Difficulty Control for Blockchain Systems

Dmitry Meshkov[a], Alexander Chepurnoy[a]

[a] *IOHK Research*

Abstract

TODO

Keywords: Blockchain, Decentralized consensus, Peer-to-peer networks, Proof-of-Work

1. Introduction

Blockchain systems have attracted a growing amount of interest from various communities after publication of Bitcoin whitepaper [1] in 2008. Bitcoin security relies on the distributed protocol that maintains the blockchain, called mining, in which network nodes tries to solve computational puzzle. Other blockchain systems may relies on different computational puzzles [**?**] or even on virtual mining [**?**], while all of them use some algorithm that changes puzzle difficulty dynamically. This algorithm for retargeting the difficulty is required to make blockchain system predictable and fix latency between blocks.

Fixed latency between blocks is important for several reasons. Too often blocks leads to situation, when for a lot of miners block propagation time become bigger, then latency between blocks. This leads to significant increasing of number of blockchain forks that complicates the consensus[2] and reduce effective hash rate in blockchain system. On the other hand, increasing of the latency between blocks leads to decreasing of the network throughput[3] and may be critical for high-loaded blockchain systems like bitcoin, where blocks are already 70% full today[4]. Increasing latency by 50% in bitcoin network will

Email address: dmitry.meshkov@iohk.io (Dmitry Meshkov)

mean, that some transactions will be never included into blockchain. Moreover this will lead to infinite growth of unconfirmed transactions pool, meaning it is likely that most bitcoin transactions will not even relay, much less confirm.

Most of blockchain systems relies on quite a naive difficulty retargeting algorithm, that assumes, that total computational power, involved in mining process, don't change over time. Using more complicated retargeting algorithms with incorrect assumptions[5] may lead to incorrect time interval between blocks even for simple case of constant hash rate as, for example, observed in Nxt, where observed mean time between blocks is 2 times bigger, then expected in whitepaper[6]. Moreover, too ofter difficulty recalculation leads to wide distribution of time intervals between blocks and makes blockchain system unpredictable[5]. Varying network computational power makes this algorithms inefficient for difficulty recalculation, e.g. continuous growth of computational power leads to decreasing mean latency between blocks and average block time in Bitcoin network is 1.07 times lower, then expected. Noteworthy, that exponential growth of computational power, which is the situation observed in practice in accordance with Moores law[7], is the absolutely worst case (regarding the maximal block rate) possible for Bitcoins difficulty retargeting algorithm[8]. On the other side, target recalculation algorithm should be easy enough and use integer arithmetic for all computational steps, since all nodes in the peer-to-peer network have to agree absolutely on the calculated difficulty.

Original Bitcoin white-paper, states that the security of the system is guaranteed as long as there is no attacker in possession of half or more of the total computational power used to maintain the system [1]. Most of the models used in the literature to discuss double-spending attacks assume that mining difficulty is constant [**?**]. However difficulty is not constant, and can be manipulated by the attacker. The Difficulty Raising Attack, introduced in [9], enables the attacker to discard n-depth block, for any n and any attacker hash power, with probability 1 if he[1] is willing to wait enough time. The fact that there is no way

[1]For simplicity we choose to adapt the male form. This was chosen by flipping a coin.

to determine whether a block have been computed on its declared time or not, have been used as part of other attacks [10, 11]. In secton 2 we introduce new way of attack for blochcking systems, that manipulates difficulty for decreasing effective hash rate, required for block generation.

New researches of difficulty control proposes better functions for difficulty recalculation. For example paper [8] introduces target recalculation function, designed to work perfectly not just for constant hash rate but also if the hash rate grows exponentially (with a constant but unknown rate). Since it's good for situation, observed in practice in Bitcoin network, there are still a lot of open questions for future research. Is it possible to create such a function, suitable for random fluctuations in the hash rate? Is it possible to create such a function, simple enough to use integer arithmetic for all computational steps? Is it possible to create such a function, stable for attackers manipulations?

TODO transition to newxt section

2. Bitcoin Mining

First, let us briefly describe how the mining process works. The original concept is described in section 4 of the Bitcoin whitepaper [1], and disscussed in plenty of papers [8, 12, 13]. We refer to the basic unit of mining work in Bitcoin as a scratch-off puzzle (SOP). Miner generates candidates to the SOP by iterating thought *nonse* and calculating SHA-256 hash of a blocks header. For a block to be valid it must value hash value less than the current *target* that is which is usually expressed in terms of the network difficulty D as $\frac{1}{D}$. Thus, each hash attempt yields a valid block with probability $\frac{1}{D}$ and block frequency is $\frac{R}{D}$ where R is effective network hash rate.

Every M blocks ($M = 2016$ for Bitcoin) difficulty is recalculated as

$$D_{i+1} = D_i \cdot \frac{MT}{S_m} \tag{1}$$

where T is expected time interval between blocks and S_m is actual time, it took to generate M blocks. Real time ($9min20sec$) interval is less, then desired time

interval $T = 10min$ because of continiuos growth of network computational power. Difficulty recalculation interval $M = 2016$ has been chosen in such a way to recalculate difficulty every 2 weeks, while for real network it is a bit smaller. This time intrerval is big enough to see increasing computational power of the network: right after target recalculation block time is close to desired 10 minutes, whereas at the end of *epoch* it's less then 9 minutes (see figure 1).

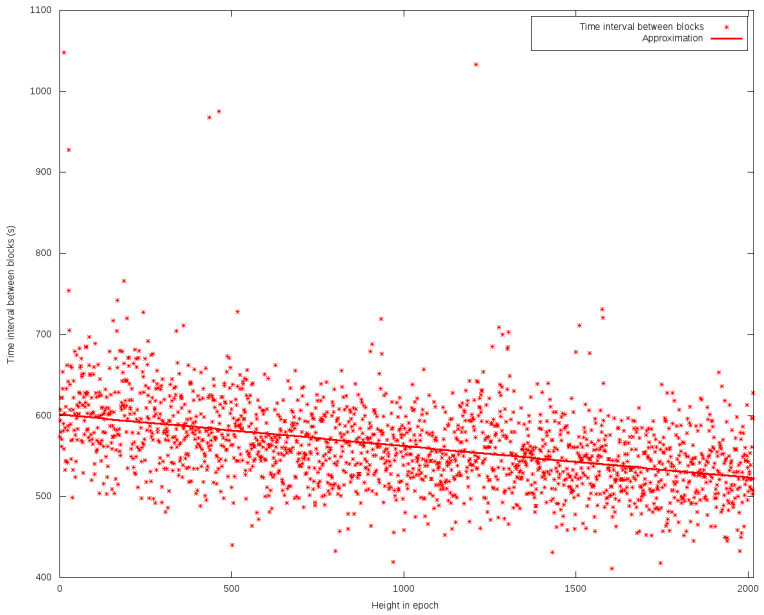

Figure 1: Average block time between difficulty recalculation

TODO transition to next section

3. Fork Switch Attack

On this section we show a new way of manipulating difficulty with profit to an attacker. The main idea is very simple. The attacker may disable his mining, and wait for the next difficulty racalculation. New difficulty will be lower, cause it don't include computational power of the attacker, and he turns on his mining again. After next difficulty recalculation (which will include the attacker computational power) he turns off his mining again, and wait for next difficulty

recalculation. Such a simple algorithm allows him to mine during epochs with low difficulty, and don't mine during epoch with the low one, that will decrease computational resourses expended for block generation. Of course in such a situation the attacker will create less blocks, that in situation, when he mine all the time. However, lows of economy dictate that the cost of computational resourses invested into mining should be around the expected reward, and attackers profit, gained from every block may overbalance his wastes caused by decreasing number of created blocks. Moreover, in a situation with a lot cryptocurency forks he is able to mine other forks in epoches with big difficulty and gain more blocks with more profit from each, that in situation, when he mine single cryptocurrency permanently.

TODO subsection?

Let's calculate attacker profit in circumstances of Bitcoin difficulty recalculation function and constant hash rate (which is natural for this function). Assume the situation when the network has some hash rate R_0 provided by honest miners and the attacker has additional hash rate $R_a = R_0 \cdot p$ that it can turn on or off according to its purposes. Usual miner with such a power will mine all the time with difficulty $D_0 = (R_0 + R_a) \cdot T$ and will mine $\frac{M \cdot R_a}{R_0 + R_a}$ blocks per epoch (M blocks) in average spending $M \cdot R_a \cdot T$ computational power for them. The attacker will start mining when difficulty $D_0 = R_0 \cdot T$ is calculated from honest miners R_0 only. He will mine $\frac{M \cdot R_a}{2(R_0 + R_a)}$ blocks per epoch in average spending $\frac{M \cdot T \cdot R_a \cdot R_0}{2(R_0 + R_a)}$ computational power for them. Consequently usual miner will spend $(R_0 + R_a)T$ computational power per block, whereas attacker will just spend $R_0 T$ computational power per block. The cost of computational resourses invested into mining should be around the expected reward. If B is block reward and C is hash calculation cost, the usual miner profit is

$$(\frac{M \cdot R_a}{R_0 + R_a}) \cdot B - M \cdot R_a \cdot T \cdot C = M \cdot R_a \cdot (\frac{B}{R_0 + R_a} - TC) \tag{2}$$

per epoch. At the same time attackers profit is

$$\frac{M \cdot R_a}{2 \cdot (R_0 + R_a)} \cdot (B - T \cdot R_0 \cdot C) \tag{3}$$

and attack become profitable when

$$\frac{B}{C} < T \cdot (R_0 + 2 \cdot R_a). \tag{4}$$

Note, that if attacker is able to switch between different forks his epoch profit is twice bigger and is bigger then regular miner profit at any B and C.

Remarkable, that under such an attack mean time between blocks will be

$$T_a = \frac{T}{2}(\frac{R_0 + R_a}{R_0} + \frac{R_0}{R_0 + R_a}) = T(1 + \frac{p^2}{2(1+p)}) \tag{5}$$

which is strictly bigger then desired time T. Please notice that we regard p as the ratio between the hash-powers of the attacker and the honest network so it changes from 0 to ∞.

TODO transition to next section

4. An Improved Difficulty Control

The difficulty-update algorithm employed by Bitcoin works as designed: If the hash rate is constant, it yields the desired block rate. However it does not achieve the desired block rate in other situations and vulnerable to attack, described in 3. In this section we are going to propose an alternative difficulty-update algorithm that works better, then Bitcoin's one.

First, let us describe properties of *ideal* difficulty update algorithm:

1. It should be resistant to known types of difficulty-update attacks.
2. It should lead to desired block rate for random fluctuations in the hash rate.
3. It should be simple enough to use integer arithmetic for all computational steps.

Security is the most important feature of blockchain systems and should be regarded with the highest priority. Incorrect block rate is not considered a big problem in the Bitcoin community, but it may be important for more advanced

applications of blockchain systems. Implementation of the *ideal* difficulty update algorithm in subclass of integer programming is desired for different platforms compatibility. This rule is not required, because, as mentioned in [8], it's possible to include non-integer algorithm parameters as part of the block, but it provides another way of difficulty manipulating to an attacker.

In this section we are going to regard difficulty update algorithm, based on well-knowen linear least squares method[14]. In the simpliest case of pair linear regression ($y = kx + b$) it's coefficients may be calculated as follows:

$$\begin{cases} k = \frac{\overline{xy} - \overline{x}\,\overline{y}}{\overline{x^2} - \overline{x}^2} \\ b = \overline{y} - k\overline{x} \end{cases} \tag{6}$$

Note, that for accurate difficulty prediction we should use few last observed difficulties, rather than only one, as implemented in Bitcoin, but it's possible to use this algorithm right after second epoch.

We regard it as the good candidate for difficulty update algorithm, because:

1. It should reduce profit of the attack, described in Section 3. Calculations of the attacker profit are described is Section 5
2. It leads to desired block rate for linear changes in the hash rate. This means, that regarding block rate, linear algorithm is better, then Bitcoin's one, in all cases, except constant hash rate, when they lead to the same result.
3. It is simple enough to use integer arithmetic for all computational steps with high fidelity.

5. Simulations

Finally, we want to present simulation results that show how our method proposed in Section 4 improves over Bitcoins difficulty update algorithm. We'll regard $difficulty$ growth in this section, keeping in mind the fact, that it's closely related with network hash rate, which is usually considered in literature.

All calculations have been performed with open-sourced programs, avaliable at Scorex project GitHub page [15].

TODO Linear?

5.1. Exponential Difficulty

First let us regard exponential difficulty growth, wich is the situation observed in practice in Bitcoin network. As we already mentioned, exponential difficulty growth is the absolutely worst case possible for Bitcoins difficulty retargeting algorithm. For simplicity we regard a situation, when hash rate growth 10% each epoch, more complicated research of exponential difficulty growth can be found in [8]. Figure 2 presents difficulty as the function of epcoh, which is 2016 blocks in Bitcoin.

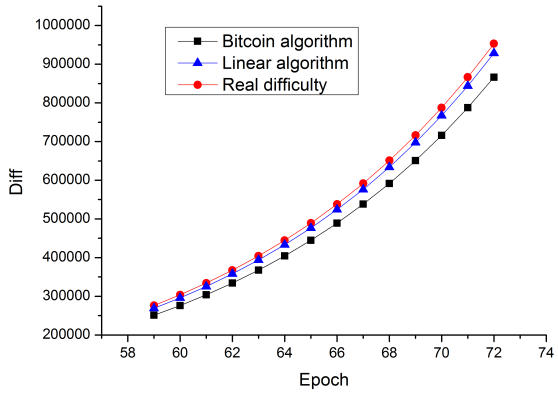

Figure 2: Real difficulty (red) and difficulties calculated from bitcoin (black) and linear (blue) algorithms in situation of exponential hash rate growth

Note, that difficulty, calculated from Bitcoin algorithm is always significantly lower, then the real one. This leads to $9min5sec$ time interval between blocks, which is 10% lower then desired $10min$ interval. Difficulty, calculated from Linear algorithm is also always lower, then the real one, while it's much closer to it. Mean time interval between blocks is $9min45sec$, which is much closer to the desired one.

While difficulty update algorithm, proposed in [8] leads to much better results for exponential difficulty growth with a constant rate, we should note, that our algorithm is much simplier and may be implemented with integer arithmetic only. Moreover, exponential difficulty growth is the simplification of the difficulty growth law, and it may be incorrect to expect it in some situations.

5.2. Fork Switch Attack

Now let's conside a situation, regarded in Section 3: an attacker, containing R_a computational power (for simplicity we suppose $R_a = 0.2 \cdot R_a$ in this section) turn on and turn off his mining to manipulate difficulty and minimize computational power, expended for block mining. Figure 2 represents difficulty as the function of epcoh for this situation.

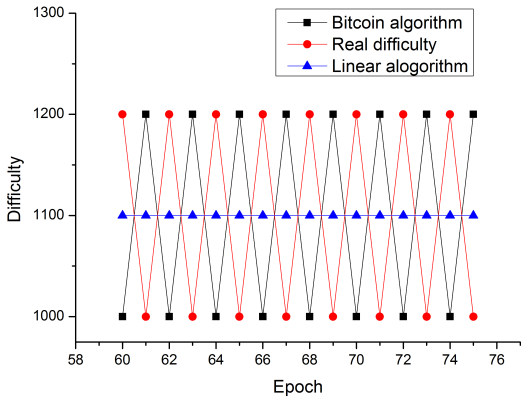

Figure 3: Real difficulty (red) and difficulties calculated from bitcoin (black) and linear (blue) algorithms in situation of attack, described in section **??**sec:attack)

Note, that the difficulty, calculated from the Bitcoin algorithm is always in antiphase with the real one and the attacker spends his computational power only when difficulty is low. Bitcoin difficulty update algorithm leads to $10min10sec$ mean delay between blocks, which is in good correlation with 5. Linear algorithm also leads to enlarged time interval between blocks $10min5sec$, but it's deviation from desired time is 2 times lower. Obviously, attacker profit is also 2

times lower in situation with linear difficuilty update algorithm, which may be regarded as a good result.

Thus, linear difficulty control algorithm, proposed in Section 4 is better, then the Bitcoin one in all situations both in terms of block rate and in terms of attacker profit.

6. Conclusions

In this paper we analyze the new way of attack on the blockchain, based on manipulating mining difficulty. This attack decrease computational power, spended by the attacker for block mining and increase mean time interwal between blocks. It is especially favourable in situation, when the cost of computational resourses invested into mining is around the expected reward and there are enough forks to switch mining between them.

To improve the stability of block times and decrease the attacker profit, we proposed an alternative difficulty update algorithm, based on linear regression. It was found that this algorithm is better then the Bitcoin one both in terms of block rate and in terms of attacker profit, while it's still simple enough to be computed with integer arithmetic only.

References

[1] S. Nakamoto, Bitcoin: A Peer-to-Peer Electronic Cash System (2008). arXiv:43543534534v343453, doi:10.1007/s10838-008-9062-0.
URL http://s.kwma.kr/pdf/Bitcoin/bitcoin.pdf

[2] C. Decker, R. Wattenhofer, Information propagation in the bitcoin network, in: Peer-to-Peer Computing (P2P), 2013 IEEE Thirteenth International Conference on, IEEE, 2013, pp. 1–10.

[3] R. Böhme, T. Okamoto, On scaling decentralized blockchains, 2016.
URL http://fc16.ifca.ai/bitcoin/papers/CDE+16.pdf

[4] B. Armstrong, What happened at the satoshi roundtable (2016).
URL https://medium.com/@barmstrong/what-happened-at-the-satoshi-roundtable-6c11a10d8cdf#.pvoqt832u

[5] andruiman, Nxt forging algorithm: simulating approach.
URL https://www.scribd.com/doc/243341106/Nxt-forging-algorithm-simulating-approach

[6] andruiman, Nxt forging algorithm: simulating approach.
URL http://wiki.nxtcrypto.org/wiki/Whitepaper:Nxt

[7] G. E. Moore, Cramming more components onto integrated circuits, reprinted from electronics, volume 38, number 8, april 19, 1965, pp. 114 ff., IEEE Solid-State Circuits Newsletter 3 (20) (2006) 33–35.

[8] D. Kraft, Difficulty control for blockchain-based consensus systems, Peer-to-Peer Networking and Applications (2015) 1–17.

[9] L. Bahack, Theoretical bitcoin attacks with less than half of the computational power, arXiv preprint arXiv:1312.7013.

[10] The timejacking attack.
URL http://culubas.blogspot.com

[11] ArtForz, The time wrapping attack.
URL https://bitcointalk.org/index.php?topic=43692.msg521772#msg521772

[12] A. Miller, A. Juels, E. Shi, B. Parno, J. Katz, Permacoin: Repurposing bitcoin work for data preservation, in: Security and Privacy (SP), 2014 IEEE Symposium on, IEEE, 2014, pp. 475–490.

[13] I. Eyal, E. G. Sirer, Majority is not enough: Bitcoin mining is vulnerable, in: Financial Cryptography and Data Security, Springer, 2014, pp. 436–454.

[14] C. L. Lawson, R. J. Hanson, Solving least squares problems, Vol. 161, SIAM, 1974.

[15] Scorex blockchain framework.
URL `https://github.com/ScorexProject`