

Too Fast Blocks, Too Furious Adjustments: The Satoshi Drift Level-Out

DEMETRIS KYRIACOU, Imperial College London, UK

MATTHIEU BABAK, ENSTA Paris, France

IAIN STEWART, Imperial College London, UK

WILLIAM J KNOTTENBELT, Imperial College London, UK

Government-issued currencies are maintained by trusted authorities such as central banks, centralised payment systems, etc. In digital currencies where “code is law”, there are no centralised authorities and many of the functionalities are replicated by algorithms. For Proof of Work (PoW) blockchains, one of the most important algorithms is the difficulty adjustment algorithm (DAA). In an ever-changing hash rate environment, the duty of a DAA is to adjust mining difficulty to ensure block rate stability. This is of paramount importance because in the short term, it controls transaction throughput and in the long term, it controls the issuance of the native coin. Designing an effective DAA can be challenging because the real hash rate of a blockchain network is unobservable and can only be estimated based on previous interblock times. For Bitcoin (BTC) things are relatively simple. The network is vast and largely immune to hash rate fluctuations so mining difficulty can be adjusted infrequently (approx. every two weeks). The remaining PoW blockchains however, such as Bitcoin Cash (BCH), are more susceptible to hash power fluctuations and benefit from a more adaptive DAA which adjusts the mining difficulty after every block. Such DAAs already exist, have been battle-tested and have proven effective at maintaining average interblock times close to the target. Nevertheless, existing DAAs tend to focus on short-term stabilisation of block production and ignore long-term stabilisation thus allowing drift accumulation. In fact, both BTC and BCH halvings of April 2024 occurred almost 9 months ahead of schedule due to this drift. To our knowledge, no DAA had systematically addressed the problem of accumulated drift prior to our earlier work originally presented in BlockTEA 2025 [29]. This paper is an extension of our work which greatly generalises, and yet paradoxically simplifies, the DAA given therein.

Additional Key Words and Phrases: Difficulty Adjustment Algorithm, Mining difficulty, Proof of Work, Bitcoin Cash

ACM Reference Format:

Demetris Kyriacou, Matthieu Babak, Iain Stewart, and William J Knottenbelt. 2026. Too Fast Blocks, Too Furious Adjustments: The Satoshi Drift Level-Out. 1, 1 (September 2026), 26 pages. <https://doi.org/XXXXXXX.XXXXXXX>

1 Introduction

Over recent years, various difficulty adjustment algorithms (DAAs) have been proposed and implemented in blockchains that support Proof-of-Work (PoW) consensus mechanisms. Some of them proved to be stable and robust while others deteriorated to relics amidst disaster.

The primary purpose of a DAA is to regulate interblock times of a PoW blockchain by adjusting the difficulty of mining a block. In an ideal world where the real total hash rate of a blockchain network is known, adjusting difficulty would be trivial. In practice, however, the hash rate of a network is unobservable and can only be estimated using

Authors’ Contact Information: Demetris Kyriacou, demetris.kyriacou22@imperial.ac.uk, Imperial College London, London, UK; Matthieu Babak, matthieu.babak@ensta-paris.fr, ENSTA Paris, Palaiseau, France; Iain Stewart, i.stewart@imperial.ac.uk, Imperial College London, London, UK; William J Knottenbelt, w.knottenbelt@imperial.ac.uk, Imperial College London, London, UK.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

© 2026 Copyright held by the owner/author(s). Publication rights licensed to ACM.

Manuscript submitted to ACM

Manuscript submitted to ACM

data about previously mined blocks such as interblock times and past difficulties. Although some miners proudly and openly declare their hash power, some of them remain rather cryptic. Additionally, there are miners who are not always loyal to mining on a single blockchain but instead perform what is called coin-hopping strategies to maximise their profit [28]. Interestingly, the hash power behaviour of the network resembles that of financial markets. Both are driven by underlying human, technological, and architectural components. Another common trait is that poorly designed algorithms or subtle interactions between algorithms can disrupt their orderly regulation [2].

However, even if the DAA does a good job in controlling interblock times in the short-term, there is another problem that can be developed gradually in the long-term. *Drift* or *skew* is created when blocks are, on average, being mined faster or slower than normal. For BTC it happens mainly because the DAA cannot keep up with the increase of hash rate due to increase in mining popularity and hardware capabilities. Consequently, it accumulates drift every time there is a major leap forward in mining technology.

In August 2017, disputes within the Bitcoin community over scalability and functionality led to the creation of Bitcoin Cash (BCH) [26, 27] as a hard fork. After the hard fork, Bitcoin Cash continued its own, more turbulent path, as an independent blockchain and experienced another schism as a result of Bitcoin Cash’s hash war [39]. The blockchain was split into what is still referred to as Bitcoin Cash (BCH) [5] and Bitcoin SV (BSV) [7]. This paper focuses on the former. To this day, BTC uses its original DAA with biweekly difficulty adjustments, while BCH has experimented with various DAAs that adjust the mining difficulty at every block [3, 11]. However, both of them have piled on more drift to the inherited pre-fork drift, and as a result, their halvings in April 2024 happened almost 9 months prior to their originally-scheduled dates.

In this paper, we choose to focus on BCH due to its unsettled past of DAAs along with the fact that it represents one of the most popular PoW blockchains whose mining difficulty is adjusted at every block. Additionally, BCH’s vulnerability to hash rate fluctuations makes it more challenging and interesting to investigate. Lastly, our research builds on previous work by Ilie et al. [25].

NEFDA DAA proposed in [25] ensured effective difficulty adjustment with stable block production rate without long-term autocorrelation. However, it was prone to the drift accumulation problem.

NEF(DA)² DAA that we presented in [29] inherits the properties of NEFDA but also solves the drift accumulation problem without a significant increase in computational complexity. However, it cannot eliminate drift for exponential and super-exponential hash rate functions. Additionally, the mathematical formulae and the code are convoluted which hinders generalisation.

In this paper, we present NEF(DA)^N, the generalised algorithm that incorporates NEFDA and NEF(DA)² as special cases of levels 1 and 2, respectively. This algorithm inherits the properties of its predecessors and mitigates the drift accumulation problem even for extreme cases of hash rate. Additionally, with the introduction of *liquid heights*, the formulae and code are significantly simplified. Although primarily designed for BCH, our algorithm is intended to be blockchain-agnostic and makes minimal assumptions about the underlying blockchain. Even the ideal interblock time is integrated into the algorithm as a tunable parameter. This is evident both from the formulae (see Section 5.5) and the simulation code [17]. Therefore, our algorithm is compatible with many PoW blockchains and is particularly suited to blockchains where difficulty is frequently adjusted, ideally on a block-by-block basis.

The paper begins with a review of various DAAs used in BCH before examining the individual components of the generalised NEF(DA)^N and linking them together. Subsequently, the algorithm is theoretically analysed and tested experimentally to prove its effectiveness and robustness in a realistic environment. The source code is available at [16] and [17].

2 Related work

2.1 BCH DAAs and Grasberg

Bitcoin Cash has adopted multiple DAAs since the hard fork with Bitcoin, all of them are analysed in depth in Section 4. A noteworthy proposal for BCH that was never adopted was Amaury Sechet’s Grasberg DAA [37]. Grasberg DAA was based on a negative exponential filter and included a drift correction mechanism. It aimed to ensure a stable and more predictable block-production rate and minimise the profit gap between loyal mining and coin-hopping mining. However, the proposal was rejected as it was deemed too complicated and inadequately substantiated. Furthermore, a few members of the BCH community argued against the need for drift correction [22, 31, 38]. The main difference between our DAA and Grasberg is that ours is very simple and we provide a detailed analysis of the mathematical background, as well as extensive simulations.

2.2 Kimoto Gravity Well algorithm and Dark Gravity Wave algorithm

Kimoto Gravity Well (KGW) algorithm is the DAA used for Megacoin [10] that adjusts difficulty at every block. It was introduced in 2013 by an anonymous user called Dr Kimoto Chan. Its purpose was to combat the hash-rate fluctuation issue caused by coin-hopping strategies in PoW altcoins such as Megacoin [13]. The algorithm uses a negative exponential curve called *Kimoto* and its reciprocal $1/Kimoto$ to keep track of hash power fluctuations. The more rapid the change in average recent hash power, the more responsive KGW is [6, 32]. Unfortunately, this DAA proved to be vulnerable to time-warp attacks where malicious users could manipulate the algorithm to create windows of low difficulty. Thus, KGW algorithm became obsolete [3].

Dark Gravity Wave (DGW) algorithm is a DAA introduced for Dash [20] by its creator Evan Duffield. Dash, formerly known as Darkcoin, is a Bitcoin protocol fork that initially used the KGW algorithm. When KGW’s vulnerability to time-warp attacks was revealed, Evan Duffield developed DGW to replace it [14]. DGW uses a combination of simple and moving averages to adjust the mining difficulty of every block. It is still used by Dash and has also been adopted by other digital currencies [3, 15].

2.3 DigiShield algorithm

DigiShield DAA was first introduced for DigiByte in 2014 [18] and was later adopted by other blockchains such as ZCash [41] and Dogecoin [19]. The algorithm adjusts difficulty at every block using a tempered simple moving average window of 17 blocks. This algorithm is still in use although its over-responsiveness can lead to massive fluctuations in the difficulty even for a relatively stable network hash rate. In fact in the case of ZCash, it is not uncommon for the difficulty to swing by more than 50% within less than two hours [1] and by more than 8% within consecutive blocks [42]. Consequently, throughout the day, ZCash experiences short periods with bursts of new blocks and long periods of sparse new blocks.

2.4 Linear Weighted Moving Average algorithm

Linear Weighted Moving Average (LWMA) algorithm was presented by a researcher under the pseudo-name zawy12 as part of an extensive work on, and comparison of, difficulty adjustment algorithms [40]. LWMA is also an algorithm that adjusts difficulty at every block using a moving window of 60 blocks. However, contrary to DigiShield, LWMA uses a filter that smoothens out linearly. Subsequently, the impact of the most recent interblock times in the calculation of the new difficulty is much more significant than the impact of interblock times earlier within the window.

3 Background

3.1 Proof of Work (PoW) Mining

Hash rate is a measurement of a computational unit's ability to produce outputs of a hashing function. It is calculated as hashes per second and in blockchain terms it is the main method of evaluating computational power [34]. Therefore, throughout this paper, the terms hash rate and hash power will be used interchangeably.

PoW is based on the idea that mining a block is computationally demanding, so that no malicious user has the ability to mine blocks faster than the rest of the network. The only way a malicious user can do that consistently is by possessing more than 50% of network's hash rate [9].

Each valid block consists of a collection of transactions along with a header that contains important information about the block such as its timestamp and the root hash of transactions. One of the block header fields is called *nonce*. The sole purpose of a nonce is to pseudorandomly alternate the output of a hash function that takes block header as input. By requiring the hash of a block header to be smaller than a predefined threshold or *target*, the trivial process of creating a block becomes a challenging cryptographic puzzle called PoW mining [21, 33, 36]. The probability of a hash being smaller than or equal to the target is:

$$P(\text{hash} \leq \text{target}) = \frac{\text{target} + 1}{2^{256}}$$

since every 256-bit hash has 2^{256} possible values $[0, 2^{256} - 1]$

Also, each attempt to mine a block is a Bernoulli trial with a probability of success equal to $\frac{\text{target} + 1}{2^{256}}$. Thus, the expected work or the number of hashes to mine a single block is:

$$\mathbb{E}[W] = 2^{256} / (\text{target} + 1) \approx 2^{256} / \text{target}$$

The idea that a higher target means that it is more difficult to mine a block while a lower target means it is easier can be perceived as somewhat counter-intuitive. Thus, the notion of *difficulty* D was introduced. Difficulty is the scaled reciprocal of target and is calculated as:

$$D = \frac{2^{256}}{\text{target}}$$

Consequently, the expected number of hashes to mine a single block can also be expressed as:

$$\mathbb{E}[W] = D$$

If the real hash rate H_n at any given time was observable and since the ideal interblock time T is constant, setting the difficulty for each block would be trivial:

$$D_n = \mathbb{E}[W] = H_n \cdot T$$

However, in reality it is unattainable to observe the actual hash rate of the network *a priori*. In fact, it is impossible to precisely calculate the real hash rate even *a posteriori*. For this reason, DAAs are based (sometimes only implicitly) on an estimation $\widehat{H}_n$ of the network's hash rate.

Lastly, it should be noted that the initial target and difficulty D_0 of Bitcoin were approximately equal to 2^{224} and 2^{32} respectively. However, it is common practice to normalise difficulty values by dividing with 2^{32} so that $D_0 = 1$.

3.2 Difficulty Adjustment Algorithm (DAA)

A Difficulty Adjustment Algorithm (DAA) is a vital element of every PoW blockchain. Its main task is to control mining difficulty to modulate the time between the creation of every new block in the blockchain.

In the short term, a DAA should ensure a stable block-production rate and relatively steady transaction throughput that makes the blockchain dependable. The reliability of a blockchain improves user experience and is also reflected in its coin's value. In the long run, a DAA controls coin issuance, timing of the halvings, and miners' profit. For example, by lowering miners' reward, scarcity is maintained and this is a measure to protect from inflationary effects. Moreover, lowering supply increases demand and affects the price of the coins [12]. Regardless of supply curve choice, the DAA is responsible for long-run timing of blocks along that curve.

Today, more than fifteen years later, Bitcoin is the most well-established cryptocurrency and its network hash power is vast and invariant to minor oscillations. As a result, its primitive original DAA [33] that adjusts mining difficulty on a biweekly basis is sufficient. But, alas, not every PoW blockchain enjoys the same broad and stable support by miners as BTC. Almost all PoW altcoins are plagued by highly volatile hash power, mainly due to *coin-hopping* strategies, and cannot afford to wait a fortnight to alter mining difficulty.

Coin-hopping is when a miner switches the entirety or a part of his hash power from mining one coin to another. There are various factors that influence a miner's coin-hopping strategy, like mining difficulty or coin value, but the main motive is often single and clear, maximising profit. Since various PoW blockchains use the same mining mechanism, it is only a matter of flipping a switch for a miner to "hop" to mining the most profitable coin at any time [28].

When performed on a large scale, coin-hopping can cause serious fluctuations in an altcoin network's total hash rate and DAAs are required to be able to keep up with this. As a result, numerous DAAs were developed during the previous decade, many of which proved to be resistant to hash rate fluctuations and capable of maintaining relatively stable block-production rates. What was not accounted for though is the long-term effect of DAAs' inability to keep up with the general increase of hash power in blockchains, primarily due to technology advances.

3.3 Drift (Skew)

Satoshi Nakamoto foresaw the increasing hardware speed and mining interest back in 2008 when the Bitcoin whitepaper was issued [33]. Indeed, as you can see in Fig. 1, BTC's difficulty has a tendency to increase over the years. Furthermore, there are two obvious rapid jumps in difficulty. The first was around 2010 with the introduction of GPU mining, and the second was around 2013 with the introduction of ASIC mining.

Of course, BTC's DAA was able to cope with the difficulty increase and restore the order of block-production rate without significant impact on transaction throughput. However, as shown in Fig. 2, drift was gradually built up because – on average – blocks were mined faster than every 10 minutes. BTC's DAA remained the same and was never adjusted to mitigate the problem of drift. As a result, the 2nd, 3rd and 4th BTC halvings occurred 139, 234 and 252 days (respectively) ahead of schedule (see Fig. 2 or Table 3 in Appendix B) [8, 16, 24]. Meanwhile, when BCH forked out of BTC in August 2017, it inherited any already-accumulated drift. Its stormy quest to find a suitable DAA only made the drift problem worse. Thus, BCH's 3rd and 4th halvings happened 267 and 269 days (respectively) ahead of their ideal time (see Fig. 2 or Table 3 in Appendix B).

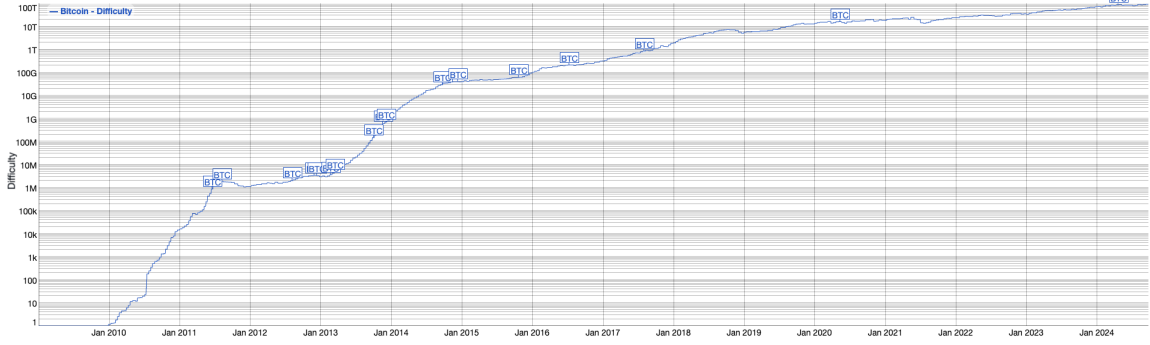


Fig. 1. BTC difficulty history, [Public domain], via BitInfoCharts. (<https://bitinfocharts.com/comparison/bitcoin-difficulty.html>).

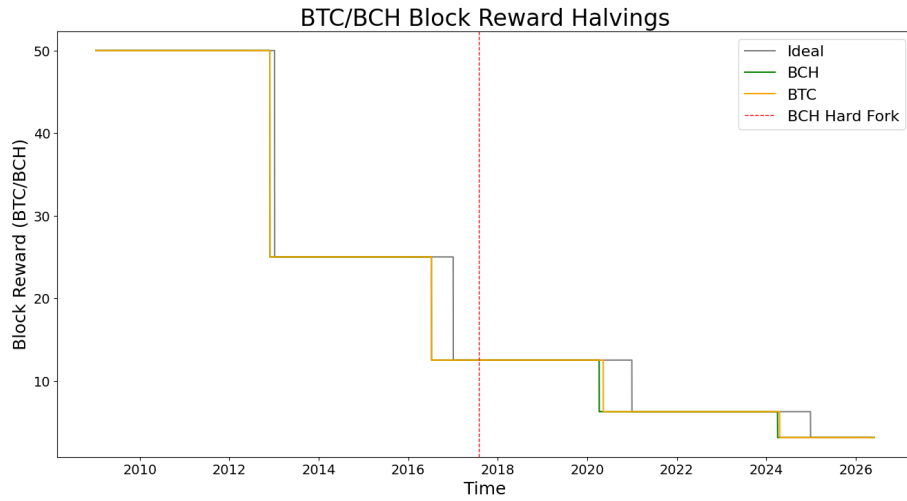


Fig. 2. BTC/BCH Block Reward Halvings.

4 Current and Legacy DAAs of BCH

4.1 BTC DAA

Prior to the August 2017 fork, Bitcoin Cash used Bitcoin's DAA according to which the mining difficulty is updated every 2016 blocks. The DAA formula prevents extreme changes in difficulty by capping adjustments, allowing the new difficulty to be no more than four times higher or lower than the previous value [35]. This DAA has allowed Bitcoin network to maintain a stable block-generation rate of approximately one block every ten minutes. The formula is:

$$D_{new} = D_{current} \cdot \max \left(\min \left(\frac{2016 \cdot T}{t}, 4 \right), 0.25 \right) \quad (1)$$

where:

- D_{new} is the difficulty of the next 2016 blocks.
- $D_{current}$ is the difficulty of the previous 2016 blocks.

- T is the ideal inter-block time and is equal to 600 seconds.
- t represents the time difference in seconds between the timestamps of the latest block (B_n) and the 2016th previous block (B_{n-2015}).
- The factor is constrained between 0.25 and 4 to prevent drastic changes.

As shown in Figs. 3–4, this DAA works very well for BTC. The probability density function of interblock times converges to an exponential distribution, the probability mass function of number of blocks per hour is converging to a Poisson distribution and there is no autocorrelation between interblock times (i.e. correlation between the time series of interblock times and a delayed copy of itself). As explained in previous sections though, it is not viable for most altcoins and it does not solve the drift problem.

4.2 Emergency DAA (EDA)

In anticipation of the BTC–BCH fork, the BCH community decided that a new DAA was needed to ensure the survival of Bitcoin Cash. Although it was impossible to accurately predict the post-fork hash power of the new blockchain, it was certain that BTC would keep the lion’s share. There was also a great uncertainty about the effect of the split on BCH’s value [43]. Hence, Emergency DAA (EDA) was an adjustment to the original DAA that lowered difficulty immediately if there was a huge drop in hash power. More specifically, if the interval between the median time past (MTP) of the current block n and the MTP of block $n - 6$ was greater than 12 hours, the difficulty would drop directly by 20% [35].

4.3 CW-144

Bitcoin cash survived the split with the help of EDA DAA but it did not come away intact. As anticipated, the network’s hash power experienced a free fall and left the blockchain vulnerable to fluctuations. Figs. 3–4 reveal the extent of hash rate fluctuation and the desperate need for a more stable DAA. Amaury Sechet’s CW-144 [4] was promising to provide just that. CW-144 was activated on November 2017, only three months after the activation of the EDA.

CW-144 adjusted difficulty after each block and used a 144-period simple moving average to estimate hash rate [35]. The formula used to calculate the new difficulty for each block is:

$$D_n = \widehat{H}_n \cdot T = \frac{\sum_{i=old}^{new} D_i}{T_{elapsed}} \cdot T \quad (2)$$

where:

- $\widehat{H}_n$ is the estimated hash rate after mining the n -th block.
- T is the ideal time between two blocks, i.e., 10 minutes.
- B_{new} is the block with the median timestamp between the latest block B_n and the previous two B_{n-1} and B_{n-2} .
- B_{old} is the block with median timestamp between blocks B_{n-144} , B_{n-145} and B_{n-146} .
- $T_{elapsed}$ is the elapsed time between B_{new} and B_{old} .

As shown in Fig. 4, CW-144 achieved an average interblock time close to optimal. However, it caused a serious autocorrelation issue. In simple words, CW-144 caused large and consistent oscillations in difficulty throughout the day. This created brief periods of low difficulty and large numbers of new blocks, and large periods of high difficulty and low number of new blocks. As a result, not only did it not protect BCH from coin-hopping but it made it more vulnerable to it instead. On top of that, the BCH–BSV fork happened, causing even more stability issues to the already tormented BCH network, as can be seen in Fig. 3.

4.4 ASERT/NEFDA

By 2020 it was obvious that BCH's DAA was suffering and was in urgent need of an upgrade. A few different solutions were put forward, and after a heated debate within BCH community, it was decided to adopt a proposal that used a negative exponential filter. Two very similar proposals were submitted independently by two separate teams under different names, ASERT [23, 30] and NEFDA [25].

NEFDA's formula is:

$$D_n = \widehat{H}_n \cdot T = \left[\frac{1}{S} \sum_{i=0}^{n-1} D_i e^{\frac{t_i - t_n}{S}} \right] \cdot T \quad (3)$$

where:

- $\widehat{H}_n$ is the estimated hash rate after mining the n -th block.
- T is the ideal time between two blocks, i.e., 10 minutes.
- S is the smoothing factor.
- D_i is the difficulty after mining block i .
- t_i is the timestamp of block i .

Also, [25] shows that it can be simplified to:

$$\begin{aligned} D_n &= D_{n-1} \cdot \exp\left(\frac{T - (t_n - t_{n+1}))}{s}\right) \\ D_n &= D_0 \cdot \exp\left(\frac{nT - (t_n - t_0)}{s}\right) \\ D_n &= D_0 \cdot \exp\left(-\frac{\text{drift}}{s}\right) \end{aligned} \quad (4)$$

where:

- *drift* is the accumulated drift since the genesis block.

Figs. 3–4 show that NEFDA (ASERT) helped stabilise the BCH hash rate, brought the block time distribution much closer to ideal, and solved the autocorrelation problem.

However, although the drift term appears in (4), it appears purely *descriptively* as a tool to help estimating hash rate. (By contrast, for the following algorithms, drift appears *prescriptively* as a problem to be mitigated and kept under control as best as possible.) Additionally, the blockchain is still exposed to further drift accumulation with each next leap in mining technology [29].

5 Introducing NEF(DA)^N

5.1 NEF(DA)²

In [29] we presented NEF(DA)², a modified version of NEFDA that addresses the drift problem but also inherits NEFDA's desired properties such as effective mining difficulty adjustment, stable block-production rate and no-long-term autocorrelation. Furthermore, NEF(DA)² achieves this with a negligible increase in computational complexity.

NEF(DA)² uses a modified version of NEFDA's initial formula (see (3) in Section 4.4). For NEFDA's formula, the estimated hash rate of the network $\widehat{H}_n$ is multiplied by the ideal time between blocks T (which is equal to 10 minutes). For NEF(DA)², the constant T is replaced by a variable T_n that depends on the accumulated drift.

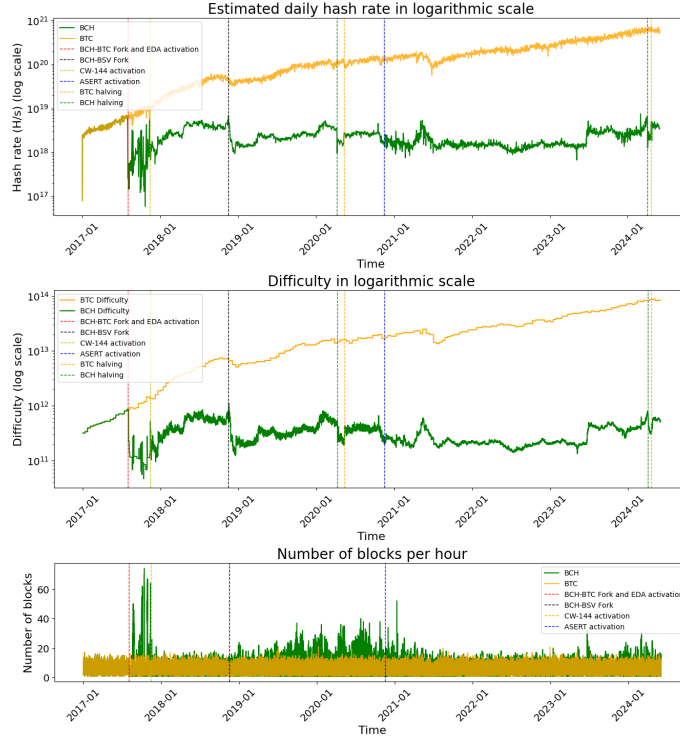


Fig. 3. BTC-BCH historical data comparison.

The formula of $\text{NEF}(\text{DA})^2$ is:

$$D_n = \widehat{H}_n \cdot T_n = \left[\frac{1}{S} \sum_{i=0}^{n-1} D_i e^{\frac{t_i - t_n}{S}} \right] \cdot T e^{\frac{t_0 - t_n + nT}{S}} = \left[\frac{1}{S} \sum_{i=0}^{n-1} D_i e^{\frac{t_i - t_n}{S}} \right] \cdot T e^{\frac{-\text{drift}}{S}} \quad (5)$$

In [29] we proved mathematically and experimentally that $\text{NEF}(\text{DA})^2$ is effective. However, the recursive formula used to calculate the difficulty of a block using the difficulty of the previous block is somewhat convoluted and not very intuitive. Additionally, for a hash rate that increases exponentially (let alone super-exponentially), $\text{NEF}(\text{DA})^2$ cannot solve the drift problem, and it leaves a residual that allows the accumulation of more drift. Lastly, because the formula is convoluted, it cannot easily be generalised and adapted to withstand hash rate functions that increase super-exponentially.

In this paper, we present a generalised algorithm that eliminates the drift problem for any realistic scenario of hash rate increase. The generalised algorithm introduces the concept of drift correction levels that can be chained one after the other to enhance the algorithm's ability to alleviate the drift problem. NEFDA and $\text{NEF}(\text{DA})^2$ can be categorised as special cases of the generalised algorithm that have 1 and 2 levels respectively.

Lastly, due to the innovative notion of *liquid heights*, the formula for $\text{NEF}(\text{DA})^2$ and any other $\text{NEF}(\text{DA})^n$ becomes very elegant and comprehensible. Moreover, the methodology for deriving the formula is much more intuitive and reflects programming code that is shorter and more optimised.

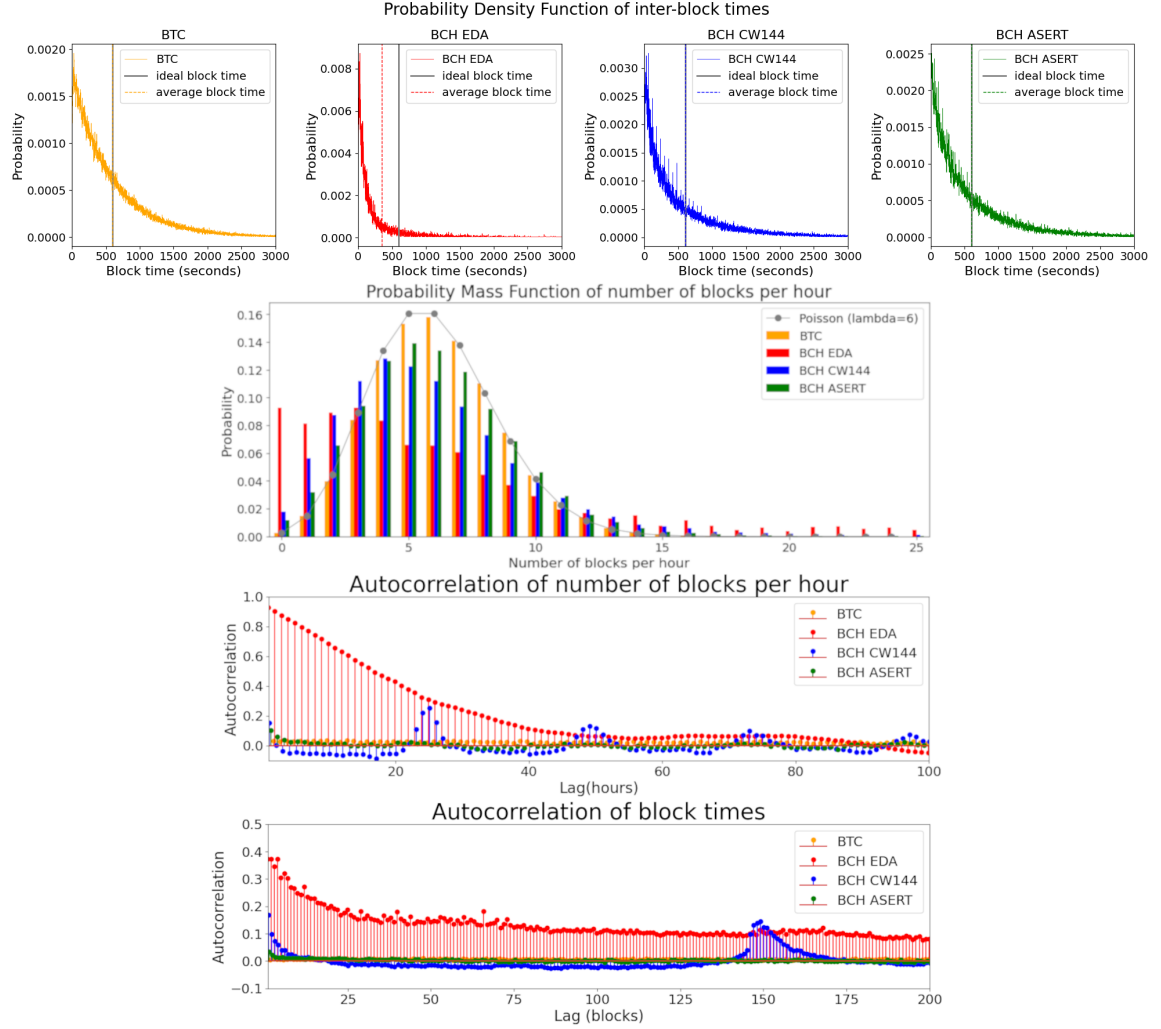


Fig. 4. DAA block time, and autocorrelation comparisons.

5.2 Mind the gap

As seen in Section 3.1, an effective DAA is constantly trying to estimate network's hash rate $\widehat{H}_n$ and match the difficulty D_n which are linked with: $D_n = \widehat{H}_n \cdot T$. To allow direct comparison, from now on we will be using the normalised terms *relative hash rate* $\tilde{H}$ and *relative difficulty* $\tilde{D}$ given by:

$$\tilde{H} = \widehat{H}/H_0$$

$$\tilde{D} = D/D_0$$

where:

- $H_0 = \frac{D_0}{T} \approx 7\,000\,000$ hashes/s is the initial hash rate of the Bitcoin network.

- $D_0 = 2^{32}$ is the initial difficulty of the Bitcoin network.

Let us assume a simple case with NEFDA algorithm and a network's hash rate that is constant and equal to 1, instantaneously increases by a factor of 30 and then remains constant at that value. Fig. 5 shows how drift is accumulated because difficulty $\tilde{D}$ lags behind hash rate $\tilde{H}$. This creates a gap between $\tilde{H}$ and $\tilde{D}$ (red area), yielding the (multiplicative) residual $\tilde{H}/\tilde{D}$.

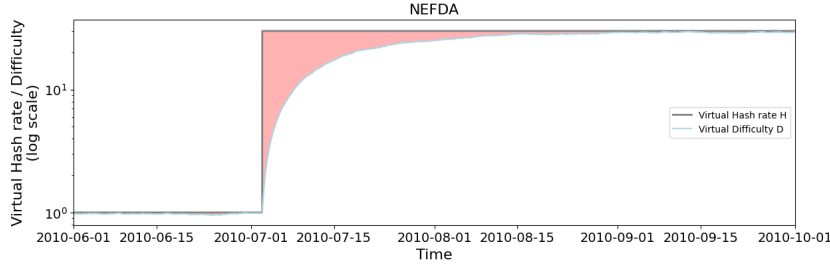


Fig. 5. $\tilde{D}$ and $\tilde{H}$ comparison for NEFDA.

5.3 Unlocking levels

NEFDA algorithm leaves a residual that causes drift accumulation. In [29] we addressed this problem with an ad-hoc adjustment to the code. The key idea of the generalised algorithm is that instead of adjusting the code, the residual can be dealt with by running the same algorithm again. Each instance of the same algorithm is given a *level* of its own.

Instead of tracking the estimated hash rate $\tilde{H}$, each new level tries to match the residual $\tilde{H}/\tilde{D}$ left from the previous level. The same process can be repeated and more levels can be added, each one on the residual left by the previous ones (see Fig. 6). Additionally, as can be seen in Figure 6, the green areas have the reverse effect and they eliminate residual and therefore drift. By repeating the process and adding more levels, drift can quickly be eliminated even for hash rate functions that are steeply increasing.

Some important notes about levels:

- Each level attempts to eliminate the residual left from all previous levels. We refer to this residual as the level's *virtual hash rate* $\tilde{H}_l$.
- Each level thinks that it is unique and sees $\tilde{H}_l$ as the estimated hash rate that it needs to match with its difficulty. We refer to this difficulty as *virtual difficulty* $\tilde{D}_l$ (see Fig. 6).
- The residual left by each level $\tilde{H}_l/\tilde{D}_l$ is the input $\tilde{H}_{l+1}$ of the next level.
- Each level has a smoothing factor ω_l that controls the level's responsiveness.
- The *real difficulty* D_l of a level l is calculated as the product of the initial difficulty D_0 and all the $\tilde{D}_l$ up to level l (i.e. thereby undoing all the divisions that created the virtual world that this level lives in).
- The *final difficulty* D_L given to miners is equal to the real difficulty of the last layer.

5.4 Liquid heights

The mining difficulty D represents the amount of work put by miners to mine a new block (see Section 3.1). The generalised algorithm consists of the same algorithm applied to many levels, but only the last of those levels corresponds

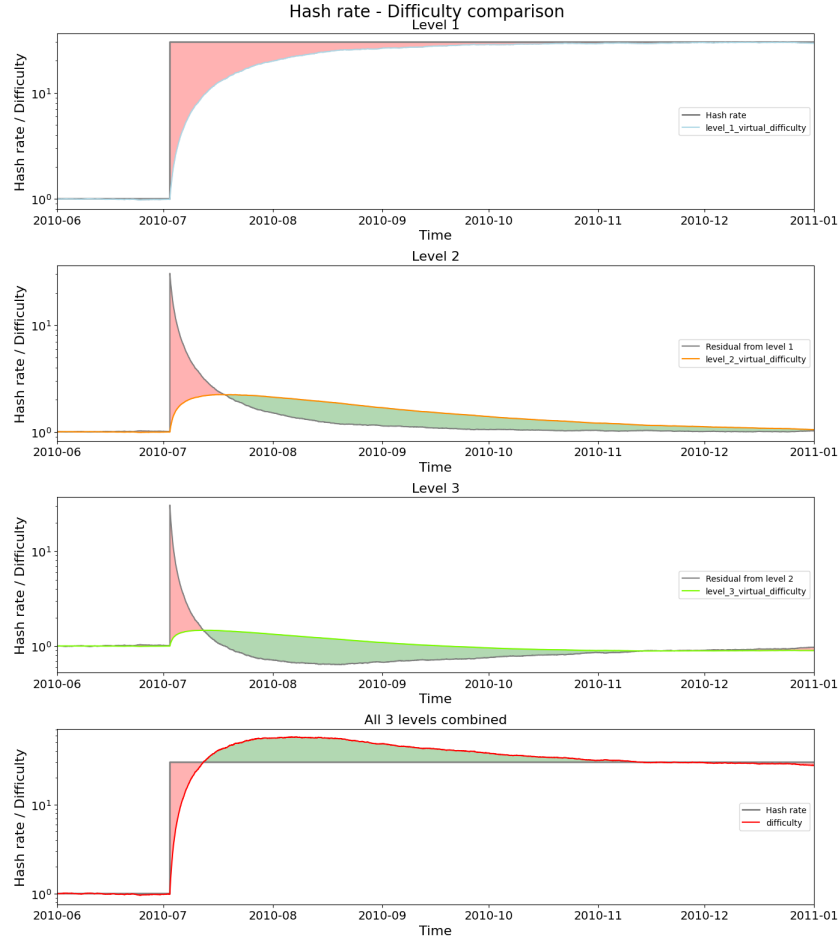


Fig. 6. Hash rate – Difficulty comparison for a 3-level algorithm

to the real blockchain. Therefore, miners mine blocks according to the difficulty D_L of the last level, and each new block n corresponds to D_L^n amount of work.

However, each level receives the same amount of work that corresponds to the amount of work required to mine a new block at the last level. The problem is that intermediate levels have different real difficulties D_l and generally $D_l \neq D_L$, for $l < L$. In other words, intermediate levels expect to receive work that corresponds to their own real difficulty D_l but instead receive work that corresponds to D_L .

The solution to this problem is to perceive block heights of intermediate levels as real numbers. Instead, **for every new block n of the last level, the height of each intermediate level will increase by a non-integer value equal to D_L^n/D_l^n** . This amount of blocks, or *block fluid*, is equal to the amount of blocks that D_L amount of work would have created with D_l difficulty. This is the definition of a *liquid height*.

Liquid heights are illustrated in Fig. 7. On the left, we illustrate the traditional definition of block height. It is presented as a tower of containers stacked on top of each other. Each container has different width that corresponds to the mining

difficulty for that block but all containers have exactly the same height, equal to 1 unit or block. Every time a new container is added, the total height increases by 1 unit.

On the right side of Fig. 7, we illustrate liquid heights as tubes that contain block fluid. Additionally, the tubes have varying cross-sections and they are different from each other, this simulates the real difficulties D_l^n .

Each time a new block is mined for the last level, the same amount of work fluid, which corresponds to D_L , is added to every tube. Nevertheless, since each tube has different cross-section (corresponding to the level's real difficulty at the time), the liquid-level rises differently for each tube. Only in the last tube the liquid-level is guaranteed to increase at a constant rate of exactly 1 unit at a time. The liquid-level of all the other tubes increases by a non-integer amount.

The introduction of liquid heights allows us to calculate the individual *drift*_{*l*} for each level *l*. This unlocks the possibility of significantly simplifying the mathematical expressions and the programming code for the generalised algorithm and is a major improvement to the previously presented NEF(DA)² [29].

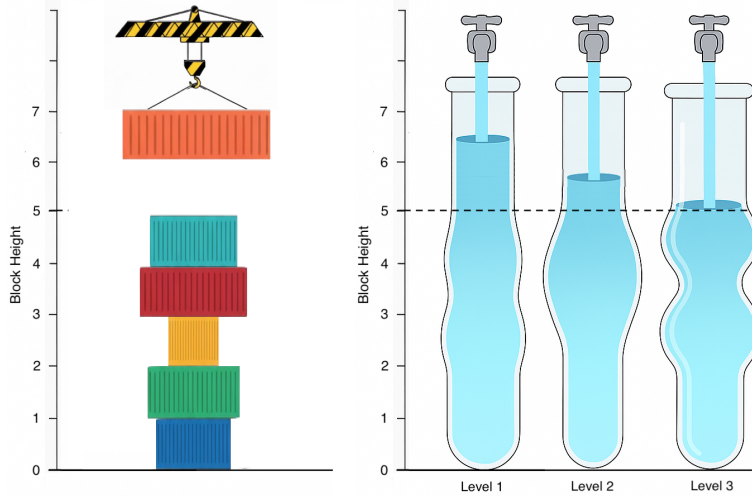


Fig. 7. Illustration of liquid heights.

5.5 NEF(DA)^N formulae

The idea of NEF(DA)^N is to apply L levels of NEFDA's algorithm. Note that throughout this section, we will be using the notation D to refer to *real difficulties* and $\tilde{D}$ to refer to *virtual difficulties*, i.e. the difficulties as perceived by the level without knowledge about the existence of other levels. At any given time t_n , each level l has a real difficulty D_l^n , a virtual difficulty $\tilde{D}_l^n$ (detailed in Section 5.3), alongside a height h_l^n and a drift $drift_l$ (detailed in Section 5.4). From NEFDA's formula 4 and the observations from Sections 5.3 and 5.4, we derive the following formulae for each level:

$\forall \text{ level } l \in [1, L]:$

$$drift_l = (t_n - t_0) - (h_l^{n-1} \cdot T) \quad (6)$$

$$\tilde{D}_l^n = \exp(-drift_l \cdot \omega_l) \quad (7)$$

$$D_l^n = \begin{cases} D_0 \cdot \tilde{D}_l^n, & l = 1, \\ D_{l-1}^n \cdot \tilde{D}_l^n, & l > 1, \end{cases} \quad (8)$$

$$h_l^n = h_l^{n-1} + D_L^n / D_l^n \quad (9)$$

where:

- t_n is the current time
- ω_l is the level's smoothing factor given as a frequency (reciprocal seconds).
- h_l^{n-1} is the height of the previous block produced in level l
- $\tilde{D}_l^n$ is the virtual difficulty of level l at time n
- D_l^n is the real difficulty of level l at time n
- D_L^n is the real difficulty of the last level L and the one given to miners at time n

for $L = 1$, from formulae 6 - 9 we get:

$$drift_1 = (t_n - t_0) - h_1^{n-1} \cdot T$$

$$\tilde{D}_1^n = \exp(-drift_1 \cdot \omega_1)$$

$$D_1^n = D_0 \cdot \tilde{D}_1^n = D_0 \cdot \exp(-drift_1 \cdot \omega_1)$$

$$h_1^n = h_1^{n-1} + 1$$

which are equivalent to the formulae of NEFDA [25]. Therefore, **NEF(DA)^N with L=1 is equivalent to NEFDA.**

for $L = 2$, from formulae 6 - 9 we get:

$$D_1^n = D_0 \cdot \tilde{D}_1^n$$

$$D^n = D_2^n = D_1^n \cdot \tilde{D}_2^n = D_0 \cdot \tilde{D}_1^n \cdot \tilde{D}_2^n$$

$$h_1^n = h_1^{n-1} + D_2^n / D_1^n = h_1^{n-1} + \tilde{D}_2^n$$

$$h_2^n = h_2^{n-1} + 1$$

which are equivalent to the formulae of NEF(DA)² [29] (see Appendix A). Therefore, **NEF(DA)^N with L=2 is equivalent to NEF(DA)².**

Generally, for L levels we get:

$$\begin{aligned} D_l^n &= D_0 \times \tilde{D}_1^n \times \tilde{D}_2^n \times \dots \times \tilde{D}_l^n \\ &= D_0 \cdot \prod_{i=1}^l \tilde{D}_i^n \\ &= D_0 \cdot \prod_{i=1}^l \exp(-drift_i \cdot \omega_i) \\ &= D_0 \cdot \exp\left(-\sum_{i=1}^l drift_i \cdot \omega_i\right) \end{aligned} \quad (10)$$

$$\begin{aligned}
h_l^n &= \begin{cases} h_l^{n-1} + D_L^n / D_l^n & , \quad l < L, \\ h_l^{n-1} + 1, & l = L \end{cases} \\
&= \begin{cases} h_l^{n-1} + \tilde{D}_{l+1}^n \cdot \tilde{D}_{l+2}^n \cdot \dots \cdot \tilde{D}_L^n & , \quad l < L, \\ h_l^{n-1} + 1, & l = L \end{cases} \\
&= \begin{cases} h_l^{n-1} + \prod_{i=l+1}^L \tilde{D}_i^n & , \quad l < L, \\ h_l^{n-1} + 1, & l = L \end{cases} \\
&= \begin{cases} h_l^{n-1} + \exp\left(-\sum_{i=l+1}^L \text{drift}_i \cdot \omega_i\right) & , \quad l < L, \\ h_l^{n-1} + 1, & l = L \end{cases}
\end{aligned} \tag{11}$$

5.6 Performance analysis

In this section, we examine the performance of NEF(DA)^N and explain how each additional level eliminates drift by an exponential nesting order. Since some of the hash rates discussed in this analysis are extremely large (such as triply-nested or arbitrarily multiply-nested exponential functions), we make a continuous time assumption instead of blocks separated by discrete time gaps. The simulation section that follows studies more realistic hash rate functions and uses discrete time which seems to be in agreement with the continuous time results of this section. As for some of the unrealistically gigantic hash rate functions discussed here, they may not be of significant practical relevance, but they are included as theoretical boundary cases to demonstrate that the (continuous-time idealised) algorithm copes even with them. Meanwhile, it is not impossible for a network to briefly experience such extreme periods of hash rate growth (e.g., during a leap in mining technology), yet our algorithm stays on schedule even during these rare scenarios.

As presented in [29], NEFDA (i.e. level 1) accumulates drift even for a linearly increasing hash rate. Moreover, NEF(DA)² (i.e., level 2) can eliminate drift for up to polynomial hash rate functions. For exponentially increasing hash rate functions, NEF(DA)² can only eliminate drift growth and drift converges to a constant number. As for nested exponential hash rate functions (such as $H(t) = e^{e^{\text{poly}(t)}}$), NEF(DA)² breaks down completely.

To measure the effectiveness of each level of the generalised DAA, we begin with the following assumptions:

$$\tilde{H}_l \cdot T = e^{f(t)} \cdot g(t)$$

(We use $\tilde{H}_l \cdot T$ rather than $\tilde{H}_l$ to make it a pure number instead of a reciprocal of time.)

$$\tilde{D}_l = e^{f(t)}$$

Under this assumption, the residual $\tilde{H}_{l+1} = \tilde{H}_l / \tilde{D}_l$ (multiplied by T) is just $g(t)$ so $e^{f(t)}$ has been annihilated by this level. Now we solve for $f(t)$; if there is no solution, or if the only $f(t)$ allowed is logarithmic w.r.t. $g(t)$ – thereby undoing

Table 1. Drift accumulation per hash rate function and number of levels

Hash rate	NEFDA 1 level	NEF(DA) ² 2 levels	NEF(DA) ³ 3 levels	NEF(DA) ⁴ 4 levels
Step function	$\rightarrow c$	$\rightarrow 0$	$\rightarrow 0$	$\rightarrow 0$
$O(t)$	$O(\ln t)$	$\rightarrow 0$	$\rightarrow 0$	$\rightarrow 0$
$O(t^2)$	$O(\ln t)$	$\rightarrow 0$	$\rightarrow 0$	$\rightarrow 0$
$O(e^t)$	$O(t)$	$\rightarrow c$	$\rightarrow 0$	$\rightarrow 0$
$O(e^{e^t})$	$O(e^t)$	$O(t)$	$\rightarrow c$	$\rightarrow 0$
$O(e^{e^{e^t}})$	$O(e^{e^t})$	$O(e^t)$	$O(t)$	$\rightarrow c$

the exponential we were hoping for – then we are *not* removing one exponential nesting order. If on the other hand $f(t)$ is of the same broad class as $g(t)$ (e.g., a polynomial of different degree), then the level has successfully annihilated an exponential nesting order.

$$\frac{dh_l}{dt} = \frac{\tilde{H}_l}{\tilde{D}_l} = \frac{\tilde{H}_l T}{\tilde{D}_l} \cdot \frac{1}{T}$$

From (6):

$$\frac{d}{dt} \text{drift}_l = 1 - \frac{\tilde{H}_l T}{\tilde{D}_l}$$

Moreover, remembering (7):

$$\begin{aligned} \tilde{D}_l &= \exp(-\text{drift}_l \cdot \omega_l) \\ \ln \tilde{D}_l &= -\text{drift}_l \cdot \omega_l \\ \frac{d}{dt} \ln \tilde{D}_l &= \left[\frac{\tilde{H}_l T}{\tilde{D}_l} - 1 \right] \cdot \omega_l \\ \frac{df(t)}{dt} &= \left[\frac{\tilde{H}_l T}{\tilde{D}_l} - 1 \right] \cdot \omega_l \\ \frac{df(t)}{dt} &= [g(t) - 1] \cdot \omega_l \\ f(t) &= \omega_l \cdot \int [g(t) - 1] dt \end{aligned}$$

Therefore, $f(t)$ and $g(t)$ can be of the same broad class and the level successfully downgrades $\tilde{H}_l$ to a $\tilde{H}_{l+1}$ of an exponential nesting order smaller than $\tilde{H}_l$. Additionally, from formula (7): $\text{drift}_l = -\ln(\tilde{D}_l)/\omega_l$. Thus each level eliminates drift by an exponential nesting order which aligns with the findings of [29]. Hence, to eliminate drift for hash rate functions that increase at an exponential rate or higher, a NEF(DA)^N DAA of at least 3 levels is required. Table 1 illustrates drift accumulation for different numbers of levels and different orders of hash rate functions.

Theoretically, there is no limit on the number of levels that can be added. In practice, there is most likely no need for a DAA of more than 3 levels, as a scenario of long-term nested exponential hash rate growth is unrealistic.

6 Simulation

In this section we present the results of the simulations we designed to test the behaviour of NEF(DA)^N. Similarly to a real environment, the DAA cannot observe the hash rate and must adjust the mining difficulty based only on the

previous interblock times. The simulations compare the behaviour of $\text{NEF}(\text{DA})^N$ with BTC DAA (see Section 4.1), DigiShield DAA (see Section 2.3), and LWMA DAA (see Section 2.4). Each simulation contains 4 different versions of $\text{NEF}(\text{DA})^N$, a single-level, a double-level, a triple-level, and a quadruple-level. Each simulation consists of 30 runs of all 7 algorithms with the same unobservable hash rate function for a duration of 16 years from 2009 to 2025. The initial simulations are run with a noiseless and predetermined hash rate function, while the following ones use a hash rate function that adapts to difficulty changes, replicating a more realistic mining behaviour.

The simulations produce data (such as timestamps, difficulties) and metadata (such as interblock times, unobservable hash rate, virtual difficulties and drift). The visualisation of data and metadata is split into two parts. The first part is the sanity check that ensures that $\text{NEF}(\text{DA})^N$ DAAs inherit the properties of NEFDA, such as effective difficulty adjustment, stable block-production rate, and no long-term autocorrelation. Once the sanity check is over and we have verified the correct behaviour of the $\text{NEF}(\text{DA})^N$ DAAs, we plot the diagrams for the hash rate, the difficulty, and the drift. These diagrams allow comparison between the DAAs and verify the findings of Section 5.6. We also present the mean final drift accumulated with 95% confidence intervals in Table 2. Curious readers are encouraged to investigate the GitHub repository [17] and experiment with custom hash rate functions.

6.1 Noiseless simulations

Step hash rate function. This hash function (see Fig. 9) has a single instant increase by a factor of 30 in 2010 and is the same function discussed in Sections 5.2 and 5.3. The sanity check graphs for this simulation shown in Fig. 8 verify experimentally that all DAAs have a stable block-production rate without long-term autocorrelation. To avoid repetitiveness, sanity check graphs will be omitted for the following simulations. Also, in alignment with [29] and Table 1, single-level $\text{NEF}(\text{DA})^N$ cannot eliminate the drift which converges to a constant, while even one additional level is sufficient to solve the problem (see Fig. 9). BTC DAA behaves similarly to the single-level $\text{NEF}(\text{DA})^N$ while DigiShield and LWMA have very volatile difficulty and, counter-intuitively, develop a positive drift which is further discussed in the constant hash rate simulation.

Constant hash rate function. This hash function (see Fig. 10) retains the same hash rate throughout the simulation. All 4 $\text{NEF}(\text{DA})^N$ and BTC DAA behave very similarly, with minimal difficulty fluctuations. On the other hand, the difficulty of DigiShield and LWMA fluctuates violently which also leaves a residual bias that leads to the accumulation of positive drift (see Table 2). BTC DAA also has a slight residual bias leading to some positive drift. Therefore, just due to the random nature of block mining and although the mining power of the network is stable, these three algorithms cannot achieve a perfectly stable block-production rate unlike $\text{NEF}(\text{DA})^N$. For DigiShield and LWMA specifically, the difficulty also fluctuates wildly during the day, leaving the blockchain vulnerable to coin-hopping. This behaviour is a consequence of the very short moving window that the two algorithms use and makes them over-responsive.

Exponential hash rate function with step. This function (see Fig. 11) simulates a more aggressive Moore’s law where hash rate power doubles approximately every 3 months and has an initial 30x jump in 2010. It follows the formula:

$$H(t) = \begin{cases} H_0 & \text{if } t < t_{2010.5} \\ 30 \cdot H_0 \cdot e^{\frac{t-t_{2010.5}}{10 \cdot \text{FORTNIGHT}}} & \text{if } t \geq t_{2010.5} \end{cases}$$

This simulation confirms that $\text{NEF}(\text{DA})^N$ with at least three levels can annihilate drift for an exponential hash rate while the drift of a double-level converges to a constant and single-level’s becomes linear as described in Table 1. Also,

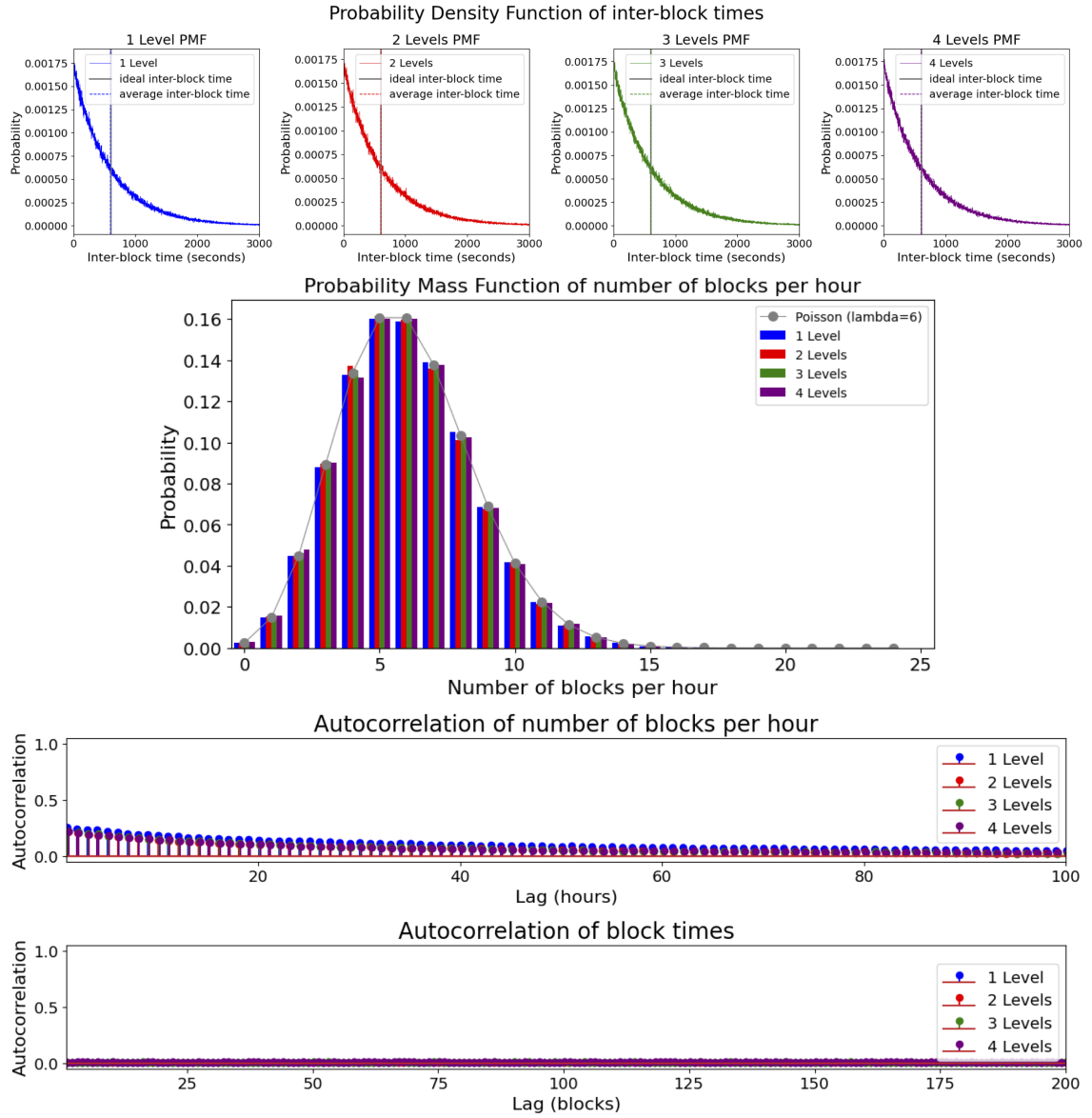


Fig. 8. Sanity check graphs for noiseless step hash rate.

according to Table 2, BTC DAA's drift accumulation verifies the arguments in Section 3.3. Finally, the over-responsive DigiShield and LWMA retain their residual bias and somewhat mitigate the drift problem. However, this is coincidental and comes with vulnerability to coin-hopping strategies, further discussed in the Adaptive hash rate simulations.

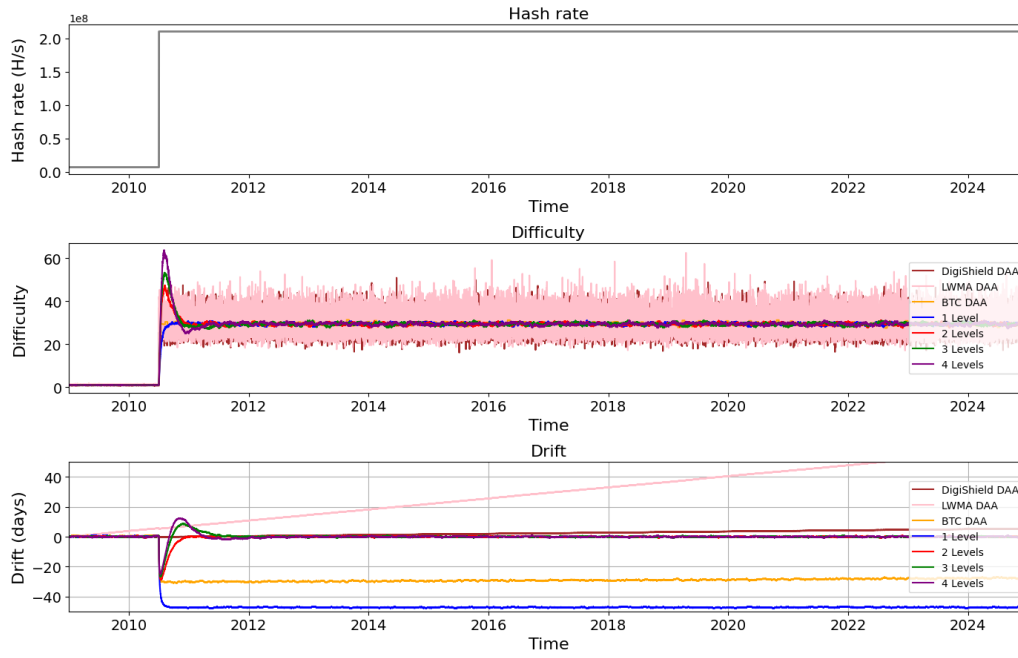


Fig. 9. Comparison for noiseless step hash rate.

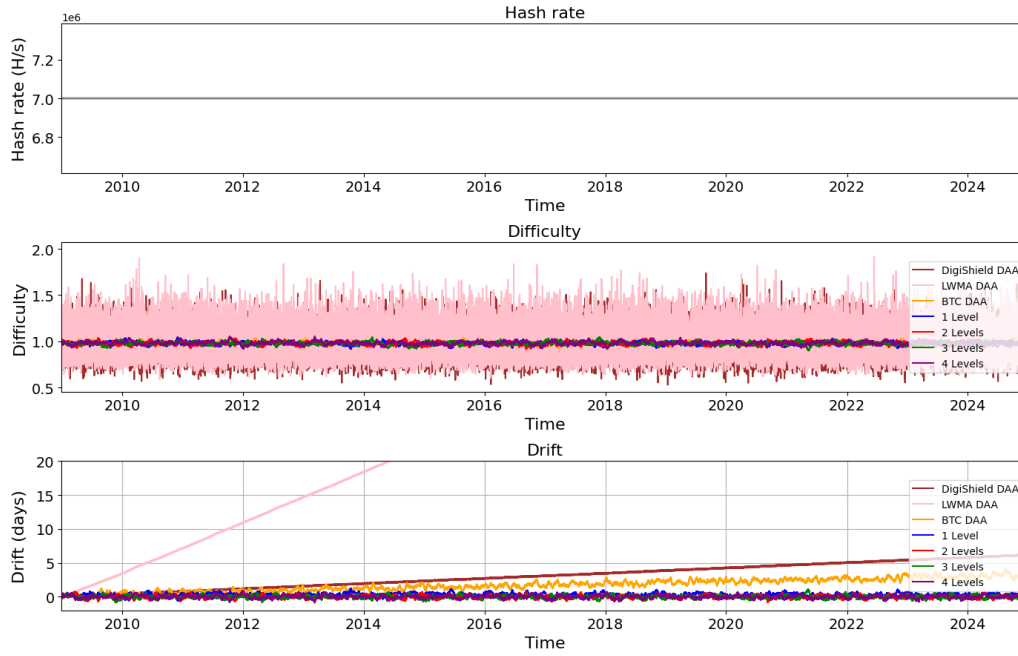


Fig. 10. Comparison for noiseless constant hash rate.

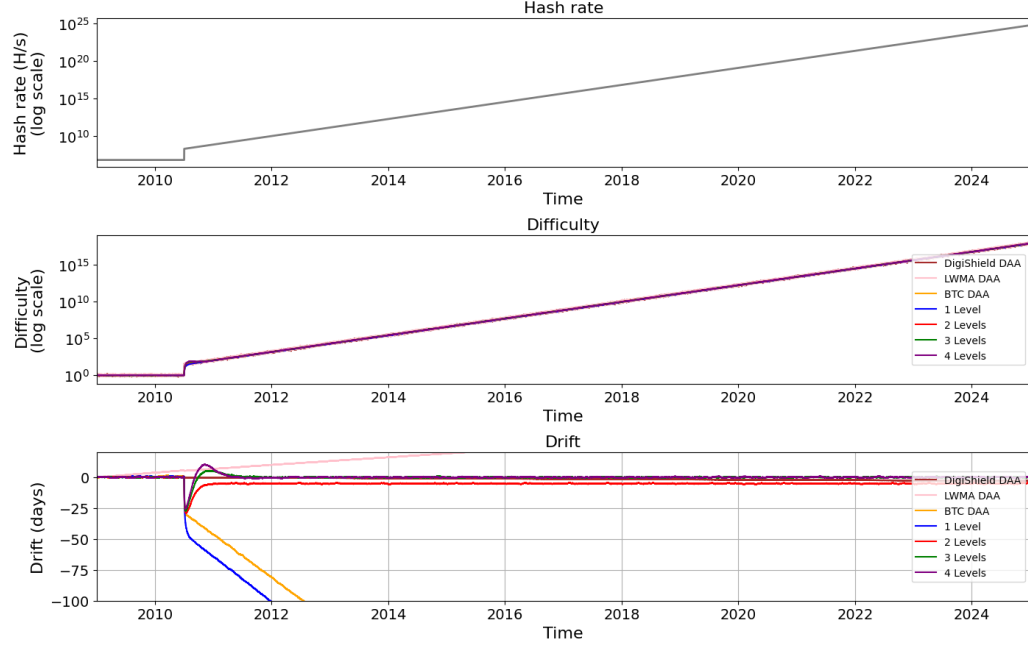


Fig. 11. Comparison for noiseless exponential hash rate function with step in 06/2010.

Super-Exponential hash rate function with step. This is an exponential of quadratic hash rate function with a 30x jump in 2010 (see Fig. 12). It follows the formula:

$$H(t) = \begin{cases} H_0 & \text{if } t < t_{2010.5} \\ 30 \cdot H_0 \cdot e^{\frac{t-t_{2010.5}}{10 \cdot \text{FORTNIGHT}} + \left(\frac{t-t_{2010.5}}{15 \cdot \text{FORTNIGHT}}\right)^2} & \text{if } t \geq t_{2010.5} \end{cases}$$

This growth rate is probably beyond realistic for a long period of time and serves mainly as a stress test, yet the simulation displays how powerful NEF(DA)^N is, even with just 3 layers. The drift graph of Fig. 12 and Table 2 show that the 3-level and the 4-level algorithm can achieve an impressive behaviour eliminating drift while the other two fail. Meanwhile, BTC DAA, DigiShield and LWMA cannot keep up with the growth of the hash rate.

6.2 Adaptive hash rate simulations

In the real world, miners have their own incentives and are sensitive to changes in difficulty, coin value, and electricity prices. To simulate this strategic behaviour, we developed an adaptive hash rate function. More specifically, we consider three different classes of miners: *loyal miners*, *variable miners*, and *greedy miners* with hash rates H_l , H_v , and H_g respectively. The total maximum available hash rate is controlled by a predetermined hash rate function but at any given moment, the network's current hash rate is given by:

$$H = H_l + H_v + H_g \quad (12)$$

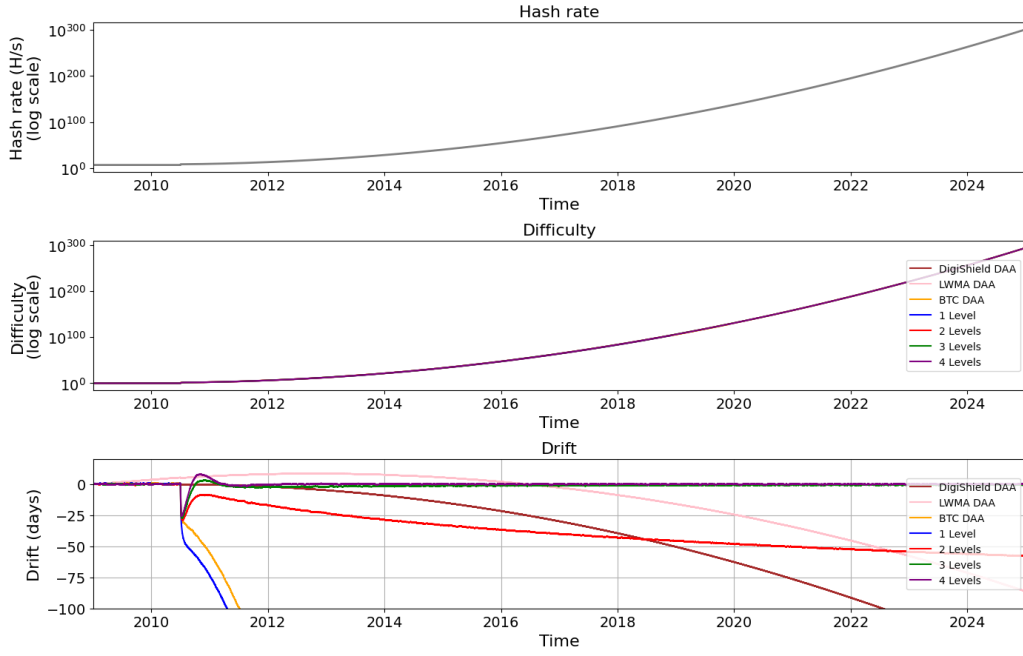


Fig. 12. Comparison for noiseless super-exponential hash rate function with step in 06/2010.

$$H_v = \begin{cases} 4 \cdot H_l, & \text{if } R_D \leq 1 - \epsilon \\ \frac{4 \cdot H_l}{1 + e^{-\frac{1-R_D}{\epsilon^2}}}, & \text{if } 1 - \epsilon < R_D \leq 1 + \epsilon \\ 0, & \text{otherwise} \end{cases} \quad (13)$$

$$H_g = \begin{cases} 4 \cdot H_l, & \text{if } R_D \leq 1 - \epsilon/3 \\ 0, & \text{otherwise} \end{cases} \quad (14)$$

where:

- R_D is the ratio of the average difficulty of the latest 6 blocks divided by the average difficulty of the 6 previous.
- H_l is following a predetermined function similar to the hash rate function of the noiseless simulations.
- $\epsilon = 0.05$

Adaptive step hash rate function. This hash rate function uses the same base as the noiseless step hash rate simulation, but this time miners can adapt to difficulty fluctuations and therefore the total hash power of the network becomes very unpredictable. Fig. 13 illustrates vividly how robust NEF(DA)^N is to coin-hopping strategies and how vulnerable the other 3 algorithms are. It is also a good indication that even Bitcoin itself could have been suffering from coin-hopping if its total network hash rate was not massive compared to the rest of PoW blockchains.

Exponential hash rate function with step. In this simulation, the base hash rate function is the same as the one of the noiseless exponential with step. Although not clearly visible due to the scale of the graphs (see Fig. 14), this simulation once again verifies the robustness of NEF(DA)^N against coin-hopping strategies.

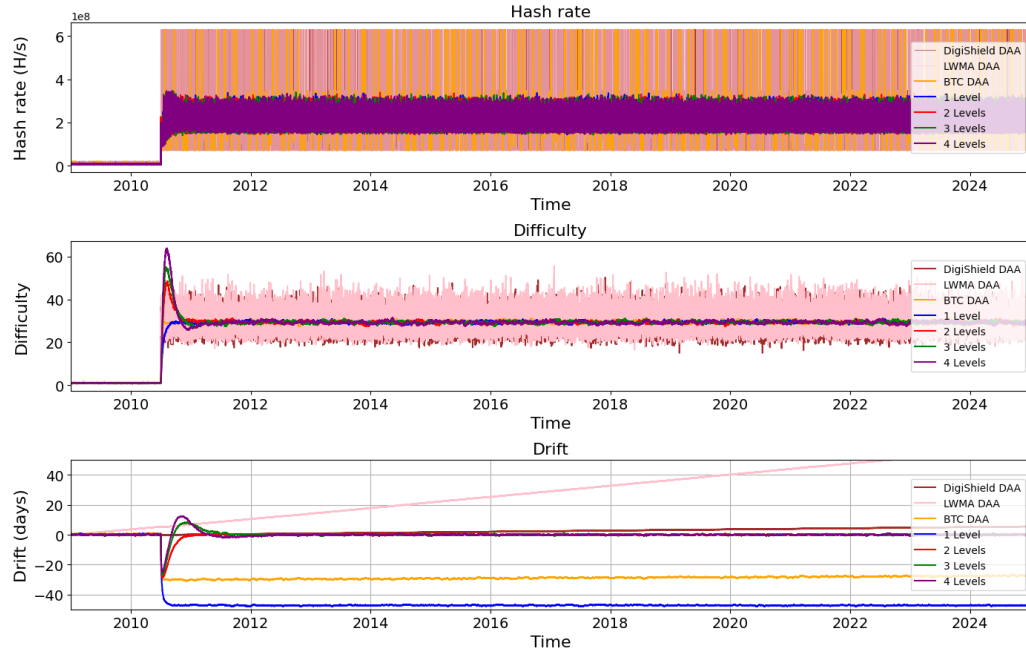


Fig. 13. Comparison for adaptive step hash rate.



Fig. 14. Comparison for adaptive exponential hash rate function with step in 06/2010.

Table 2. Mean final drift accumulation in days with 95% confidence intervals (30 runs)

Hash rate	NEFDA	NEF(DA) ²	NEF(DA) ³	NEF(DA) ⁴	BTC DAA	LWMA	DigiShield
Noiseless functions							
Step	-47.33 ± 0.07	-0.00 ± 0.08	-0.06 ± 0.10	-0.03 ± 0.10	-27.74 ± 0.17	14.90 ± 0.05	5.45 ± 0.02
Const	0.36 ± 0.07	0.02 ± 0.07	0.02 ± 0.08	0.04 ± 0.08	3.14 ± 0.13	59.24 ± 0.13	6.19 ± 0.02
Exp w/ jump	-575.35 ± 0.08	-5.12 ± 0.07	-0.01 ± 0.06	0.06 ± 0.10	-532.89 ± 0.15	6.71 ± 0.06	-2.97 ± 0.01
S.Exp w/ jump	-3327.70 ± 0.03	-57.80 ± 0.03	-0.54 ± 0.04	-0.00 ± 0.03	-2961.89 ± 0.04	-85.78 ± 0.11	-144.42 ± 0.02
Adaptive functions							
Step	-47.33 ± 0.10	0.04 ± 0.06	-0.09 ± 0.06	0.03 ± 0.09	-27.70 ± 0.18	14.89 ± 0.06	5.43 ± 0.01
Exp w/ jump	-575.24 ± 0.09	-5.14 ± 0.07	0.05 ± 0.07	0.02 ± 0.06	-532.78 ± 0.13	6.62 ± 0.06	-2.97 ± 0.02

6.3 Smoothing factor effect

Both NEFDA [25] and NEF(DA)² [29] have a smoothing factor that adjusts the responsiveness of the algorithm and is set so as to make the instant-response rate to a hash rate jump be one reciprocal fortnight. Smaller smoothing factors would make the algorithm more responsive but also more vulnerable to hash rate fluctuations as displayed in the case of DigiShield and LWMA (see Section 6.2). NEF(DA)^N can have multiple smoothing factors, as each level has its own. As before, we set the smoothing factors so that their sum – which gives the instantaneous response rate – is equal to a reciprocal fortnight. After experimenting with different values, we observed that as long as the sum of all the smoothing factors remains constant and the smoothing factors are of the same order, the algorithm tends to behave similarly.

6.4 Limitations

Our evaluation relies primarily on synthetic hash rate scenarios and a stylised adaptive miner model rather than real blockchain traces. Thus, the model cannot fully capture factors such as block propagation delays, timestamp manipulation, parameter sensitivity across diverse network settings, and complex strategic miner behaviour. Finally, we acknowledge that applying drift correction to a blockchain that has already accumulated a significant amount of drift can be overly drastic, even for large smoothing factors. NEF(DA)^N would immediately adapt to neutralise the accumulated drift, which could lead to a long initial period of increased difficulty and adversely impact network throughput. Nevertheless, NEF(DA)^N can use any block as an anchor block (e.g., the first block after on-chain activation or the BCH-BTC split) to partially eliminate accumulated drift and prevent further accumulation.

7 Conclusion

In this paper, we reviewed the present and past DAAs of BCH to reveal design flaws, some of which were addressed to a limited extent in our previous work [29]. Here, we have expanded on this previous work by proposing NEF(DA)^N, a generalised DAA that incorporates the existing NEFDA [25] as well as the NEF(DA)² [29] as special cases. By inheriting the desirable properties of the two algorithms, NEF(DA)^N achieves a stable block-production rate by effectively adjusting mining difficulty without introducing long-term autocorrelation between interblock times. Moreover, we have theoretically analysed and experimentally substantiated that this simple and intuitive algorithm can alleviate drift even for unrealistically extreme hash rate increases and outperform the state-of-the-art DAAs. Lastly, its straightforward formulae and concise code can readily replace existing DAAs and provide long-term block production stability with negligible – or even negative – additional coding complexity. Our work thus provides a generalised and robust yet elegantly simple and intuitive Difficulty Adjustment Algorithm compatible with many PoW blockchains.

References

- [1] 2Miners. Last accessed: 2026. ZCash Difficulty. <https://2miners.com/zec-network-difficulty>
- [2] Lise Arena, Nathalie Oriol, and Iryna Veryzhenko. 2018. Too Fast, Too Furious? Trading algorithmique et instabilité des marchés financiers. *Systèmes d'information & management* Volume 23 (2018), 81–106. Issue 2. doi:10.3917/sim.182.0081
- [3] Imran Bashir. 2018. *Mastering blockchain : distributed ledger technology, decentralization, and smart contracts explained* (2nd ed.). Packt Publishing Ltd, Birmingham.
- [4] Bitcoin ABC. 2017. Difficulty Adjustment Algorithm Update. <https://www.bitcoinabc.org/2017-11-01-DAA/>
- [5] Bitcoin Cash. Last accessed: 2026. Bitcoin Cash – Peer-to-Peer Electronic Cash. <https://bitcoincash.org/>
- [6] Bitcoin Stack Exchange. 2014. How does the Kimoto Gravity Well regulate difficulty? <https://bitcoin.stackexchange.com/questions/21730/how-does-the-kimoto-gravity-well-regulate-difficulty>
- [7] Bitcoin SV. Last accessed: 2026. The Original Bitcoin Blockchain: Bitcoin SV (BSV). <https://bitcoinsv.com/>
- [8] BitDegree. Last accessed: 2026. Bitcoin Cash Halving Countdown. <https://www.bitdegree.org/halving/bitcoin-cash-halving>
- [9] Vitalik Buterin. 2014. Ethereum: A Next-Generation Smart Contract and Decentralized Application Platform.
- [10] Kimoto Chan. 2013. Megacoin: A Peer-to-Peer Electronic Cash System. <https://c3.coinlore.com/pdf/megacoin-white-paper.pdf>
- [11] CoinDesk. Last accessed: 2026. Bitcoin Cash Price – BCH Price Index and Live Chart. <https://www.coindesk.com/price/bitcoin-cash/>
- [12] Luke Conway. 2025. Bitcoin Halving: What It Is and Why It Matters for Crypto Investors. <https://www.investopedia.com/bitcoin-halving-4843769>
- [13] CryptoFrenzy. 2014. Multipools vs Gravity Well. <https://cryptofrenzy.wordpress.com/2014/02/09/multipools-vs-gravity-well/>
- [14] Dash. Last accessed: 2026. Features – Dash documentation. <https://docs.dash.org/en/stable/docs/user/introduction/features.html>
- [15] Dash. Last accessed: 2026. Proof of Work – Dash documentation. <https://docs.dash.org/projects/core/en/stable/docs/guide/block-chain-proof-of-work.html>
- [16] demKyr. 2025. BCH-DAA. GitHub repository. <https://github.com/demKyr/BCH-DAA>
- [17] demKyr. 2026. BCH-DAA-2. GitHub repository. <https://github.com/demKyr/BCH-DAA-2>
- [18] DigiByte. 2014. DigiByte Community Infopaper. <https://www.digibyte.org/docs/infopaper.pdf>
- [19] Dogecoin. 2018. dogecoin pow. GitHub repository. <https://github.com/dogecoin/dogecoin/blob/master/src/pow.cpp>
- [20] Evan Duffield and Daniel Diaz. 2015. Dash: A Privacy-Centric Crypto-Currency.
- [21] Ethereum. 2025. Proof-of-work (PoW). <https://ethereum.org/en/developers/docs/consensus-mechanisms/pow/>
- [22] freetrader. 2020. On Drift correction. <https://read.cash/@freetrader/on-drift-correction-39974f66>
- [23] freetrader, Jonathan Toomim, Calin Culianu, and Mark Lundberg. 2020. 2020-NOV-15 ASERT Difficulty Adjustment Algorithm (aserti3-2d). <https://upgradespecs.bitcoincashnode.org/2020-11-15-asert/>
- [24] IG. Last accessed: 2026. Bitcoin halving 2028: what to know. <https://www.ig.com/ae/bitcoin-btc/bitcoin-halving>
- [25] Dragos I. Ilie, Sam M. Werner, Iain D. Stewart, and William J. Knottenbelt. 2021. Unstable throughput: When the difficulty algorithm breaks. In *Proc. IEEE International Conference on Blockchain and Cryptocurrency, ICBC 2021*. Institute of Electrical and Electronics Engineers Inc. doi:10.1109/ICBC51069.2021.9461086
- [26] The Investopedia Team. 2025. What Is Bitcoin Cash (BCH), and How Does It Work? <https://www.investopedia.com/terms/b/bitcoin-cash.asp>
- [27] Siddhant Kejriwal. 2024. Bitcoin vs Bitcoin Cash: An In-Depth Look at BTC & BCH. <https://coinbureau.com/analysis/bitcoin-vs-bitcoin-cash/>
- [28] Tamás Király and Lilla Lomoschitz. 2018. *Profitability of the coin-hopping strategy*. Technical Report. Egervary Research Group, Budapest.
- [29] Demetris Kyriacou, Matthieu Babak, Iain Stewart, and William J. Knottenbelt. 2025. Fast Blocks and Furious Adjustments: Satoshi Drift. In *Blockchain Technology and Emerging Applications: 4th EAI International Conference, Proceedings*. Springer.
- [30] Mark B. Lundberg. 2020. Static difficulty adjustments, with absolutely scheduled exponentially rising targets (DA-ASERT)-v.2. <https://www.bitcoinunlimited.info/nexa/da-asert.pdf>
- [31] Mark B. Lundberg. 2020. Thoughts on Grasberg DAA. https://www.reddit.com/r/btc/comments/hx4kgz/thoughts_on_grasberg_daa/
- [32] Megacoin. 2013. Megacoin. GitHub repository. <https://github.com/megacoin/megacoin/blob/master/src/main.cpp>
- [33] Satoshi Nakamoto. 2008. Bitcoin: A Peer-to-Peer Electronic Cash System. <https://www.bitcoin.org/bitcoin.pdf>
- [34] Scott Nevil. 2025. Hash Rate: How It Works and How to Measure. <https://www.investopedia.com/hash-rate-6746261>
- [35] Reference Cash. 2022. Difficulty Adjustment Algorithm. <https://reference.cash/protocol/blockchain/proof-of-work/difficulty-adjustment-algorithm>
- [36] Reference Cash. 2022. Proof of Work. <https://reference.cash/protocol/blockchain/proof-of-work>
- [37] Amaury Sechet. 2020. Announcing the Grasberg DAA. <https://read.cash/@deadalnix/announcing-the-grasberg-daa-ff52e96d>
- [38] Jonathan Toomim. 2020. Dark secrets of the Grasberg DAA. <https://read.cash/@jtoomim/dark-secrets-of-the-grasberg-daa-a9239fb6>
- [39] Kurt Wuckert Jr. 2020. History on Hash War. <https://coingeek.com/hash-war-history/>
- [40] zawy12. 2018. difficulty-algorithms. GitHub repository. https://github.com/zawy12/difficulty-algorithms/blob/master/test_DAs.cpp
- [41] ZCash. 2023. zcash pow. GitHub repository. <https://github.com/zcash/zcash/blob/master/src/pow.cpp>
- [42] ZCash Block Explorer. Last accessed: 2026. Block Explorer. <https://mainnet.zcashexplorer.app/blocks>
- [43] Antony Zegers. 2017. Bringing Stability to Bitcoin Cash Difficulty Adjustments. <https://mengerian.medium.com/bringing-stability-to-bitcoin-cash-difficulty-adjustments-eae8def0efa4>

A Proof that $\text{NEF}(\text{DA})^2$ is equivalent to $\text{NEF}(\text{DA})^N$ with 2 levels

For $\text{NEF}(\text{DA})^2$:

$$D_n = \widehat{H}_n \cdot T_n = \left[\frac{1}{S_1} \sum_{i=0}^{n-1} D_i e^{\frac{t_i - t_n}{S_1}} \right] \cdot T e^{\frac{-\text{drift}}{S_2}} = FTW_n \cdot \frac{T}{S_1} e^{\frac{-\text{drift}}{S_2}}$$

$$FTW_n = \sum_{i=0}^{n-1} D_i e^{\frac{t_i - t_n}{S_1}}$$

$$\text{drift} = (t_n - t_0) - (n \cdot T)$$

Therefore:

$$FTW_{n+1} = \sum_{i=0}^n D_i e^{\frac{t_i - t_{n+1}}{S_1}} = \sum_{i=0}^n D_i e^{\frac{t_i - t_n + t_n - t_{n+1}}{S_1}} = e^{\frac{t_n - t_{n+1}}{S_1}} \sum_{i=0}^n D_i e^{\frac{t_i - t_n}{S_1}}$$

$$= e^{\frac{t_n - t_{n+1}}{S_1}} \left[D_n + \sum_{i=0}^{n-1} D_i e^{\frac{t_i - t_n}{S_1}} \right] \quad (15)$$

$$= e^{\frac{-\Delta t}{S_1}} [D_n + FTW_n]$$

$$D_{n+1} = FTW_{n+1} \cdot \frac{T}{S_1} \cdot e^{\frac{t_0 - t_{n+1} + (n+1)T}{S_2}} = FTW_{n+1} \cdot \frac{T}{S_1} \cdot e^{\frac{t_0 - t_n + t_n - t_{n+1} + nT + T}{S_2}}$$

$$= e^{\frac{-\Delta t}{S_1}} [D_n + FTW_n] \cdot \frac{T}{S_1} \cdot e^{\frac{t_0 - t_n + nT}{S_2}} \cdot e^{\frac{-\Delta t + T}{S_2}}$$

$$= e^{\frac{-\Delta t}{S_1}} \cdot \frac{T}{S_1} \cdot e^{\frac{t_0 - t_n + nT}{S_2}} \cdot e^{\frac{-\Delta t + T}{S_2}} \cdot D_n + e^{\frac{-\Delta t}{S_1}} \cdot \frac{T}{S_1} \cdot e^{\frac{t_0 - t_n + nT}{S_2}} \cdot e^{\frac{-\Delta t + T}{S_2}} \cdot FTW_n$$

$$= e^{\frac{-\Delta t}{S_1}} \cdot \frac{T}{S_1} \cdot e^{\frac{t_0 - t_n + nT}{S_2}} \cdot e^{\frac{-\Delta t + T}{S_2}} \cdot D_n + e^{\frac{-\Delta t}{S_1}} \cdot e^{\frac{-\Delta t + T}{S_2}} D_n \quad (16)$$

$$= D_n \cdot e^{\frac{-\Delta t}{S_1}} \cdot e^{\frac{-\Delta t}{S_2}} \cdot e^{\frac{T}{S_2}} \left[1 + \frac{T}{S_1} \cdot e^{\frac{t_0 - t_n + nT}{S_2}} \right]$$

$$\stackrel{1+x \approx e^x}{\approx} D_n \cdot e^{\frac{-\Delta t}{S_1}} \cdot e^{\frac{-\Delta t}{S_2}} \cdot e^{\frac{T}{S_2}} \cdot \exp \left(\frac{T}{S_1} \cdot e^{\frac{t_0 - t_n + nT}{S_2}} \right)$$

For $\text{NEF}(\text{DA})^N$ with 2 levels:

$$D_1^n = D_0 \cdot \exp(-\text{drift}_1 \cdot \omega_1)$$

$$D^n = D_2^n = D_0 \cdot \exp(-\text{drift}_1 \cdot \omega_1 - \text{drift}_2 \cdot \omega_2)$$

$$\text{drift}_1 = (t_n - t_0) - (h_1^{n-1} \cdot T)$$

$$\text{drift}_2 = (t_n - t_0) - (h_2^{n-1} \cdot T)$$

$$h_1^n = h_1^{n-1} + D_2^n / D_1^n$$

$$h_2^n = h_2^{n-1} + 1$$

$$n = h_2^n$$

Therefore:

$$\begin{aligned}
D^{n+1} &= D_0 \cdot \exp((t_0 - t_{n+1} + h_1^{n+1}T)\omega_1 + (t_0 - t_{n+1} + h_2^{n+1}T)\omega_2) \\
&= D_0 \cdot \exp\left(\left[t_0 - t_n + t_n - t_{n+1} + (h_1^n + \frac{D_2^{n+1}}{D_1^{n+1}})T\right]\omega_1 \right. \\
&\quad \left. + [t_0 - t_n + t_n - t_{n+1} + (h_2^n + 1)T]\omega_2\right) \\
&= D_0 \cdot \exp([t_0 - t_n + h_1^n T]\omega_1 + [t_0 - t_n + h_2^n T]\omega_2) \\
&\quad \cdot \exp\left(\left[t_n - t_{n+1} + T\frac{D_2^{n+1}}{D_1^{n+1}}\right]\omega_1 + [t_n - t_{n+1} + T]\omega_2\right) \\
&= D_n \exp\left(\left[-\Delta t + T\frac{D_2^{n+1}}{D_1^{n+1}}\right]\omega_1 + [-\Delta t + T]\omega_2\right) \\
&= D_n \cdot e^{-\Delta t \cdot \omega_1} \cdot e^{-\Delta t \cdot \omega_2} \cdot e^{T \cdot \omega_2} \cdot \exp\left(\frac{D_2^{n+1}}{D_1^{n+1}} \cdot \omega_1 T\right) \\
&= D_n \cdot e^{-\Delta t \cdot \omega_1} \cdot e^{-\Delta t \cdot \omega_2} \cdot e^{T \cdot \omega_2} \cdot \exp(\omega_1 T \cdot e^{(t_0 - t_n + nT)\omega_2})
\end{aligned} \tag{17}$$

By setting $\omega_i = 1/S_i$ the expressions (16) and (17) are the same. This analysis applies at exact moments of solving a block. Since the miners are given real time difficulty updates, we also need an analysis of intermediate times between solving one block and the next. Such an analysis is very similar to the one above and we omit it for brevity. Thus, NEF(DA)^2 is equivalent (subject to the mild approximation) to NEF(DA)^N with 2 levels.

B Comparison table of BTC/BCH halvings to the ideal

Table 3. Comparison of BTC/BCH halvings to the ideal

	Ideal	BTC	BCH
Genesis Block (block #0 - reward 50 BTC)			
Date	2009-01-09	2009-01-09	
Halving 1 (block #210000 - reward 25.0 BTC)			
Date	2013-01-06	2012-11-28	
Days since last halving (deviation)	1458 (0)	1419 (-39)	
Days since genesis (deviation)	1458 (0)	1419 (-39)	
Halving 2 (block #420000 - reward 12.5 BTC)			
Date	2017-01-03	2016-07-09	
Days since last halving (deviation)	1458 (0)	1319 (-139)	
Days since genesis (deviation)	2916 (0)	2738 (-178)	
Halving 3 (block #630000 - reward 6.25 BTC)			
Date	2020-12-31	2020-05-11	2020-04-08
Days since last halving (deviation)	1458 (0)	1402 (-56)	1369 (-89)
Days since genesis (deviation)	4374 (0)	4140 (-234)	4107 (-267)
Halving 4 (block #840000 - reward 3.125 BTC)			
Date	2024-12-28	2024-04-20	2024-04-03
Days since last halving (deviation)	1458 (0)	1440 (-18)	1456 (-2)
Days since genesis (deviation)	5832 (0)	5580 (-252)	5563 (-269)