Difficulty Control for Blockchain-Based Consensus Systems

Daniel Kraft
University of Graz
Institute of Mathematics, NAWI Graz
Universitätsplatz 3, 8010 Graz, Austria
Email: daniel.kraft@uni-graz.at

March 18th, 2015

Abstract

Crypto-currencies like Bitcoin have recently attracted a lot of interest. A crucial ingredient into such systems is the "mining" of a Nakamoto blockchain. We model mining as a Poisson process with time-dependent intensity and use this model to derive predictions about block times for various hash-rate scenarios (exponentially rising hash rate being the most important). We also analyse Bitcoin's method to update the "network difficulty" as a mechanism to keep block times stable. Since it yields systematically too fast blocks for exponential hash-rate growth, we propose a new method to update difficulty. Our proposed method performs much better at ensuring stable average block times over longer periods of time, which we verify both in simulations of artificial growth scenarios and with real-world data. Besides Bitcoin itself, this has practical benefits particularly for systems like Namecoin. It can be used to make name expiration times more predictable, preventing accidental loss of names.

Keywords: Crypto-Currency, Bitcoin Mining, Namecoin, Nakamoto Blockchain, Poisson Process

Published by Springer in **Peer-to-Peer Networking and Applications**, DOI **10.1007/s12083-015-0347-x**. The final publication is available at `http://link.springer.com/article/10.1007/s12083-015-0347-x`.

1 Introduction

In recent years, so-called "crypto-currencies" have attracted a growing amount of interest from various communities. Among them, Bitcoin [14] is the most widely known and the initial system that managed to provide a digital payment and money system without any central instance, clearing house or issuer of monetary tokens. Instead, all transactions are performed and validated by a peer-to-peer system, where each node is "equal" and none has any special authority.

Roughly speaking, the system works by keeping a global ledger with "account balances", where each account (Bitcoin address) is represented by an asymmetric cryptographic key pair. Transactions can only be performed by the owner of the private key, since other network participants only accept them as valid if they carry a valid signature for the address' public key. A major difficulty, however, is to ensure that the entire peer-to-peer network reaches a *consensus* about the current state of the ledger. In particular, the owner of an address may create two mutually conflicting transactions, spending the same balance twice to different recipients. This may lead to some parts of the network considering the first recipient to be the new owner of the coins and rejecting the second transaction, while the other part of the network has it the other way round. This is called *double spending*. Earlier proposals for digital payment systems, such as Chaumian cash [5], had to rely on central instances to detect and prevent double spending attempts.

Bitcoin's main innovation is the introduction of a proof-of-work system similar to HashCash [3] that allows the network to reach consensus even in the face of potential double spendings and in a *completely decentralised fashion*. A brief introduction into the basic mechanics of this process, called *mining*, is given in Section 2. Roughly speaking, mining network participants use their processing power to solve a proof-of-work puzzle. Whenever a solution is found, a new *block* is generated and attached to the so-called *blockchain*. This data structure represents the network's consensus about the transaction history. If a node manages to find a new block, it is allowed to award itself a certain number of bitcoins. This creates strong economic incentives for the network as a whole to find a consensus. As more and more processing power is added to the network, the rate at which new blocks are

found increases. This is undesirable, because it increases the amount of newly created bitcoins on one hand, and also causes problems due to network latency on the other hand. A thorough investigation of the latter issue can be found in [7]. Thus, the Bitcoin network regulates the block frequency by adjusting the proof-of-work difficulty dynamically.

In this paper, we want to present a mathematical model for the mining process itself and use it to analyse the properties of Bitcoin's algorithm for retargeting the difficulty. We will particularly focus on the case of exponentially rising hash rate, which is the situation observed in practice in accordance with Moore's law [13]. We will see that Bitcoin's retargeting method yields blocks that are found too frequently in this situation. This is empirically well known in the Bitcoin community and not considered a big problem. However, it can pose a bigger problem for applications based on the same technology but with different goals. In particular, the blockchain system can also be used to create a naming system that goes beyond "Zooko's triangle" [20], [19]: In Namecoin [1], a Nakamoto blockchain is used to provide a name-value database that is secure, completely decentralised and allows for human-readable names. This has a lot of very interesting potential applications, including an alternative to centralised domain-name systems and the secure exchange of public keys linked to human-readable identity names. To prevent names from being lost forever if the owner's private key is lost accidentally, names in Namecoin *expire* after a certain number of blocks (currently 36,000) if they are not renewed in time. Blocks that are constantly found too frequently cause the expiration to happen too early in terms of real time. Consequently, name owners that are not cautious enough risk missing the renewal deadline and losing their names. While individual block times are, of course, random, fluctuations average out over a full expiration period of many blocks. It is thus very desirable to better understand the systematic "error" introduced by the difficulty-retargeting algorithm and, potentially, remove it by choosing a different method for controlling the difficulty. This allows to better match expiration times to real time, which is much easier to handle for users of the system.

To put our work into perspective, we would also like to refer to other recent publications concerning Bitcoin mining: [4], [11], [16], [18] All of them deal with possible attacks on mining that would allow an attacker to double spend transactions, which is a different focus from our work. Most of the models used in the literature to discuss such attacks assume that mining difficulty is constant. Consequently, the difficulty-update mechanism is not taken into account at all. We, on the other hand, are not interested in double-spending attacks. Our focus is the explicit modelling of the difficulty update, which is a feature that sets our model apart from those existing in the literature. It is also worthwhile to mention that there exists a variety of forks of the Bitcoin code and network. Some of these so-called "altcoins" implement also changes to the difficulty update. However, we are not aware of any academic literature analysing or modelling the changed methods. Instead, changes are mostly made in an empirical, ad-hoc fashion. The goal of these changes is to counteract extreme difficulty changes on a short time scale if miners quickly switch between different networks. Our work is different, since we assume a stable base of mining power, and are interested in the behaviour of the difficulty on *much longer* time scales.

Section 3 will be devoted to modelling the mining process itself without considering difficulty changes. In Section 4, Bitcoin's difficulty-update method will be analysed, and in Section 5, we propose an improved update formula. Section 6 and Section 7, finally, will be used to analyse our models both in theory and with practical simulations (including for real-world data).

2 Bitcoin Mining and the Blockchain

Before we start our modelling, let us briefly describe how the mining process works. For a thorough discussion of the involved concepts, see chapters 2, 7 and 8 of [2]. A description can also be found in subsection 2.1 of [11] and section 2 of [16]. The original introduction of the concept is section 4 of the Bitcoin whitepaper [14].

All transactions that change the distributed Bitcoin ledger are grouped into *blocks*. Each block represents thus an "atomic update" of the ledger's state. In order for a block to be valid, it has to fulfil a proof-of-work condition: A particular cryptographic hash involving the block's content is formed, and must be below a threshold value. In other words, nodes wishing to publish new blocks have to do a brute-force search for a partial hash collision. This ensures that a block cannot be changed without redoing all the work involved in finding this hash collision.

In addition to current transactions, each block also contains a reference to a preceding block. In other words, from a given block, a chain of other blocks linking it to the initial network consensus (the *genesis block* that is hardcoded into the Bitcoin client) can be constructed. Such a data structure is called a *Nakamoto blockchain*. Following the chain of blocks and performing the encoded transactions allows one to construct a precisely defined state of the global ledger corresponding to each block. The client is designed to always look for the "longest" branch in the tree of all known blocks. (Actually, the branch which contains the most proof-of-work. But for a basic understanding, one can very well imagine it to be the longest branch.) The ledger state corresponding to this longest branch is considered the "true" state. Furthermore, also mining nodes always build their new blocks onto the longest known chain.

This has an important implication: Assume an attacker wants to revert a transaction to reclaim ownership over bitcoins that were already spent. In order to construct such an "alternative reality" and to have the network accept it, the attacker now has to build a chain of blocks forking off the main chain before the point in time when the coins were spent. But the alternative chain will only be accepted if it becomes longer than the already existing chain. Since this requires redoing all the proof-of-work that was involved in the main chain, the attack will only succeed with non-vanishing probability if the attacker controls more processing power than the entire "honest" network combined. (This is called a "51% attack".) In practice, this is almost impossible to do given the existing mining power of the Bitcoin network.

3 Modelling the Mining Process

Now, we are ready to derive a general stochastic model for the mining process described in Section 2. In particular, we will argue that the mining of blocks can be described by an inhomogeneous Poisson process (see, for instance, [17] for a general discussion). Our model will consider the hash rate $R(t)$ as well as the network difficulty D as given input parameters, and we will derive the probability distribution of the resulting individual block times, the time for M blocks (corresponding to the expiration period), and their expectation values. Later on, starting in Section 4, we will consider concrete scenarios for $R(t)$ as well as letting D be controlled by some retargeting algorithm. (In other words, depend, in turn, on the realised block times.) An overview of the notation used in the models throughout this and the following sections can be found in Appendix A.

As we have seen above, solving the proof-of-work process for a valid block works by calculating cryptographic hashes in a brute-force way. We may assume that each hash value is drawn from a uniform distribution, say on the interval $[0, 1]$. A block is found if the drawn value is less than a *target value*, which is usually expressed in terms of the *network difficulty* $D > 0$ as $\frac{1}{D}$. Thus, each hash attempt yields a valid block with probability $p = \frac{1}{D}$. (In practice, the possible hash values are actually 256-bit integers and difficulty is measured in other units. However, this does not matter for our considerations here, other than a constant factor.) From these assumptions, it follows that the number $N(t)$ of blocks found after some time t is described by a *Poisson process*. If we denote the frequency of hashes calculated per time by $R(t)$, then the intensity of this process is given by

$$\lambda(t) = R(t)p = \frac{R(t)}{D}.$$

We are mainly interested in the *time for finding M blocks*. If we denote the interarrival times of N by X_i, $i \in \mathbb{N}$, then the time for M blocks is the random variable

$$S_M = X_1 + X_2 + \cdots + X_M = \sum_{k=1}^{M} X_k.$$

The following result is well-known, and can be found, for instance, in [17]:

Theorem 1. *Let λ be continuous as function of t and define*

$$m(t) = \int_0^t \lambda(\tau)\, d\tau.$$

We will also assume that m is strictly increasing (thus bijective) and that $\lim_{t\to\infty} m(t) = \infty$.

The probability distribution of S_M is then given by

$$\mathrm{P}\left(S_M \leq t\right) = \mathrm{P}\left(N(t) \geq M\right) = \sum_{k=M}^{\infty} \frac{m(t)^k}{k!} e^{-m(t)}.$$

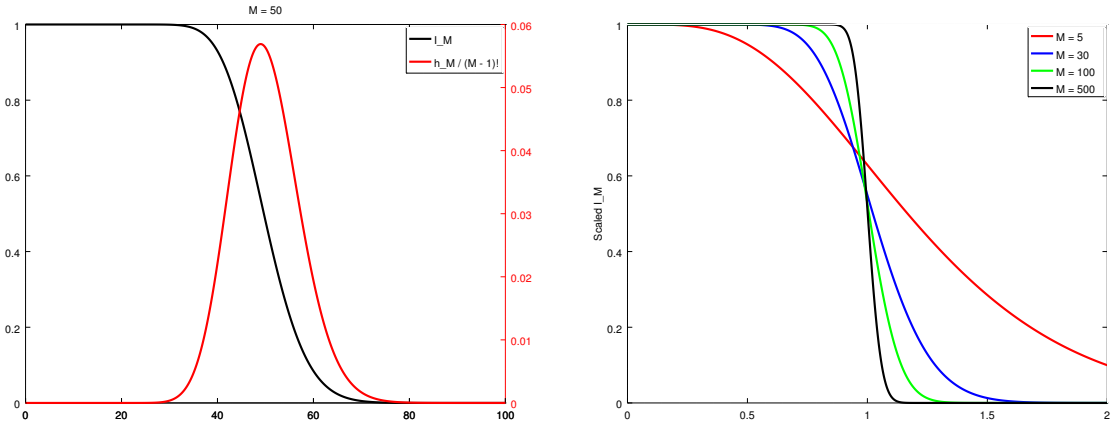
It can be described by the density function

$$f(S_M, t) = \lambda(t) e^{-m(t)} \frac{m(t)^{M-1}}{(M-1)!}. \tag{1}$$

The next goal will be to calculate (as far as possible) the *expectation value* $\mathbb{E}\left(S_M\right)$. As a first step, note that the substitution $u = m(t)$ can be used to calculate

$$\int_0^t \lambda(\tau) e^{-m(\tau)} m(\tau)^{M-1}\, d\tau = -\Gamma\left(M, m(\tau)\right)\big|_0^t. \tag{2}$$

Here, $\Gamma(\cdot,\cdot)$ denotes the *incomplete gamma function*. For more details, see Chapter 8 of [8]. Noting that $\Gamma(M, 0) = \Gamma(M) = (M-1)!$, this relation also implies that (1) is properly normalised.



(a) (b)

Figure 1: The functions I_M and h_M for $M = 50$ (left), and $\tilde{I}_M$ for different values of M (right).

Lemma 1. *Under the conditions of Theorem 1,*

$$\int_0^t \tau f(S_M, \tau)\, d\tau = -\frac{\tau}{(M-1)!}\Gamma\left(M, m(\tau)\right)\Big|_0^t + \frac{1}{(M-1)!}\int_0^t \Gamma\left(M, m(\tau)\right)\, d\tau. \tag{3}$$

If we assume in addition that $\lambda(t) \geq \underline{\lambda}$ for $t \to \infty$ and some $\underline{\lambda} > 0$, then this gives in particular:

$$\mathbb{E}\left(S_M\right) = \frac{1}{(M-1)!}\int_0^\infty \Gamma\left(M, m(\tau)\right)\, d\tau \tag{4}$$

Proof. (3) follows via integration by parts and (2). For $\mathbb{E}\left(S_M\right)$, note that the additional assumption ensures that $m(t) \geq \underline{\lambda} t + C$, which in turn gives

$$\lim_{t\to\infty} t \cdot \Gamma\left(M, m(t)\right) = 0.$$

This implies that the boundary term vanishes, yielding (4). □ □

Before we continue by examining concrete scenarios for the hash-rate development, we would like to stress that (4) is, unfortunately, hard to calculate for non-trivial functions m. However, note that the integrand (with x instead of $m(\tau)$) can be more explicitly written as

$$I_M(x) = \frac{\Gamma\left(M, x\right)}{(M-1)!} = \frac{\Gamma\left(M, x\right)}{\Gamma\left(M, 0\right)} = \frac{\int_x^\infty \tau^{M-1}e^{-\tau}\, d\tau}{\int_0^\infty \tau^{M-1}e^{-\tau}\, d\tau} = \frac{\int_x^\infty h_M(\tau)\, d\tau}{\int_0^\infty h_M(\tau)\, d\tau},$$

where we have introduced the auxiliary function $h_M(\tau) = \tau^{M-1}e^{-\tau}$. In particular $I_M(x) \in (0, 1]$ for all $x \geq 0$, and I_M is strictly decreasing with $I_M(0) = 1$ and $I_M(x) \to 0$ asymptotically as $x \to \infty$. This behaviour can also be clearly seen in Figure 1a, which shows I_M and h_M for $M = 50$. The range where I_M shows the transition from 1 to 0 is where h_M provides non-vanishing "mass" in the integral, so roughly "around" the maximum of h_M.

Lemma 2. *h_M has its global maximum at $\tau_0 = M - 1$.*
It is strictly increasing on $[0, \tau_0]$ and strictly decreasing on $[\tau_0, \infty)$.

Proof. This follows immediately when considering the sign of $h'_M(\tau) = \tau^{M-2}e^{-\tau}(M - 1 - \tau)$. □ □

These considerations motivate us to scale the argument of I_M such that the transition happens, for all values of M, at the same position. Thus, let us introduce $\tilde{I}_M(x) = I_M((M-1)x)$. This function is shown in Figure 1b for different values of M. One can clearly see that it approaches the step function

$$I_\infty(x) = \begin{cases} 1 & x < 1 \\ 0 & x > 1 \end{cases}$$

in the limit $M \to \infty$. Also note that the values of M that are of practical interest are even larger than the ones shown in the plot. For instance, $M = 2{,}016$ is the number of blocks between changes to the difficulty in the Bitcoin protocol. Hence, it makes sense to simplify the calculation of (4) by approximating $\tilde{I}_M \approx I_\infty$. To further justify this approximation, we can also show a formal convergence property:

Lemma 3. *With the notations as above:*

1. $\lim_{M\to\infty}(M-1)\frac{h_M((M-1)\tau)}{(M-1)!}=0$ *for all* $\tau\geq 0, \tau\neq 1$.
2. $\tilde{I}_M\to I_\infty$ *as* $M\to\infty$, *pointwise for all* $x\geq 0, x\neq 1$.

Proof. The first part is trivial for $\tau=0$. For $\tau>0$ and $\tau\neq 1$, note that

$$1+\log\tau-\tau<0 \text{ and } M!\geq\sqrt{2\pi M}\cdot M^M e^{-M}.$$

The latter is a version of Stirling's approximation (see 5.6.1 in [8]). Thus we get

$$\begin{aligned}\lim_{M\to\infty}(M-1)\frac{h_M((M-1)\tau)}{(M-1)!} &= \lim_{M\to\infty} M\frac{h_{M+1}(M\tau)}{M!}\leq\lim_{M\to\infty}\frac{M(M\tau)^M e^{-M\tau}}{\sqrt{2\pi M}\cdot M^M e^{-M}}\\ &= \lim_{M\to\infty}\sqrt{\frac{M}{2\pi}}\cdot e^{M(1+\log\tau-\tau)}=0.\end{aligned}$$

For the second part, assume first that $x>1$ is fixed. Then

$$\tilde{I}_M(x)=\int_{(M-1)x}^{\infty}\frac{h_M(\tau)}{(M-1)!}\,d\tau=\int_x^\infty(M-1)\frac{h_M((M-1)\tau)}{(M-1)!}\,d\tau. \tag{5}$$

Furthermore, if $\tau\geq x>1$ and M is sufficiently large,

$$\begin{aligned}\sqrt{M}\cdot e^{M(1+\log\tau-\tau)} &= \exp\left(\frac{\log M}{2}+M(1+\log\tau-\tau)\right)=\exp\left(M\left(\frac{1}{2}\frac{\log M}{M}+1+\log\tau-\tau\right)\right)\\ &\leq\exp\left(\frac{1}{2}\frac{\log M}{M}+1+\log\tau-\tau\right)\leq e^{2+\log\tau-\tau}.\end{aligned}$$

Since this function is integrable over $\tau\in[1,\infty)$, we can use Lebesgue's dominated convergence theorem (Theorem 3 on page 20 of [10]) to get $\tilde{I}_M(x)\to 0$ as $M\to\infty$ by applying the first part to (5).

It remains to show $\tilde{I}_M(x)\to 1$ if $x<1$. For this, note first that

$$\tilde{I}_M(x)=1-\int_0^{(M-1)x}\frac{h_M(\tau)}{(M-1)!}\,d\tau=1-\int_0^x(M-1)\frac{h_M((M-1)\tau)}{(M-1)!}\,d\tau.$$

Consequently, it suffices to show

$$\lim_{M\to\infty}\int_0^x(M-1)\frac{h_M((M-1)\tau)}{(M-1)!}\,d\tau=0.$$

If we use monotonicity of h_M (see Lemma 2), it follows that the integrand can be estimated by its value at the upper boundary. Hence

$$\int_0^x(M-1)\frac{h_M((M-1)\tau)}{(M-1)!}\,d\tau\leq x\cdot(M-1)\frac{h_M((M-1)x)}{(M-1)!}\to 0$$

according to the first part. □ □

Having this result in hand, we can use $\tilde{I}_M\approx I_\infty$ also for still finite values of M to approximate $\mathbb{E}(S_M)$. To apply this in a concrete situation, the following reformulation of (4) is useful:

Lemma 4. *Assume that the conditions of Theorem 1 are satisfied and that* $\lambda(t)\geq\underline{\lambda}$ *for all* $t\geq 0$ *and some* $\underline{\lambda}>0$. *Then, with the notation from above,*

$$\mathbb{E}(S_M)=(M-1)\int_0^\infty\left(m^{-1}\right)'((M-1)u)\cdot\tilde{I}_M(u)\,du. \tag{6}$$

Proof. Note first that our conditions ensure that m is strictly increasing, thus m^{-1} exists. Since $m'(\tau)=\lambda(\tau)\geq\underline{\lambda}>0$ holds true for all τ, also m^{-1} is continuously differentiable. Thus we can apply the substitution $\tau=m^{-1}((M-1)u)$, which turns (4) into (6). □ □

Finally, if we replace $\tilde{I}_M$ by I_∞ in (6), we get

$$\mathbb{E}(S_M)\approx(M-1)\int_0^1(m^{-1})'((M-1)u)\,du=\int_0^{M-1}(m^{-1})'(\tau)\,d\tau=m^{-1}(M-1). \tag{7}$$

3.1 Constant Hash Rate

As a first scenario, consider the case that $R(t) = R$ is constant over time. Consequently, as long as the difficulty is also not changed by the network, $\lambda = \frac{R}{D}$ is also constant, and $m(t) = \lambda t$. Theorem 1 and Lemma 1 are clearly applicable in this case, yielding

$$\mathbb{E}(S_M) = \frac{1}{(M-1)!}\int_0^\infty \Gamma(M, \lambda\tau)\, d\tau = \frac{1}{\lambda}\frac{\Gamma(M+1)}{(M-1)!} = \frac{M}{\lambda}. \tag{8}$$

Note that this result can also be interpreted as M average interarrival times for the homogeneous Poisson process with intensity λ. We would also like to remark that (8) is exact. The approximation (7) would, in this case, yield the slightly off result $\mathbb{E}(S_M) \approx \frac{M-1}{\lambda}$.

3.2 Exponential Hash Rate

The second scenario of interest, which seems to occur in practice due to Moore's law [13], is that of an *exponentially rising hash rate*:

$$R(t) = R(0)e^{\alpha t} \;\Rightarrow\; \lambda(t) = \lambda_0 e^{\alpha t} \;\Rightarrow\; m(t) = \frac{\lambda_0}{\alpha}\left(e^{\alpha t} - 1\right) \tag{9}$$

See Figure 4a for an impressive demonstration that exponential growth does, indeed, occur in practice. We assume $\alpha > 0$, since otherwise the hash rate would decay instead and our basic assumption of $m(t) \to \infty$ with $t \to \infty$ would not hold. The case $\alpha = 0$, of course, degenerates to the constant hash-rate scenario discussed already above.

In this situation, Theorem 1 and Lemma 1 can again be employed to calculate $\mathbb{E}(S_M)$. It is more complicated this time, though, since $\Gamma(M, m(\tau)) = \Gamma\left(M, \frac{\lambda_0}{\alpha}\left(e^{\alpha\tau} - 1\right)\right)$ is quite a difficult function to integrate in practice.

Lemma 5. *For $M = 1$, we have*

$$\mathbb{E}(S_1) = \mathbb{E}(X_1) = \frac{e^{\frac{\lambda_0}{\alpha}}}{\alpha}\Gamma\left(0, \frac{\lambda_0}{\alpha}\right).$$

Proof. Note first that $\Gamma(M, m(\tau)) = \Gamma(1, m(\tau)) = e^{-m(\tau)}$ in this special case. With the substitution $u = e^{\alpha\tau}$, we get

$$\mathbb{E}(S_1) = \int_0^\infty e^{-m(\tau)}\, d\tau = \int_1^\infty \frac{1}{\alpha u}\exp\left(-\frac{\lambda_0}{\alpha}(u-1)\right) du = -\frac{e^{\frac{\lambda_0}{\alpha}}}{\alpha}\,\Gamma\left(0, \frac{\lambda_0}{\alpha}u\right)\Big|_1^\infty = \frac{e^{\frac{\lambda_0}{\alpha}}}{\alpha}\Gamma\left(0, \frac{\lambda_0}{\alpha}\right).$$

□ □

Unfortunately, Lemma 5—while interesting—does not help much in practice. It only gives us the expectation value of the first arrival time and not the expected time for M blocks. In practice, the change in hash rate *during a single block interval* is negligible (while it is not for a larger number of blocks). For larger values of M, we can instead employ the approximation (7). Since in the current case

$$m(t) = \frac{\lambda_0}{\alpha}\left(e^{\alpha t} - 1\right) \;\Rightarrow\; m^{-1}(u) = \frac{1}{\alpha}\log\left(\frac{\alpha}{\lambda_0}u + 1\right),$$

the expected time for M blocks can be approximated by

$$\mathbb{E}(S_M) \approx m^{-1}(M-1) = \frac{1}{\alpha}\log\left(\frac{\alpha}{\lambda_0}(M-1) + 1\right). \tag{10}$$

4 Bitcoin's Difficulty Retargeting

With the model of the mining process itself developed above in Section 3, we will now look one step further: How to *control* the difficulty D in such a way that the block frequency roughly matches a desired value. In other words, we are interested in updating D based on past observations about the network status in order to get $S_M \approx MT$. T denotes the desired time between individual blocks. For Bitcoin, we have $T = 10\,\text{min}$ in practice. The concrete method used in the Bitcoin protocol works like this:

1. Start with some initial difficulty D_0, yielding the initial block rate $\lambda_0 = \frac{R(0)}{D_0}$.
2. Keep D_0 constant over the *retargeting interval* of M blocks. For Bitcoin, $M = 2{,}016$ (or approximately two weeks). The actual network hash rate $R(t)$ and thus also $\lambda(t)$ may change.

3. After M blocks, measure the realised value of S_M (i. e., the actual time it took to find these blocks) and adapt the difficulty to
$$D^+ = D_0 \cdot \frac{MT}{S_M}. \tag{11}$$
Note that it will often be more convenient to consider the "relative difficulty update" instead:
$$\delta = \frac{D^+}{D_0} = \frac{MT}{S_M}.$$

4. Repeat this process with D^+ as initial difficulty.

It is easy to see that (11) increases the difficulty if blocks are found too frequently ($S_M < MT$), and decreases the difficulty if it is the other way round. This behaviour ensures that the block rate is stable even if the hash rate is not. In this section, we want to consider how well this method works under certain scenarios for actual hash-rate development. For this, there will be *three* time points of interest:

- The initial time at which we start with difficulty D_0 and block rate λ_0 is denoted by $t = 0$.
- The value of S_M over the retargeting interval is called t_0. This is the actual time spent that will be compared to MT in (11). It can be considered as a random variable or its realisation, depending on the circumstances.
- The time after *another* M blocks have been found, i. e., the *next* difficulty update happens. We will denote this value (which is also a random variable) by $t_0 + S_M$.

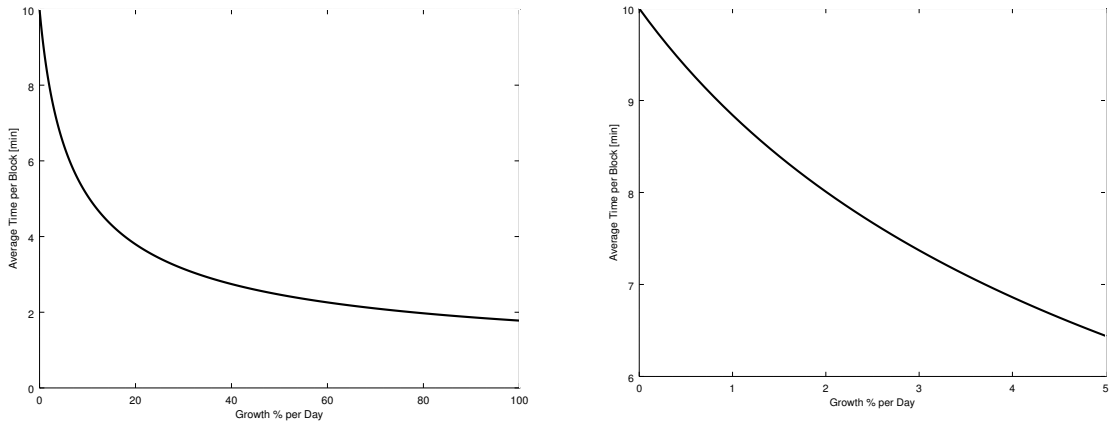
We will use t_0 as the "reference time": At this point, we can already look back over the first retargeting interval. This information is used to calculate a new difficulty D^+, which is then used for the next M blocks. Looking into the future to the third time point, we can estimate S_M and compare S_M again to MT. This tells us how well the difficulty update at t_0 achieves its goal of enforcing $S_M \approx MT$.

4.1 Constant Hash Rate

Let us again consider, as a first scenario, that the hash rate $R(t) = R$ is constant over time. In this case, (11) implies for the updated intensity:
$$\lambda^+ = \frac{R}{D^+} = \frac{R}{D_0}\frac{t_0}{MT} = \lambda_0 \frac{t_0}{MT}$$
Note that $\mathbb{E}(t_0) = \frac{M}{\lambda_0}$ according to (8), so that
$$\mathbb{E}(\lambda^+) = \frac{\lambda_0}{MT}\mathbb{E}(t_0) = \frac{1}{T}.$$
If we consider λ^+ to be fixed at this value and no longer a random variable, we see that this is precisely the right result:
$$\mathbb{E}(S_M) = \frac{M}{\lambda^+} = MT \tag{12}$$
is exactly what we want it to be. In fact, (11) was probably designed exactly to yield this property for a constant hash rate. This means that one difficulty update with (11) is enough to bring the block rate to its target—independently of the initial rate.

4.2 Exponential Hash Rate

Assume now again an exponentially rising hash rate $R(t) = R(0)e^{\alpha t}$ according to (9). In this case, the difficulty update (11) works out to yield
$$\lambda^+ = \frac{R(t_0)}{D^+} = \lambda_0 \frac{t_0}{MT} \cdot e^{\alpha t_0}. \tag{13}$$
There are two important things to note here: First, this λ^+ only gives the intensity of the mining process *at time* t_0. Since the hash rate continues to grow, $\lambda(t)$ will also grow over time. Second, in addition to the factor $\frac{t_0}{MT}$ from the difficulty update itself, we now also have the term $e^{\alpha t_0}$. It represents the exponential increase in hash rate over the retargeting interval, which changes λ^+ on top of the change due to the difficulty update itself. Using the approximate $\mathbb{E}(t_0)$ from (10), this yields
$$\lambda^+ = \lambda_0 \frac{\frac{\alpha}{\lambda_0}(M-1)+1}{MT} \cdot \frac{1}{\alpha}\log\left(\frac{\alpha}{\lambda_0}(M-1)+1\right) = \frac{M-1+\frac{\lambda_0}{\alpha}}{MT}\log\left(\frac{\alpha}{\lambda_0}(M-1)+1\right) = g(\lambda_0). \tag{14}$$
If we watch the evolution over many retargeting periods, the block rates at retargets can be calculated by *iterating* this function g on the initial λ_0. We reach a "stable situation" if this process converges to some limit block rate λ^* as a fixed point of g.



(a) (b)

Figure 2: Average "equilibrium" block time $\frac{t_0^*}{M}$ from (16) for the Bitcoin network. We assume exponential growth of the hash rate and show how the resulting block time depends on the assumed growth rate.

Lemma 6. *g is strictly decreasing and convex. It has a unique fixed point λ^*, which is given by*

$$\lambda^* = (M-1)\alpha \cdot \left(e^{W(MT\alpha)} - 1\right)^{-1}. \tag{15}$$

W denotes the Lambert W-function, see Section 4.13 of [8].

Proof. If we introduce $x = \frac{\lambda}{\alpha}$, $K = M - 1$ and $u = \frac{K}{x}$, then

$$g(\lambda) = \frac{1}{MT}(K+x)\log\left(\frac{K}{x}+1\right) \quad \Rightarrow \quad g'(\lambda) = \frac{\alpha}{MT}\left(\log(u+1) - u\right).$$

Since g' is negative and increasing in λ, this shows that g is strictly decreasing and convex. Note also that $\lim_{\lambda\to 0^+} g(\lambda) = \infty$, which implies that $g(\lambda) = \lambda$ must have a unique solution λ^*.

To show (15), we can explicitly solve the equation

$$(K+x)\log\left(\frac{K}{x}+1\right) = MT\alpha \cdot x.$$

With $v = \frac{K}{x} + 1$, it turns into

$$v \log v = v^v = MT\alpha,$$

which can be solved using W. Undoing all the substitutions, we get the final solution (15). □ □

What Lemma 6 tells us is this: If we start a retargeting interval with λ^*, the block rate will increase exponentially together with the hash rate, but it will, on average, be set back to λ^* at the next difficulty update. The total time t_0 for one retargeting interval can be calculated from (15) and (10). Even simpler and more direct, though, is to note that (13) is, for the fixed point λ^*, equivalent to

$$MT\alpha = \alpha t_0^* \cdot e^{\alpha t_0^*} \quad \Rightarrow \quad t_0^* = \frac{1}{\alpha} W(MT\alpha). \tag{16}$$

For the parameters of the real Bitcoin network, this function (actually, $\frac{t_0^*}{M}$ to get the average time *per block*) is shown in Figure 2. In practice, growth rates of 1%–2% have been observed.

Lemma 7. *t_0^* is strictly decreasing as a function of α. Furthermore,*

$$\lim_{\alpha\to 0} t_0^*(\alpha) = MT.$$

Proof. For $\alpha > 0$ and writing $W = W(MT\alpha)$ for simplicity, we have

$$t_0^*(\alpha)' = \frac{MT}{\alpha} e^{-W}\left(\frac{1}{1+W} - \frac{1}{MT\alpha} W e^W\right) = \frac{MT}{\alpha} e^{-W}\left(\frac{1}{1+W} - 1\right) = -\frac{MT}{\alpha} e^{-W}\frac{W}{1+W} < 0.$$

The limit for $\alpha \to 0$ follows using differentiability of W at 0. □ □

This result shows us that (16) is consistent with (12) for the limit $\alpha \to 0$, which means no growth. It also shows that if $\alpha > 0$ is fixed, we get blocks that are systematically too fast even with difficulty updates happening as designed. One final thing remains to be seen for this analysis: While we know that an "equilibrium" fixed point λ^* exists and what its properties are, it is not yet clear whether the iteration (14) actually converges towards λ^*. This iteration is what happens to the Poisson intensities in practice if we start with $\lambda_0 \neq \lambda^*$. The following result gives the answer:

Theorem 2. *(14) converges linearly to λ^* for arbitrary initial $\lambda_0 > 0$.*

Proof. Recall the basic properties of g from Lemma 6, in particular that g is strictly decreasing. Together with the limit

$$\lim_{\lambda \to \infty} g(\lambda) = \frac{M-1}{MT},$$

this implies that $g(\lambda) \geq \frac{M-1}{MT}$ for all $\lambda > 0$. Since we are interested in the iterates of (14), we can restrict ourselves to $\lambda \geq \frac{M-1}{MT}$ without loss of generality. Also recall that g' is negative and strictly increasing. Thus

$$|g'(\lambda)| \leq \left| g'\left(\frac{M-1}{MT}\right)\right| = 1 - \frac{\log(\alpha MT + 1)}{\alpha MT} < 1$$

for all those λ. With this result, linear convergence follows by Banach's fixed-point theorem. □ □

5 An Improved Difficulty Update

We have seen above in Section 4 that the difficulty-update strategy employed by Bitcoin works as designed: If the hash rate is constant, it yields the target block rate ever after the first difficulty update. Even if the hash rate is exponentially rising, it is able to control the block rate towards a "stable situation" (see Theorem 2). However, it *does not* achieve the target block rate for the latter situation. This is not a big deal for the Bitcoin system to function in practice, but it may be undesirable for more advanced applications of Bitcoin's blockchain technology. In this section, we will propose an alternative difficulty-update method that can be used to improve accuracy of the resulting block rate for exponentially increasing hash rate.

This section considers mainly the situation of "exponentially rising hash rate" from above, thus we continue with the notation used in the description before: We assume that $R(t) = R(0)e^{\alpha t}$ is the hash rate, $\lambda_0 = \frac{R(0)}{D_0}$ the initial block rate, M the number of blocks in a retargeting interval and $t_0 = \mathbb{E}(S_M)$ the expected (or observed) time for all M blocks of the initial retargeting interval. According to (10), this time is approximately given by

$$t_0 = \frac{1}{\alpha} \log\left(\frac{\alpha}{\lambda_0}(M-1) + 1\right). \tag{17}$$

Furthermore, we will denote the time for M blocks in the *second* retargeting interval by S_M. Our goal is to find the difficulty update D^+ such that $S_M \approx TM$. Assuming for a moment that α is known (which is not true in practice), we can use (17) to calculate λ_0 as well as the "required" block rate $\lambda^+ = \lambda(t_0)$ immediately after the difficulty update:

$$\lambda_0 = \frac{\alpha(M-1)}{e^{\alpha t_0} - 1}, \quad \lambda^+ = \frac{\alpha(M-1)}{e^{\alpha TM} - 1}.$$

Thus

$$\frac{\lambda^+}{\lambda_0} = \frac{R(0)e^{\alpha t_0}}{R(0)} \frac{D_0}{D^+} = \frac{e^{\alpha t_0} - 1}{e^{\alpha TM} - 1},$$

which further gives

$$\delta = \frac{D^+}{D_0} = \frac{e^{\alpha TM} - 1}{1 - e^{-\alpha t_0}} \approx \frac{TM}{t_0}. \tag{18}$$

Using the first-order approximation $e^x \approx 1 + x$ to simplify the exponential functions yields the indicated approximation, which corresponds to Bitcoin's update (11). Without this approximation, the update would give us the "ideal" new difficulty. Unfortunately, though, it is not readily applicable since α is unknown in a practical situation. However, we can use the relation (17) to estimate α from the measured time t_0. While this requires us to know the also unknown λ_0, we can estimate *both* parameters together if we have *another data point*! By looking at the times t_1 and t_2 for the last *two* difficulty-retargeting intervals, we can do that. Here, t_2 is the time just measured for the previous M blocks (which was called t_0 before), and t_1 is the time for the M blocks preceding those. (I. e., the t_2 of the last difficulty retargeting.) These times are readily available in practice, as is the change in difficulty that happened between both intervals. This is the value calculated in the last difficulty update, and we denote it by δ_0. Except for the change in difficulty (which will be factored into the calculations), the ratio of both times is an indication of the growth rate α. Using (10), the times will approximately be

$$t_1 = \frac{1}{\alpha} \log\left(\frac{\alpha}{\lambda_1}(M-1) + 1\right), \quad t_2 = \frac{1}{\alpha} \log\left(\frac{\alpha}{\lambda_2}(M-1) + 1\right).$$

Here, $\lambda_2 = \lambda_1 \cdot \frac{e^{\alpha t_1}}{\delta_0}$ is the "initial" hash rate at the start of the second measurement interval. It is already increased due to the hash-rate growth, but may be adjusted also according to the last difficulty update. These equations yield further

$$\frac{e^{\alpha t_1}}{\delta_0} = \frac{\lambda_2}{\lambda_1} = \frac{e^{\alpha t_1} - 1}{e^{\alpha t_2} - 1}. \tag{19}$$

Our next goal is to solve (19) for α. Note that this is now possible, since the unknown λ_1 has been eliminated from the equation. Substituting $u = e^{\alpha t_1}$ and $\tau = \frac{t_2}{t_1}$ turns (19) into the fixed-point equation

$$u = \delta_0 \frac{u - 1}{u^\tau - 1} = g(u). \tag{20}$$

Note that g can be continuously extended by defining $g(1) = \frac{\delta_0}{\tau}$. We would like to solve $u = g(u)$ by iterating for a fixed point. We will show below in Theorem 3 that (20) has indeed a unique fixed point and that the iteration with g converges to it under certain conditions. If this is done, we can define the *time-ratio update* as follows:

1. Use timing data from the previous two retargeting intervals to calculate t_1, t_2 and $\tau = \frac{t_2}{t_1}$. The previous difficulty update δ_0 is also known.
2. Solve (20) for u by iterating with g. We can choose $u_0 = \frac{\delta_0}{\tau} = g(1)$ as initial iterate.
3. Use (18) to derive the update as:

$$\alpha = \frac{1}{t_1} \log(u), \quad \delta^+ = \frac{D^+}{D_0} = \frac{e^{\alpha T M} - 1}{1 - e^{-\alpha t_2}}$$

   For $\alpha = 0$ (i. e., approximately constant hash rate), the correct limit of the update is $\delta^+ = \frac{TM}{t_2}$. This matches (11).

This update ensures optimal difficulty under the assumption of exponential hash-rate growth (or decay). It is, however, important to remark that this update strategy is *much more complicated* than those employed in practice so far (in particular (11)). This creates certain difficulties, since all nodes in the peer-to-peer network have to agree absolutely on the calculated difficulty. Thus, it is necessary to implement the proposed algorithm very carefully and to use fixed-point or integer arithmetic for all computational steps. To lift the burden at least a little, we propose the following rule: Each miner that solves a block with a difficulty update is required to provide the calculated value of α as part of the block. This means that the miner is free to solve (19) in whichever way it prefers. For instance, it can use sufficiently precise floating-point arithmetic and its own choice of the number of iteration steps to perform. It could even use a method to solve the equation that is totally different from the approach suggested here. Validating nodes can then simply check that (19) is fulfilled with small enough residual and that the new difficulty is calculated correctly from α. This requires "only" strict network consensus about the evaluation of the exponential function and basic arithmetic operations. The most difficult part, namely solving (19), is not prescribed by the consensus rules.

We will now proceed to state some basic properties of the function g, which is a necessary first step in order to show convergence of the fixed-point iteration (20):

Lemma 8. *Consider the domain $[0, \infty)$ and $\tau > 0$. Clearly $g(0) = \delta_0$.*

If $\tau = 1$, then $g(u) = \delta_0$ for all $u \geq 0$. For the other cases, we have:

1. *If $\tau \in (0, 1)$, then g is strictly increasing and strictly concave. Furthermore,*

$$\lim_{u \to \infty} g(u) = \infty \text{ and } \lim_{u \to \infty} g'(u) = 0.$$

2. *For $\tau > 1$, g is strictly decreasing and strictly convex.* $\lim_{u \to \infty} g(u) = 0$.

Theorem 3. *The fixed-point equation (20) has a unique solution u^* for every $\tau, \delta_0 > 0$.*

If $\tau \leq 1$ or $\delta_0 < 1$, then the fixed-point iteration with g converges linearly towards u^* for arbitrary initial value $u_0 \geq 0$.

Proof. If $\tau = 1$, then the iteration converges in a single step to $u^* = \delta_0$. We use the properties stated in Lemma 8. The existence of a unique fixed point follows from these properties.

To show convergence for the case of $\tau \in (0, 1)$, consider an iterate $u < u^*$. In this case, $u < g(u) < u^*$. Consequently, the iteration sequence is strictly increasing and bounded. This means that it converges, and the limit must necessarily be a fixed point of g. By uniqueness of u^*, the limit must be u^*. A similar argument can be applied for iterates $u > u^*$. We know that the convergence must be at least linear since $|g'(u^*)| < 1$, which means that g must be a contraction in some neighbourhood of u^*. This neighbourhood is reached in a finite number of steps.

Finally, assume $\tau > 1$ and $\delta_0 < 1$. In this case, g is a contraction on $[0, \infty)$, since convexity implies

$$\left|g'(u)\right| \leq \left|g'(0)\right| = \delta_0 < 1$$

for arbitrary $u \geq 0$. □ □

Unfortunately, it is not guaranteed that the fixed-point iteration (20) converges for *all* potential values of τ and δ_0. If the solution u^* turns out to be a repulsive fixed point, one can fall back to interval bisection or some other method of solving (20). This is another reason why we propose to leave the solution method out of the consensus rules. We believe, however, that this will be rarely necessary in practice since the parameter values required to make u^* repulsive are already quite "extreme". See also Table 2.

6 Worst-Case Behaviour of Bitcoin's Algorithm

To get some further understanding about the behaviour of Bitcoin's difficulty-retargeting algorithm (see Section 4), we want to discuss how it behaves in worst-case scenarios. This is useful because it makes clear how much a malicious actor or other "external event" (e. g., hardware failure or bankruptcy of a miner) can disrupt the average block times. For a system like Namecoin, this is particularly interesting because this directly influences name expiration times. The time scales considered will be multiple retargeting intervals. On shorter scales (i. e., a couple of blocks) no retargeting takes place and thus the expected times can be directly derived from the mining model developed in Section 3. We acknowledge that it would be very interesting to do the analysis below particularly also for the time-ratio update proposed by us, but for now we want to defer this analysis to a future work.

For this analysis, we will assume that an attacker has the capability to control the network hash rate $R(t)$ arbitrarily within some bounds $\left[\underline{R}, \overline{R}\right]$, $\underline{R} > 0$. This corresponds to the situation that the network has some "base hash rate" R_0 provided by honest miners and the attacker has additional hash rate R_a that it can switch on or off according to its purposes. This yields $\underline{R} = R_0$ and $\overline{R} = R_0 + R_a$. Note that we will assume that the hash rate is constant over each single retargeting interval, since our focus is on the difficulty updates. They are not influenced by the precise time-dependence of the hash rate within a single interval. We consider n retargeting intervals $I_1, \ldots, I_n$. The (constant) hash rates over these intervals are denoted by $r_1, \ldots, r_n \in \left[\underline{R}, \overline{R}\right]$. Furthermore, we denote the hash rate of the retargeting interval prior to I_1 by r_0. This quantity is important since it characterises the "initial situation" and influences the difficulty during I_1. We assume that the attacker can also control r_0 in the same bounds. Under these assumptions, the expected time J_k for the M blocks of I_k, $k = 1, \ldots, n$, is

$$J_k = \frac{M}{\lambda_k} = M\frac{D_k}{r_k} = MT\frac{r_{k-1}}{r_k}$$

according to (8) and (11). The average block time over all n retargeting intervals, which is the quantity of interest, is thus

$$J(r) = J(r_0, \ldots, r_n) = \frac{J_1 + J_2 + \cdots + J_n}{nM} = \frac{T}{n}\sum_{k=1}^{n}\frac{r_{k-1}}{r_k}.$$

Since J is continuous and the set of admissible r is compact, this finite-dimensional optimisation problem has both a maximum and a minimum. These are the most extreme block times that can be reached by the attacker, and our goal below is to calculate them. As a first step, note that r_k, $k = 1, \ldots, n-1$, influences precisely two terms in this sum: J_k and J_{k+1}. If we consider all other variables as fixed, then r_k has to maximise or minimise the auxiliary function

$$g_k(r_k) = \frac{r_{k-1}}{r_k} + \frac{r_k}{r_{k+1}}.$$

The following result, giving the core properties that we will use about this function, is easy to see using elementary calculus:

Lemma 9. *g_k is strictly convex. Over $\left[\underline{R}, \overline{R}\right]$, g_k achieves its maximum (only) on the boundary of the interval and has a unique minimum at*

$$r_k^* = \sqrt{r_{k-1} \cdot r_{k+1}} \in \left[\underline{R}, \overline{R}\right]. \tag{21}$$

If $r_{k-1} = r_{k+1}$, then clearly $r_k^* = r_{k-1} = r_{k+1}$. Otherwise, r_k^* is strictly between *r_{k-1} and r_{k+1}.*

6.1 Minimal Block Time

If the goal of the attacker is to *minimise* $J(r)$, then it is easy to see that $r_0 = \underline{R}$ and $r_n = \overline{R}$ should be chosen. Furthermore, all other values must satisfy the optimality condition (21) stated in Lemma 9.

Theorem 4. *The minimum r^* of J is achieved at*

$$r_k^* = \underline{R} \cdot e^{k\Delta} = \underline{R}^{1-\frac{k}{n}} \cdot \overline{R}^{\frac{k}{n}}, \quad \Delta = \frac{\log \overline{R} - \log \underline{R}}{n}. \tag{22}$$

Proof. We already know that J must achieve a minimum at some r. Also, $r_0 = \underline{R}$ and $r_n = \overline{R}$, which matches (22), are clear. For all other r_k, we know that (21) holds. Rewriting this equation yields

$$\sqrt{\frac{r_k}{r_{k+1}}} = \sqrt{\frac{r_{k-1}}{r_k}} \quad \Leftrightarrow \quad \log r_{k+1} - \log r_k = \log r_k - \log r_{k-1}.$$

Hence, all these differences must be equal to Δ. This shows (22). □ □

Thus we have shown that the average block time becomes minimal if the hash rate is raised from $\underline{R}$ to $\overline{R}$, or, in other words, the attacker turns R_a on, *exponentially*. It is interesting to note that this result shows that exponential hash-rate growth, as it appears in practice, is the *absolutely worst case* (regarding the block rate) possible for Bitcoin's update algorithm! The actual block rate for this case was already derived above, so that the minimum value of J is (approximately) given by (16).

6.2 Maximal Block Time

To achieve, on the other hand, a *maximal delay* of blocks (corresponding to a maximiser of J), clearly $r_0 = \overline{R}$ and $r_n = \underline{R}$ should be chosen. Furthermore, it will turn out that the optimal sequence of r's is a "bang-bang" alternation between $\underline{R}$ and $\overline{R}$. While increasing the hash rate yields, of course, a relatively fast retargeting interval at first, it also causes the next difficulty to be high. Decreasing the hash rate afterwards, however, gives "doubly slow" blocks over the next retargeting interval. On average, this works out to increase the total time for all intervals taken together.

Theorem 5. *J achieves its global maximum at*

$$r_k^* = \begin{cases} \overline{R} & k \text{ even and } k \neq n \\ \underline{R} & k \text{ odd or } k = n \end{cases} \tag{23}$$

for $k = 0, \ldots, n$. For the value, it holds that

$$\lim_{n\to\infty} J\left(r^*\right) = T \cdot \frac{\underline{R}^2 + \overline{R}^2}{2\underline{R}\overline{R}}.$$

Proof. Let r be a maximiser. Lemma 9 implies that r must be "bang-bang", i. e., $r_k \in \left\{\underline{R}, \overline{R}\right\}$ for all $k = 0, \ldots, n$. Furthermore, $r_0 = \overline{R}$ and $r_n = \underline{R}$. Assume that we have a pair of neighbouring intervals with the same rate, e. g., $r_m = r_{m-1}$ for some $m = 1, \ldots, n-1$. In this case, we can define another point r' that must also be a maximiser by "shifting all rates to the left" and thus "moving" the constant pair to the end:

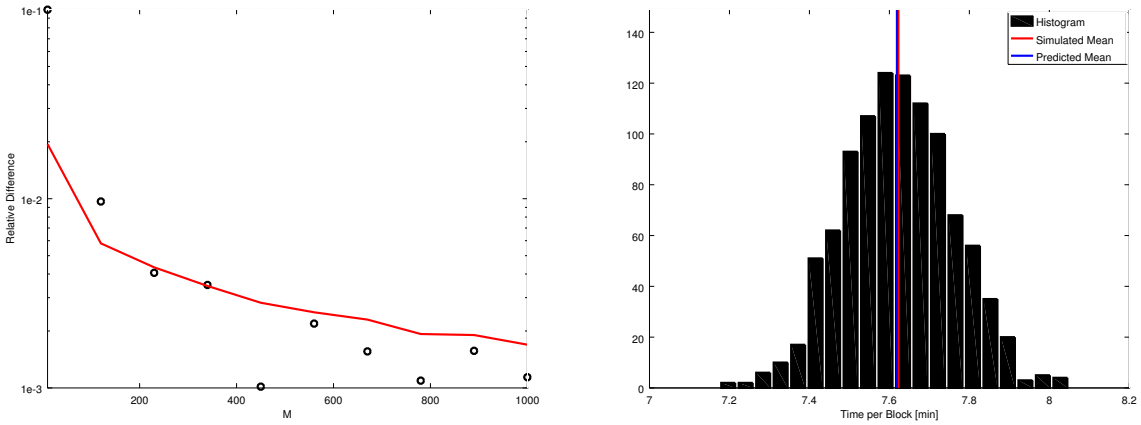
$$r'_k = \begin{cases} r_k & k = 0, \ldots, m-1 \\ r_{k+1} & k = m, \ldots, n-2 \\ r_n & k = n-1, n \end{cases}$$

Then, the cost stays indeed the same, since

$$\begin{aligned} J(r') &= \frac{T}{n} \sum_{k=1}^{n} \frac{r'_{k-1}}{r'_k} = \frac{T}{n} \left(\sum_{k=1}^{m-1} \frac{r_{k-1}}{r_k} + \frac{r_{m-1}}{r_{m+1}} + \sum_{k=m+1}^{n-2} \frac{r_k}{r_{k+1}} + \frac{r_{n-1}}{r_n} + \frac{r_n}{r_n} \right) \\ &= \frac{T}{n} \left(\sum_{k=1}^{m-1} \frac{r_{k-1}}{r_k} + \frac{r_{m-1}}{r_m} + \frac{r_m}{r_{m+1}} + \sum_{k=m+2}^{n-1} \frac{r_{k-1}}{r_k} + \frac{r_{n-1}}{r_n} \right) = \frac{T}{n} \sum_{k=1}^{n} \frac{r_{k-1}}{r_k} = J(r). \end{aligned}$$

Hence, we know that there exists a maximum that alternates between $\overline{R}$ and $\underline{R}$ at each new interval and only (potentially) has $r_m = r_{m+1} = \cdots = r_n$ at the very end. However, it is impossible to have $r_{n-2} = r_{n-1} = r_n = \overline{R}$ since in that case, $r_{n-1} = \underline{R}$ would yield a strictly larger value of J according to Lemma 9. It follows that at most $r_{n-1} = r_n$ is possible, but $r_{n-2} \neq r_{n-1}$ must hold. This implies the form of the maximiser given in (23). The actual value is then easy to derive, which further gives the asserted limit. □ □

Note that the maximiser (23) is not necessarily unique. The proof of Theorem 5 shows, though, that *all* maximisers have the same structure: Alternation between $\overline{R}$ and $\underline{R}$, with at most a single pair of neighbouring intervals that have the same hash rate (if n is even).



(a) (b)

Figure 3: Convergence of (10) to the simulation result. The daily hash-rate growth (expressed by α) is 5%. Left: Difference between (10) and the mean of 5,000 simulated times (black circles) and 5% quantile of $|S_M - \mathbb{E}(S_M)|$ of the simulation (red), both relative. Right: Histogram for 1,000 simulations with $M = 2{,}016$ together with their mean (red) and the prediction of (10) (blue).

7 Simulation Results

Finally, we also want to present simulation results that validate the mining model developed (in particular, (10) and (16)) and that show how our method proposed in Section 5 improves over Bitcoin's difficulty-update formula. We also failed to provide a worst-case analysis for this method similar to Section 6, but will, at least, simulate how it behaves under selected "extreme" scenarios in Subsection 7.2. In Subsection 7.3, we will compare the actual block times observed in the real Bitcoin and Namecoin networks and try to estimate how they would have evolved if our proposed method had been used to control difficulty. This will show that it stabilises block rates indeed much better, even in practice and not just in theory. All calculations have been performed with GNU Octave [9]. Where not stated to the contrary, we used the configuration parameters of the real Bitcoin network: $M = 2{,}016$ and $T = 10\,\text{min}$. The initial hash rate and difficulty were chosen to yield the "expected" block rate $\lambda_0 = \frac{1}{T}$ at $t = 0$.

7.1 Simulating the Mining Process

Let us start by considering how to simulate the mining process itself, given the hash rate and difficulty over time. This knowledge also enables us to calculate $\lambda(t)$ for each $t \geq 0$. At any given time point, the time it takes to find the *next* block is *exponentially distributed* (as can be seen from (1)). We will, furthermore, assume that the hash rate is constant at least over the time it takes to find a single block. In this case, $m(t) = \lambda t$ and the time for the next block can be constructed as a random number following an exponential distribution with the current λ. The generation of such random numbers is already a feature of GNU Octave (using the "Ziggurat" method [12]), and we make use of it. With this in hand, we can simply generate one block time after another, adapting λ at each step to the current time, to simulate the real mining process and generate a realisation of the total time S_M for M blocks.

Our goal in this first subsection is to show that the approximation made in deriving (10) is valid. Indeed, Figure 3a shows that (10) becomes a very good approximation as M grows: The error made falls well below the statistical fluctuations of the simulations themselves even for values of M that are still smaller than in practice. This is also evident from Figure 3b, which shows that the simulated times for M blocks vary much more than the difference between simulation and prediction (the vertical lines). Finally, the match between simulation and prediction is also shown in Table 1 for various assumptions on the daily growth rate. We can see from the table that (10) is accurate (particularly compared to the inherent fluctuations) also in all of these scenarios.

Growth	Simulated	Predicted	Rel. Error	5% Quantile
−2%	11.753 min	11.747 min	0.049%	0.169%
0%	10.002 min	10.000 min	0.018%	0.149%
1%	9.356 min	9.358 min	0.020%	0.130%
2%	8.833 min	8.822 min	0.120%	0.116%
5%	7.624 min	7.619 min	0.071%	0.109%
10%	6.353 min	6.351 min	0.035%	0.094%

Table 1: Comparison between (10) and the mean calculated from 1,000 simulation results for $M = 2{,}016$ and various assumptions for the daily growth rate. The second and third columns show the simulated and predicated average time per block, and the fourth and fifth columns show the relative error between these as well as the relative 5% quantile of $|S_M - \mathbb{E}(S_M)|$.

7.2 Simulation of the Difficulty Updates

Next, we will include difficulty updates into our simulations. In particular, we will compare how the average block time behaves when using Bitcoin's difficulty update (11) and the proposed time-ratio update (see Section 5). The quantity of interest is always the average block time (it should be as close to $T = 10\,\text{min}$ as possible) over a longer period of time. We compare it over 25 retargeting intervals as well as 100. The former quantity corresponds to roughly one year of real time. We think that this is a good estimate for a practical name expiration period. The latter gives more perspective with respect to the behaviour in the limit of long time spans, as it reduces artefacts introduced during the initial intervals.

These simulations will not only be done for the "standard case" of strictly exponential growth, but also cover possible attack scenarios. Motivated by Theorem 5, we are particularly interested what happens when the hash rate alternates between two extremes. Thus, we use the following three parameters to characterise our scenarios:

$\underline{\alpha}$ Minimal growth of the hash rate, assumed to occur on every second retargeting interval.

$\overline{\alpha}$ Maximal growth of the hash rate, occurring on all other intervals.

R_a Hash rate that an attacker turns on and off alternatingly. It is assumed to be turned on when $\overline{\alpha}$ is active and turned off when growth is described by $\underline{\alpha}$. The initial non-attack hash rate is normalised to 1. Thus, the total hash rate is $R(t) = 1 \cdot e^{\alpha t} + R_a$, where $e^{\alpha t}$ represents the accumulated growth over time for some "effective α".

If $R_a = 0$ and $\underline{\alpha} = \overline{\alpha}$, we have a non-attack scenario with fixed hash-rate growth. If $R_a = 0$ but $\underline{\alpha} < \overline{\alpha}$, we see how fluctuations in the hash-rate growth affect the resulting block times. For $R_a \neq 0$, we get the scenario proven to be worst-case for Bitcoin's difficulty update. While we have not provided a proof for the time-ratio update, it is not too far fetched to assume that also in this case, the alternation scenario is bad. We have also tried out variations (e. g., turning R_a on and off for longer than just a single interval) but not found results that are worse.

Table 2 shows some of the resulting block times for selected scenarios. For each scenario, we include both a simulation where S_M is calculated deterministically from (10) and one where every block is simulated as before in Subsection 7.1. The difference between both is minor, though, as the large number of individual blocks making up one or even multiple retargeting intervals already removes a lot of statistical fluctuations. The last column shows whether the fixed-point iteration (20) fails to converge for the given scenario. Where this happens, we fall back to interval bisection to solve the equation. Note, though, that this happens only under extreme conditions. The values shown are the relative differences between the actual time per block and the expected one of $T = 10\,\text{min}$. Negative values mean that the actual blocks are too fast. In the upper part of the table, we have scenarios of stable exponential growth (no kind of "attack") with various growth rates. This is what happens mostly in practice, and it is easy to see that the Bitcoin update results in vastly too fast blocks. This was discussed already in Subsection 4.2, and the simulated differences match the predictions of (16). The time-ratio update, on the other hand, is able to stabilise the block times towards T almost perfectly. This is especially true over the longer observation period where the effect of the initial λ_0 is not so strong any more. The lower part of the table shows "attack" scenarios. Even for them, the time-ratio update does not perform too badly. It does worse than the Bitcoin update for the cases with no intrinsic growth but R_a turned on and off alternatingly. Note, though, that even the already quite extreme case with $R_a = 50\%$ distorts the block times less than a simple growth of 2% per day for Bitcoin's update. Furthermore, when an attacker controls a significant portion of the network's hash rate, there are by far worse things to do than messing with the difficulty update (e. g., the dreaded "51% attack"). Finally, the last entry of the table combines an attack with large hash rate and exponential growth. Here, the time-ratio update is again quite good; this is due to the fact that the exponential growth makes the attacker's share of the total hash rate dwindle over time.

$\underline{\alpha}$	$\overline{\alpha}$	R_a	Deterministic?	Over 25 Intervals Bitcoin	Over 25 Intervals Time Ratio	Over 100 Intervals Bitcoin	Over 100 Intervals Time Ratio	FP Failed?
−2%	−2%	0%	×	52.9%	2.5%	54.5%	0.6%	×
−2%	−2%	0%		53.2%	2.7%	54.8%	0.8%	
0%	0%	0%	×	0.0%	0.0%	0.0%	0.0%	
0%	0%	0%		0.0%	0.0%	0.0%	0.1%	
2%	2%	0%	×	−19.6%	−1.3%	−19.8%	−0.3%	
2%	2%	0%		−19.6%	−1.0%	−19.8%	−0.2%	
5%	5%	0%	×	−35.2%	−2.5%	−35.5%	−0.6%	
5%	5%	0%		−35.1%	−2.5%	−35.5%	−0.6%	
10%	10%	0%	×	−48.8%	−3.5%	−49.1%	−0.9%	
10%	10%	0%		−48.8%	−3.5%	−49.1%	−0.9%	
−2%	5%	0%	×	−13.8%	0.1%	−14.9%	0.0%	
−2%	5%	0%		−13.6%	0.1%	−14.8%	0.1%	
0%	0%	10%	×	0.4%	1.3%	0.4%	0.8%	
0%	0%	10%		0.6%	2.1%	0.4%	1.0%	
0%	0%	30%	×	3.3%	7.9%	3.2%	6.6%	
0%	0%	30%		3.5%	7.7%	3.2%	6.5%	×
0%	0%	50%	×	8.0%	18.0%	7.8%	15.9%	×
0%	0%	50%		8.1%	19.2%	7.8%	16.8%	×
0%	0%	70%	×	13.8%	30.2%	13.7%	27.2%	×
0%	0%	70%		14.3%	30.3%	14.0%	27.3%	×
0%	0%	100%	×	24.0%	51.3%	24.0%	46.5%	×
0%	0%	100%		25.1%	52.3%	24.3%	45.2%	×
1%	3%	100%	×	−15.8%	6.0%	−18.8%	1.5%	×
1%	3%	100%		−15.3%	6.1%	−19.0%	1.4%	×

Table 2: Comparison between resulting block times for Bitcoin's difficulty update and the proposed time-ratio update. Various attack and non-attack scenarios are included. The main columns show the *relative difference* between the actual time and the desired $T = 10\,\text{min}$ per block.

7.3 Rewriting History

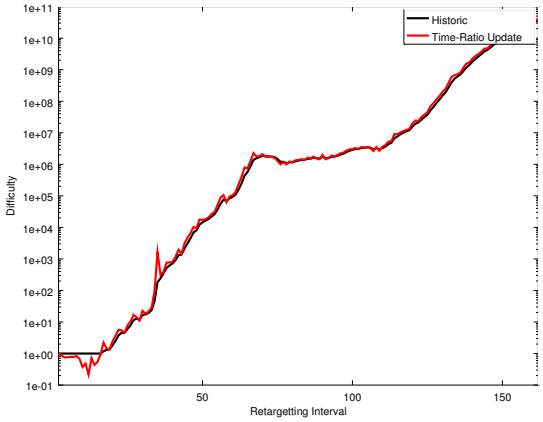
Of course, the fundamental principle of a Nakamoto blockchain is to make rewriting history *extremely hard*. As a final demonstration, we will try it nevertheless (in a different way): We use historic data about block times and difficulty values from the Bitcoin and Namecoin blockchains as input into the time-ratio update. This gives us the most realistic test of our proposed method that is possible. To do so, we assume that the hash rate at any given moment in time is given from the outside and the same even if we switch out the difficulty-retargeting algorithm. The block times, on the other hand, change according to the difficulty. Consequently, whenever the time-ratio update prescribes a difficulty different from the historic one, we can transform the measured block times by the ratio of both difficulty values. This yields block times sampled with respect to the changed difficulty. There are, however, a few caveats which we want to mention before presenting the simulation results:

- The real Bitcoin and Namecoin systems have special rules to control the difficulty besides the update described in Section 4. In particular, difficulty is capped by a lower floor and the difficulty-update ratio δ is clipped, when necessary, to be in the interval $\left[\frac{1}{4}, 4\right]$. The former constraint was active at the very beginning of the Bitcoin blockchain, as can be seen on the left of Figure 4a. The latter dampens excessive fluctuations in the hash rate. None of them was active any more after the systems matured sufficiently. For simplicity, such constraints are left out of our modelling. It would be easy to add them in practice, though, if that is deemed beneficial. Thus, the initial parts of the simulation results should be taken with a grain of salt. The later retargeting intervals are much more important to assess the difficulty updates for the context of a matured system.
- We will show how the expiration time of a name differs from its expected value, since that is a quantity of actual interest for a system like Namecoin. For simplicity, we assume that the expiration period is 18 retargeting intervals, or 36,288 blocks. For Namecoin, it is actually 36,000 now and was lower in the very beginning. While Bitcoin does not support names at all in reality, we will nevertheless consider the real-time duration of this expiration period for the Bitcoin network's historical data as well.
- By changing the difficulty, we also shift the times of blocks around in terms of real time. We do not account for this change in the hash-rate assumptions. Instead, our method of calculating the new block times assumes that the hash rate is the same as it was historically *per block* and not per real time.

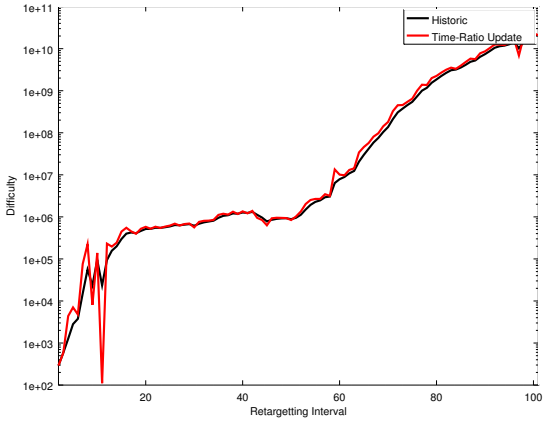
For our simulations, we consider full retargeting intervals and average the block times over them. The very first interval is excluded, since no correct timing is known for it (as it is not clear when mining the initial block *started*). For Bitcoin, we show the 2nd–162nd retargeting intervals (Jan-27-2009 to Oct-23-2014). For Namecoin, the intervals 2–101 (May-11-2011 to Nov-01-2014) are used. Figure 4 compares the actual difficulty and block times (in black) to the ones that result from using the time-ratio update (red). The left column is for the Bitcoin network, while the right is for Namecoin. Both networks show qualitatively the same behaviour. The top row depicts the difficulty development itself. It is clearly visible that it behaves exponentially. The time-ratio update yields a higher difficulty, particularly where growth happens. This corrects the systematically too fast blocks that happen due to Bitcoin's update mechanism. The middle row shows how the average block time behaves in comparison to the desired one (horizontal blue line). It can be seen that it fluctuates wildly for both difficulty-update methods. While Bitcoin's update is seemingly more robust with respect to fluctuations, also the time-ratio update is not too bad. (The wild swings that happen early are a result of very large changes in the hash rate. One can see that also the historic block times are extreme during these times. Additional rules like the ones mentioned above could help to guard against such situations.) Note, though, that the time-ratio update produces fluctuations centred around the desired value, while the Bitcoin update does not. The bottom row, finally, shows the expiration period of names in real time (relative to the desired one). The value corresponds to a name registration at the start of some retargeting interval. Consequently, fewer intervals are included on the x-axis as in the other graphs. Names registered after these intervals are not yet expired in the data set and thus no expiration time can be given for them. The strange "block" between intervals 20 and 40 in Figure 4e is produced solely because of the single retargeting interval with extremely high block time. Figure 4e and Figure 4f make it particularly clear that the time-ratio update is superior to the current method if stability of the expiration period is a goal: Except for effects produced by the initially extreme swings in hash rate, the time-ratio method manages to control the expiration period almost precisely to the desired value. Bitcoin's difficulty-retargeting method does not.

8 Conclusion

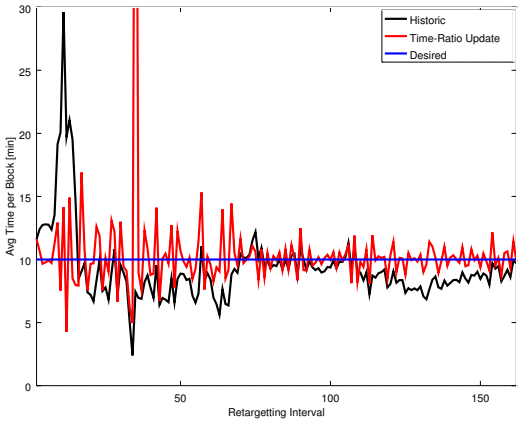
We have developed a model that allows to predict average block times in Bitcoin mining. It is possible to account for the effect of changes in the network hash rate over time, and to predict how the difficulty-update mechanism of Bitcoin reacts. Unfortunately, this currently applied method yields too fast blocks while the hash rate is exponentially rising. To improve the stability of block times, we proposed an alternative difficulty control that is



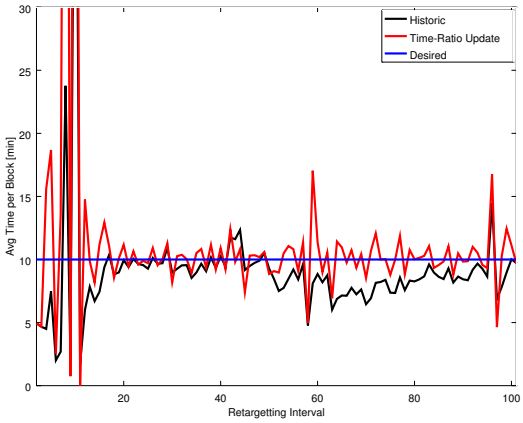
(a) Bitcoin, Difficulty



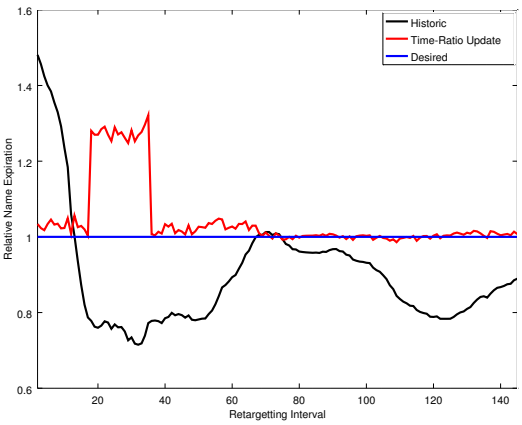
(b) Namecoin, Difficulty



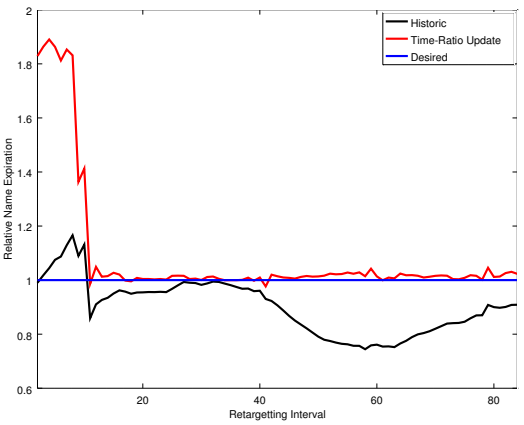
(c) Bitcoin, Average Block Time



(d) Namecoin, Average Block Time



(e) Bitcoin, Name Expiration



(f) Namecoin, Name Expiration

Figure 4: Comparison of various quantities for the historic Bitcoin (left) and Namecoin (right) blockchains. Black is the actually observed value, while red is the value that results from the time-ratio update being applied instead. The blue line denotes the desired value based on $T = 10\,\text{min}$ for the two bottom rows.

designed to work "perfectly" not just for constant hash rate but also if the hash rate grows exponentially (with a constant but unknown rate). We have seen that even for real-world data (of the historical Bitcoin and Namecoin blockchains), the method leads to significant improvements with respect to our goal.

It is important to note that the focus of this work is really on a *long-term perspective*. Fluctuations of the block time between single retargeting intervals seem to be increased by our proposed difficulty update rather than decreased, but the mean value over longer times is nevertheless much closer to the desired block time. This long-term stability is very useful in practice in certain situations, name expiration in Namecoin being one prime example.

We tried to lay a fundament for further research about this topic. However, there still remain a lot of open topics to revisit in more detail. In particular, we propose the following open questions for future research:

- We assumed that the hash rate is time-dependent, but in a deterministic way. Can our model be extended to allow for random fluctuations in the hash rate, e. g., based on the framework of Cox processes [6]?
- Is it possible to do a rigorous worst-case analysis similar to what we have done in Section 6 also for the proposed time-ratio update?
- Develop an actual implementation of the time-ratio update that is suitable for application in practice. This includes, in particular, an implementation of the necessary calculations ((18) and solving (19)) that must be part of the strict network consensus rules. Some ideas for this have been discussed in Section 5, but still a lot of further thought is required to turn them into a working system. It may also be a good idea to include some rules into the difficulty update such that it is more stable with respect to extreme hash-rate changes during the initial stages of a new system.

Acknowledgements

The author would like to thank the Bitcoin and Namecoin communities for valuable input as well as pointing out that more stable name expiration times are an interesting research question in the first place. This work is supported by the Austrian Science Fund (FWF) and the International Research Training Group IGDK 1754.

A Notation Used in the Models

t	time, continuous on $[0, \infty)$		
$R(t)$	hash rate at time t		
D	mining difficulty		
$\lambda(t)$	expected block frequency		$\lambda(t) = R(t)/D$
M	number of blocks per retargeting interval		
$N(t)$	number of blocks found after time t	(random variable)	
X_i	time between blocks $i-1$ and i	(random variable)	
S_M	time to find all M blocks of a retargeting interval	(random variable)	$S_M = \sum_{i=1}^{M} X_i$
λ_0	initial block frequency		
α	exponential growth rate of R		$R(t) = \lambda_0 e^{\alpha t}$
T	desired time between blocks		
t_0	measured time for the last retargeting interval	(realised value of S_M)	
D_0	initial difficulty		
D^+	difficulty after an update		
δ	relative difficulty update		$\delta = D^+/D_0$
λ^+	block frequency right after a difficulty update		$\lambda^+ = R(t_0)/D^+$
R_0	base hash rate of the network		
R_a	hash rate controlled by an attacker		
$\underline{R}$	lower bound for the hash rate during an attack		$\underline{R} = R_0$
$\overline{R}$	upper bound for the hash rate during an attack		$\overline{R} = R_0 + R_a$

References

[1] Namecoin. `http://www.namecoin.org/`

[2] Antonopoulos, A.M.: Mastering Bitcoin: Unlocking Digital Cryptocurrencies. O'Reilly Media (2014)

[3] Back, A.: A partial hash collision based postage scheme. `http://www.hashcash.org/papers/announce.txt`

[4] Bahack, L.: Theoretical Bitcoin Attacks with less than Half of the Computational Power. arXiv:1312.7013 (2013). `http://arxiv.org/abs/1312.7013`

[5] Chaum, D.: Blind signatures for untraceable payments. In: Advances in Cryptology Proceedings, vol. 82, pp. 199–203 (1983)

[6] Cox, D.R.: Some Statistical Methods Connected with Series of Events. Journal of the Royal Statistical Society, Series B **17**(2), 129–157 (1955)

[7] Decker, C., Wattenhofer, R.: Information Propagation in the Bitcoin Network. In: Proceedings of the 13-th IEEE International Conference on Peer-to-Peer Computing (2013)

[8] NIST Digital Library of Mathematical Functions. `http://dlmf.nist.gov/`, Release 1.0.9 of 2014-08-29. Online companion to [15]

[9] Eaton, J.W., Bateman, D., Hauberg, S.: GNU Octave version 3.0.1 manual: a high-level interactive language for numerical computations. CreateSpace Independent Publishing Platform (2009). URL `https://www.gnu.org/software/octave/doc/interpreter`. ISBN 1441413006

[10] Evans, L.C., Gariepy, R.F.: Measure Theory and Fine Properties of Functions. Studies in Advanced Mathematics. CRC Press (1992)

[11] Ittay Eyal and Emin Gün Sirer: Majority is not Enough: Bitcoin Mining is Vulnerable. In: 18th International Conference on Financial Cryptography and Data Security. Barbados (2014)

[12] Marsaglia, G., Tsang, W.W.: The Ziggurat Method for Generating Random Variables. Journal of Statistical Software **5** (2000)

[13] Moore, G.E.: Cramming more components onto integrated circuits. Electronics Magazine (1965)

[14] Nakamoto, S.: Bitcoin: A Peer-to-Peer Electronic Cash System. `https://bitcoin.org/bitcoin.pdf`

[15] Olver, F.W.J., Lozier, D.W., Boisvert, R.F., Clark, C.W. (eds.): NIST Handbook of Mathematical Functions. Cambridge University Press, New York, NY (2010). Print companion to [8]

[16] Rosenfeld, M.: Analysis of Hashrate-Based Double Spending. arXiv:1402.2009 (2014). `http://arxiv.org/abs/1402.2009`

[17] Ross, S.M.: Simulation, fifth edn. Academic Press (2013)

[18] Shomer, A.: On the Phase Space of Block-Hiding Strategies. Cryptology ePrint Archive, Report 2014/139 (2014). `http://eprint.iacr.org/`

[19] Swartz, A.: Squaring the Triangle: Secure, Decentralized, Human-Readable Names. January 6th, 2011

[20] Wilcox-O'Hearn, Z.: Names: Decentralized, Secure, Human-Meaningful: Choose Two