Collaborative Block Reinforcement in Blockchain Systems

Matej Pavlovic

1 Introduction

The goal of this proposal is to minimize the amount of work that is "wasted" by unsuccessful miners in blockchain-based systems like Bitcoin. We believe that this can shorten the delay between the time when a transaction is proposed and the time when the transaction can be considered confirmed with high probability.

In classic blockchain-based systems like Bitcoin, miners try appending a new block to the end of a blockchain by searching for a nonce that, after being fed through an irreversible hash function together with the new block contents and the digest of the predecessor block, produces a digest that is smaller than some threshold d (for *difficulty* of mining - the smaller d, the higher the expected time until a miner finds the right nonce). A miner that successfully appends a new block to the blockchain receives some form of reward.

Let the last block of the blockchain be A. After some miner finds a nonce that allows it to create a block B that is a successor of A, it broadcasts the new block B (containing the new nonce). On reception of B, other miners abandon their search for the nonce that would allow them to append a new block after A, and start searching for a nonce that would allow them to append a new block after B (we say they start mining on top of B). In general, some significant amount of computing power needs to be invested in order to produce a new block.

The valid chain is defined to be as the longest one. The more computing power is invested in appending blocks to a particular chain, the harder it becomes to create another chain that is longer (since prolonging a chain requires a significant amount of computing power). Thus, the acceptance of a block is reinforced by the decision of miners to mine on top of it. In general, if the majority of miners[1] decide to append blocks to a chain containing the block, then the block will be accepted. In some sense, each block is backed by the amount of computation invested in creating it and in appending other blocks to it.

If the chain *forks*, i.e. if two miners find a valid successor blocks of A (let us call them B and B'), only one of these two blocks will be accepted. If the

[1] Technically, we should say here "miners that together possess a sufficiently big fraction of the computing power". For simplicity, we will refer to this simply as the "majority of miners".

majority of miners decide to mine on top of B, then the chain appended to B will grow faster on expectation and thus will most probably become longer than the chain appended to B'. The "longest chain is the valid one" property, together with the difficulty of prolonging any chain, ensure that miners have the incentive to mine on top of the last block of the longest chain, and thus prolong it even more. Note that the reward associated with blocks that are not part of the longest chain is none.

In practice, a block A is considered accepted if it is part of the longest chain and at least a blocks have been appended to it. After a blocks have been appended, one can assume that quickly appending a different chain to the predecessor of A is too difficult. Moreover, if the majority of the miners mine on top of the chain that contains A, this chain will grow faster on average than any parallel chain would.

The problem with this approach is at the end of the chain: every time a new block is appended, the race for the next block starts from scratch. While this property is important regarding selfish mining attacks, big amounts of computation resources and energy are wasted by miners that are not successful in finding the new block. In other words, the system works while a majority of miners *try to append* to the same block, while only one miner actually *does append* a new block. In particular, after creating a new block, there is no a-posteriori proof that the majority tried appending to the same predecessor block. Thus, the work done by the majority of miners can be considered wasted every time the race for the new block restarts.

Our main idea is to allow miners which unsuccessfully mined on top of a block A to use their work to reinforce a new successor block B, regardless of which miner found B. A miner reinforcing another miner's block is also rewarded. We make the mining process involve a proof of work (POW) that is independent of the particular block that the miner tries to append, and thus transferable to any new block. When some miner succeeds in finding a new block, it broadcasts this new block, asking other miners to reinforce the block with their independent POWs generated while mining on top of the same predecessor block. The finder then waits for some amount of time to receive these reinforcements, chooses which to use for its block and broadcasts the chosen ones. Instead of defining the longest chain as the valid one, we say that the chain with the most associated amount of reinforcement is valid (while the mere existence of a block would still constitute an important part of that block's reinforcement). If parametrized properly, non-malicious (i.e., also rational) miners have an incentive to reinforce an existing block, instead of continuing to mine for a competing block. As a result, a bigger amount of the computing power can be provably associated with a single block, and thus fewer additional appended blocks are necessary to consider some block confirmed. This, in turn, addresses the problem of high latency of transaction confirmation, that is particularly pressing in Bitcoin. The next section presents a more detailed description of our new protocol.

2 The Protocol

This section describes the protocol for collaborative block reinforcement. Our protocol is based on Bitcoin, which, on a high level, is the following:

1. Miners mine on top of what they believe is the last block of the blockchain by trying to compute a POW involving both the last block and the new block to be appended.
2. When a miner m succeeds, it broadcasts the new block, together with its POW.
3. Miners start mining on top of the new block.

In Bitcoin, the valid blockchain is defined to be the longest one. Thus, in some sense, miners "vote" for a chain by mining on top of its last block.

Our protocol extends and generalizes Bitcoin in various ways. We present our protocol in its simplest working form, intentionally leaving out some optimizations that would reduce its computation and/or communication complexity. For example, in many cases, a block contained in a broadcast message can be replaced by just the digest of the block.

A simplified high-level sketch of the protocol is as follows:

1. Miners mine on top of what they believe is the last block of the blockchain, independently of what the next block will be.
2. During this mining process, some POW is generated at each miner locally, even if that miner does not succeed in appending a new block.
3. When a miner m succeeds in computing a POW that is sufficient a new block, it *proposes* a new block by broadcasting it, together with its POW.
4. The proposal of a new block is interpreted as a request for support by the other miners, which may decide to reinforce that block by sending their POWs to m.
5. m, after collecting the reinforcements from other miners, commits the block by broadcasting the reinforcements.
6. Miners start mining on top of the new block (it is only possible to mine on top of committed blocks).

Unlike Bitcoin, our protocol defines the valid blockchain to be the *heaviest* one, i.e., the linear chain with the largest cumulative amount of POW associated with its blocks.[2] The POW associated with a block consists of the POW required to append that block, and the POWs that are used to reinforce it.

[2]The GHOST rule generalization is still applicable here, i.e. a block can be considered valid if it has the heaviest subtree. We consider linear chains for simplicity.

The above sketch just illustrate the normal operation of the protocol. The rest of this section provides more detailed description of the protocol steps, as well as ways to handle situations outside the normal operation. We also describe how rational miners are incentivized to follow the protocol.

2.1 Proof of work and block proposal

Let miner m mine on top of block A in order to append a new block B. In order to decouple the POW from the new block, the mining process will not directly involve B. Instead, we allow m to choose what block to append after generating a POW on top of A.

Generating the POW is similar to Bitcoin, with a few differences. It works as follows. m tries to find a nonce n, such that $h(A, n, m_{pub}) < d$, where h is a hash function, m_{pub} is m's public key and d is the mining difficulty parameter. Even if m does not find a nonce that would produce a hash $< d$, it may save nonces that produce hashes m considers reasonably small. These nonces, although not sufficient for appending a new block, still constitute a POW that can be later used to reinforce a block found by another miner. An alternative approach proposed by Garay et al. would work as well, using different parts of the output of the hash function for mining for new blocks and mining for reinforcements.

In practice, a distinct difficulty parameter d_r may be necessary, setting an upper bound on the hash value allowed for block reinforcement, as discussed in section 3.2.

If m succeeds in finding a nonce that produces a sufficiently small hash, it has the right to *propose* a new block B. m proposes a block by broadcasting it.

2.2 Reinforcing and committing blocks

When a miner m' receives the proposition for block B, it may decide to reinforce B by associating its POW with B. This is done by broadcasting a message $\langle B, m'_{pub}, \mathcal{R}_{m'} \rangle_{m'}$, where $\mathcal{R}_{m'} = \{n_0, n_1, \ldots\}$ are nonces that produced small hash values during the mining process of m'. The notation $\langle \ldots \rangle_{m'}$ signifies that the message is signed by m'.

After m, the finder of block B, receives enough reinforcement messages, it *commits* B by broadcasting the message $\langle B, \mathcal{R} \rangle_m$, where $\mathcal{R}$ is the set of the collected reinforcement messages. In order to limit the time and the bandwidth cost, we introduce a parameter $\mathcal{R}_{max}$, which limits the maximal size of $\mathcal{R}$. $\mathcal{R}$ is only valid if $|\mathcal{R}| < \mathcal{R}_{max}$.

As long as m adheres to the $\mathcal{R}_{max}$ limit, it is solely up to m to decide how much reinforcement is "enough" and which of the received reinforcement messages to include in $\mathcal{R}$. We discuss the implications of this in sections 3.1 and 3.1.

2.3 Preventing double reinforcements

In order to keep the probability of forks small, it is necessary to ensure that a POW can only be used to reinforce a single block. By including block A in the mining process, we ensure that only successors of A can be reinforced. However, in case of a fork, where both B and B' are successors of A, a miner might try to reinforce both new blocks to maximize the chance to be rewarded, regardless of which of the two blocks becomes part of the heaviest chain.

To prevent this, we use the concept of *proof of misbehavior* (POM). In our case, any pair of messages $(\langle B, m_{pub}, \mathcal{R}_m \rangle_m, \langle B', m_{pub}, \mathcal{R}'_m \rangle_m)$ with $\mathcal{R}_m \cup \mathcal{R}'_m \neq \emptyset$, constitutes a POM against miner m. To discourage miners from reinforcing multiple blocks using the same POW, we take away from them the associated reward if they do so. To this end, similarly to Bitcoin, we delay awarding reinforcement-related rewards k blocks after the block they are associated with. The first miner that finds one of these k blocks and is in possession of a POM against m may claim the associated reward by adding a special transaction (containing the POM) to the block.

Note that this might incentivize nodes holding a POM to not cooperate in disseminating some of m's reinforcement messages, in order to prevent other nodes from obtaining a POM and thus to increase the own chance to claim the associated reward. This is, however, not a problem, since it also, although in a different way, prevents double reinforcements.

2.4 Reinforcement transfer

The above reinforcement protocol exacerbates the problem of the block withholding attack. In this attack, a malicious miner does not disclose a block after discovering it, but instead keeps it secret and starts mining on top of it in the background. With the protocol described so far, such a miner could even propose a block, gather reinforcement from others, and then delay committing the block to win extra time during which it is the only miner able to mine on top of it. Even worse, the submitted reinforcements make it even harder for another competing block[3] to be accepted.

To address this problem, we allow miners to *transfer* their reinforcements from one block to another competing block. In this section we describe only the mechanism of reinforcement transfers; for a discussion of the impact on selfish mining, see section 3.3

It is important that simple revocation is not possible, since this would enable sudden decreases in the weight of potentially unlimited parts of the blockchain, if many colluding miners decided to revoke their reinforcements. If we only allow reinforcement transfers, revoking the reinforcement of a block involves some amount computation, namely at least the computation necessary to find a competing block to transfer the reinforcement to.

To transfer its reinforcement from block B to a competing block B', a miner m broadcasts a message $\langle B, B', m_{pub} \mathcal{R}_m \rangle_m$, where $\mathcal{R}_m$ contains the nonces

[3] A competing block is a block with the same predecessor.

associated with the reinforcement m wants to transfer. A pair of messages $(\langle B, m_{pub}, \mathcal{R}_m \rangle_m, \langle B, B', m_{pub}, \mathcal{R}'_m \rangle_m)$ is, for the purposes of block reinforcement, equivalent to $\langle B', m_{pub}, \mathcal{R}'_m \rangle_m$, but it does not constitute a POM together with the original message $\langle B, m_{pub}, \mathcal{R}_m \rangle_m$. It can, however, be used in a POM together with some other message.

The reinforcement is only considered transferred if it is actually used to reinforce B'. It may well happen that a miner decides to transfer its reinforcement from B to B' when B' is proposed, but the finder of B' decides to not use it for committing B'. In such a case, the reinforcement keeps contributing to the weight of block B. However, it can still be used in a POM against m.

The reward associated with the reinforcement is "sticky". This means that both B and B' contain the reinforcement reward for m, even if m transferred its reinforcement from B to B'. This is crucial, since if the reward was transferred together with the reinforcement, any miner would be able to remove coins from the main chain at any time, simply by creating a fork and transferring it's reward to it. Note that giving both competing blocks contain the reward does not pose a double-reward problem, since only one of these blocks is valid at any point in time.

2.5 Block weight

The weight of a block corresponds to the work that was invested in appending it to the blockchain. Let some amount of work w be necessary on expectation to obtain a hash value $< h$. Assuming the output of the used hash function to be uniformly random, twice as much work is necessary on expectation to obtain a hash value $< h/2$.

Thus if a nonce produces a hash h, we define the *weight* w of the POW it represents by

$$w = \frac{d}{h} \tag{1}$$

The weight of block A, denoted w_A is then computed as the sum of the weights of all reinforcements backing that block. Thus, if $\mathcal{H}_A$ is the set of hashes corresponding to all nonces across all reinforcements of block A, the weight of A is computed as

$$w_A = \sum_{h \in \mathcal{H}_A} \frac{d}{h} \tag{2}$$

The POW that allowed the miner to append the block (having, by definition, a weight of at least 1) is not treated specially in the block weight computation. It must, however, be present among the block reinforcements, otherwise the block is considered invalid.

Note that a miner that found a hash sufficiently small to append a new block may still decide to use his POW to support another block instead. This might be a reasonable choice if a new block has already been proposed (but not yet committed), and it is likely that that block will become part of the stable chain

(since it probably gathered reinforcement from the rest of the network, or is at least closer to doing so).

2.6 Rewards

As in Bitcoin, the reward for mining a block consists of two parts: 1) the newly created coins that the finder of the block can claim using the so-called coinbase transaction, and 2) the transaction fees associated with the transactions in the block.

The total reward for a block is computed the same way as in Bitcoin, but we redistribute this reward among the finder and the reinforcers as follows:

- A fixed fraction r_f of the reward goes to the finder of the block.
- The remaining reward $(1 - r_m)$ is distributed among reinforcers, proportionally to the weights they contributed.

Considering a normalized block reward to be 1, the reward for the finder is r_f, while a miner m reinforcing the block receives $r_m = \frac{w_m}{w_{total}}$, where w_m is the cumulative weight of all m's reinforcements and w_{total} is the total weight of all reinforcements of the block, not counting the weight of the finder's POW used to append the block.

3 Discussion

3.1 Choosing reinforcements by the finder of a block

The reward distribution proposed in 2.6 creates a tension between the incentives of a block finder to include reinforcements from other miners. On one hand, upon finding a block, a miner is incentivized to maximize the weight of this block by including as many reinforcements as possible. The heavier the block, the more likely it is to be included in the main chain, guaranteeing the finder's reward. On the other hand, in order to maximize its reward, a finder of a block could refuse to include other than its own reinforcements in that block. However, doing so would decrease the weight of the block, and the finder would risk loosing all its reward to a competing block. Thus, while a miner does have the ability to, in some sense "unfairly", increase its own reward, it can only do so to a limited extent.

Refusing too much reinforcement is risky for a miner, since other miners, knowing the amount of "unused" reinforcement, are incentivized to continue mining for a competing block. Such a competing block would, with high probability, be heavier, and thus more likely to be included in the main chain.

A miner could still opportunistically try to refuse others' reinforcement, hoping that no competing block would appear. If a competing block does appear, the miner would use its POW to reinforce the competing block, in order to obtain at least a smaller reward. Such behavior, however, can easily be prevented by either disallowing reinforcements by a POW that has already been

used to create a block (by considering it a POM), or by completely separating the POWs for block appending and reinforcement (using the "2-for-1 trick" of Garay et al.).

3.2 Lower-bounding reinforcement weight

Given $\mathcal{R}_{max}$, the limit on the size of the set of reinforcements in a block, the finder might still face a dilemma which reinforcements to choose among equally heavy ones. Additionally, allowing reinforcements of any weight might unnecessarily flood the network with reinforcements of small weight, hindering propagation of the more important ones. For this reason, we use the parameter d_r, effectively setting the lower bound on the weight of allowed reinforcements. In practice, one might define $d_r = \alpha \cdot d$ as a multiple of the block mining difficulty d for some constant factor α.

Note that, given the constant size of the representation of a POW (regardless of its weight), a rather small number of the heaviest POWs accounts for most of the cumulative weight of all the POWs generated for a block. Thus, d_r can be set such that, on expectation, $|\mathcal{R}| \ll \mathcal{R}_{max}$ even if all valid reinforcements are included in $\mathcal{R}$, while still capturing most of the work done by miners.

3.3 Selfish mining

As stated in section 2.4, we employ reinforcement transfers to fight against selfish mining, to which our protocol would otherwise be more susceptible. If a miner finds a block B, it has two choices, it has three choices:

1. Propose B, gather reinforcements for it, and commit B in "reasonable" time (i.e., follow the protocol).
2. Propose B, gather reinforcements for it, but not commit it B (i.e., withhold the commitment).
3. Withhold B altogether.

In case (2), the selfish miner can start mining on top of B that includes reinforcements gathered from the rest of the system. However, if the selfish miner takes too long to commit its block and a competing block B' appears, honest miners can transfer their reinforcements to B'. As long as the selfish miner controls less than half of the hashing power, B' is likely to become heavier than B. Still, the selfish miner might wait until the first appearance of B' and commit B while B' still propagates, and the results of Eyal and Sirer apply. In such a case, reinforcers are still rewarded for their work.

In case (3), the selfish miner can only use its own reinforcements (and, implicitly, reinforcements of colluding miners) for B, decreasing the potential weight of B.

In general, our protocol does not completely defend against the selfish mining strategy. However, it might decrease the efficiency of this strategy, since it:

- prevents all honest work from being wasted even if a selfish miner temporarily succeeds (honest miners are still rewarded for their reinforcements) and
- makes the growth of the chain weight much more predictable by incrementing it in finer-grained steps, decreasing the probability of a selfish miner "being lucky" and secretly pre-computing a heavier chain using little hashing power.