see e.g. Section 4.1.2.

Further, we can write (9) as

$$s_B(\mathcal{B}) = \frac{h_B \cdot \eta_B \cdot \sum_{t\in\mathcal{B}} b(t)}{\sum_{t\in\mathcal{B}} v(t)} = \frac{h_B \cdot \eta_B \cdot \overline{b(t)}}{\overline{v(t)}}, \tag{12}$$

i.e. in terms of the average transaction where the sample is the whole block $\mathcal{B}$. For a message with average size and output values, we can conclude by comparing (6) and (12) that the current block-based bitcoin security is identical to the individual one according to Definition 1.

However, if e.g. an average sized transaction has a much higher output value than the average one is, the current block-based security may be insufficient. On the contrary, if e.g. an average sized transaction has a much lower output value than the average one is, this security may be an overkill. Let us make the following assumption:

Assumption 1 *If one transacts, he expects the same security $s(t)$ regardless of transaction value $v(t)$.*

This formally under the validity of model (6) means that there exists a $c \in \mathbb{R}^+$ such that transaction difficulty $\delta(t) = c \cdot v(t)$ for all admissible transactions t, implying $s(t) = c$. This c is certainly at least time dependent, but it is independent on individual transactions t.

3.2 Transaction fees

Currently, Bitcoin users pay fees that are only proportional to transaction size $b(t)$. From the perspective of miners, this fee scheme is an optimal one. Their main running costs are payments for electric power consumption used to hash blocks of transactions. In Equation (66) all the other variables except $b(t)$ are transaction invariant. First, all the transactions are included in a block hashed by all the miners with their combined hash rate h_B. Second, blocks are created on average every 10 minutes.

However from the usefulness perspective, as already informally argued in Section 2.3, the current fee scheme based only on $b(t)$ is far from optimal. We will look for an optimal fee scheme(s) from both user and miner perspectives in a relaxed setting where strict Bitcoin constraints on hash rate and hashing time are removed.

In (10), the definition of transaction difficulty $\delta(t)$, both the hashing time $t_h(t) = \eta(t) \cdot b(t)$ and hash rate $h(t)$ are transaction dependent variables by design. Following (66), $\delta(t)$ multiplied by energy efficiency e of mining hardware gives us energy consumption of a transaction. Thus in any case, to fairly cover miner costs, users should pay transaction fees f proportional to $\delta(t)$:

$$f[\text{₿}] \propto \delta(t)[\text{H}]. \tag{13}$$

In effect, users then still pay fees proportional to transaction size $b(t)$ but also proportional to variable hashing capacity used $h(t)$ and its employment time, determined by $\eta(t)$.

The transaction difficulty $\delta(t)$ is the numerator of (6), which defines the security of individual transactions $s(t)$. Thus $s(t)$, except being proportional to the long term miner commitment c through $h(t)$, is also proportional to the energy used to sign the transaction, i.e. to the main running cost of miners, making it a rather robust concept. The more electrical energy is used to sign a transaction, the more of it is needed to double spent it.

Finally, if we want to adapt the Bitcoin ecosystem to be conformant to Assumption 1, we have to find mappings $t_h(t) = \eta(t) \cdot b(t)$ and $h(t)$ and constant $c \in \mathbb{R}^+$ such that transaction difficulty $\delta(t) = t_h(t) \cdot h(t) = c \cdot v(t)$ for all admissible transactions. Assumption 1 thus implies that the transaction fees f should be always proportional to the transaction value $v(t)$:

$$f[\text{₿}] \propto v(t)[\text{₿}]. \tag{14}$$

3.3 Definition of the extended transaction

Now, we are ready to introduce our main vehicle of change. Let us extend the Bitcoin transaction t by two-parameters:

Definition 2 *$t_e(\dots,\lambda,\eta)$ is an extended Bitcoin transaction, where $\lambda \in (0,1]$ determines the fraction of all mining nodes which include this transaction into their blocks and the time investment η defined in (4) determines how long these miners on average look for an admissible hash of this transaction.*

The current Bictoin transactions t are naturally embedded in the extended transaction "space". We have $t_e(\dots,1,\eta_B) = t(\dots)$, i.e. the current Bitcoin transactions are included into blocks by all the available mining nodes and it takes the nodes on average 10 minutes to sign the blocks with the time investment η_B estimated in (5). This can be immediately expressed in difficulty language using (3):

$$\eta_B \approx \frac{2^{32}\Delta}{h_B \cdot \overline{b(\mathcal{B})}} \tag{15}$$

How to understand Definition 2 in a situation when $t \neq t_e$? For example $t_e(\dots,0.5,\eta_B)$ is a transaction mined by only a half of all mining nodes and the hashing difficulty is reduced by factor two which essentially means that the resulting hash can have a one less leading zero since

$$\eta_B \approx \frac{2^{32}(\Delta/2)}{(h_B/2) \cdot \overline{b(\mathcal{B})}}.$$

On average, a block containing such transactions will be created in the same time as current Bitcoin blocks under the constraint that the block size does not change.

Two extra parameters of the extended transaction t_e control two degrees of freedom of its difficulty defined in (10):

$$\delta(t_e) = h(t_e) \cdot \eta(t_e) \cdot b(t_e) = \lambda(t_e) h_B \cdot \eta(t_e) \cdot b(t_e), \tag{16}$$

i.e. the hashing resources used and the time invested to sign t_e.

We would like to recall that both the security $s(t_e)$ and the transaction fees $f(t_e)$ are quite naturally proportional to $\delta(t_e)$, see (6) and (13) respectively. Thus, by controlling λ and η we control transaction security and at the same time offer transaction fees proportional to this security. This is indeed not by coincidence but by respecting the Bitcoin economic realities while deriving the Bitcoin incentive based model presented in the previous sections. It also means that the model does not suffer from some obvious inconsistencies. From now on we use notation t instead of t_e since we will only consider extended transactions.

4 Hierarchical block structures (HBSs)

In the subsequent sections, we will employ the concept of extended transaction t from Definition 2 to suggest modifications of the Bitcoin block structure that allow users to transact with different levels of security, while aiming costs proportional to this security. Of course, users will be free to override any defaults and to transact using an increased level of security if they are prepared to pay

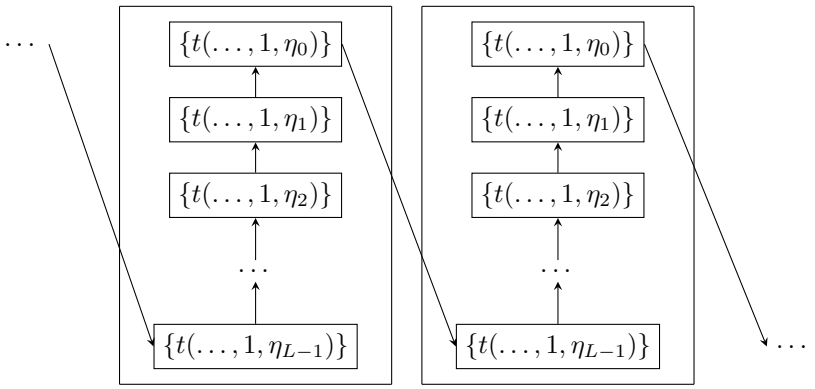

Figure 1: Blockchain of blocks with different security levels. Each sub-block represents a current Bitcoin block with a different hashing difficulty.

higher fees or to decrease the default security, if they feel that a lower one is sufficient and thus pay even less. Based on freedom of choice, a "marketplace of trust" should emerge quite naturally.

The first proposition in Section 4.1 would require minimal changes to the current Bitcoin ecosystem. Its further evolutions in Sections 4.2 and 4.3 lead to both more usability, complexity and more disruption of Bitcoin's status quo.

4.1 Single virtual computer

In this section we still assume that all transactions are seen by all mining and full nodes, i.e. Bitcoin network remains acting as a single "homogeneous" state-transition machine. Consequently, the scalability issues will remain unresolved. But we achieve the first necessary condition for Bitcoin to potentially become a successful general on-chain payment system: the ability to transact at different security levels and pay proportional fees.

We consider only transactions t signed by all miners, i.e. $\lambda(t) = 1$ for any admissible extended transaction t, which trivially implies $h(t) = \lambda(t)h_B = h_B$. The only degree of freedom of the extended transaction t we will tune here is the time investment η.

Let T denote the current average block time resulting from the difficulty Δ. In this section, instead of mining just one block in T, we propose mining a sequence of blocks containing transactions with different levels of security. As depicted in Figure 1, the blocks with $L \in \mathbb{N}^+$ different security levels governed by time investments $\eta_l, l = 0, \ldots, L-1$ are periodically repeating, forming imaginary super-blocks. In each super-block the lowest block contains transactions which require a minimal security η_{L-1} and the highest block the transactions which require the maximal security η_0 and η is monotonically decreasing in l, i.e.

$$\eta_{l+1} < \eta_l \quad \forall l \in \{0, \ldots, L-2\}. \tag{17}$$

4.1.1 Desing methodology

We approach the design of our first Bitcoin evolution prototype in two steps:

1. We segment Bitcoin transactions waiting in the mempool $\mathcal{T}$ for confirmation according to their output *values per bit* β, i.e. we handle the ratio

$$\beta(t) := \frac{v(t)}{b(t)} \tag{18}$$

   occurring in (6). Then we will assign them into individual security level sub-blocks from Figure 1.

2. Then we determine variable η_l for each level $l = 0, \ldots, L-1$ such that security complies with Assumption 1.

There certainly exists no single best way how to accomplish those tasks but we will present our best candidate and it is up to the Bitcoin community to eventually refine it, fill in details or alternatively propose a different solution altogether.

The developments are guided by performance on recent transaction data of the Bitcoin network, also because any eventual candidate for next Bitcoin protocol has to allow for smooth transition from the current state. One can employ Algorithm 5 from Appendix B to create a random sample of approximately 500 recent blocks starting from block height 650000 and ending with 700000. Our set is published on github [19], where the reader also finds the majority of code used for this article.

4.1.2 Segmentation of transactions

To illustrate the current distribution of aggregated output transaction values per bit $\beta(t)$ of Bitcoin network, we employed Algorithm 6 to plot a histogram of

$$\lg\left(\beta(t)\right),$$

where $\lg(\cdot) := \log_{10}(\cdot)$, together with the fitted log-normal probability density function, this resulting in Figure 2.

The distribution of $\beta(t)$ is certainly not perfectly log-normal, but rather skewed to the right side with a long tail. One can only speculate why but two factors seem to be in play. First, as already explained, Bitcoin is not usable for everyday transactions, certainly not on chain. Second, high $\beta(t)$ transactions do not pay fair fees to miners for their work [2]. I.e. low-value transactions are discouraged (or impossible) and high-value ones encouraged. E.g. if a whale moves 10000₿ to a cold storage, it can cost less than 10^{-6}% to do so. Nevertheless, the log-normal parametrization is already sufficiently accurate and the distribution of $\beta(t)$ should become more log-normal and symmetric if Bitcoin becomes more useful for everyday payments on chain.

[2] The author is not involved in mining business.

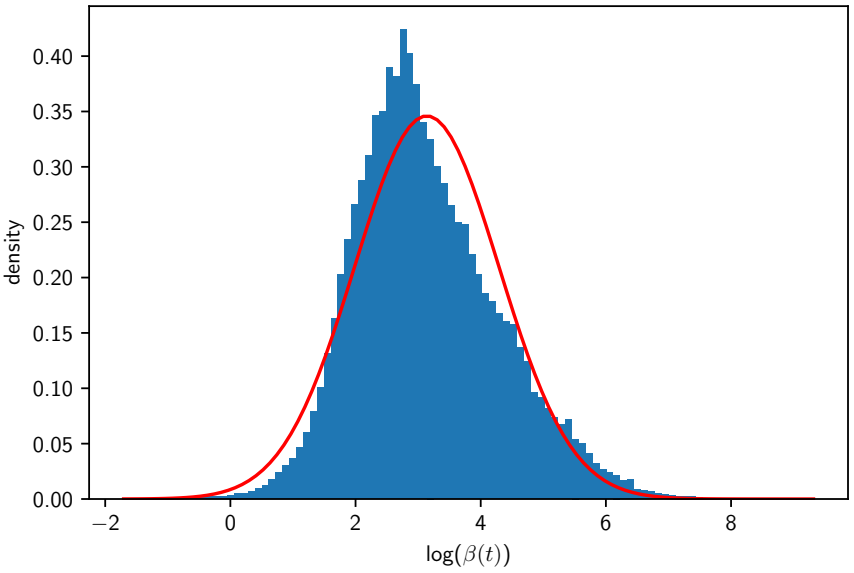

Figure 2: Distribution of per transaction aggregated output values plus the corresponding log-normal distribution

Altogether, we propose segmenting the unconfirmed Bitcoin transactions waiting in the mempool $\mathcal{T}$ into $L \in \mathbb{N}^+$ lists corresponding to individual levels from Figure 1 based on their $\lg(\beta(t))$ values. The following simple algorithm

can be employed:

Algorithm 1: A candidate for segmentation of a transaction set into subsets.

Data: $L, N \in \mathbb{N}^+$; a set of transaction tuples $\mathcal{T} = \{(t_i, \beta_i)\}_{i \in \mathcal{I}}$, $\mathcal{I} = \{i | i = 0, \ldots, N-1\}$, where t_i are some unique transaction identifiers and β_i the corresponding transacted values per bit.

Result: L sets $\mathcal{T}_i$, such that $\mathcal{T} = \cup \mathcal{T}_i$ and $\forall i, j \in \mathcal{I} : \mathcal{T}_i \cap \mathcal{T}_j = \emptyset$

```
T = sort set T in descending order of β_i;
for i ← 0 to L − 1 do
    T_i = {};
end
S_L = (lg(β_0) − lg(β_{N−1}))/L (or S_L = (⌈lg(β_0))⌉ − ⌊lg(β_{N−1}⌋)/L);
C = lg(β_0) − S_L; l = 0;
for i ← 0 to N − 1 do
    if lg(β_i) < C then
        l = l + 1;
    end
    T_l = T_l ∪ {(t_i, β_i)};
end
```

The code is practically self-explanatory. The interval between the minimal $\lg(\beta_{N-1})$ and the maximal $\lg(\beta_0)$ is uniformly divided into L subintervals and a transaction t is assigned to set $\mathcal{T}_0$ if

$$\lg(\beta(t)) \in [\lg(\beta_0) - S_L, \lg(\beta_0)]$$

and to $\mathcal{T}_l$ for $l \in \{1, \ldots, L-1\}$ if

$$\lg(\beta(t)) \in [\lg(\beta_0) - (l+1)S_L, \lg(\beta_0) - lS_L).$$

The transactions are thus segmented based only on β values and not on the frequency of their occurrence. We could instead compute L-quantiles of $\lg(\beta(t))$ for our set of transactions t. Then all sets $\mathcal{T}_l$ for $l \in \{0, \ldots, L-1\}$ would have similar cardinalities. However, it is much easier for a user to understand mapping between the security levels and the transaction values/costs when the interval $[\lg(\beta_0), \lg(\beta_{L-1})]$ is uniformly divided. Also, this choice is more forward looking, which becomes clear in the next sections.

A Python implementation of Algorithm 1 is presented in Algorithm 7 in Appendix B. We employ it to segment our Bitcoin transaction dataset. The results, obtained by running Algorithm 8, are presented in Table 1.

Let us discuss the results a little bit. We have divided our dataset containing almost $950k$ transactions from 493 blocks into only 6 levels, so that Table 1 fits on page. One immediately recognizes how extreme this segmentation really is.

Only 615 transactions, it is less than 0.1%, move more than 71% of all the transacted wealth, which on average represents barely more than 1 transaction per block. However, together they only pay fees in the hundreds of USD.

l	0	1	2	3	4	5
$\lvert\mathcal{T}_l\rvert$	615	24967	220097	571652	130420	1861
$\min(\beta(t))$	3e+07	4.4e+05	6.4e+03	9.3e+01	1.4	0.02
$\max(\beta(t))$	2.1e+09	3e+07	4.4e+05	6.4e+03	9.3e+01	1.4
$\overline{\beta(t)}$	3.6e+08	2.3e+06	6.3e+04	1.3e+03	4.5e+01	0.73
$\min(v(t))$	5.4e+10	6.8e+08	9.6e+06	7.6e+04	2.1e+03	3.1e+02
$\max(v(t))$	6e+12	6.6e+11	1.6e+11	7.4e+09	1.2e+08	3e+05
$\overline{v(t)}$	8.7e+11	6.4e+09	2.3e+08	6.9e+06	2.6e+05	3.7e+03
$\sum v(t)$	5.3e+14	1.6e+14	5.1e+13	3.9e+12	3.4e+10	6.9e+06

Table 1: Distribution of transaction values after segmentation: $L = 6$

On the other side of the spectrum in level 5 are few, almost $2k$ transactions, which move on average 3700 sats, at current exchange rate BTCUSD less then 2 USD. This is significantly less them our estimate of electricity costs per transaction of 14.3 USD from Appendix A.4. We can argue based on this observation, that costs are a limiting factor for inclusion of transactions into the blockchain. And it is indeed a rational choice to exclude them within the current set-up of Bitcoin blockchain.

4.1.3 Determining time investments

Let $\{\mathcal{T}_l\}_{l\in\mathcal{L}}$, where $\mathcal{L} := \{0, \dots, L-1\}$, be any segmentation of mempool of Bitcoin transactions by Algorithm 1. By $\mathcal{B}_l$ we depict the sub-blocks of Figure 1, such that $\mathcal{B}_l \subset \mathcal{T}_l$. All transactions within a segment $\mathcal{T}_l$ enjoy the same miner effort if included in a block $\mathcal{B}_l$, i.e. $\eta(t) = \eta_l$ for all $t \in \mathcal{T}_l$ and $l \in \mathcal{L}$. Using (6) and (10) we obtain:

$$\overline{s(t)}|_{\mathcal{T}_l} = \frac{h_B \cdot \eta_l}{|\mathcal{T}_l|} \sum_{t\in\mathcal{T}_l} \frac{b(t)}{v(t)}. \tag{19}$$

Further, all transaction t in each $\mathcal{T}_l, l \in \mathcal{L}$ have by construction similar output values per bit $\beta(t) = v(t)/b(t)$. Consequently, we may substitute in the average value $\overline{\beta(t)}|_{\mathcal{T}_l}$ of $\beta(t)$ and we obtain

$$\overline{s(t)}|_{\mathcal{T}_l} \approx \frac{h_B \cdot \eta_l}{\overline{\beta(t)}|_{\mathcal{T}_l}}, \tag{20}$$

which is a simple formula containing only h_B and η_l and average values of s and β per segment $\mathcal{T}_l$.

Under Assumption 1 we strive to secure the same security regardless of transaction value per bit β, which translates to:

$$\overline{s(t)}|_{\mathcal{T}_k} = \overline{s(t)}|_{\mathcal{T}_l} \quad \forall\, k, l \in \mathcal{L}, \tag{21}$$

i.e. Assumption 1 is satisfied in a per sub-block fashion. Using 20 we obtain practically the final condition which allows us to determine the time investments $\eta_l, l \in \mathcal{L}$ of the individual sub-blocks:

$$\frac{h_B \cdot \eta_k}{\overline{\beta(t)}|_{\mathcal{T}_k}} = \frac{h_B \cdot \eta_l}{\overline{\beta(t)}|_{\mathcal{T}_l}} \quad \forall\ k, l \in \mathcal{L}, \tag{22}$$

i.e. we look for a unknown $c_\eta \in \mathbb{R}^+$ such that

$$\eta_l \left[\frac{\mathrm{s}}{\mathrm{b}}\right] = c_\eta \left[\frac{\mathrm{s}}{\ddot{\mathrm{B}}}\right] \cdot \overline{\beta(t)}|_{\mathcal{T}_l} \left[\frac{\ddot{\mathrm{B}}}{\mathrm{b}}\right] \quad \forall l \in \mathcal{L}, \tag{23}$$

where c_η plays the role of c from the last lines of Section 3.1 under the specific circumstances of this section. It belongs to the only one factor of difficulty δ we control here: the time investment η. It is common to all levels $l \in \mathcal{L}$ and its unit (i.e. $s/\ddot{\mathrm{B}}$) helps us to understand that it essentially represents computational time invested to secure one $\ddot{\mathrm{B}}$ of transacted wealth.

Assume for a moment that a transition from a current block to a new multi-block is possible, so to say, at once. We can easily compute c_η from current global difficulty Δ. First, based on the definition of transaction difficulty (10), we spread the current hashrate in the correct way among the sub-blocks $\{\mathcal{B}_l\}_{l\in\mathcal{L}}$ from Figure 1 while achieving the same expected block creation time of 600s for the super-blocks by asking that:

$$h_B \sum_{l\in\mathcal{L}} \eta_l \sum_{t\in\mathcal{B}_l} b(t) = 2^{32}\Delta. \tag{24}$$

Thus it holds

$$600s \overset{(3)}{=} \frac{2^{32}\Delta}{h_B} = \sum_{l\in\mathcal{L}} \eta_l \cdot b(\mathcal{B}_l) \overset{(23)}{\approx} c_\eta \sum_{l\in\mathcal{L}} \overline{\beta(t)}|_{\mathcal{B}_l} \cdot b(\mathcal{B}_l), \tag{25}$$

where the last relation is only approximation, since blocks $\{\mathcal{B}_l\}$ are chosen from mempool's segmentation $\{\mathcal{T}_l\}$ by miners. It can be expected that miners will prefer the transactions paying higher fees per bit in each sub-block $\mathcal{B}_l$, which should be correlated to the value $\beta(t)$ being transacted. One could redefine (23) by using blocks instead of the mempool segmenation, but no mathematical trick can cover the fact that miners are responsible for creation of valid blocks and distributions of transactions in $\{\mathcal{B}_l\}_{l\in\mathcal{L}}$ can potentially differ from those in mempool segmentation $\{\mathcal{T}_l\}_{l\in\mathcal{L}}$.

Assume further, that $\overline{\beta(t)}|_{\mathcal{B}_l}$ can be approximated as

$$\overline{\beta(t)}|_{\mathcal{B}_l} \approx \frac{\overline{v(t)}|_{\mathcal{B}_l}}{\overline{b(t)}|_{\mathcal{B}_l}} \tag{26}$$

with a sufficient accuracy. Then we can continue simplifying (25) as follows:

$$c_\eta \sum_{l\in\mathcal{L}} \overline{\beta(t)}|_{\mathcal{B}_l} \cdot b(\mathcal{B}_l) \approx c_\eta \sum_{l\in\mathcal{L}} \frac{\overline{v(t)}|_{\mathcal{B}_l}}{\overline{b(t)}|_{\mathcal{B}_l}}|_{\mathcal{B}_l} \cdot \overline{b(t)}|_{\mathcal{B}_l}|\mathcal{B}_l| = c_\eta \sum_{l\in\mathcal{L}} v(\mathcal{B}_l), \tag{27}$$

i.e.

$$c_\eta \approx \frac{600s}{\sum_{l\in\mathcal{L}} v(\mathcal{B}_l)} = \frac{2^{32}\Delta}{h_B \sum_{l\in\mathcal{L}} v(\mathcal{B}_l)}. \tag{28}$$

We can choose (25) or (28) to determine c_η, depending on the accuracy of (26). The second expression explains very clearly what c_η represents: the ratio of expected super-block creation time $600s$ to the total transacted value of this super-block $v(\{\mathcal{B}_l\}_{l\in\mathcal{L}})$, i.e. in short time of all miners to wealth of all users.

The relation (28) also suggests how the computation of c_η should be implemented in practice. Its expression contains difficulty Δ, unknown and variable total hash rate of Bitcoin network h_B plus again variable total transacted value $v(\{\mathcal{B}_l\}_{l\in\mathcal{L}})$. Just like in the case of Δ, c_η can be computed only probabilistically. There is no reason to change what is working very well. We propose adjusting c_η every 2016 blocks with the goal to achieve average super-block time of 10 minutes.

Now, we can return back to relation (23) with an already determined c_η. Purely theoretically, one could directly use (23) to calculate η_l for each new super-block. However $\overline{\beta(t)}|_{\mathcal{B}_l}$ can fluctuate a lot, especially at the higher levels, see e.g. level 0 in Table 1, where we have only 615 transaction in 493 blocks, i.e. sometimes level 0 may be even empty. Second, since η controls difficulty of the sub-blocks, which are essentially normal Bitcoin blocks, all the attacks possible when adjusting difficulty per block (and potentially more) would be possible [17]. Indeed, difficulty is fundamentally a security mechanism and so are all the time investments $\{\eta_l\}|_{l\in\mathcal{L}}$. Thus we again propose updating them in the same fashion as difficulty of normal blocks is updated - every 2016 blocks:

$$\eta_l = \frac{c_\eta}{2016} \sum_{i=0}^{2015} \overline{\beta(t)}|_{\mathcal{B}_{l,i}} \quad \forall l \in \mathcal{L}, \tag{29}$$

where $\mathcal{B}_{l,i}, i = 0, \ldots, 2015$ for each $l \in \mathcal{L}$ are the last 2016 sub-locks in the blockchain.

Let us compute c_η and η_l for our dataset of transactions, using function `compute_c_eta_and_eta` from Algorithm 9. The estimate for c_η based on (25) reads

$$\hat{c_\eta} \approx 0.036 \frac{\mathrm{s}}{\ddot{\mathrm{B}}} \tag{30}$$

and

$$\hat{\eta} \approx (0.13, 8.2\times10^{-4}, 2.3\times10^{-5}, 4.7\times10^{-7}, 1.6\times10^{-8}, 2.6\times10^{-10}) \tag{31}$$

with the values representing time in seconds spent by all the miners to secure one bit of information. These values are corresponding to levels from Table 1. When we compare them with the corresponding estimated value of time investent for current Bitcoin blocks $\hat{\eta}_B \approx 7.15\times10^{-5}$ using (5), we see that the transactions in the first 2 sub-blocks would enjoy higher immediate security guaranties than nowadays. Looking at Table 1, it concerns transactions starting from $v = 6.8\ddot{\mathrm{B}}$. On the other hand, a complex transaction even moving as much

as $v = 1600Đ$ could be included in a sub-block $l = 2$ and pay fees more than 3 times lower than today, see (34).

Based on (31), we can estimate the time required to sign the individual levels for an average block as

$$\hat{t}_l \approx \hat{\eta}_l \cdot \overline{b(\mathcal{B}_l)} \tag{32}$$

and thus using function `compute_time_per_level` from Algorithm 9 we get:

$$\hat{t} \approx (429, 124, 44.5, 2.85, 0.0247, 5.89 \times 10^{-6}). \tag{33}$$

Here we have to, for the first time, distinguish two possibilities: whether miners are expected to broadcast each sub-block $\mathcal{B}_l, l \in \mathcal{L}$ or they should work on sub-blocks independently from each other and broadcast only the final multi-block. When miners broadcast the sub-blocks, transactions are verifiable sooner than when they only broadcast the finalized multi-blocks. However since the difference between the mining times of two successive sub-blocks belonging to a same level $l \in \mathcal{L}$ is on average 10 minutes, the confirmation will practically never be immediate and the upper bound is on average 10 minutes even if the next sub-block $\mathcal{B}_l$ is able to contain all waiting $l-$level transactions. From the usability point of view, there is thus no significant difference between these options. Since in this section we consider that all the transactions are seen by all the miners (and full nodes), one might better opt to broadcast only the finalized blocks. If we were able to transition from a current block to a multi-block at once, this would allow us to preserve the current broadcasting implementation and to introduce a high number of levels L.

However, to be able to scale PoW in the next sections, starting from Section 4.2, we have to broadcast the individual sub-locks. One observes in (33) that for the current distribution of transaction output values v, more then 4 levels of sub-blocks are practically unthinkable, since the last sub-blocks would be created in fractions of a second and broadcasting them safely takes certainly much longer. Technically, based on our understanding of state of the art, three levels are certainly possible. E.g. Ethereum's blockchain has 15 seconds block time. The fourth level might be a challenge. One may consider a hybrid approach such that the last sub-blocks are mined in a sequence and only then broadcasted together, here e.g. sub-blocks $\mathcal{B}_l, l \in \{2, 3, 4, 5\}$. This is very easy to implement, namely miners broadcast only when sufficient work has been done.

But why are the mining times for the last sub-blocks in (33) so small? The transition from a single block to a multi-level block is a classical example of the chicken and egg problem. Since transactions t with small $v(t)$ on-chain are expensive, they are rare.

4.1.4 Value transfer downstream

Whatever form the broadcasting scheme takes, we propose the following iterative approach to gradually increase the number of possible levels L. First, introduce a viable number of initial levels - e.g. three. As argued in Section 3.2, users

should pay fees proportional to difficulty $\delta(t)$, which in settings of this section equals

$$\delta(t) = h_B \cdot \eta_l \cdot b(t)$$

for a transaction t from a block $\mathcal{B}_l, l \in \mathcal{L}$. It is, the *fees per bit* ϕ_l at level $l \in \mathcal{L}$ should be proportional to η_l:

$$\phi_l \left[\frac{\text{Ƀ}}{b} \right] \propto \eta_l, \tag{34}$$

which under validity of Assumption 1 is also proportional to the average value per bit transacted within the block $\mathcal{B}_l$, see (23).

Looking at (31), transactions of a second sub-block $\mathcal{B}_1$ should thus pay $0.13/8.2 \times 10^{-4} \approx 159$-times less ϕ and those within a third sub-block $\mathcal{B}_2$ approximately 5650-times less than transactions in a first sub-block $\mathcal{B}_0$, whatever the corresponding absolute values would be. These relative differences represent strong economic incentives to choose a lower level blocks over a higher level one. Moreover, if accepted by miners, a high value transaction belonging normally e.g. in a highest 0-level block can be included in a lower lever block and user can choose to wait for a confirmation with a number of $0-$level blocks. These two forces would certainly move a lot more wealth from higher levels to lower levels, filling those in and consequently allowing to increase the number of levels L. A mechanism for automatic adaptation of L with some save maximum of e.g. 16^3 can be included into protocol. In essence, one could e.g. ask that the last sub-block time is at least a predefined constant $t_{\min}(\text{block})$:

$$t_{L-1} \approx \eta_{L-1} \cdot \overline{b(\mathcal{B}_{L-1})} > t_{\min}(\text{block}). \tag{35}$$

4.1.5 Block rewards

If sub-blocks $\mathcal{B}_l, l \in \mathcal{L}$ are never broadcasted, only the final multi-block, no changes to reward scheme are necessary. The new multi-block takes the place of the current Bitcoin block.

If sub-blocks are broadcasted, it is necessary to distribute block rewards r among the individual levels in a multi-block from Figure 1. First, the expectation that at most 21M of coins will ever exist should be preserved. To comply, it is sufficient to apply the current block reward scheme at the multi-block level, since we aim at the same average multi-block creation time of 600s . This reward then should be split among the individual sub-block based on their average creation time. We propose assigning rewards

$$r_l := \frac{\hat{t}_l}{\sum_{l \in \mathcal{L}} \hat{t}_l} r \tag{36}$$

for each $l \in \mathcal{L}$, where $\hat{t}_l$ are estimators of average time to mine sub-block $\mathcal{B}_l$, see (32), computed at the same time and in the same way as time investments

[3] From 21000000 Ƀ to 1 sat in lg-scale.

Figure 3: New hybrid blockchain with multi level blocks included between the normal Bitcoin blocks.

$\eta_l, l \in \mathcal{L}$ are computed in (29), i.e.

$$\hat{t}_l := n_l \cdot \overline{b(\mathcal{B}_l)} \approx \frac{\eta_l}{2016} \sum_{i=0}^{2015} b(\mathcal{B}_{l,i}) \quad \forall l \in \mathcal{L}, \tag{37}$$

where again $\mathcal{B}_{l,i}, i = 0, \ldots, 2015$ for each $l \in \mathcal{L}$ are the last 2016 sub-locks in the blockchain.

4.1.6 How to transition from block to multi-block?

When switching from the current Bitcoin block structure to the multi-layered solution of Figure 1, continuous operation has to be ensured. This is not a trivial task and it has to be carefully planned and its full treatment is outside of the scope of this publication. Nevertheless, in our view, it should be done in a maximally not intrusive way where new functionality is a possibility, not a dictate. Here, we present such an approach on conceptual level.

Bitcoin represents among other important things a breakthrough innovation, store of accumulated trust and even a promise of a better future for mankind. Naturally, its development is less agile and more conservative and careful than those of other relatively never blockchains. For this reason, what we propose in this paper is a stepwise and systematic road map of metamorphosis, which makes all the three main parties, i.e, users, miners and developers, comfortable enough to undergo the transition, in our view a vital one.

Consequently, we propose working in an additive fashion. The first elemental idea is presented in Figure 3. New multi-blocks are included between the current Bitcoin blocks. Initially, the current system remains practically intact. The new multi-blocks form a small seed which will compete in usefulness with the old implementation. We are persuaded that if given chance, it will grow in time and overtake the old blocks due to its superior economic and from Section 4.2 also technical properties.

Mathematically speaking, we are constructing a homotopy from an old system to a new one, i.e. instead of the state transition function (1) we will consider the following one:

$$\tau_\lambda = (1 - \lambda)\tau_o + \lambda\tau_n, \tag{38}$$

where τ_o is the current state transition function, τ_n is the new one representing the hypothetical Bitcoin blockchain with multi-blocks only and λ is a real parameter from interval $(0, 1)$ which "measures" to which degree the old blocks and to which degree the new multi-blocks are used for transactions. This homotopy is the second elemental idea.

In the beginning, only old blocks are used, i.e. $\lambda = 0$. If new multi-blocks succeed and all transactions employ them, $\lambda = 1$. Here, parameter λ is controlled predominately by users and then by miners since they decide on the inclusion of transactions into blocks. Thus, if multi-blocks did not represent a winning formula for users and miners and indeed also developers in the beginning of the road, they could never succeed.

The reader can immediately see that this homotopy mechanism can be used to relatively safely and democratically introduce any new functionality. A disadvantage in our case is that the Bitcoin software has to support two or more block versions, at least temporarily. If λ reaches almost 1, users might be theoretically forced to transfer values to new blocks within a transition period of e.g. few years. Then support for old blocks could be deprecated. However, this would be a rather harsh measure. Consequently, the system should most probably remain backward compatible. Also, once even a single multi-block is non-empty, it caries some information about the Bitcoin state s and the state transition function τ_n can not be easily disposed of. Again, values would have to be first transfered to old blocks. In short, Equation (38) is certainly no tool for experimentation and only worthy candidates τ_n should be considered.

Assume that one old block and multi-block pair is created on average every 10 minutes. Then

$$\lambda = \frac{t_n}{600s}, \tag{39}$$

where t_n is average time necessary to mine new multi-blocks. In practice, even if initially t_n is almost zero, since the first multi-block contains only header and no transactions, one has to reserve a positive time for the block propagation. Based on Figure 4 of a rather dated paper [10], we have to expect at least 1ms/B for propagation of en empty Bitcoin block to 90 percentile of the network, i.e approximately 80ms for a header of 80B and potentially much more.

Let us apply Algorithm 1 to obtain the corresponding segmentation for $L = 2$. What would be t_n if the transactions for $l = 0$ stayed in old blocks and those from $l = 1$ moved to new multi-block, here with one level only? The results are presented in Table 2. In comparison to Table 1 we added a $\overline{b(t)}$-row to illustrate that small transaction are actually more "rich". And we added η_l- and t_l-rows which are computed according to Algorithm 9. Otherwise, the table is generated in the same way as Table 1.

Even if the second segment $\mathcal{T}_1$ of "the mempool" $\mathcal{T}$ contains more transactions and is bigger, under the validity of Assumption 1, the transactions included within it enjoy the same security by using the hash rate h_B of the Bitcoin network for less then 2 seconds out of 600s, particularly $t_n \approx 1.92s$. And they should pay fees approximately $\eta_0/\eta_1 \approx 1120$ times lower on average then those transactions in $\mathcal{T}_0$. Maybe more importantly, the energy usage per transaction is also approximately 1120-times lower. It means, that a small investment of miners time, i.e. approximately $1.92/598 = .32\%$, represents the capacity to sign more than two thirds of all the transactions while still providing an adequate security and decreasing energy consumption of small value transactions three fold. We hope, that the huge potential effect on the ability of the Bitcoin

l	0	1
$\lvert\mathcal{T}_l\rvert$	245679	703933
$\overline{b(\mathcal{T}_l)}$	5.33e+02	6.66e+02
$\min(\beta(t))$	6.39e+03	0.0197
$\max(\beta(t))$	2.08e+09	6.39e+03
$\overline{\beta(t)}$	1.2e+06	1.07e+03
$\min(v(t))$	9.65e+06	3.06e+02
$\max(v(t))$	6e+12	7.4e+09
$\overline{v(t)}$	3.04e+09	5.61e+06
$\sum v(t)$	7.46e+14	3.95e+12
η_l	0.000281	2.52e-07
t_l	5.98e+02	1.92

Table 2: Distribution of transaction values after segmentation: $L = 2$

network to serve small regular transactions on-chain is obvious.

Is it necessary to motivate somehow the use of new multi-blocks? The short answers is no. First, it is reasonable to assume that transaction fees and environmental concerns on its own would be sufficient motivation to employ multi-block for small casual transactions. With growing confidence, higher and higher values would be transacted. If found desirable, the natural economic incentives which promote value transfer described in Section 4.1.4 could be strengthened. E.g. those users using multi-blocks could enjoy a higher security $t_n + \epsilon$ for some $\epsilon > 0$. However, since transaction fees are determined by market forces, such measures would be probably ineffective.

Whether an equilibrium with a $\lambda < 1$ would be found or the current blocks would stopped to be used altogether, i.e. λ would eventually reach 1, is a difficult question to answer. The answer depends on too many unknowns, among them on implementation details. We may only speculate that eventually the current blocks would probably survive and work as a bank for high value transactions, since (6) is only a model and in reality the relation between $\delta(t)$ and $v(t)$ is very likely non-linear. E.g. those making high-value transactions are maybe prepared to accept a higher risk or are only making transactions between two wallets they own and thus double-sending is of no concern or they are maybe able to wait longer for confirmation. On the other hand miners would realize this and they would include also transaction paying less than a "fair" linear-model based value into standard blocks.[4] But most importantly sub-blocks are essentially current Bitcoin blocks and one of them is the highest in hierarchy. Thus the standard block preceding a multi-block can be seen as the highest level sub-block for $l = -1$ and λ is then merely a degree-of-freedom parameter which has to be found to balance the two state transition functions: τ_o and τ_n.

To summarize, we propose:

[4]The same reasoning applies to relations among sub-blocks of a multi-block.

- Introducing multi-blocks between the standard blocks as depicted in Figure 3.
- Adapting the consensus algorithm to accommodate the state transition function (38), where λ is defined by (39), where t_n could be computed at the same time and in the same way as $\eta_l, l \in \mathcal{L}$ are computed in (29), i.e. by averaging the multi-block time through last 2016 blocks.
- Propagating the whole multi-blocks only which allows us to introduce from the beginning practically any number of levels L, where $L \approx 16$ seems to offer good granularity.
- Dividing the block reward r honestly between τ_o and τ_n, i.e. λr goes to miner finding the next multi-block and $(1-\lambda)r$ to miner finding the next standard Bitcoin block.

The l.h.s. of Equation (25) contains t_n instead of $600s$. All the formulas to compute c_η and η etc. are adapted accordingly.

4.2 Sharding levels

In Section 4.1 we introduced our first hierarchical block structure (HBS), depicted in Figure 1. It employed the concept of extended transaction t, see Definition 2, but only those signed by all the miners and seen by all the full nodes, i.e. $\lambda(t) = 1$ which implies $h(t) = h_B$. We showed that by varying time investment η, see (4), this HBS allows us to successfully align three things: transaction security, fees and electric energy consumption. This was all possible because we have recognized and already partially employed the fact that all transactions are not equal. Namely, that some of them move fortunes and some of them only pennies. But since the constraint $h(t) = h_B$ remained untouched, so did the scalability, as it is succinctly expressed in Section 1.

Assume for a moment that the constraint $h(t) = h_B$ remained untouched and we simply allowed the new multi-blocks from Figure 3 to have flexible size. They could indeed accommodate more transactions. But to propagate big blocks costs time [10] and the bigger their size, the more difficult it becomes to run a full node, promoting the type of centralization we really want to avoid.

As discussed in Appendix A.6, it is not the centralization of mining but that of control which is problematic. Miners and developers, even if they are indeed privileged, merely serve the user base of the Bitcoin community. And users cast they votes every day by buying or selling coins. This type of control is obviously fully independent on blockchain implementation including its consensus algorithm. But the second voting mechanism, the choice to run or not a full node or its specific software version has to be preserved. Altogether, simply increasing multi-blocks size is not the way to scale.

What we however can do is to distribute hashing capacity and to adapt the concept of a full node. The underlying motivation is simple. Again, when one does high-stake transactions, he will try to avoid all risks. Let us take

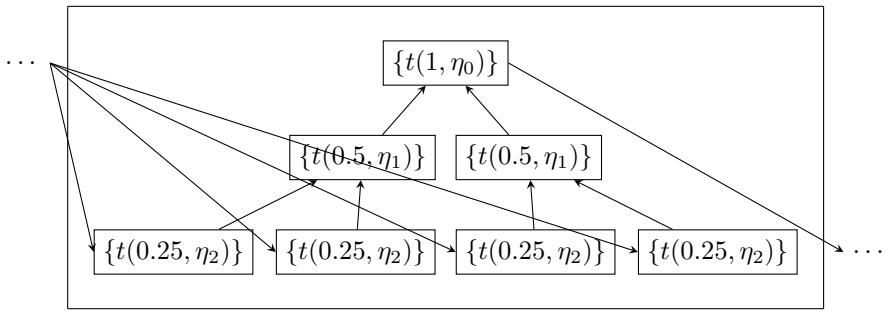

Figure 4: Sharding of individual levels for $L = 3$.

as an example a private property transaction. Buying e.g. a house without professional legal support is probably unthinkable for most people. But when paying for ice-cream we sometimes not even count change. The difference in due diligence simply stems from different values of those transactions. Which aspect matters to us the most here is how transactions are validated and recorded. Private property ownership together with the supporting documents are usually recorded in at least a county register, i.e. in a centralized database, sometimes even a publicly searchable one. On the other hand who knows a lot about your last ice cream transaction paid in cash?

This observation implies for Bitcoin transactions t that we should require those with the highest value per bit $\beta(t)$ to be signed by all miners and validated and recorded by everybody, i.e. all nodes. But those with lower values can be mined by only a fraction of the global hashrate h_B and validated and recorded by only some fraction of the "full" nodes.

We can build directly on the work presented in Section 4.1, where $\beta(t)$ decreases as the level index $l \in \mathcal{L}$ increases, i.e., in effect, also the required due diligence decreases. Thus we can exploit the first degree of freedom of an extended transaction t, see Definition 2, i.e. λ, and split the individual levels in Figure 1. More accurately, we split the mining work, it is the transactions waiting in the mempool for confirmation among the mining nodes in the network. The validation work is split as well. The highest level for $l = 0$ is not split at all. The transactions within the corresponding blocks are worked on by all the mining nodes and seen by all the full nodes. It behaves as a current bitcoin block.

One possible realization of the corresponding HBS is depicted for $L = 3$ in Figure 4. Here, we have employed a binary tree structure but any number of children per a non-leaf node c is possible. For simplicity but without any loss of generality we assume that $c = 2$ except when otherwise noted.

Formally, we still map the transactions from $\mathcal{T}$ based on their β and/or based on what users desire to one-dimensional space with the main parameter being $l \in \mathcal{L}$. The individual sub-blocks $\mathcal{B}_{l,s}, s \in \{0, \ldots, 2^l - 1\}$ for a certain level $l \in \mathcal{L}$ are principally equivalent to each other with respect to security, since they

employ the same time investment η_l.

Even if the idea is simple, the set up is much more complex than that of Section 4.1. And to reach any form of consensus in the community will be probably an equally complex and challenging process. Nevertheless, we again suggest introducing any new functionality using the homotopy mechanism from Section 4.1.6, which makes the transition process a rather safe one. It is even possible that the development of this section and Section 4.1 can be aggregated to avoid a two stage transition process.

4.2.1 Distribution of hashing resources and its security

First, in the HBS from Figure 4, we assume implicitly that the hashrate h_B can be uniformly divided among the shards. This can be a problem especially for a higher number of levels L, when some of the biggest miners can have capacity higher then is necessary to sign a single shard $\mathcal{B}_{l,s}$. We expect that such sophisticated miners have no problem to split their mining power among the individual shards if necessary.

Moreover, nowdays hashing capacity is mostly concentrated into mining pools. These again sophisticated actors would have to accommodate their business model and to distribute the hashing power according to the new multi-layer sharded block structure.

We propose assigning miners to shards $\mathcal{B}_{l,s}, s \in \{0, \dots, 2^l - 1\}$ based on a hash of their IP address or any other similar unique network identifier. One can take e.g. few leading bits/bytes of this hash to assign them a shard to work on. For convenience, we define an abstract shard function s which does precisely this:

$$s(l, \text{hash}(IP)) : (l, IP) \to s. \tag{40}$$

A Python implementation of (40) is presented in Algorithm 10. A miner obtains not only a shard s_l at level l but a branch $(s_0, s_1, s_2, \dots, s_l), s_0 = 0$ of the tree HBS, he should be working on or better to which he belongs. This branch is such that each $\mathcal{B}_{i,s_i}$ is a parent sub-block of shard $\mathcal{B}_{i+1,s_{i+1}}$ for $i \in \{0, \dots, l-1\}$. The probabilities to belong to a certain shard at any level $i \in \{0, \dots, l\}$ are uniformly distributed.

Even if an attacker achieves more than 50% hashrate for a particular shard $\mathcal{B}_{l,s}$ for a sufficiently high level $l \in \mathcal{L}$, his ability to double spent is very limited. After broadcasting its block, miners from a higher level $l-1$ are already including the hash of its block into their block secured by a much higher accumulative hashrate. To counterfeit any block later is also impossible, since after broadcasting it is available to all miners from level $l-1$ and also full nodes keeping records of this part of the tree HBS. In the end, the whole multi-block from Figure 4 is secured by cumulative hashrate of the whole network h_B when the corresponding root block $\mathcal{B}_{0,0}$ is mined. Consequently, the whole tree would be at least as secure as the current Bitcoin implementation.

One can also consider an attacker having enough hashrate to theoretically double spend at a certain level l, but to attack at a lower level $l+1$. Such

attacks are in our view highly unlikely since, simply put, monetary rewards are exponentially decreasing function of l and one time only at level $l+1$. This is in sharp contrast to continuously flowing block rewards and transaction fees at the level l at which the miner normally operates, together with a possible crushing affect on the BTC market price.

To further strengthen security, the function (40) may be randomized. The hash can include a random string determined e.g. during assemble of a previous multi-block by all the miners in a distributed fashion, so that the shards assigned by (40) are always different and not fixed. The following algorithm to generate such a global nonce comes to mind:

Algorithm 2: An algorithm for generation of a global random nonce.

Data: $L \in \mathbb{N}^+$;
Result: A random nonce $R_{0,0}$ saved at the root block $\mathcal{B}_{0,0}$.

```
1  while 1 do   // an abstract loop corresponding to multi-blocks
2      for l ← L − 1 to 0 do
3          for s ← 0 to 2^l − 1 do                          // non-blocking
4              if miner should mine B_{l,s} then
5                  ...
6                  if l = L − 1 then
7                      R_{l,s} = hash(rand())
8                  end
9                  if l < L − 1 then
10                     broadcasting and validating blocks B_{l+1,2s}, B_{l+1,2s+1}
11                     R_{l,s} = hash(rand(), R_{l+1,2s}, R_{l+1,2s+1})
12                 end
13                 Add R_{l,s} to the header of B_{l,s}
14                 mine B_{l,s}
15                 ...
16             end
17         end
18     end
19 end
```

The generation of this random nonce is local as each miner of a block $\mathcal{B}_{l,s}$ for some $0 \leq l < L-1$, $s \in 0, \ldots, 2^l - 1$ needs to know only the lower level nonces from two children blocks $\mathcal{B}_{l+1,2s}$ and $\mathcal{B}_{l+1,2s+1}$. But the result $R_{0,0}$ is available to everybody, since $\mathcal{B}_{0,0}$ is mined by all the miners and validated and recorded by all the full nodes. Instead of (40) we use

$$s(l, \text{hash}(IP, R_{0,0})) : (l, IP) \to s. \tag{41}$$

Admittedly, a sophisticated attacker can own a range of IPs and to route traffic always through one which maps to a fixed shard s, and to e.g. become a miner with more then 50% hashrate for this shard. However, information about IPs is publicly available and it is quite easy to propose countermeasures. E.g. it is more costly to control static IPv4 addresses with different 2 or 3 leading

bytes. I.e., to create a distribution of IPs which seems randomly chosen is difficult. Altogether, this measure substantially increases costs to perform any attack where concentration of hashing resources in a shard is necessary, however unlikely they are based on the previous reasoning in this section.

4.2.2 How to assign transactions to individual shards?

We again propose using a variation of function (40) to assign transactions t, i.e. work, to individual shards $s \in \{0, \dots, 2^l - 1\}$ for some $l \in \mathcal{L}$, particularly

$$s(l, \text{hash}(i(t))) : (l, i(t)) \to s, \tag{42}$$

where input $i(t)$ of transaction t is a single Unspent Transaction Output(UTXO) of some previous transaction.

The tree hierarchical block structure (HBS) from Figure 4 that we consider in this section does not allow us to support classical Bitcoin transactions with multiple inputs, only at the highest level block $\mathcal{B}_{0,0}$. Why? Since the whole point of sharding the sub-blocks $\mathcal{B}_l, l \in \mathcal{L}$ is to allow for independent signing of transactions and their validation in the individual shards $\mathcal{B}_{l,s}, s \in 0, \dots, 2^l - 1$, the mapping between UTXOs and transactions has to be unique. Otherwise, an attacker can double spend its UTXOs in possible multiple sub-blocks from set $\{\mathcal{B}_{l,s}\}$. Namely, he could create different non-disjoint subsets S_1 and S_2 of UTXOs he controls, such that $s(l, \text{hash}(S_1)) \neq s(l, \text{hash}(S_2))$ and those would be assigned to different shards $s_1, s_2 \in \{0, \dots, 2^l\}$. The UTXOs in $S_1 \cap S_2$ would be then double spent.

To aggregate multiple UTXOs in one transaction is still possible within the HBS of Section 4.1. Or indeed, one classical Bitcoin transaction with multiple inputs can be substituted by a set of simple one input transactions transacting to the same output(s). Altogether, we pay at most a small loss of convenience for possibility to scale on-chain transaction capacity.

4.2.3 Distributed computation on tree HBS

Let us now adapt the important formulas from Section 4.1 to the new HBS, since many of them require some revision. First, sub-blocks $\mathcal{B}_{l,s}$ from level $l \in \mathcal{L}$ for any shard $s \in \{0, \dots, 2^l - 1\}$ employ hashrate

$$h_l := \frac{h_B}{2^l} \tag{43}$$

and consequently transactions within those blocks enjoy, instead of (19), the following security:

$$\overline{s(t)}|_{\mathcal{T}_l} = \frac{h_B \cdot \eta_l}{2^l |\mathcal{T}_l|} \sum_{t \in \mathcal{T}_l} \frac{b(t)}{v(t)}. \tag{44}$$

By repeating the derivation from Section 4.1.3 we get the formula for time investments of individual levels under validity of Assumption 1:

$$\eta_l \left[\frac{\mathrm{s}}{\mathrm{b}}\right] = 2^l c_\eta \left[\frac{\mathrm{s}}{\ddot{\mathrm{B}}}\right] \cdot \overline{\beta(t)}|_{\mathcal{T}_l} \left[\frac{\ddot{\mathrm{B}}}{\mathrm{b}}\right] \quad \forall l \in \mathcal{L}, \tag{45}$$

where again $c_\eta \in \mathbb{R}^+$ and the interpretation of this variable does not change. We see that since hash power of miners is divided among the shards $\mathcal{B}_{l,s}$ for $l > 0$, 2^l-times more time than in Section 4.1 has to be invested to compute a hash-signature of those blocks to achieve the same security for transactions inside them. When we compare the exponential grow 2^l with exponential decrease of $\overline{\beta(t)}|_{\mathcal{T}_l}$ for a sufficiently small L (e.g. see Table 1), we see that (17) still holds.

The corresponding match to relation (24) is

$$h_B \sum_{l\in\mathcal{L}} \frac{\eta_l}{2^l} \sum_{s=0}^{2^l-1} \sum_{t\in\mathcal{B}_{l,s}} b(t) = 2^{32}\Delta, \tag{46}$$

which leads, instead of (25), to

$$600s \overset{(3)}{=} \frac{2^{32}\Delta}{h_B} = \sum_{l\in\mathcal{L}} \frac{\eta_l}{2^l} \sum_{s=0}^{2^l-1} b(\mathcal{B}_{l,s}) \overset{(45)}{\approx} c_\eta \sum_{l\in\mathcal{L}} \sum_{s=0}^{2^l-1} \overline{\beta(t)}|_{\mathcal{B}_{l,s}} \cdot b(\mathcal{B}_{l,s}). \tag{47}$$

But how do we compute c_η, since information about $\overline{\beta(t)}|_{\mathcal{B}_{l,s}}$ and $b(\mathcal{B}_{l,s})$ for each admissible (l,s)-pair is not available to every miner? We have to work in a distributed fashion as it is already done in Algorithm 2. Assume, that we again want to compute c_η based on the last 2016 multi-blocks as it was done in Section 4.1.3. Then we have

$$c_\eta \approx \frac{600s}{\sum_{l\in\mathcal{L}} \sum_{s=0}^{2^l-1} \frac{1}{2016} \sum_{i=0}^{2015} \overline{\beta(t)}|_{\mathcal{B}_{l,s,i}} \cdot b(\mathcal{B}_{l,s,i})}, \tag{48}$$

where $\mathcal{B}_{l,s,i}, i \in 0,\ldots,2015$ are the last 2016 shards for any admissible pair (l,s). Each miner belonging to shard (l,s) should compute the average "output value" for his proposition block

$$v_{l,s} := \frac{1}{2016} \sum_{i=0}^{2015} \overline{\beta(t)}|_{\mathcal{B}_{l,s,i}} \cdot b(\mathcal{B}_{l,s,i}). \tag{49}$$

A miner one level above him will control the computation during validation of the children blocks and only then include the hash of this block into the header of its block. We suggest to use recurrent relation for average

$$v_{l,s,i} = \frac{i}{i+1} v_{l,s,i-1} + \frac{1}{i+1} \overline{\beta(t)}|_{\mathcal{B}_{l,s,i}} \cdot b(\mathcal{B}_{l,s,i}) \tag{50}$$

to obtain an on-line estimate $v_{l,s,i}$ of $v_{l,s}$ for each $i \in \{0,\dots,2015\}$ with any finite initial condition $v_{l,s,-1} < \infty$. Now we are ready to present the distributed Algorithm 3 to compute c_η, see Appendix B.

Since information is distributed among the nodes of tree HBS, not only c_η but any variable dependent on such information has to be computed in a distributed way as it is done in Algorithm 3. The same is true for time investments $\eta_l, l \in \mathcal{L}$. For those, instead of (29) we have

$$\eta_l = \frac{c_\eta}{2016 \cdot 2^l} \sum_{s=0}^{2^l-1} \sum_{i=0}^{2015} \overline{\beta(t)}|_{\mathcal{B}_{l,s,i}} \quad \forall l \in \mathcal{L}, \tag{51}$$

i.e. each η_l is dependent on information from the all shards $s \in \{0,\dots,2^l\}$. Those are connected only via the root node of the tree and thus η_l can only be computed by applying a reduce to the sum of $\overline{\beta(t)}|_{\mathcal{B}_{l,s,i}}$ through shards s using intermediate results saved in the higher levels $\{0,\dots,l-1\}$ of the tree HBS. The average through recent multi-blocks for $i \in \{0,\dots,2015\}$ can be again facilitated via an on-line update analogical to (50). Final $\eta_l, l \in \mathcal{L}$ will be saved in the root block $\mathcal{B}_{0,0}$ as a vector and updated at the same time as c_η is updated in Algorithm 3. Since root blocks are available to everybody, so are $\{\eta_l\}$. Details of the algorithm are left on reader as an exercise.

Let us now reassess c_η and η_l for our dataset of transactions. First, even if the calculation of c_η changes, the value remains the same, i.e. (30) still holds. By comparing (23) with (45) we see that the updated estimate $\hat{\eta}_{tree}$ of η reads

$$\hat{\eta}_{tree,l} = 2^l \hat{\eta}_l \quad \forall l \in \mathcal{L}, \tag{52}$$

where $\hat{\eta}$ is the estimate from (31).

Second, since the sub-blocks $\mathcal{B}_l, l \in \mathcal{L}$ from Section 4.1 are here divided to 2^l shards $\mathcal{B}_{l,s}$ for $s \in \{0,\dots,2^l-1\}$, those are thus on average 2^l-times smaller and (32) implies that 2^l from (52) cancels out and time to sign a shard $\mathcal{B}_{l,s}$ is still given by (33). In consequence, the hurdles to broadcast the shards are neither lower nor higher at the first sight then those with sub-blocks in Section 4.1. But since the shards are 2^l-times smaller, it takes in principle 2^l-times less time to broadcast them. While we ignored latency issues etc., the situation is clearly significantly better, namely exponentially so.

4.2.4 Reward and transaction fees

Again, as argued in Section 3.2, users should pay fees proportional to transaction difficulty $\delta(t)$. Per shard $\mathcal{B}_{l,s}, l \in \mathcal{L}, s \in \{0,\dots,2^l-1\}$, the available hashrate is given by (43). On the other hand we have derived that the time investment η under validity of Assumption 1 satisfies (45). Altogether, for any $l \in \mathcal{L}$, 2^l cancel out and the difficulty $\delta(t_1)$ for a transaction t_1 in $\mathcal{B}_{l,s}$ is identical to the difficulty $\delta(t_2)$ for a transaction t_2 in $\mathcal{B}_l$ from Section 4.1, i.e. fees should not change. Indeed, the fees per bit ϕ_l at level $l \in \mathcal{L}$ should be proportional to

$$\phi_l \left[\frac{\ddot{\mathrm{B}}}{b}\right] \propto \frac{h_B}{2^l}\eta_l \overset{(45)}{=} c_\eta \cdot \overline{\beta(t)}|_{\mathcal{T}_l}, \tag{53}$$

which is, under validity of Assumption 1, identical to what users should pay, according to (34), for space in sub-blocks $\mathcal{B}_l, l \in \mathcal{L}$.

For distribution of rewards r, we build on the approach presented in Section 4.1.5. If the whole network hashrate h_B is uniformly divided among 2^l-shards for each level $l \in \mathcal{L}$, we have that for each $s \in \{0, \dots, 2^l - 1\}$

$$r_{l,s} := \frac{\hat{t}_l}{2^l \sum_{l \in \mathcal{L}} \hat{t}_l} r, \tag{54}$$

where $\hat{t}_l$ are estimators of average time to mine sub-blocks $\mathcal{B}_{l,s}$ given by

$$\hat{t}_l \approx \hat{\eta}_l \cdot \overline{b(\mathcal{B}_{l,s})}|_{s \in \{0,\dots,2^l-1\}} \tag{55}$$

and computed in the same distributed way as time investment estimators $\hat{\eta}_l, l \in \mathcal{L}$ are computed starting from (51). Since $\hat{t}_l, l \in \mathcal{L}$ need to be available to everybody, they have to be saved in a root block $\mathcal{B}_{0,0}$. Because the estimates $\hat{\eta}_l$ of η_l should already be saved therein, it is equivalently possible to save estimates of $\overline{b(\mathcal{B}_{l,s})}|_{s \in \{0,\dots,2^l-1\}}$ for $l \in \mathcal{L}$ and $\hat{t}_l$ can be computed using (55). Analogically to (51), we can write

$$\overline{b(\mathcal{B}_{l,s})}|_{s \in \{0,\dots,2^l-1\}} \approx \frac{1}{2016 \cdot 2^l} \sum_{s=0}^{2^l-1} \sum_{i=0}^{2015} b(\mathcal{B}_{l,s,i}) \quad \forall l \in \mathcal{L}. \tag{56}$$

Again, (56) can be computed using on-line update analogical to (50).

4.2.5 Sharding validation and storage: minimal "full" node.

Without distribution of both transaction validation and storage, the recent developments of this section would be in essence meaningless. All aspects of the Bitcoin ecosystem have to be adapted to achieve scalability on-chain.

Our core distribution approach for validation and storage is very similar to that of mining. As $\beta(t)$ decreases as the level index $l \in \mathcal{L}$ increases, we propose that a decreasing fraction of "full" nodes validates and stores transactions, proportionally to hashrate h used to sign those transactions. Looking at Figure 4, it means, that all the nodes validate and store transaction for $l = 0$, a half of the nodes for $l = 1$ and in general

$$\frac{1}{2^l} \tag{57}$$

of all the nodes validate and store transaction for level $l \in \mathcal{L}$. This defines a notion of a *minimal full node* (MFN). But nodes may choose to support a bigger part of the tree HBS, even the whole one.

Of course, the fraction of network nodes (57) has to be large enough to ensure virtually 100% availability of any record. This gives us an upper bound on number of possible levels. Currently, there are more then 14000 reachable Bitcoin nodes, which means that level $l = 7$ is supported by more then 100 nodes.

Thus $L = 8$ seems certainly feasible. To achieve e.g. $L = 16$ with 100 nodes per a leaf shard $\mathcal{B}_{15,s}$ for any $s \in \{0, \ldots, 2^{15} - 1\}$ would require approximately $100 \cdot 2^{15} \approx 3.3$M modes. Currently, this number seems illusory, but we will show that running a MFN becomes easier as L grows, which together with the obvious increase of three HBS blockchain usability should promote widespread use of decentralized finance.

We propose assigning parts of the HBS tree to candidate MFNs again based on shard function (40), see also Algorithm 10. The MFNs should validate and store all sub-blocks mined within their branch. This ensures that probabilistically all parts of the tree HBS are uniformly covered with respect to both validation and storage.

When a MFN has to validate transactions included in a shard $\mathcal{B}_{l,s}, l \in \mathcal{L}, s \in \{0, \ldots, 2^l - 1\}$ from the branch assigned to him by (40), they refer to UTXOs saved in a certain previous shard of any previous tree HBS. Thus, it has to receive from the network other sub-blocks not belonging to its branch to be able to validate those transactions. What is the number of those sub-blocks? Let us assume for simplicity that one multi-block tree contains on average $N \in \mathbb{N}^+$ transactions and those are uniformly divided among the shards. Then one shard $\mathcal{B}_{l,s}$ for any admissible pair (l, s) contains simply

$$\frac{N}{2^L - 1} \tag{58}$$

transactions, where $2^L - 1$ is the number of $\mathcal{B}_{l,s}-$shards in the binary tree HBS with L levels. To validate any of those transactions, the MFN has to download one historical branch up to some level $k < L$ from the network, where k is the security level of this historical transaction. It means that it has to download, validate and store at least temporarily up to

$$\frac{Nk}{2^L - 1} < \frac{NL}{2^L - 1} \tag{59}$$

historical shards per one own shard $\mathcal{B}_{l,s}$. For all L-shards directly assigned to a MFN, it has to download, store and validate up to

$$\frac{NL^2}{2^L - 1} \tag{60}$$

extra shards. Altogether, the ratio of all the shards which any MFN has to control to the alternative of storing the whole tree HBS is bounded by

$$r(N, L) := \frac{NL^2}{(2^L - 1)^2} + \frac{L}{(2^L - 1)}. \tag{61}$$

The scalability of MFN sharding is depicted in Figure 5 generated by running Algorithm 11. The solid line represents the ratio (61) plotted as a function of L for $N = 4200$, which is approximately the number of 250B legacy transactions savable in a current 1MB Bitcoin block. For a still feasible $L = 10$ the above

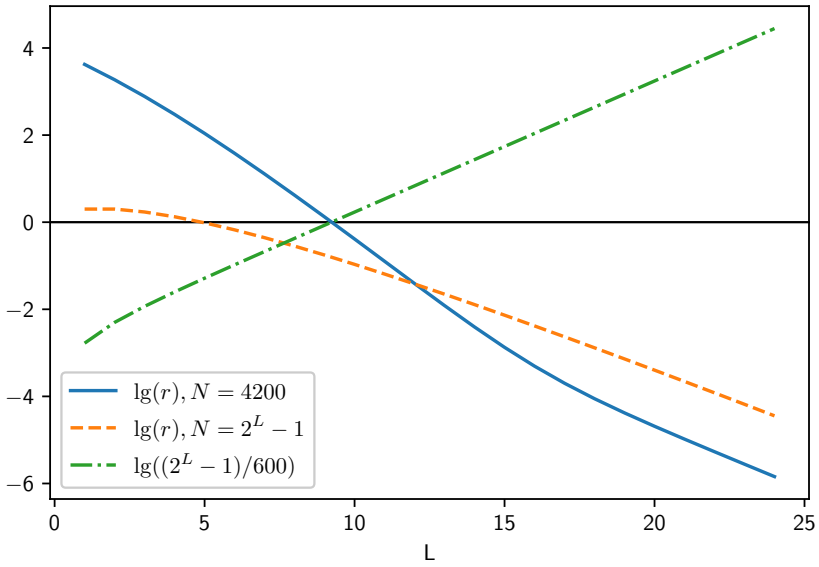

Figure 5: Sharding efficiency: result of running Algorithm 11

expression already evaluates to a number smaller then 1. This corresponds to approximately $4200/(2^{10}-1) \approx 4$ transactions per a shard $\mathcal{B}_{l,s}$ for some admissible pairs (l,s).

Of course, the goal is to scale the on-chain capabilities of Bitcoin network, i.e. to increase N. Expression (61) is minimized for $N = 2^L - 1$, which represents a hypothetical tree HBS where each node, i.e sub-block $\mathcal{B}_{l,s}, l \in \mathcal{L}$, $s \in \{0, \ldots, 2^l - 1\}$, contains on average just 1 transaction. For $N < 2^L - 1$ some sub-blocks $\mathcal{B}_{l,s}$ have to be empty. Even if this represents an intriguing possibility which should be further analyzed, we consider here only the case $N \geq 2^L - 1$. The dashed line plots the ratio (61) for $N = 2^L - 1$ as a function of L. We see that e.g. for already $L = 15$ we achieve $r \approx 0.007$, which means that a MFN has to download less than 1% of total multi-block capacity. The dash-dotted line in Figure 5 depicts the corresponding theoretical capacity of the network in number of transaction per second, under assumption of average multi-block time of $600s$. E.g. for $L = 20$ we would achieve approximately 1750 transactions per second, which is close to the average number of VISA network transactions per second.

4.2.6 Information routing on tree HBS

The last aspect we shortly comment on is the question of information routing within the network. It is a rather complex problem whose full treatment lies outside of the scope of this conceptual publication. Nevertheless, the commu-

nication has to be scaled as well, otherwise it would be a bottleneck. It seems wise to employ the natural topology of tree HBS.

Let us image e.g. that a node wants to broadcast a transaction that should be signed at a certain level $l, 0 < l \leq L-1$. By computing (42) for its transaction the broadcasting node identifies a branch $(s_0, s_1, \ldots, s_l)$ of the tree HBS to which this transaction belongs. From its routing table, it can identify a safe number of nodes which belongs to the corresponding shard s_1 based on (40) at level $l = 1$. Those nodes should have a sufficient knowledge about their part of the tree HBS, saved in their routing table. They will then broadcast the transaction to the right nodes from shard s_2 at level $l = 2$. Those can broadcast it among them and route to those from s_3 at $l = 3$ etc. It is, the communication happens along the corresponding branch of tree HBS. The broadcasting node should receive a confirmation that at least one mining node or a desirable number of them working at level l is aware of the corresponding transaction.

The routing table may or may not be distributed. E.g. we estimated in Section 4.2.5 that we need 3.3M modes for $L = 16$ at a minimum of 100 nodes per a leaf shard. A routing table would then have a size proportional to approximately 13MB for 4B per an IP address, which is arguably not problematic. However, with e.g. billions of nodes in mind, it is still sensible to consider a non-distributed routing table? If a node knows n random peers, probability that all those do not belong to a certain shard at level l is given by

$$p(n) = \left(1 - \left(\frac{1}{2}\right)^l\right)^n, \tag{62}$$

which yields that it has to know at least

$$n = \frac{\lg(1-q)}{\lg\left(1 - \left(\frac{1}{2}\right)^l\right)} \tag{63}$$

nodes to be sure that with probability $q = 1 - p$ at least one of n nodes belongs to the considered shard. Then the broadcasting node could communicate with this node directly with a high probability, without using the briefly proposed schema above. E.g. for $p = 1 \times 10^{-10}$, $L = 16$ and the lowest level $l = L - 1$ one gets "only" $n \approx 754500$. However for $p = 1 \times 10^{-10}$, $L = 24$ and the lowest level $l = L - 1$ one gets more than 193M nodes. And worse, for unsatisfactory $p = 0.1$, $L = 24$ and the lowest level $l = L - 1$ one gets still more than 19M nodes. To conclude, a distributed routing table is a must for a high number of levels L. But we think that all aspects of any next evolutions of Bitcoin network should be implemented as efficiently as possible from day one. Of course, taking into account all criteria, such as security and simplicity etc. A communication scheme which exploits the tree HBS topology fully is desirable.

4.3 Concurrent levels

In Section 4.2 we proposed a tree hierarchical block structure (HBS), depicted in Figure 4, which could theoretically allow us to scale Bitcoin blockchain to

thousands of on chain-transactions per second, while building on and respecting previous developments in Section 4.1, where, recalling again, we successfully align three things: transaction security, fees and electric energy consumption. Moreover all processes including mining, validation and storing of blocked transactions are distributed and nodes has to handle only a small fraction of the full state s.

However, at least one significant issue remains unresolved. Among any too sets of confirmed transactions recorded in sub-blocks $\mathcal{B}_{l,s}, s \in \{0, \ldots, 2^l - 1\}$ of two successive "multi-blocks" at any level $l \in \mathcal{L}$, there lie all sub-blocks of the current multi-block for higher levels $\{l-1, l-2, \ldots, 0\}$ and the next multi-block for lower levels $\{L-1, L-2, \ldots, l+1\}$. Confirmation times of those sets of transactions differ on average by multi-block creation time T, which we assumed to be equal to the current block time of $600s$. Consequently, a user has to wait at most 10 minutes for its transaction to be included in a sub-block $\mathcal{B}_{l,s}$ even if the tree HBS from Section 4.2 has enough capacity to store all the waiting transactions. In short, transactions are not timely.

There exist few ways how to mitigate this problem. Obviously, transactions which need to be really quick can employ some off-chain mechanism such as Lightning Network [2]. While we fully recognize the importance of "perpendicular" approaches, we would still prefer an on-chain solution, if one can be found.

The obvious direct approach is to decrease T to few seconds, e.g. $T = 5s$. Based on behavioral research [11], tolerable waiting time for information retrieval is $2s$ and $10s$ is about the limit for keeping user's attention focused. These "multi-block" times are very short and push the tree HBS from Section 4.2 to uncharted waters. Surely, if the number of transactions per second is kept constant, size reduction of sub-blocks $\mathcal{B}_{l,s}, l \in \mathcal{L}$, $s \in \{0, \ldots, 2^l - 1\}$, would be proportional to reduction in T. However, it is an open question how efficiently can all the aspects of state transition (1) be rescaled, especially with regards to information synchronization within the network. It is entirely possible that this approach can lead to an acceptable solution. A fine treatment of this possibility is however outside of scope of this publication. In the forthcoming text we propose a more systematic solution with has a far higher potential of success.

Let us recall that Bitcoin transactions t are in essence signed by expenditure of computation resources, expressed by hash rate $h(t)$, for a certain time, expressed by time investment $\eta(t)$ defined in (4). Transaction security can only be controlled by controlling those two degrees of freedom (DOFs), see Section 3.3, particularly Equation (16). The tree HBS from Section 4.2 depicted in Figure 4 has been the first HBS we proposed in this paper which is designed to simultaneously employ those two DOFs. While it certainly does not represent a unique possibility with respect to this, it is a natural extension of the multiblock idea from Section 4.1. So how to achieve timeliness of transactions while keeping this tree HBS or altering it only minimally and thus preserve everything we have achieved so far?

The answer is concurrency. One has to relax the sequential dependence in Figure 4.2. A transaction t at a level $l \in \mathcal{L}$ should not wait for all sub-blocks of

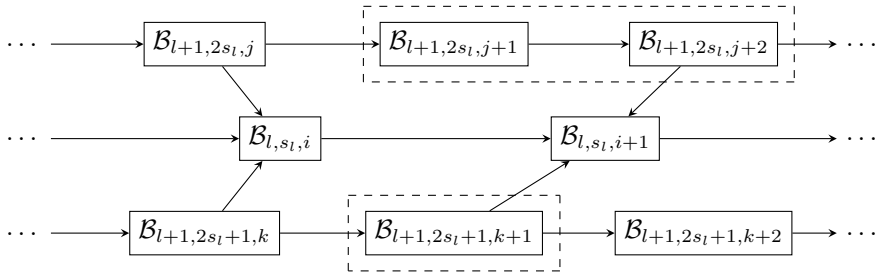

Figure 6: Concurrent chains at levels $l, l+1 \in \mathcal{L}$ and some branch $s_l \in \{0, \ldots, 2^l - 1\}$.

the current multi-block for higher levels $\{l-1, l-2, \ldots, 0\}$ and of the next multi-block for lower levels $\{L-1, L-2, \ldots, l+1\}$ from its branch $\{s_0, s_1, \ldots, s_{L-1}\}$ to be mined. It should be included in a block $\mathcal{B}_{l,s_l}$ as quickly as possible.

The idea is presented visually in Figure 6. It depicts in the middle one blockchain containing $l-$level blocks $\mathcal{B}_{l,s_l,i}$ for a level $l \in \{0, \ldots, L-2\}$ and $i = 0, 1, \ldots$ surrounded by its two corresponding $l+1$ blockchains, one containing blocks $\mathcal{B}_{l+1,2s_l,j}$ for $j = 0, 1, \ldots$ and the other blocks $\mathcal{B}_{l+1,2s_l+1,k}$ for $k = 0, 1, \ldots$. Each of those blockchains corresponds to one node of the tree HBS depicted for $L = 3$ in Figure 4, namely the nodes determined by coordinate pairs (l, s_l), $(l+1, 2s_l)$ and $(l+1, 2s_l+1)$. The tree HBS with L levels has together at most $2^L + 1$ nodes, i.e. we obtain the same amount of concurrent blockchains. As displayed in Figure 6, those blockchains still to a great extend respect the hierarchical topology of the underlying tree HBS from Section 4.2 and they are certainly not independent. For example, hashes of lower level blocks are included into higher level proposition blocks after performing of all necessary controls to avoid double spending etc.

What changes? We allow each blockchain $(l, s), l \in \mathcal{L}, s \in \{0, \ldots, 2^l - 1\}$ to grow without its blocks immediately being checked and signed by higher levels $l-1, \ldots, 0$. In essence it means, that validation by mining higher level sub-blocks/shards is postponed in time. And we also allow multiple blocks in a chain (l, s) to be created without all their hashes being included in a higher level block. This does not substantially alter the tree HBS topology. A sequence of such blocks can be merely seen as a single larger block, see dashed blocks in Figure 6. But it allows us to increase timeliness because very small blocks can be added to a chain (l, s) if necessary.

Practically all results from Section 4.2 apply to this concurrent setup without a change or only with slights modifications. We lose the strict order of shards $\mathcal{B}_{l,s}$ within tree HBS from Figure 4 but concurrent blockchains are one-to-one mapped to individual nodes (l, s) of this tree HBS, while its hierarchy is still enforced even if more loosely. Thus this loss of strict sub-block order has only temporal implications for the finality of state s when the situation is compared with the setup of Section 4.2.

In the end, any finality of Bitcoin transactions is only probabilistic and thus

prone to subjective perception, see Section 2. Nowadays, users often wait for multiple confirmations till they consider a transaction settled and this partially irrespective of the wealth being transacted. In doing so they follow classical Byzantine fault tolerance model [16]. It seems that they have no other choice. At least at first glance, see A.6, this is the model governing the security of current Bitcoin blocks. All transaction within those share the same risks and guaranties. If one wants to double spend even a single transaction, he has to implicitly attack all transactions in the corresponding block. This is completely different from situation in HBSs starting already from Section 4.1, supported by our simple incentive based security model from Section 3.1. Users have a choice to pick a level of security that they find appropriate based mainly on wealth being transacted. Naturally, the notion of finality is fluid as well.

If one pays for an ice-cream or receives the payment for icecream, he is probably satisfied with the transaction being included in a low lever block for a certain low enough $l \leq l_{\max}$, where $l_{\max}$ is the minimal security level both transacting parties can initially accept. They would not wait till this transaction is confirmed by creation of a highest level block $\mathcal{B}_{0,0}$. E.g. the seller would accept the very small risk that finally a longer chain may be found, not referring to their transaction. As he currently accepts e.g. the risk that a very small percentage of coins are counterfeits.

On the other hand, if one receives a payment e.g. in a property transaction, he would require a much lower $l_{\max}$, such that at least a $2^{-l_{\max}}-$ fraction of nodes validate and store the data and he would consider the transaction settled only when for the corresponding block $\mathcal{B}_{l,s}$ there exists a hashpath up to the root level $l = 0$ within the graph obtained by repeating Figure 6 for all pairs $(l, s), l \in \mathcal{L}, s \in \{0, \ldots, 2^l - 1\}$. And even wait till a higher number of root level blocks $\mathcal{B}_{0,0}$ is added to the blockchain $(0, 0)$, similarly to the current practice.

4.3.1 Running a MFN node

While for an L level tree HBS, there are at most $2^L + 1$ concurrent blockchains, each minimal full node (MFN), defined in Section 4.2.5, directly handles, i.e. validates and stores, only L of them, those contained in its branch of the tree HBS $\{s_0, s_1, \ldots, s_{L-1}\}$, determined by (40). For a MFN the problem is linear in L. The ratio of transaction information necessary to fulfill its duties with respect to all information contained in all $2^L + 1$ concurrent blockchains is estimated in (61).

Today CPUs have mostly at least 4 cores, many supporting hyper-threading, meaning that up to $L = 8$ each blockchain can be theoretically easily mapped onto a single hardware supported thread. In practice, modern CPU can easily handle 3 or 4 IO intensive threads per core, which leads to $L = 12$ if one core is reserved for operating system only.

How many transactions does a MFN process and then has to store? Assume that n transactions per second are uniformly divided among the $2^L - 1$

concurrent blockchains. Then a MFN has to validate and store

$$s_{MFN} = \frac{nL}{2^L - 1} \tag{64}$$

transactions per second and needs to download for each of them at most one chain composed of maximally L sub-blocks for validation purposes, see Section 4.2.5. A minimal amount of communication is achieved if each of those sub-blocks contains just one transaction. For this optimal case we obtain that the number of transactions per second one MFN has to download is bounded by

$$d_{MFN} = \frac{nL^2}{2^L - 1} + L. \tag{65}$$

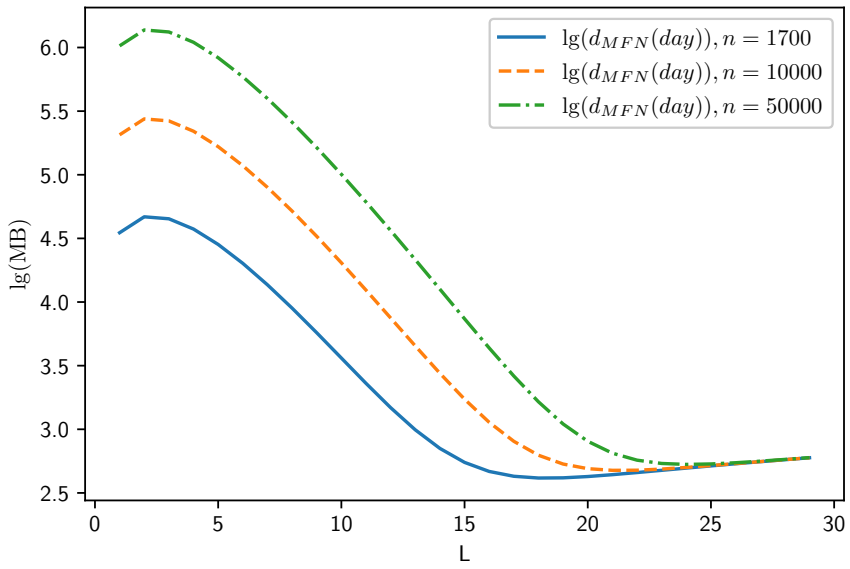

Figure 7: An upper bound of minimal download per day in MB of a single MFN: result of running Algorithm 12

Clearly, the requirement for storage (64) are monotonically decreasing function of $L \geq 1$. In Figure 7, there is estimated daily download in megabytes for different values of n, starting from 1700t/s which is close to the average number of VISA network transactions per second. This figure is generated by running Algorithm 12 which of course employs relation (65). We assume an average transaction size of 250B. We see that to minimize communication, a certain minimal L has to be reached, such that the linear term in (65) starts to dominate.

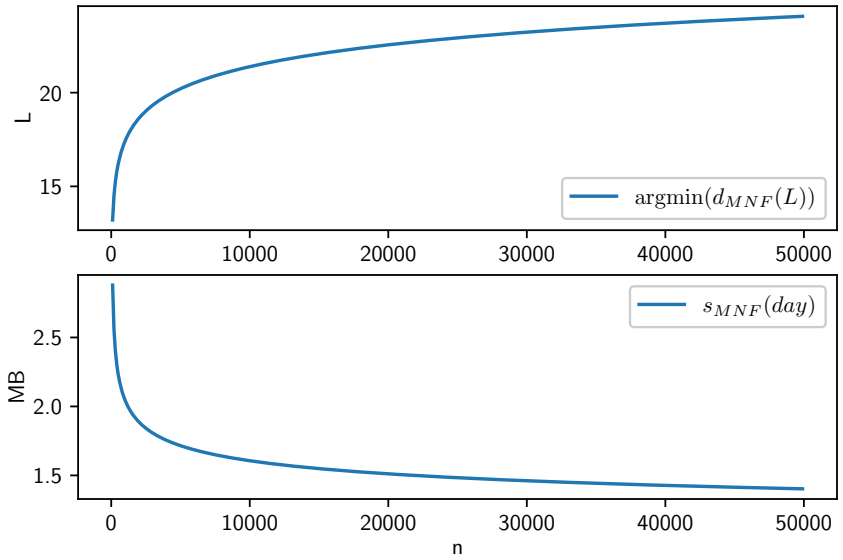

Figure 8: $L = \text{argmin}(d_{MFN}(L, n))$ for different transaction throughputs per second n and corresponding storage per one MFN per day: result of running Algorithm 13

Moreover, for each $n \in \mathbb{N}^+$ there exist one minimum of (65). By running Algorithm 13 we compute those optimal L for different transaction throughputs n. And we compute the corresponding storage per day for single MFN using (64). The result is presented in Figure 8. By employing those L, we obtain per day downloads in range from approximately 300MB for $L = 13$ to approximately 530MB for $L = 24$. It means that theoretically a 52kbits/s connection is sufficient (neglecting upload). Moreover, the per day MFN storage is monotonically decreasing function of n for L values minimizing the connection throughput, dropping from merely ~ 2.9MB for $L = 13$ to ~ 1.4MB for $L = 24$.

The scaling potential should be now obvious. It could be possible to run a MFN node for $L = 13$ which needs a little bit more then 1GB of transaction storage per year and can run on a 2G wireless connection. By scaling further, the storage actually drops to approximately 500MB per year for $L = 24$ and the 2G connection is still sufficient, neglecting latency issues.

5 Conclusion

In this paper, we have argued that a solution to the problem of the Bitcoin on-chain scaling lies outside of the comfort zone of the current concept of its

blockchain, see e.g. Appendix A.5. Nowadays, transactions requiring completely different security guaranties, considering the monetary values they move, are included into the same block. They are signed by the aggregate hashrate of all the participating miners and validated and stored by all the full nodes. Within these constraints, a scaled up Bitcoin blockchain simply cannot be found.

We have proposed a multi-scale approach, where extended Bitcoin transactions t having two tunable parameters are uniquely assigned to a certain shard $\mathcal{B}_{l,s}, s \in \{0, \dots, 2^l - 1\}$ of the network at a certain level $l \in \mathcal{L}$. The default level l assigned to a transaction is controlled by its output value per bit β, but l can be freely chosen by the user. The lower l, the higher the number of hashes used to secure t and more expensive it becomes to double spend. On the contrary, both electric energy consumption and fees decrease as l grows.

The sub-blocks $\mathcal{B}_{l,s}$ are organized in a (binary) tree topology with coordinates (l, s). The transactions t are uniquely assigned to individual nodes/branches of this tree by a shard function s, see (42). This function is an essential concept, which facilitates probabilistic assignment of not only transactions t to nodes (l, s) of the tree, but also miners and other participants to those nodes, see (40) and Section 4.2.5. Metaphorically, transactions meet their miners and minimal full nodes (MFNs) at coordinates (l, s) with uniform probabilities in s. A MFN is an updated version of full node. It validates and stores only transactions along its branch $(s_0, s_1, \dots, s_{L-1})$ assigned to it by (40). Consequently, the state s is distributed among all MFNs.

We have suggested that an efficient information exchange scheme among the participants should fully utilize the (binary) tree topology. In Section 4.2 this topology is strictly enforced, see Figure 4, but in Section 4.3 only loosely, see Figure 6. In the latter, we essentially have $2^L + 1$ interlocked concurrent blockchains. This constitutes our final scaling proposal which achieves all objectives. It offers thousands of timely transactions per second with a variable level of trust together with aligned network fees and energy usage. And the state s is distributed in a way which allows practically anybody to participate and run a MFN.

Many results in this paper are based on a simple security model (6) together with Assumption 1. While 6 is certainly only a rough approximation of the reality and the expectations of some or even all the users can differ from those in Assumption 1, it is of no concern. The solution found in Section 4.3 is sufficiently flexible and self correcting. First, the users are free to choose an appropriate security level $l \in \mathcal{L}$. Second, miners may or may not include those transactions into their proposition blocks at the fees offered by the users. A true equilibrium between costs and guaranties will be found by market forces.

The Bitcoin elites, at first sight, have no economic incentive to support the changes proposed in this publication that include aiming for fees proportional to transaction value $v(t)$. Some of them are certainly satisfied with the current role of BTC as a store of value, volatile but a store of value nevertheless. But other crypto plutocrats may recognize that without resolving inconsistency between the costs and fees as block rewards decrease, not only will ₿ never become a universal mean of payment, but it will become progressively insecure and

eventually at risk of becoming extinct, see [13]. The implications for the store of value proposition are obvious.

Actually, mathematically speaking, there exists at least one very early adopter that could find this contribution significant. The one who wrote:
"Craig Steven Wright is a liar and a fraud. He doesn't have the keys used to sign this message.

The Lightning Network is a significant achievement. However, we need to continue work on improving on-chain capacity.

Unfortunately, the solution is not to just change a constant in the code or to allow powerful participants to force out others.

We are all Satoshi"

We are highly optimistic about the Bitcoin future. The community will certainly recognize the economic discrepancies described in this paper and build upon the solutions proposed. In the end, Bitcoin ecosystem is as democratic as one can hope it to be. If developers suggest solutions and users choose to support them, the elites have to bend the knee or risk their fortune to evaporate, see Section 2.4 of [13]. There is more then enough time for discussions, solutions and implementations of them. But complacency and postponement are never good tendencies. It is better to start right now or others will take the lead.

6 Acknowledgement

I have never participated in Bitcoin's development or in any other crypto based project. This paper is written in my limited spare time and for prospective public good. If you liked what you read, you may donate and support my financial freedom to think and create. Please, do not be ashamed to donate any small contribution. In the end, $10^6 \cdot 100sats = 1฿$. Thank you.

Before sending any coins, please double check that the addresses here are the same as the ones published on my github account [19] or check the sha256sum of this paper there. To communicate with a limited trust, please use my email[5].

SegWit: bc1qrfunay7uekg5aj578suh6c6n8d9lqhwwf5h3ps
Legacy: 1EZRjWCagpegLcydVPSB4zFPgEdZ4wYuKu

References

[1] Adam Back. Hashcash - A Denial of Service Counter-Measure. http://www.hashcash.org/papers/hascash.pdf, 2002.

[5] "Total paranoia is just total awareness." – Charles Manson

[2] Ferenc Béres, István A. Seres, and András A. Benczúr. A Cryptoeconomic Traffic Analysis of Bitcoin's Lightning Network. *Cryptoeconomic Systems*, 2020.

[3] Vitalik Buterin. A Proof of Stake Design Philosophy. https://medium.com/@VitalikButerin/a-proof-of-stake-design-philosophy-506585978d51.

[4] Vitalik Buterin. Ethereum Whitepaper. https://ethereum.org/en/whitepaper/.

[5] Agustín Carstens, Jerome H. Powell, Gillian Tett, and Jens Weidmann. How can central banks innovate in the digital age? https://www.bis.org/events/bis_innovation_summit_2021/agenda.html, 2021.

[6] Aaron Clauson, Andrew Chow, Anthony Towns, Bruno Garcia, Fabian Jahr, fanquake, Hennadii Stephanov, Jon Atack, Luke Dashjr, MarcoFalke, Pieter Wuille, practicalswift, radymmcmillan, Sjors Provoost, Vasil Dimov, and W.J. van der Laan. 0.21.1 Release Notes. https://bitcoin.org/en/releases/0.21.1.

[7] Bitcoin community. Difficulty. https://en.bitcoin.it/wiki/Difficulty.

[8] Wikipedia contributors. Bitcoin Cash — Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/wiki/Bitcoin_Cash, 2021.

[9] Brian Fabian Crain and Sébastien Couture. EB65 – Adam Back & Greg Maxwell: Sidechains Unchained. https://www.youtube.com/watch?v=jE_telgnlw3M&t=4775s.

[10] Christian Decker and Roger Wattenhofer. Information Propagation in the Bitcoin Network. In *13-th IEEE International Conference on Peer-to-Peer Computing*, 2013.

[11] Fiona Fui-Hoom Nah. A study on tolerable waiting time: how long are Web users willing to wait? *Behaviour & Information Technology*, 23(3):153–163, 2004.

[12] Cyril Grunspan and Ricardo Pérez-Marco. Double Spend Races. *International Journal of Theoretical and Applied Finance*, 21(08):1850053, 2018.

[13] Hasu, James Prestwich, and Brandon Curtis. A model for bitcoin's security and the declining block subsidy. https://uncommoncore.co/wp-content/uploads/2019/10/A-model-for-Bitcoins-security-and-the-declining-block-subsidy-v1.02.pdf, 2019.

[14] Thomas Kerber, Aggelos Kiayias, Markulf Kohlweiss, and Vassilis Zikas. Ouroboros Crypsinous: Privacy-Preserving Proof-of-Stake. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 157–174, 2019.

[15] Eric Lombrozo, Johnson Lau, and Peter Wuille. Segregated Witness (Consensus layer). https://github.com/bitcoin/bips/blob/master/bip-0141.mediawiki.

[16] Satoshi Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System. https://bitcoin.org/bitcoin.pdf, 2008.

[17] nullc. https://old.reddit.com/r/Bitcoin/comments/mtugta/mentor_monday_april_19_2021_ask_all_your_bitcoin/gv86j6b/.

[18] Meni Rosenfeld. Analysis of Hashrate-Based Double Spending. https://arxiv.org/abs/1402.2009, 2014.

[19] Qui Somnium. Hbs. https://github.com/quisom/hbs, 2021.

A On Bitcoin, its economics, energy consumption and scaling

A.1 In short, how Bitcoin works?

Bitcoin *transactions* are stored in *blocks* of a predefined maximal size. Each subsequent block contains a hash of a previous block, thus proving that its creator accepts all the transactions recorded in this and all the earlier blocks. Consequently, the blocks form a *blockchain*. *Mining nodes* in the network compete to create a next valid block of transactions by finding in parallel and with a minimal amount of coordination a nonce for their proposition block such that its hash is smaller than a predefined number which determines the difficulty of finding such a hash. This difficulty is regularly adjusted, reflecting any change of the compound computational power of all the nodes, such that the block creation has a specified expected speed (e.g. one 1 MB block per 10 minutes). When a hash for a certain block is found that fulfills the difficulty criterion and thus proving that the set amount of computational time has been spent to secure the transactions contained in the block, this block is added to the blockchain. The creator of the block then obtains a certain amount of newly minted bitcoin called *block reward*, this being the main incentive to take part in signing of the transactions.

A.2 Deflationary nature

Some argue that Bitcoin's technological constraints such as its scalability issues are a secondary problem to its ultimately deflationary nature. They claim that bitcoin's appreciation in market value discourages people from using it for payments. It is a reasonable hypothesis, since, its finite supply is caped at 21M and ultimately it should be a deflationary coin. Its inflation rate is governed by block reward which halves every 210000 mined blocks, i.e. about every 4 years. It was initially 50 bitcoins (BTC), currently being 6.25 BTC. Bitcoin was programmed to be better in scarcity than gold. And there are other aspects that

make it superior to gold, e.g. its divisibility or its zero mass which implies practically zero costs holding the asset. Even the most influential people in finance now recognize or at least admit that BTC shows some gold-like properties and behavior. Let us quote Federal Reserve Chairman Jerome Powell talking about cryptoassets and bitcoin in particular: *"... There are also not particularly in use as means of payment. It's more a speculative asset that's essentially a substitute for gold, rather than for the dollar...."* [5].

Inflationary fiat/crypto currencies are definitely easier to be disposed of when paying for goods or services. If BTC's value rises when it is still inflationary due to the ongoing minting process, what will happen when this stops?

But rather than focusing on the question whether BTC is the new gold, we focus in this paper on what Bitcoin was meant to be and it clearly has not yet become: a successful general payment system. Without removing the existing technological hurdles, we will never really know if people are or will ever be prepared to use BTC for daily payments and in this way to potentially spread real wealth to others, maybe being even "slightly" altruistic. In the end, it is difficult to assess whether a deflationary monetary system could work, especially because societies are used to practically constant presence of inflation. Philosophically speaking, human life time is similarly deflationary. Each day we have one day less of it.

A.3 Current state of blockchain affairs

There exist more than 5000 blockchain based cryptoassets at the time of writing this article, precise number not being important. Most of them are merely *tokens* utilizing one of a handful of public blockchain implementations which support smart contracts. The pioneering one is Ethereum [4], the number two player with still less than a half of Bitcoin's market cap. They together represent more than 60% of more than 2 trillion USD invested in public blockchains.

The Ethereum project extends in a very meaningful way Bitcoin's codebase and builds a truly programmable blockchain. Bitcoin itself is a virtual machine evaluating small programs but limited by design. Arguably, its Taproot upgrade from November 2021 expands substantially its smart contract abilities, potentially reducing Ethereum's significant usefulness advantage in this domain [6]. Both Bitcoin and Ethereum use PoW based consensus algorithms but Ethereum ecosystem is long planned to switch to *Proof of Stake* (PoS) based one [3].

Proof of Stake based consensus algorithms, among others, are proposed as alternatives to PoW to mitigate some of its shortcomings, mainly its negative ecological impact. In short, the blocks are created by validators that are chosen randomly with probability proportional to their staked holdings of the underlining coin. They are rewarded for creating a valid block and/or punished for misbehaving by loosing a part of or the whole stake. The idea is simple and intriguing at first but a complex one to apply at a closer look: see e.g. [9] for critical counterarguments. It suffices to note that the Ethereum network is still running on PoW even after more than 5 years of PoS development.

Among PoS based blockchain platforms, Cardano clearly stands out. Its

market capitalization rose from roughly 1 billion USD in 2020 to approximately 100 billion USD in 2021, becoming the number tree player in the public blockchain market. This may be to a great extent a speculation move into a limited smart contract enabled PoS blockchain space, driven by the rising need for an environmentally friendly blockchain technology. Nevertheless, Cardano has in our view a unique selling proposition since its development is guided by evidence-based methods and its consensus algorithm Ouroboros is formally verified under different models in a sequence of peer-reviewed scientific papers, see [14] and the references wherein.[6]

The practical evidence that PoS can be implemented robustly enough to secure values is indeed comparatively limited. PoW is much more battle tested and clearly wins with respect to its simplicity which helps to build trust. Moreover, Bitcoin and Ethereum are much more then just their consensus algorithms, supporting in return the PoW momentum. Only time will tell whether PoS wins similar levels of market approval, this almost certainly being dependent on the success of Ethereum's transition to PoS.

A.4 Bitcoin energy consumption

The power efficiency e of modern commercially available ASIC miners is approximately 30 J/TH. Given the current approximate total hashrate h_B of $1.20e8$ TH/s, corresponding to 1.2M of modern Antminers S19j Pro, we arrive at the power of 3.6 GW, equivalent to $3-4$ conventional nuclear rectors.

It is equivalent to $3.6e6$ kWh. At electricity price p e.g. 0.1$ per kWh, we arrive at one hour energy costs of 360000$. Since one block is mined on average every 10 minutes, energy costs per block are cca 60000$. For a legacy transaction which size $b(t)$ equals to 250B, costs are

$$\approx \frac{250\text{B}}{1024^2\text{B}} \cdot 60000\$ = 14.3\$$$

if blocks are assumed to be fully filled. Altogether, a general formula for transaction electric energy consumption tec is

$$tec[\text{kWh}] := e[\text{J/TH}] \cdot h_B[\text{TH/s}] \cdot \frac{600\text{s}}{3.6e6} \cdot \frac{b(t)[\text{B}]}{1024^2\text{B}} \tag{66}$$

and the cost per transaction cpt is then simply

$$cpt[\$] := tec[\text{kWh}] \cdot p[\$/\text{kWh}].$$

Certainly, these are lower estimates, since not all the hardware used to mine bitcoin is recent and it does not include all the indirect energy use. Nevertheless, this simple estimate yields that Bitcoin network uses approximately 32TWh of electricity per year - far less than normally estimated.

[6] It is up to a competent reader to independently evaluate whether the models and their assumptions are realistic.

But indeed, because of the block reward, even mining on older hardware is profitable. We may obtain an upper estimate of Bitcoin energy use, if we know the average price of electric energy used for mining and assume that miners are rational and stop mining when it starts to be unprofitable.

Let BTCUSD be the bitcoin exchange rate with USD, ϕ the transaction fee per bit and r the block reward. The total block reward tr is

$$tr[\$] = r[\text{฿}] \cdot \text{BTCUSD} + 1024^2\text{B} \cdot 8\phi[\$/\text{b}].$$

For a rational miner

$$tr[\$] > p[\$/\text{kWh}] \cdot E[\text{kWh/block}], \tag{67}$$

where E is the total electrical energy consumption of the Bitcoin network to mine a block which we estimated above as

$$E \approx e[\text{J/TH}]h_B[\text{TH/s}]\frac{600\text{s}}{3.6e6}.$$

From inequality (67) we get an upper bound on electric energy consumption based on the market prices. Currently the ϕ is stabilized under 0.001875 \$/b and BTCUSD is above 40k. Consequently, the network total consumption should not be higher then

$$E[\text{kWh/block}] < \frac{6.25 \cdot 40000 + 8 \cdot 0.001875 \cdot 1024^2}{0.1} \approx 2.66\text{GWh}$$

for the electricity price of 0.1 \$/kWh. This leads to the upper bound of approximately 140 TWh per year. This is much closer to the other estimates. Nevertheless, we think that the real consumption is much lower, since electric energy is the main cost of mining and professional miners have all incentives to use as efficient hardware as possible.

It is important to note that transaction costs are normally relatively small with respect to the block reward. Even during its historical maximum of $\phi \approx 0.03$ \$/b, the block reward was a bigger chunk of total reward, even if comparable.

As the block reward (i.e. the inflation rate) of bitcoin will decrease, the electricity consumption should be governed more end more by transaction fees only and per transaction it should be at least a few magnitudes lower, if we want the Bitcoin network to succeed.

A.5 Mining economics and scaling

Mining is a very competitive business and we assume that the majority of miners does not behave altruistically but they rationally maximize their profits. Their financial incentive to sign transaction is composed of two parts: block rewards and transaction fees.

The cost structure of bitcoin mining is complex. Nevertheless, except all the initial investments to start a mining business, including the equipment acquisition costs, the electric energy consumption is the main production cost, e.g. denominated in USD.

To enlighten the peculiar nature of Bitcoins minting scheme we can employ a simple mind experiment: Let us imagine for a moment we fix the whole Bitcoin ecosystem including the energy consumption and its USD denominated price p[\$/kWh], only halving the block reward in BTC. For miners to recuperate the block reward loss, the transaction fees and/or the market price of BTC have to increase.

Let us now further assume that the BTC market price will stagnate at preceding levels after such a hypothetical halving. Then there exists an immediate pressure to increase the transaction fee per byte to recuperate the USD denominated loss.

Conversely, let us now assume that the market price of BTC rises proportionally after this hypothetical halving, i.e. it at least doubles since the inflation rate is halved. Then there is no increased pressure to hike the transaction fees, since the USD dominated reward of miners is at least preserved. Nevertheless, the transaction fee at least doubles in USD denominated terms.

In both cases, the halving is making bitcoin payment system more expensive and thus less competitive. In reality, a mix of the above two extremal possibilities happens. Both the BTC market price rises, even if very non-linearly, and the transaction fees rise as well. Moreover, eventual more than doubling of BTC market price incentives old miners to increase capacity and new miners to join, increasing the overall energy consumption of the Bitcoin network and rising the environmental concerns.

If the transaction throughput rose at least proportionally, i.e. if it at least doubled after each halving, it could be argued that real costs per transaction are preserved. But it would be desirable that utilization increases quicker or even dramatically, such that the energy usage per transaction tec, estimated in Equation (66), could drop significantly as well.

According to Equation (66), tec can be reduced by reducing the total hash rate h_B, by increasing the power efficiency e of mining equipment, by increasing the block generation speed, by decreasing the transaction size or finally by increasing the block size. Let us shortly analyze the individual possibilities.

First, any uneven, substantial reduction of h_B theoretically negatively impacts Bitcoin's network security, since potential attack abilities of some miners increase. This is certainly not desirable, so if at all possible, only a smooth, even and slow h_B decrease could be promoted.

On the other hand, incentives to increase the power efficiency e of mining hardware are innate and e also rises at a steady pace.

One successfully implemented *Bitcoin Improvement Proposal* (BIP) which inter alia also decreased transaction size is Segregated Witness [15].

Further, the most direct way how to decrease tec and at the same time to allow for quick/instant payments is to generate new blocks more frequently. Moreover, it is proven practically possible by Ethereum's implementation of

PoW, where new blocks are generated approximately once every 15 seconds [7] This would only require to distribute the reward into multiple block inside the current 10 minute interval, so that the Bitcoin's inflation rate is preserved.

The last obvious possibility is to increase maximum size of blocks. It is a very sticky issue on which some very intelligent people disagree [8]. In short, the supporters of a block size increase emphasize an on-chain medium of exchange function. The main argument against is that larger blocks would make full nodes more expensive to operate and this would lead to further centralization which weakens Bitcoin's values proposition. Actually, the same counterargument can be employed against the above mentioned increase in block generation frequency.

We agree that the decentralized nature of Bitcoin is the most sacred property of Satoshi Nakamoto's invention and should be preserved at all costs. We explain in the next Section that even if centralization of mining is partially inevitable, it is indeed crucial to promote running full Bitcoin nodes and thus any substantial increase in block size or generation speed is problematic.

Inevitably, any viable scaling idea has to represent a substantial evolution of blockchain technology as we know it. One such an off-chain approach is Lighting Network [2]. In this paper we propose an on-chain solution.

A.6 On centralization of mining

First, we postulate that centralization is practically inevitable in any system where an increase in efficiency leads to a corresponding increase in profits. This is certainly true for bitcoin mining. The actors who are first able to exploit the inefficiencies of such a system, acquire more wealth which in turn allows then to extract more of the inefficiency. As inefficiency potential is diminishing in time, the first players have an enormous advantage.

In natural real world systems, the external conditions often change, so new inefficiencies and opportunities to exploit them emerge. Bitcoin is however a man-made system governed by strict rules of its protocol which can be changed only by consensus and thus rather stable and consequently it supports inertia, resulting naturally in centralization of mining, mainly due to economies of scale.

This kind of centralization is not a problem in isolation, even if it seems to be the case according to classical Byzantine fault tolerance (BFT) analysis, established for Bitcoin's consensus protocol already in [16]. Satoshi Nakamoto however only estimated the probability of successful double-spending attack. For a more detailed analysis see [18], where the correct probabilistic model is employed. Moreover, the paper briefly discusses economics of double spending as well. Finally, a rigorous mathematical treatment of Bitcoin's BFT probabilistic model is presented in [12].

However, the bitcoin mining is already centralized to such an extend that BFT is insufficient to ensure its security. E.g. only the five biggest known mining pools control together more than 50% of hashrate. BFT implies that a majority attacker can easily double-spend, censor transactions and even claim

[7] They are however much smaller.

all block rewards for himself. Had it been rational to execute these attacks, it would have been rather easy for these players to organize. But it is against their best interest as already recognized in [16]: *"If a greedy attacker is able to assemble more CPU power than all the honest nodes, he would have to choose between using it to defraud people by stealing back his payments, or using it to generate new coins. He ought to find it more profitable to play by the rules, such rules that favour him with more new coins than everyone else combined, than to undermine the system and the validity of his own wealth."*

This observation is well established in [13], where a simple but therefore a robust cost-reward based Bitcoin security model is presented. The main conclusion is that miners are committed to preserve trust in Bitcoin network, since their long-term infrastructure investments are non-repurposable due to their high level of specialization. Any loss of trust normally results in BTC price plummeting, affecting the margins significantly. Following the estimates in the paper, at the current issuance of 6.25 BTC per block only a 5% sustainable market price drop would result in almost a 10K BTC loss for an attacker with 60% hashrate majority.

We point out that the authors in [13] did not analyze a sophisticated attacker which hedges against BTC market drop by e.g. a short position in bitcoin futures. However, such an attack is prohibitively expensive since e.g. Interactive Brokers apply for short bitcoin futures (BRR) positions margins of 150% of daily settlement price. Moreover the liquidity of bitcoin futures is limited with open interest currently around 1.5 billion USD. If deep bitcoin derivatives markets develop, such attacks become more probable. Arguably, Bitcoin's ecosystem inclusion into the current financial system may represent its biggest existential threat.

So if the mining centralization is currently not an existential problem, which type of decentralization is necessary to be preserved? Blocks created by the miners are ultimately validated by Bitcoin users running full nodes. If a majority of these users is against the blocks proposed by the miners or their behavior in general, they can organize off-chain to suspend Nakamoto consensus. For details and precedents from Bitcoin's history, see again [13]. This is the ultimate circuit breaker. Imagine e.g. a last-resort capital punishment scenario when the users would change the hashing algorithm "overnight", rendering the special mining equipment practically immediately useless. This ability has to be preserved at all costs. It requires that many users have to be able to run full-node Bitcoin clients. More precisely, the Bitcoin users have to be able to validate transactions included into the blocks. This together with the ability of users to sell their bitcoins, secures a healthy functioning of Bitcoin's network.

B Algorithms

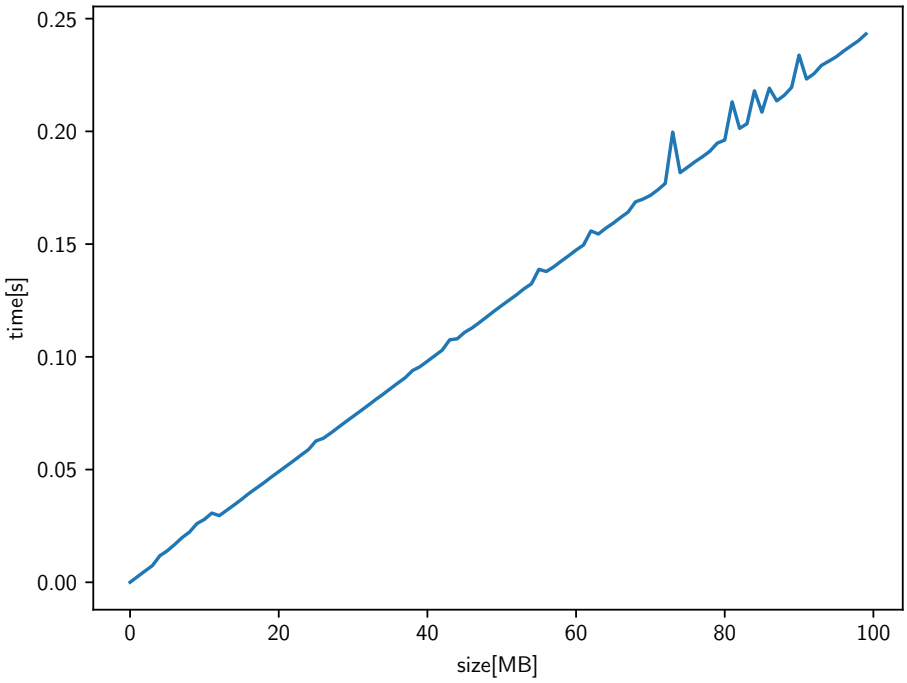

Figure 9: Linearity of hashing time: result of running Algorithm 4

Algorithm 3: An algorithm to compute c_η in tree HBS from Figure 4.

Data: $L \in \mathbb{N}^+$; multi-block index number $i \in \mathbb{N}, i = 0$.
Result: Every 2016 blocks, a new c_η will be available and recorded in $\mathcal{B}_{0,0}$.

```
1  while 1 do   // an abstract loop corresponding to multi-blocks
2      for l ← L − 1 to 0 do
3          for s ← 0 to 2^l − 1 do                          // non-blocking
4              ...;
5              if miner should mine B_{l,s,i} then
6                  if l < L − 1 then
7                      broadcasting and validating blocks B_{l+1,2s,i}, B_{l+1,2s+1,i}
                         while also checking v_{l+1,2s,i} and v_{l+1,2s+1,i};
8                  end
9                  ...;
                   // block assemblage
10                 v_{l,s,i} ← use (50);
11                 s_{l,s,i} = v_{l,s,i};
12                 if l < L − 1 then
13                     s_{l,s,i} = s_{l,s,i} + s_{l+1,2s,i} + s_{l+1,2s+1,i};
14                 end
15                 add v_{l,s,i} and s_{l,s,i} to block header;
16                 if i = 2015 ∧ l = 0 then
17                     c_η = 600s/s_{l,s,i};
18                     add c_η to header;
19                 end
20                 mine B_{l,s,i};
21                 ...;
22             end
23         end
24     end
25     i = i + 1;
26     i = i  mod 2016;
27 end
```

Algorithm 4: Hashing time scales linearly with size.

```
import hashlib
import time
import os
import matplotlib.pyplot as plt
plt.rcParams['pgf.texsystem']= 'pdflatex'

n = 100
x = range(n)
y = []
for i in x:
    m = os.urandom(i*1024**2)
    time_b = time.time()
    hashlib.sha256(m).hexdigest()
    time_e = time.time()
    y.append(time_e - time_b)

fig = plt.subplots()
plt.plot(x,y)
plt.xlabel("size[MB]")
plt.ylabel("time[s]")
plt.savefig('hash_time.pgf')
```

Algorithm 5: Prepares Bitcoin transaction dataset for analysis.

```
import os
from blockchain_parser.blockchain import Blockchain
import random
import pandas as pd

columns_blk = ['block_height', 'block_hash', 'version',\
    'previous_block_hash', 'merkle_root', 'timestamp',\
    'bits', 'nonce', 'difficulty']
types_blk = ['uint32', 'str', 'uint32', 'str', 'str',\
    'uint32', 'uint32', 'uint32', 'float']
dtypes_blk = {c: t for (c, t) in zip(columns_blk, types_blk)}
columns_tx = ['block_height', 'txid', 'hash', 'version', 'n_inputs',\
    'n_outputs', 'is_segwit', 'is_coinbase', 'size', 'output_value']
types_tx = ['uint32', 'str', 'str', 'uint32', 'uint32',\
    'uint32', 'bool', 'bool', 'uint32', 'uint64']
dtypes_tx = {c: t for (c, t) in zip(columns_tx, types_tx)}
blk_df = pd.DataFrame(columns = columns_blk)
blk_df = blk_df.astype(dtype = dtypes_blk)
tx_df = pd.DataFrame(columns = columns_tx)
tx_df = tx_df.astype(dtype = dtypes_tx)

blockchain = Blockchain(os.path.expanduser('.../btc/blocks'))
bx_list = []
for blk in blockchain.get_ordered_blocks(\
    os.path.expanduser('.../btc/blocks/index'),\
    start = 650000, end = 700000):
    if (random.random() > 0.01):
        continue
    h = blk.header
    df = pd.DataFrame([[blk.height, blk.hash, h.version,\
    h.previous_block_hash, h.merkle_root, h.timestamp,\
    h.bits, h.nonce, h.difficulty]], columns = columns_blk)
    blk_df = blk_df.append(df)
    tx_list = []
    for tx in blk.transactions:
        sum_out = 0
        for tout in tx.outputs:
            sum_out = sum_out + tout.value
        tx_list.append([blk.height, tx.txid, tx.hash, tx.version,\
        tx.n_inputs, tx.n_outputs, tx.is_segwit, tx.is_coinbase(),\
        tx.size, sum_out])
tx_df = tx_df.append(pd.DataFrame(tx_list, columns = columns_tx))

tx_df.reset_index(drop = True, inplace = True)
blk_df.reset_index(drop = True, inplace = True)
tx_df.to_hdf('transactions.h5', key='tx_df', mode='w')
blk_df.to_hdf('transactions.h5', key='blk_df')
```

Algorithm 6: Algorithm to plot graph in Figure 2.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
plt.rcParams['pgf.texsystem']= 'pdflatex'

T = pd.read_hdf('transactions.h5', 'tx_df')
T = T[T['output_value'] > 0]
vpb = T['output_value']/(8*T['size'])

count, bins, ignored = plt.hist(np.log10(vpb),\
bins = 100, density = True)

mu = np.sum(np.log10(vpb))/len(vpb)
sigma = np.sqrt(np.sum((np.log10(vpb) - mu)**2)/(len(vpb)-1))

x = np.linspace(min(bins), max(bins), 100)
pdf = (np.exp(-(x - mu)**2 / (2 * sigma**2))
/ (sigma * np.sqrt(2 * np.pi)))
plt.plot(x, pdf, linewidth = 1.5, color = 'r')
plt.xlabel(r'\lg($\beta(t)$)')
plt.ylabel('density')
plt.savefig('tx_vpb.pgf')
plt.show()
```

Algorithm 7: A Python implementation of Algorithm 1.

```
import numpy as np
import pandas as pd

def segment_transactions(L, T, key):
    T = T.sort_values(ascending = False, by = key)

    M = T[key].values[0]
    m = T[key].values[-1]

    S_L = (np.log10(M)-np.log10(m))/L
    #S_L = (np.ceil(np.log10(M))-np.floor(np.log10(m)))/L

    indx = [0]*(L+1)
    indx[0] = 0
    s = T[key]
    C = np.log10(M) - S_L; l = 0
    for r, v in enumerate(s):
        if np.log10(v) < C:
            l = l+1
            indx[l] = r
            C = C - S_L
    indx[L] = len(s)
    Ts = []
    for l in range(L):
        Ts.append(T[indx[l]:indx[l+1]])
    return Ts
if __name__ == "__main__":
    tx_df = pd.read_hdf('transactions.h5', 'tx_df')
    tx_df.reset_index(drop = True, inplace = True)
    tx_df = tx_df[tx_df['output_value'] > 0]
    tx_df['vpb'] = tx_df['output_value']/(8*tx_df['size'])
    Ts = segment_transactions(6, tx_df, 'vpb')
```

Algorithm 8: Algorithm to obtain latex code of Table 1.

```
def generate_latex_table(Ts):
    text = """\def\\arraystretch{1.1}
\\begin{table}
\\begin{tabular}{|"""
    text += 'c|'*(len(Ts)+1)+"""}
\hline
$l$"""
    for i in range(len(Ts)):
        text += f' & {i}'
    text +="\\\ \hhline{|"
    text += '=|'*(len(Ts)+1)+"""}
$|\mathcal{T}_l|$"""
    for i in range(len(Ts)):
        text += f' & {len(Ts[i])}'
    text += """ \\\ \hline
min($\\beta(t)$)"""
    for i in range(len(Ts)):
        text += f' & {Ts[i]["vpb"].min():.2}'
    text += """ \\\ \hline
max($\\beta(t)$)"""
    for i in range(len(Ts)):
        text += f' & {Ts[i]["vpb"].max():.2}'
    text += """ \\\ \hline
$\overline{\\beta(t)}$"""
    for i in range(len(Ts)):
        text += f' & {Ts[i]["vpb"].mean():.2}'
    text += """ \\\ \hline
min($v(t)$)"""
    for i in range(len(Ts)):
        text += f' & {Ts[i]["output_value"].min():.2}'
    text += """ \\\ \hline
max($v(t)$)"""
    for i in range(len(Ts)):
        text += f' & {Ts[i]["output_value"].max():.2}'
    text += """ \\\ \hline
$\overline{v(t)}$"""
    for i in range(len(Ts)):
        text += f' & {Ts[i]["output_value"].mean():.2}'
    text += """ \\\ \hline
$\sum v(t)$"""
    for i in range(len(Ts)):
        text += f' & {Ts[i]["output_value"].sum():.2}'
    text += """ \\\ \hline
\end{tabular}
\caption{...}
\label{...}
\end{table}"""
    return text
```

Algorithm 9: Functions to estimate c_η, $\{\eta_l\}$ and time investment per level for our example dataset. Input Ts is obtained by Algorithm 7.

```
def compute_c_eta_and_eta(Ts, num_blocks = 493):
    s = 0
    for l in range(len(Ts)):
        s += Ts[l]['vpb'].mean()*8*Ts[l]['size'].sum()
    c_eta = 600*num_blocks*10**8/s
    eta = []
    for l in range(len(Ts)):
        eta.append(c_eta*Ts[l]['vpb'].mean()/10**8)
    return c_eta, eta

def compute_time_per_level(eta, Ts, num_blocks = 493):
    t = []
    for l in range(len(Ts)):
        t.append(eta[l]*8*Ts[l]['size'].sum()/num_blocks)
    return t
```

Algorithm 10: A python implementation of shard function (40).

```
def shardf(l, string):
    b = hashlib.sha256(bytes(string, 'utf-8')).digest()
    shifts = []
    for r in range(l//8+1):
        shifts = shifts + [int(i) for i in f'{b[r]:08b}']
    shards = [0]*(l+1)
    for i in range(l):
        if shifts[i] == 0:
            shards[i+1] = 2*shards[i]
        else:
            shards[i+1] = 2*shards[i]+1
return shards[l], shards
```

Algorithm 11: Algorithm to plot graph in Figure 5.

```
import numpy as np
import matplotlib.pyplot as plt
plt.rcParams['pgf.texsystem']= 'pdflatex'

def ratio_blocks_mfn_vs_full(N, L):
    return N*L**2/(2**L-1)**2 + L/(2**L-1)

n_trans = 4200
L = 25
x = range(1, L)

y1 = [np.log10(ratio_blocks_mfn_vs_full(n_trans,i)) for i in x]
y2 = [np.log10(ratio_blocks_mfn_vs_full(2**i-1,i)) for i in x]
y3 = [np.log10((2**i-1)/600) for i in x]

fig, ax = plt.subplots()
ax.axhline(y = 0, color = 'k', linewidth = 1)
ax.plot(x, y1, label = '$\lg(r), N = 4200$', linestyle = '-')
ax.plot(x, y2, label = '$\lg(r), N = 2^L-1$', linestyle = '--')
ax.plot(x, y3, label = '$\lg((2^L-1)/600)$', linestyle = '-.')
ax.legend()
plt.xlabel('L')

plt.savefig('sharding_efficiency.pgf')
```

Algorithm 12: Algorithm to plot graph in Figure 7.

```
import numpy as np
import matplotlib.pyplot as plt
plt.rcParams['pgf.texsystem'] = 'pdflatex'

def d_MFN_day(L, n):
    return (n*L**2/(2**L-1) + L)*86400*250/1024**2

n_1 = 1700
n_2 = 10000
n_3 = 50000
L = 30
x = range(1, L)

y1 = [np.log10(d_MFN_day(i, n_1)) for i in x]
y2 = [np.log10(d_MFN_day(i, n_2)) for i in x]
y3 = [np.log10(d_MFN_day(i, n_3)) for i in x]

fig, ax = plt.subplots()
ax.plot(x, y1, label = f'$\lg(d_{{MFN}}(day)), n = {n_1}$',\
    linestyle = '-')
ax.plot(x, y2, label = f'$\lg(d_{{MFN}}(day)), n = {n_2}$',\
    linestyle = '--')
ax.plot(x, y3, label = f'$\lg(d_{{MFN}}(day)), n = {n_3}$',\
    linestyle = '-.')
ax.legend()
plt.xlabel('L')
plt.ylabel('$\lg(\mathrm{MB})$')

plt.savefig('download_mfn.pgf')
```

Algorithm 13: Algorithm to plot graph in Figure 8.

```
import numpy as np
from scipy import optimize
import matplotlib.pyplot as plt
plt.rcParams['pgf.texsystem']= 'pdflatex'

def s_MFN_day(L, n):
    return n*L/(2**L-1)*86400*250/1024**2

def d_MFN(L, n):
    return n*L**2/(2**L-1) + L

n_range = range(100, 50000, 100)
res = [optimize.minimize(d_MFN, 10, args = [n])\
    for n in n_range]
suc = [(r['success']) for r in res]
if all(suc):
    print('OK!')

L_argmin = [L['x'][0] for L in res]
s_mfn = [s_MFN_day(L, n) for L, n in zip(L_argmin, n_range)]

fig, ax = plt.subplots(2, 1)
ax[0].plot(n_range, L_argmin, label =\
    '$\mathrm{argmin}(d_{MNF}(L, n))$', linestyle = '-')
ax[1].plot(n_range, s_mfn, label = '$s_{MNF}(day)$'\
    , linestyle = '-')
ax[0].legend()
ax[1].legend()
ax[1].set_xlabel('n')
ax[0].set_ylabel('L')
ax[1].set_ylabel('MB')

plt.savefig('optimal_L_mfn.pgf')
```