[Short Paper:] Revisiting Difficulty Control for Blockchain Systems

No Author Given

No Institute Given

Abstract. The Bitcoin whitepaper [1] states that the security of the system is guaranteed as long as honest miners control more than half of the current total computational power. The whitepaper assumes a static difficulty, thus it is equally hard to solve a cryptographic proof-of-work puzzle for any given moment of the system history. However, the real Bitcoin network is using an adaptive difficulty adjustment mechanism.
In this paper we introduce and analyze a new kind of attack on a mining difficulty retargeting function used in Bitcoin. A malicious miner is increasing his mining profits from the attack, named coin-hopping attack, and, as a side effect, an average delay between blocks is increasing.
We propose an alternative difficulty adjustment algorithm in order to reduce an incentive to perform coin-hopping, and also to improve stability of inter-block delays. Finally, we evaluate the presented approach and show that the novel approach performs better than the original algorithm of Bitcoin.

1 Introduction

Blockchain systems have attracted significant amount of interest after the Bitcoin whitepaper [1] was published in 2008. Bitcoin security relies on a distributed protocol which maintains a distributed ledger. In the protocol miners are trying to find a partial hash collision in order to generate a valid block by iterating over nonce field values.

Alternative systems may rely on other types of computational puzzles rather than finding a partial hash collision, e.g., [2,3]. Nevertheless, all of them assume some algorithm that changes the difficulty of the puzzle dynamically. An algorithm for difficulty readjustment is required in order to make an open blockchain system working stable in the face of participants joining and leaving the system (resulting in constantly changing available computational power for solving the puzzles), and also to stabilize mean latency between blocks.

The difficulty readjustment algorithm in Bitcoin assumes that the total computational power involved in the mining process does not significantly change from epoch to epoch. In contrast, real networks show that a significant variance in computational power happens over long periods. For example, we show in this paper that due to continuous growth of computational power in the Bitcoin network a mean delay between blocks differs from an expected value by 7%.

Noteworthy, exponential growth of computational power, often observed in practice, is the absolutely worst case (regarding the mean block delay divergence) for the Bitcoin's difficulty readjustment algorithm [4].

In this paper we also consider a new type of miner behavior with regards to difficulty readjustment which provides unfair advantage to the miner, and also makes inter-block delays worse. We call the discovered strategy the coin-hopping attack following the "pool-hopping" term raised in [5]. In this attack, an adversarial miner is switching from mining one coin to another in the beginning of an epoch, then he is switching back in the beginning of next epoch when difficulty becomes lower. We show how adversarial mining profit is increasing for Bitcoin's difficulty readjustment function, and how inter-block delay suffers from the coin-hopping attack.

As a solution for the significant variance in computational power and also in order to reduce incentive of the described coin-hopping strategy, we propose an alternative difficulty readjustment procedure. We show that the proposed solution is better suited for exponential growth of the total mining power. It also reduces profit and negative side-effects of the coin-hopping attack.

1.1 Related Work

In this section we provide an overview of known formal and informal studies with regard to the dynamic nature of the difficulty parameters in Bitcoin. Following the well known paper of Garay et al. [6], generalizing the Bitcoin backbone protocol in a static difficulty setting, a newer paper from the same authors [7] is providing a positive answer on whether basic security properties of the Bitcoin backbone protocol (common prefix, chain quality and chain growth) hold in case of dynamic difficulty, in a cryptographic setting with an arbitrary adversary. Nevertheless, studying concrete attacks against the real protocol is still needed.

The Timejacking attack [8] allows an attacker to first shift the network time at a victims node (which is calculating network time by averaging timestamps it gets regularly from neighbors) and then force the victim node to reject a block with a specially crafted timestamp (other nodes are accepting). The time wrapping attack [9] is exploiting the fact that Bitcoin is using difference in timestamps between last and first block in an epoch, instead of the last block in an epoch and last block in a previous epoch. By using specially crafted timestamps for the last block of each epoch, an attacker can produce more blocks for a time window with more work contributed to his chain. The difficulty raising attack, introduced in [10], allows an attacker to discard n-depth block, for any n, and for any computational power of the attacker, with probability 1 if he is willing to wait long enough.

Another paper [4] is introducing an alternative difficulty readjustment function designed to work better than Bitcoin's not just for almost constant mining power but also when the power is growing exponentially with time. We provide comparison with this function in the paper in one of the following sections.

The paper is organized as follows: in Section 2 we provide a detailed view of Bitcoin's readjustment function. In Section 3 we introduce the coin-hopping

attack, followed by the definition of an improved difficulty readjustment function described in section 4. Section 5 provides results of evaluation of the presented approach.

2 Bitcoin Mining

The concept of Bitcoin mining was introduced in Section 4 of the Bitcoin whitepaper [1], and then discussed in detail in the papers [4],[7]. Bitcoin miner generates a block by iterating over a *nonce* value and calculating the hash of a block with the nonce value included. For a block $\mathcal{B}$ to be valid, a value of a hash function has to be less than the current *target* T, $hash(\mathcal{B}) < T$, where *hash* is an ideal cryptographic hash function. Hardness to find a block could be expressed also via *difficulty* D as $D = \frac{1}{T}$. If output of the *hash* function is μ bits long then the probability to generate a block by doing q requests to the hash function is $\frac{T \cdot q}{2^\mu} = \frac{q}{D \cdot 2^\mu}$. We define miner's *hashrate* R as $R = \frac{q_s}{2^\mu}$, where q_s is number of queries done by miner s per time unit. The probability to generate a block within a time unit is then $\frac{R}{D}$. In our analysis we assume that number of blocks mined over long period of time is proportional to hashrate of a miner. However, there are known strategies to mine a disproportionally high number of blocks, such as [**?**], and the strategies are in correspondence with a general result in [6], which is introducing *chain quality property*. The property sets an upper bound on number of blocks an adversary can generate over a sufficiently long period, however, this number can be higher than the relative hashrate of the adversary; the result got under an assumption of static difficulty. Adversarial manipulations with difficulty can be combined with selfish mining and other strategies to achieve disproportionally high number of blocks, making previous results worse, but this is out of scope of this paper: here, we study manipulations with difficulty in isolation.

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

$$D_{i+1} = D_i \cdot \frac{M \cdot |\Delta|}{S_m} \tag{1}$$

where $|\Delta|$ is the expected time interval between blocks and S_m is the actual time spent to generate M blocks. For the Bitcoin network, the observed time interval of $\approx$*9 minutes 20 seconds* is less than the planned value of $|\Delta| = 10$ *minutes* due to continuous growth of the computational power of the network. Difficulty recalculation interval $M = 2016$ has been chosen to recalculate difficulty every 2 weeks on average. The epoch length is big enough to see the computational power of the network being changing over it: mean delay is close to the planned 10 minutes right after target recalculation, whereas at the end of an epoch it is less than 9 minutes in average.

The next section describes an attack against the recalculation algorithm.

3 Coin-hopping Attack

We consider the following attack involving an adversarial miner $\mathcal{A}$:

– There are at least 2 possible coins (C_1,C_2) $\mathcal{A}$ can contribute to. Without a loss of generality, we assume that each of them provides about the same profitability of the mining activity.
– $\mathcal{A}$ is mining coin C_2 before the beginning of an epoch A. At the beginning of A he is switching to mine coin C_1.
– Without the contribution of miner $\mathcal{A}$ the total mining power of the C_2 network for the epoch decreases.
– For an epoch B right after epoch A, the difficulty of C_2 is to be readjusted to a lower value. So $\mathcal{A}$ starts mining C_2 again with a lower difficulty.

We call this strategy a *coin-hopping attack*.

To calculate the profit the adversarial miner gains from this attack, we use Bitcoins' difficulty recalculation function and assume a constant network hashrate (with respect to the rest of the network, without the adversarial miner). We denote the hashrate of miners not participating in the coin-hopping attack as R_0 in both C_1 and C_2, and we denote the hashrate of the adversarial miner as $R_a = R_0 \cdot p, 0 < p < 1$. Before epoch A the adversary is mining coin C_2, thus the difficulty of the C_2 network is $D_0 = (R_0 + R_a) \cdot |\Delta|$ (see Section 3.1 in [4]). During the epoch A the difficulty of the C_2 network is still D_0, and $\mathcal{A}$ switches to mine coin C_1 at a difficulty $D_1 = R_0 \cdot |\Delta|$ calculated from honest miners hashrate R_0 only. During the epoch B the adversary starts mining of C_2, now at difficulty D_1, while honest miners on chain C_1 continue to mine it with higher difficulty D_0. After that $\mathcal{A}$ continues to switch between chains C_1 and C_2 always mining on the chain with lower difficulty D_1, spending $R_0 \cdot |\Delta|$ computational power per block, whereas honest miners spend $(R_0 + R_a)|\Delta|$ computational power per block.

Every epoch honest miners with hashrate R_0 will generate $\frac{M \cdot R_0}{R_0 + R_a}$, blocks, whereas $\mathcal{A}$ will generate $\frac{M \cdot R_a}{R_0}$ blocks. If $\mathcal{W}$ is block reward, the additional profit of the adversary is calculated as the difference of what he mines based on the lower difficulty in contrast to the difficulty he would mine at without hopping between the coins:

$$\begin{aligned} \mathcal{W} \cdot M \cdot \frac{R_a}{R_0} - \mathcal{W} \cdot M \cdot \frac{R_a}{R_0 + R_a} = \mathcal{W} \cdot M \cdot \frac{R_a^2}{R_0 \cdot (R_a + R_0)} = \\ = \mathcal{W} \cdot M \cdot \frac{R_0^2 \cdot p^2}{R_0 \cdot (R_0 \cdot p + R_0)} = \mathcal{W} \cdot M \cdot \frac{p^2}{1 + p} \end{aligned} \tag{2}$$

Remarkably, under such an attack the mean time between blocks in both chains C_1 and C_2 will be

$$T_a = \frac{T}{2}\left(\frac{R_0 + R_a}{R_0} + \frac{R_0}{R_0 + R_a}\right) = T\left(1 + \frac{p^2}{2(1 + p)}\right) \tag{3}$$

which is bigger than the planned time T.

4 Improved Difficulty Adjustment

The difficulty adjustment algorithm employed by Bitcoin works as designed: if the hash rate of the network is constant, it yields to the desired block rate. However it does not achieve the desired block rate in other situation, and is vulnerable to the attack described in 3. In this section we propose an alternative difficulty adjustment algorithm.

First, we state properties of an ideal difficulty update algorithm:

1. It should be resistant to known types of attacks based on difficulty manipulation.
2. It should lead to an almost constant desired block rate for random fluctuations in the network hashrate.

We propose a difficulty adjustment algorithm based on the well-known linear least squares method[11], we name it *linear algorithm*. In the simplest case of pair linear regression $y = kx + b$, coefficients may be calculated as follows:

$$\begin{cases} k = \frac{\overline{xy} - \bar{x}\bar{y}}{\overline{x^2} - \overline{x}^2} \\ b = \overline{y} - k\overline{x} \end{cases} \tag{4}$$

Difficulty of the $i-th$ epoch D_i can be caclulated from the observed difficulties of previous N epochs $D_{i-1}, ..., D_{i-N}$ as follows:

$$\begin{cases} k = \frac{2\sum_{n=i-N}^{i-1}(D_n \cdot n) - (2i-N-1)\cdot\sum_{n=i-N}^{i-1} D_n}{N((i-N)^2 + (i-1)^2 + (2i-N-1)^2/2)} \\ b = \sum_{n=i-N}^{i-1} D_n/N - k(2i-N-1)/2 \\ D_i = k \cdot i + b \end{cases} \tag{5}$$

Note that for accurate difficulty prediction we use N last observed difficulties, rather than just one, as implemented in Bitcoin, but it is still possible to use this algorithm right after the second epoch of the history.

The next section provides an evaluation of the linear algorithm.

5 Evaluation

In this section we present simulation results that show that the linear algorithm proposed in Section 4 outperforms Bitcoin's difficulty update algorithm in three experiments. The first experiment is about exponential difficulty growth, which is the worst case for the original algorithm, as the previous study [4] shows. The second one is comparing two algorithm on real difficulty data from Bitcoin history. In the third experiment we do comparison of algorithms for a case of the coin-hopping attack. For all the experiments, $N = 4$ (we use data from last 4 epochs to calculate a difficulty value for a new epoch).

5.1 Exponential Difficulty Growth

First, we observe exponential difficulty growth, which occurs in practice in the Bitcoin network. Exponential difficulty growth is the absolutely worst case possible for Bitcoin [4]. In the experiment we consider a situation where network hashrate is increasing by 10% each epoch (more complicated research of exponential difficulty growth can be found in [4]). Figure 1 presents how Bitcoin and linear algorithms perform over epochs.

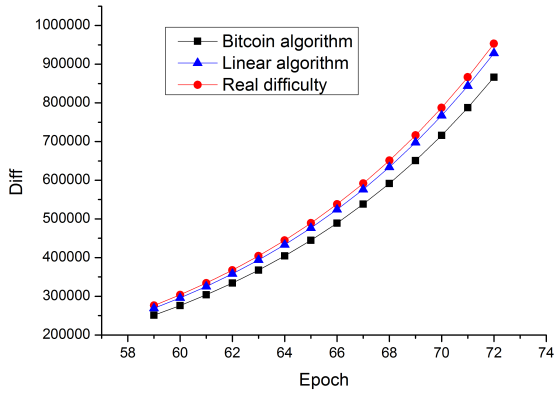

Fig. 1. Real difficulty values (red), values calculated by Bitcoin (black) and linear (blue) algorithms

Note that the difficulty calculated from Bitcoin algorithm is always significantly lower than the real one. This leads to average delay between blocks of about *9 min 5 sec*, which is ≈10% lower than the planned *10 min* value. Difficulties calculated by the linear algorithm are also always lower than the real ones, but closer to them. Mean delay between blocks when linear algorithm is used is about *9 min 45 sec*, which is closer to the planned value. The algorithm currently used in the Bitcoin network has an average error of about 9.1%, while our algorithm has an error of about 1.9%.

While a difficulty readjustment algorithm proposed in [4] leads to better results for exponential difficulty growth with a constant rate, we note that our algorithm is simpler and can be implemented with integer arithmetic only.

5.2 Real World Data

We compare the real Bitcoin network data with difficulty values calculated by the algorithm used in Bitcoin, and we do the same with values calculated by the linear algorithm.

Results show that in average Bitcoin algorithm has an error of about 12.3% while our approach has an error of about 8.4%. Thus our approach performs about 33% better than the approach currently used in the Bitcoin network.

5.3 Coin-hopping Attack

We consider the coin-hopping attack as described in the Section 3, with an attacker possessing 20% of total computational power of network. The attacker repeatedly turns on and then turns off his mining to manipulate difficulty and produce more blocks. Figure 2 represents difficulty over epochs for this scenario.

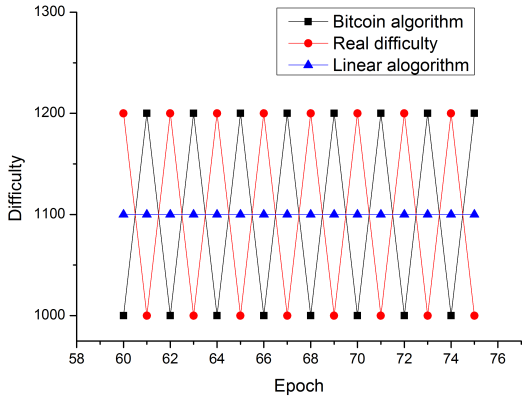

Fig. 2. Real difficulties (red), difficulties calculated from Bitcoin (black) and linear (blue) algorithms in the coin-hopping attack

Note that the difficulty calculated with the Bitcoin algorithm is always in antiphase with the real one. The Bitcoin difficulty update algorithm leads to *10 min 10 sec* mean delay between blocks, which is in good correlation with the Equation 3. The linear algorithm also leads to bigger than planned mean delay between blocks of *10 min 5 sec*, which is about two times lower difference in comparison with the algorithm of Bitcoin. Obviously, the profit of the attacker then is also 2 times lower.

Thus the linear difficulty control algorithm, proposed in Section 4 is better than the one used in Bitcoin for the coin-hopping attack scenario, both in terms of block rate and attacker's profit.

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. 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.
3. A. Biryukov, D. Khovratovich, Equihash: Asymmetric proof-of-work based on the generalized birthday problem, Ledger 2.
4. D. Kraft, Difficulty control for blockchain-based consensus systems, Peer-to-Peer Networking and Applications (2015) 1–17.
5. M. Rosenfeld, Analysis of bitcoin pooled mining reward systems, arXiv preprint arXiv:1112.4980.
6. J. Garay, A. Kiayias, N. Leonardos, The bitcoin backbone protocol: Analysis and applications, in: Advances in Cryptology-EUROCRYPT 2015, Springer, 2015, pp. 281–310.
7. J. A. Garay, A. Kiayias, N. Leonardos, The bitcoin backbone protocol with chains of variable difficulty.
8. The timejacking attack.
   URL `http://culubas.blogspot.com`
9. ArtForz, The time wrapping attack.
   URL `https://bitcointalk.org/index.php?topic=43692.msg521772#msg521772`
10. L. Bahack, Theoretical bitcoin attacks with less than half of the computational power, arXiv preprint arXiv:1312.7013.
11. C. L. Lawson, R. J. Hanson, Solving least squares problems, Vol. 161, SIAM, 1974.