A Proportionate Response

Adam Perlow
adam@zenprotocol.com

Nathan Cook
nathan@zenprotocol.com

March 25, 2017[*]

Summary

We propose a mechanism for Bitcoin holders to protect themselves in the event of a contentious hard fork without threatening the immediate destruction of the existing capital investment of miners. The mechanism allows for Bitcoin holders to send miners a strong, honest signal of approval or disapproval.

Contents

1 Introduction

Contentious hard forks of the Bitcoin blockchain radically change the incentives held by Bitcoin holders, miners and even service providers. Whereas

[*] Version 1.01 (revised 26 March, 2017).

these incentives currently all respect a common interest in the value of bitcoins themselves, even the strong danger of a contentious hard fork can destroy this common interest, motivating economic and technical attacks by one faction against another.

Should Bitcoin holders or service providers come into conflict with miners, the latter may interfere with the former by creating bad blocks—bad, that is, from the point of view of Bitcoin holders. The only recourse holders have against miners is the “nuclear option”—a change of the Proof of Work algorithm (currently based on SHA-256). The threat of such an action, which would destroy substantial expected value for existing miners, together with the twin threat of malicious block production, creates a strategic situation not dissimilar from the Cold War’s MAD—Mutually Assured Destruction. It is hardly surprising that such an environment would create substantial tension and uncertainty. To deescalate the situation, users—whether all of them, a majority, or a substantial minority—need a non-nuclear option. This is the merit of a proportionate response.

We propose a such a mechanism. It may be employed by any group of Bitcoin holders, service providers and miners—should the situation merit it. The mechanism itself requires a hard fork, and is for that reason best employed in situations where a hard fork is already necessary or desirable.

2 Proposal

Our proposal has two parts. Firstly, Bitcoin is forked to allow multiple hash functions at different difficulties. Secondly, a target ratio of blocks generated by each hash function is established by Bitcoin holders. The initial mix may include SHA-256, or even solely SHA-256 – so miners do not have to be immediately disadvantaged in any way. The target ratio may change every 2016 blocks by Bitcoin holder vote, and the difficulty of each hash function updates algorithmically to target this duly elected ratio of blocks. Some basic limits on the rate of difficulty change are put in place to protect miners (and everybody else) from the effects of a single vote. Naturally, the vote is weighted by bitcoins held, reflecting Bitcoin holders’ position as that of shareholders in Bitcoin as a whole.

3 Network difficulty updates

The network difficulty update algorithm is unchanged from its current implementation: if mining over a 2016 block period takes longer than an average of 10 minutes per block, the difficulty decreases, and if shorter, the difficulty increases, etc. For reference, we give pseudocode for network difficulty updates at algorithm 1.

Algorithm 1 Update network difficulty

```
procedure UpdateNetworkDifficulty
    $actual\_time \leftarrow timestamp\_last\_block - timestamp\_2016\_blocks\_ago$
    $target\_time \leftarrow 10 \text{ minutes} * 2016$
    $factor \leftarrow \frac{target\_time}{actual\_time}$
    $clamped\_factor \leftarrow \min(\max(factor, 1/4), 4)$
    $difficulty \leftarrow difficulty * clamped\_factor$
    return
end procedure
```

Whenever the network difficulty updates, so does the difficulty for each hash function, by the same factor. So for example, if the network difficulty increased from 500 billion to 550 billion (a 10% increase, or a factor of 1.1), and the hash function difficulty of SHA-256 was 200 billion, then the factor of 1.1 would be applied for a new hash function difficulty of 220 billion.

3.1 Best chain

Nodes select the best chain based on sum of network difficulty for each block, as in the current Bitcoin protocol.

4 Block ratio targeting

Independently of network difficulty updates, the Bitcoin protocol will also target a ratio of blocks mined under each hash function. The period over which this ratio is measured may vary independently from the network difficulty update period, but for simplicity we assume that the two updates occur simultaneously.

For n hash functions $h_1, h_2, \ldots, h_n$ the target ratio is represented as a list of n non-negative integers summing to 2016. The initial ratio may be set as needed given the current situation – even as $[2016, 0, \ldots, 0]$, where the first list element matches with the first hash function, SHA-256. After each 2016 blocks, the target ratio is compared to the actual ratio of blocks mined under each algorithm, and the hash function difficulties (which may also be set to required initial values) are adjusted as given in algorithm 2.

This procedure targets the desired hash function ratio by 1) adjusting the hash function difficulties by no more than a factor of two in either direction, 2) adjusting all hash function difficulties to keep the same time per block (by the factor α given on line 9). A simple modification would make it easier to boost a hash function with very high difficulty, for instance by substituting line 6 with

$$q_i \leftarrow \min(\max(\frac{b_i}{r_i}, (2 + (\lg d_i)/100)^{-1}), 2)$$

Algorithm 2 Update hash function difficulties according to target ratio

```
procedure TargetRatio
    $[r_1, \ldots, r_n] \leftarrow \mathit{target\_ratio}$
    $[b_1, \ldots, b_n] \leftarrow \mathit{blocks\_per\_hash\_function}$
    $[d_1, \ldots, d_n] \leftarrow \mathit{hash\_function\_difficulties}$
    for $i \leftarrow 1, n$ do
        $q_i \leftarrow \min(\max(\frac{b_i}{r_i}, 1/2), 2)$
        $b'_i \leftarrow b_i / q_i$
    end for
    $\alpha \leftarrow \frac{1}{2016} \sum_1^n b'_i$
    for $i \leftarrow 1, n$ do
        $d'_i \leftarrow \min(\max(\alpha q_i d_i, \mathit{minimum\_difficulty}), \mathit{maximum\_difficulty})$
    end for
    $\mathit{hash\_function\_difficulties} \leftarrow [d'_1, \ldots, d'_n]$
    return
end procedure
```

Example. Suppose there are two hash functions, SHA-256 and BLAKE2, and the target ratio is 1 to 1 (or 1008 blocks to 1008 blocks). If the numbers of blocks mined under SHA-256 and BLAKE2 in the last 2f016 block period were in fact 1512 and 504, respectively, then reason as follows: if the SHA-256 difficulty were $3/2$ times higher, on average only 1008 blocks would have been produced in the same time period. Similarly, if the BLAKE2 difficulty were half its actual value, then 1008 blocks would have been produced. These difficulty adjustments are made for the next 2016 period, before applying the network difficulty update to alter the expected time to produce those 2016 blocks.

In actual usage, a maximum factor of two may be too high, representing too great a threat to miners. A maximum of 1.1–1.2 would lead to a factor over one year of approximately 13–137.

5 Allocating blocks to hash functions: a public choice problem

A very wide variety of schemes exist to determine a choice between different options. Each scheme has two parts: the *process* of making a choice, and *how the choice is calculated.* The process should be as simple as possible without harming its purpose of advancing the interests of Bitcoin holders. We describe a simple process below, deferring discussion of alternatives until the end of this paper.

5.1 Voting process

Under the simple process, votes are public as soon as they are made. We suggest a voting period of about one week, timed to conclude a few hours before the network difficulty update: if the first block at a particular network difficulty is block 1, the vote will occur between blocks 1000 and 2000. A Bitcoin holder votes by attaching an `OP_RETURN` output to a transaction shaped as follows:

```
RETURN deed01 <32 bytes> <n bytes>
```

This calls `OP_RETURN` to make the transaction unspendable and prunable, while carrying 1) a three byte identifier (`deed01`), 2) a 32-byte block hash, and 3) n bytes representing n hash functions. Each byte of the vote is interpreted as an integer between 0 and 255, giving the Bitcoin holder the ability to choose his preferred allocation to a precision of better than 0.5%.

A valid vote must contain n bytes with values summing to 255 (note: an invalid vote does not make the transaction invalid). The 32-byte block hash must match the first block of the current difficulty period. This pins the vote to a particular history and a particular voting period, preventing replay attacks in which votes are reused on a secretly mined chain, or withheld until a later voting period.

The vote is weighted by the total value of each input of maturity at least 1020 blocks. This enables the holder of any particular bitcoin to vote with it exactly once per voting period.

5.2 Calculating the allocation

We have a known quantity of blocks – 2016 in each period – and want to assign them to a set of "candidates", i.e. the hash functions. We assume that each interested party has some preferred allocation, a second best allocation, and so on. We would like any system of deciding a block allocation to have two properties:

Unanimity If everyone agrees on a block allocation, that allocation should be chosen.

Strategy-proof It should be in each participant's interest to vote for his most preferred allocation.

Unanimity is easy to justify: if all Bitcoin holders agree on what should happen(!) then that thing should happen. **Strategy-proof** systems make it simpler to decide how to vote: Bitcoin holders should find it as easy as possible to do the thing that will forward their interests.

Unfortunately, it is mathematically impossible to satisfy both of these properties at the same time.[1] The best we can do is to minimize the effect of strategic voting. Luckily, computer simulation shows [Lindner et al., 2011] that some voting rules lead to only a very small amount of strategy, with a few percent of voters at most finding it rational to vote strategically. We advocate counting by a modified *Normalized Median Rule*, or *NMedian rule*.

5.3 NMedian vote counting

Count each satoshi as a single vote. For each hash function, find the median vote—an integer or half-integer between 0 and 255. This leads to a list of n such medians. Normalize them by multiplying each by a factor such that the sum is 2016, the number of blocks to be allocated. This then becomes the target ratio for the next period.

5.4 D'Hondt scaling

In practice, there may be no multiplier which leads to an exact sum of n integers to 2016. An exact algorithm that leads to almost the same outcome (not sufficiently different to induce strategic voting) is to allocate the 2016 blocks via the D'Hondt rule with no threshold.[2] As a median may lie halfway between two integers, multiply each median by two, then treat the result as that number of votes for its hash function. The D'Hondt rule is used in numerous polities and is not described further here.[3]

5.5 Tracking the vote

Nodes have to track the vote in each period, but need not track the votes themselves: a live update algorithm is possible. Each node should store, for each of the n hash functions, an int64 array of size 256.[4] For each valid vote, and for each of the n hash functions, the corresponding array should be incremented at the index corresponding to the actual vote, by an amount equal to the number of mature satoshis spent.

So for example, a vote `[100,100,40,15]` of 200 million satoshis increments

- array 1 by 200 million at index 100,

[1] By the Gibbard-Satterthwaite theorem—a nice overview is given by [Svensson and Reffgen, 2014].

[2] It is not absolutely necessary to use a target ratio summing to 2016, or consisting only of integers, but doing so both makes the ratio easier to understand and minimizes the amount of floating point computation required—a very useful property for ensuring clients stay in consensus.

[3] See `http://www.ucl.ac.uk/~ucahhwi/dhondt.pdf`

[4] This enables a "radix" median algorithm, which is much simpler and more efficient than more conventional median-finding.

Algorithm 3 Update target ratio according to bitcoin votes

```
procedure UpdateTarget
    $\mathbf{s} \leftarrow [\mathbf{0}, \ldots, \mathbf{0}]$
    for all $v \in votes$ do
        $w \leftarrow satoshis(v)$
        for $i \leftarrow 1, n$ do
            $\mathbf{s_i}[v_i] \leftarrow \mathbf{s_i}[v_i] + w$
        end for
    end for
    for $i \leftarrow 1, n$ do
        $m_i \leftarrow median(\mathbf{s_i})$
    end for
    $target\_ratio \leftarrow dHondt([m_1, \ldots, m_n])$
    return
end procedure
```

- array 2 by 200 million at index 100,
- array 3 by 200 million at index 40, and
- array 4 by 200 million at index 15.

Medians are easily calculated by summing over each array until half the total number of satoshi-votes are counted.

The total number of satoshis that could ever be voted corresponds to 21 million bitcoins, or 2.1×10^{15} satoshis. This number fits easily in an int64. The total additional storage required is $n \times 256 \times 8$ bytes, or 2 KiB per hash function, independently of the number of votes. Updating the vote count and calculating the new target ratio are both $O(n)$ in the number of hash functions.

6 Discussion

The purpose of any Proof of Work scheme is to secure a blockchain. Consider three classes of attack: network-based, interactive, and non-interactive.

A network-based attack targets the network connexions between miners and nodes in an attempt to disrupt message propagation. The proposed scheme does not significantly alter the quantity of network traffic or alter network connectivity, so is unlikely either to introduce or to prevent network-based attack vectors.

An interactive attack combines mining power with observation of the current blockchain state to produce a "bad", or "cheating", blockchain. An example of such an attack is the so-called selfish mining/mining cartel attack [ByteCoin, 2010] [Todd, 2013a] [Eyal and Sirer, 2014]. Such attacks occur over scales much shorter than that under which our proposed mechanism re-weights hash functions.

Non-interactive attacks use pure quantity of hash power, without observation of the blockchain. In Bitcoin's current scheme, the only such attack is the 51% attack, which requires a majority of SHA-256 hash power. Our scheme makes analysis of resistance to such attacks more complex, by weakening the link between network difficulty and the difficulty of any particular hash function. Lest comparisons be drawn to "proof of stake" systems, we point out than an attacker must still solo-mine thousands of blocks before gaining any possibility of advantage.[5]We can still be confident that a successful attacker must create a chain with greater *total network difficulty*, but a closer analysis is required of whether the ability to manipulate the target ratio makes it significantly easier to do so. Future work will simulate various attackers and to determine the necessary conditions of an non-interactive attack.

The voting process creates an economic incentive for miners to censor votes that weaken their chosen hash functions. This is not in itself a problem, provided some non-censoring miners exist—the problem arises if a miner cartel attempts to orphan blocks containing such votes. One method of improving the censorship-resistance of votes, timelock cryptography, is given at [Todd, 2013b].

Aside from on-chain voting in general, we discuss some objections to the particular method used:

- Public votes
- Last mover advantage
- "Activism"
- Depositors
- Cold storage

The voting process as described above makes all votes public as they are cast. This is not as such a disadvantage, but many will prefer anonymous voting. The literature on cryptographically anonymous voting is *extensive*—see for example [Baudron et al., 2001]—but the specific requirements of on-chain voting may make it impractical.[6] On the other hand, votes could easily be cast anonymously and revealed before the count. Many such commitment

[5] An attacker must 1) mine until he can change the target ratio 2) mine enough for the hash difficulties to converge to the target 3) mine difficult enough blocks to create a better quality chain than the public chain.

[6] For example, consider the difficulty of anonymizing a weighted vote. This would very likely require a wealthy holder to split his assets into pieces small enough for each to masquerade as the holdings of other users.

schemes are simple enough to implement in Bitcoin script (for instance, hash-based commitment).

A fixed voting window may encourage waiting until the last minute to vote, in the hope of catching less-interested voters unaware. Similarly, highly committed "activist shareholders" may exercise their voting rights disproportionately to others. Two possible requirements suggest themselves: quorum and referendum. Under quorum, a certain quantity of coins must be committed to make an adjustment to the target block ratio. Under referendum, a vote is not automatically held every 2016 block period, but must be called for by a quorum of coin holders. Wallets should respond to a successful referendum by alerting users.

Depositors of bitcoins with third parties may wish to vote. One solution is for their creditors to offer to vote on their behalf—just as some mining pools already offer their pool members this service.[7]

Lastly, it is a potential security issue to move bitcoins in cold storage simply to vote on hash functions. This problem could be solved by making the vote *detachable*. Under this scheme, a user depositing to cold storage adds a spendable "detach" output to his transaction. This detached voting output is then "colored" with a voting right equal to the value of one of the other outputs, valid as long as that "cold" output remains unspent. The color can be carried from a txin to a txout, with the same 1020 block maturity rule applicable. This enables a Bitcoin holder to keep his bitcoins in cold storage, while carrying voting rights in a hot wallet.

. . . .

Deployment requires simulation of both adversarial mining and adversarial voting. Rather more testing is required to implement shareholder-controlled multi-hash than would be necessary for a simple switch to a different PoW. Ideally this takes place well in advance of the need for such a significant change to the Bitcoin consensus protocol. Bitcoin wallets would be well advised to provide an interface for coin voting. New parameter sets are introduced, on which agreement would be necessary: the hash function set, the initial weightings, and the maximum block ratio adjustment. We reiterate: this proposal works best when deployed by a group that is already prepared to commit to *some* hard fork. Good luck.

References

Olivier Baudron, Pierre-Alain Fouque, David Pointcheval, Jacques Stern, and Guillaume Poupard. Practical multi-candidate election system. In *Proceedings of the twentieth annual ACM symposium on Principles of distributed computing*, pages 274–283. ACM, 2001.

[7] https://blog.slushpool.com/voting-options-simplified-core-bu-65eadb322950, archived at https://archive.is/KgXDP

ByteCoin. Mining cartel attack, 2010. URL https://bitcointalk.org/index.php?topic=2227.msg30083#msg30083.

Ittay Eyal and Emin Gün Sirer. Majority is not enough: Bitcoin mining is vulnerable. In *International Conference on Financial Cryptography and Data Security*, pages 436–454. Springer, 2014. URL http://fc14.ifca.ai/papers/fc14_submission_82.pdf.

Tobias Lindner, Klaus Nehring, and Clemens Puppe. Which voting rule is more manipulable? results from simulation studies, Sep 2011. URL http://micro.econ.kit.edu/downloads/Lindner_WhichVotingRuleIsMoreManipulable.pdf.

Lars-Gunnar Svensson and Alexander Reffgen. The proof of the Gibbard––Satterthwaite theorem revisited. *Journal of Mathematical Economics*, 55: 11–14, 2014. ISSN 0304-4068. doi: http://dx.doi.org/10.1016/j.jmateco.2014.09.007. URL http://www.sciencedirect.com/science/article/pii/S0304406814001177.

Peter Todd. [bitcoin-development] we can all relax now, Nov 2013a. URL https://lists.linuxfoundation.org/pipermail/bitcoin-dev/2013-November/003607.html.

Peter Todd. [bitcoin-development] censorship-resistance via timelock crypto for embedded consensus systems, Dec 2013b. URL http://www.mail-archive.com/bitcoin-development@lists.sourceforge.net/msg03524.html.