WWHM Difficulty Adjustment Algorithm in Masari

Author: Cryptochangements
September 2, 2018

Abstract

This paper details the cryptocurrency Masari's implementation and transition to a Weighted-Weighted Harmonic Mean difficulty adjustment algorithm in order to smoothly adjust its difficulty target so that the chain maintains a steady two-minute block time. The original idea for this algorithm comes from Tom Harold and was modified by Scott Roberts. The original C++ implementation was done by Thaer Khawaja of the Masari Core Team. This implementation has since been adopted by several other cryptocurrency projects, and has paved the way for other difficulty adjustment algorithms such as LWMA.

1 Introduction

Most cryptocurrencies utilize proof-of-work to guard against network spam. In a typical proof-of-work scheme, miners cryptographically hash some information from the blockchain; typically, the most recent block header and a nonce, until the combination of the block header and the nonce fall under certain conditions. The conditions that it must meet are called the difficulty. Typically the difficulty requires the hash digest to be lower than given value. This way not every hash is accepted by the network, the miner must "hunt" for the correct hash. That hash is then presented to the network and verified to prove that the miner has done work by hunting for that hash. If the difficulty is low, then it will be easy to find a valid hash. Therefore a valid hash can be found quickly.

The size of the network constantly changes as miners join and leave so the difficulty needs to constantly re-target. However, the original implementations of cryptocurrency difficulty adjustment algorithms used in older cryptocurrencies such as Bitcoin Core or Bytecoin, the original CryptoNote reference implementation, are rather naive and adjust slowly. Because of this slow adjustment, an attacker with a large portion of the network's hashrate can take advantage of this latency by mining with a large amount of compute power; such that the current network difficulty is easy for them and can flash mine several blocks in rapid succession until difficulty finally adjusts. Masari suffered this kind of attack in its early days of development and as a result, the Masari developers worked with Scott Roberts to develop a Weighted-Weighted Harmonic Mean (WWHM) difficulty adjustment algorithm. This allowed Masari to adjust its difficulty much quicker and prevent this kind of attack.

2 Flash-mining in Masari

Relatively shortly after the launch of Masari, during the month of October 2017 the Masari developers noticed unusual activity on the blockchain. They had noticed that the network hash rate had rapidly increased several fold within a very short window of time coming all from a single miner pool. As a result, blocks were found at an incredibly fast rate, much faster than the two minute target. As soon as the difficulty had adjusted to account for this new network hash speed this miner disconnected from

the network . This left the network with an inflated difficulty resulting in slow block times until the difficulty finally adjusted again.

2.1 How the attack works

Masari is a fork of the cryptocurrency Monero and therefore inherited Monero's difficulty adjustment algoithm. This algorithm is rather naive and assumes that the network hash rate doesn't change much between adjustment periods. This simple algorithm uses the most recent 720 blocks (approximately 1 day with a 2 minute block time) as a sample size for difficulties (D_i) and uses the timestamps (T_i) from these blocks to get the total solve time (S). The difficulties are summed such that $d = \sum_{i=1}^{720} D_i$ and the total solve time is found such that $S = T_{720} - T_1$. From this information the network hash rate (h) is estimated as $h = d/S$. The next difficulty is then calculated by multiplying the target time (t) by the estimated hash rate $t * h$. In practice this should work such that if the network hash rate is around 4,000 hashes per second over a consistent period of time then the difficulty should be around 480,000 to keep a steady 120 second block time as $4,000 * 120 = 480,000$. This means that it should take around 480,000 tries before a miner solves a block. If a new miner comes in with a drastically higher hash rate, say 15,000 hashes per second then it would only take the new miner around 32 seconds to find a block at the current difficulty as $480,000/15,000 = 32$. This new miner can then continue mining 32 second blocks until the difficulty adjusts 720 blocks later, which would take approximately 6.4 hours at that rate. After the difficulty finally retargets, the difficulty would be about 2,280,000 as $(4,000+15,000)*120 = 2,280,000$. The attacking miner can then leave the network as he is no longer reaping the benefits from the slow difficulty adjustment. With this newly increased difficulty for what is now a small amount of hashing power leaves block times at around 570 seconds on average as $2,280,000/4,000 = 570$.

3 WWHM In Theory

A Weighted-Weighted Harmonic Mean based algorithm weighs newer solved blocks more heavily than older solved blocks when calculating difficulty to provide a more accurate estimation of the network's conditions in the near future.

3.1 The Algorithm

The only data from the blockchain that the algorithm requires is the timestamps (T_i) and difficulties (D_i) from the most recent blocks within a specified block window (w) on the blockchain. In Masari $w = 60$ but it can be changed based on the necessities of the users of this algorithm. The target in seconds (c), in Masari's case 120, is also a parameter of this function. The first step is to get the solvetimes ($S_i | i = 1, ..., w$) for the blocks within the block window which can be expressed as (S_i = T_i - T_{i-1}). Next the sum of the weighted solvetimes (H) is calculated. Solvetimes are weighted by multiplying the solvetime by its index in the series so that later elements in the series are weighted heavier. This can be expressed as:

$$H = \sum_{i=1}^{w} S_i \cdot i$$

The constant k is also used which will be defined as $k = (w+1)/2 \cdot c$. The difficulties within the last w blocks (d) also need to be summed such that

$$d = \sum_{i=1}^{w} D_i$$

Finally, the next difficulty can be calculated by multiplying the sum of the previous difficulties by k and then dividing by the sum of the weighted solvetimes.

$$d \cdot k/H$$

4 WWHM Implementation

4.1 Pseudo-code

```
#define BLOCK_WINDOW 60;

function getNextDifficulty(timestamps, difficulties, target_seconds) {
  for (i = 1; i < BLOCK_WINDOW; i++) {
    solvetime = timestamps[i] - timestamps[i];
    if (solvetime > target_seconds * 10)
      solvetime = target_seconds * 10;
    weighted_solvetimes += solvetime * i;
    difficulties_sum += difficulties[i];
  }
  min_timespan = target_seconds * BLOCK_WINDOW / 2;
  if (weighted_solvetimes < min_timespan)
    weighted_solvetimes = min_timespan;

  target = (BLOCK_WINDOW + 1) / 2 * target_seconds;
  return difficulties_sum / target * weighted_solvetimes;
}
```

4.2 Sanity checks in production code

Because this algorithm was created with the intent to be used in the context of a currency verified by proof-of-work, there must be some sanity checks to make sure that the adjustment isn't too drastic. The first check is in the pseudo-code from 4.1 is a limit on the maximum solve time. If a solve time is greater than 10 times the target, which in the case of Masari would be 20 minutes, then that maximum solve time should be used. The second check is on the minimum solve time. If the solve time recorded on the blockchain is less than half of the product between the target solve time and the block window, in Masari's case this is 1 minute, then the minimum solve time should be used.

5 Conclusion

WWHM was a leap forward in difficulty adjustment algorithms, improving greatly upon previously used in older cryptocurrencies. It quickly worked to prevent flash

mining attacks in Masari and other cryptocurrencies quickly after its implementation, making it a success.

Resources

1. Masari reference codebase https://github.com/masari-project/masari
2. C++ implementation used in Masari (next_difficulty_v3) https://github.com/masari-project/masari/blob/master/src/cryptonote_basic/difficulty.cpp