SMA Algorithm

The difficulty adjustment approach we suggest is a form of a Moving Window Average. In this approach each block determines its expected difficulty on the spot based on data in the blocks it is connected to. This is in contrast to Bitcoin's fixed window approach which determines that the global difficulty will be adjusted every epoch. The main reason for that is that in a DAG constellation it is hard to even define what an epoch is, as different nodes can get the same DAG in a different order. None the less, there are several reasons to prefer this approach in the chain model as well, especially for small coins whose network can be overwhelmed by a feasibly powerful attacker.

Most of the discussion is based on the ideas implemented by Zawy12, which are hence documented.

The following assumes that there is some universal "correct" time, and that all nodes have access to a time oracle with some error, and an ability to recognize that their clock is out of sync and readjust it. This model essentially delegates timekeeping responsibilities to the nodes themselves under some assumptions of their ability to do so.

The importance of a time oracle is that local times are the only measure of how long it took to place a block, which is crucial for appropriate retargeting. We need our averaging method to take enough blocks into account to smooth out such errors, without making it non-responsive.

Worse yet, malicious miners may use the timestamp leeway to manipulate the difficulty, while being too strict will cause splits.

Suggested Algorithm in a Naive World

The expected difficulty of a proposed block B is calculated as follows:

Let B_n be the nth *from last* blue block in $B.past$ (so that $B_0 = B$ and $B_{B.blueScore} = Genesis$).

Let $R = \max\{B_i.timeStamp - B_j.timeStamp \mid 1 \leq i,j \leq N+1\}$, in a perfect world (where all clocks are synchronized and nobody lies about timestamps) R should tell us how long it took to create N blocks, and we desire $R/N = \lambda$. We hence define the *adjustment factor* $\alpha = R/(N\lambda)$.

Lastly we take the average target $V = \frac{1}{N}\sum_{n=1}^{N} B_n.target$ and calculate $B.target = V \cdot \alpha$.

Handling Bad Timestamps

One main problem with the above algorithm is that timestamps are not accurate, due to propagation, clock drifting and malfeasance.

In order to overcome this, it is required to have rules for rejecting blocks based on the offset of the timestamps.

Handling timestamps which are too far into the past can be handled by comparing the timestamp to those of the previous blocks. Handling timestamps far into the future is much more problematic as it requires use of a clock.

We hence make two physical assumption:

- There is some universal time which is "correct"
- The nodes have access to a time oracle whose error is given by the random variable $\mathcal{E}$
- Nodes are able to detect when their clock is out of sync

These assumptions are an idealized model of the fact that there are a lot of timekeeping entities which can easily verify each other (if Microsoft decide to make their NTP 1% slower, someone with a cesium clock will recognize the error), and most of them are freely accessible. Given a common reference time the problem of synchronization reduces to the problem of timekeeping (we no longer need a wall clock, just a stopwatch). The error accounts for accumulated errors across timekeeping entities, propagation delay's etc.

Under these considerations, we need to choose a parameter $\mathcal{F}$ which regulates the allowed error by stating that B is rejected unless

- $B.timeStamp \geq median\{B_1, \ldots, B_{2\mathcal{F}-1}\}$ (block not too much into the past)
- $B.timeStamp \leq T + \lambda\mathcal{F}$ (block not too much into the future)

Thus, for example, if a node recognizes that its time is not within these criteria (i.e. that it can not place a block without lying about the time) it may deduce that its clock is out of sync.

Effects of $\mathcal{F}$ on difficulty

As stated, giving more leeway allows for more adversarial manipulation.

If I am an attacker wishing to decrease difficulty, I need to use modified timestamps to make α appear as large as possible.

Assuming that in days of peace $R \approx N\lambda$, the worst an attacker can do is to push the least timestamp about $\lambda\mathcal{F}$ into the past, and the latest timestamp exactly $\lambda\mathcal{F}$ into the future, getting at $R \approx N\lambda + 2\lambda\mathcal{F} = \lambda(N + 2\mathcal{F})$.

In this case we get that $\alpha \approx N/(N+2\mathcal{F})$ or $\frac{1}{\alpha} \approx 1 + 2\frac{\mathcal{F}}{N}$.

This means we can't make $\mathcal{F}$ irresponsibly large. E.g. if $\mathcal{F} = N/2$ then an attacker can potentially make the difficulty half as easier than necessary by placing just two blocks. (Of course, this attack will only be as effective for a single block, and will have a declining effect for the next $\mathcal{F}$ blocks.)

If we want to limit timestamp attacks to a create a multiplicative error of at most $(1+\delta)$ we need to have $2\frac{\mathcal{F}}{N} \leq \delta$ or $2\mathcal{F}/\delta \leq N$.

In can be crudely stated that large $\mathcal{F}$ protects us from synchronization problems at the expense of allowing some difficulty manipulation, while choosing large N protects us from such manipulations while making the algorithm less responsive. It is possible to ameliorate this lack of responsiveness by means other than decreasing N, such as choosing different averaging mechanisms. This is essentially what zawy works on.

Choosing $\mathcal{F}$

Given a fixed λ the choice of $\mathcal{F}$ relies on $\mathcal{E}$.

The simplest assumption one can make is that $|\mathcal{E}|$ is bound by a constant $e/2$, which amounts to saying that the clock discrepancy between two nodes is at most ϵ.

In this case, choosing $\mathcal{F}$ such that $\lambda\mathcal{F} \geq e$ suffices to ensure that no node will reject a block emitted by an honest node.

In the real world, the block delay will not be exactly λ, and it can in fact be as low as $\lambda/(1+\delta)$, so we want to make sure that $\mathcal{F} \geq (1+\delta)e/\lambda$.

(We can make a bit more realistic assumption that $\mathcal{E} \sim \mathcal{N}(0, \sigma^2)$ for some unknown σ, in this case $\mathcal{E}$ is not bounded, but arguments can be made if we allow some constant fraction of the network to be out of sync)

What about D_{max} ?

It seems that propagation delay are not much of a concern when all nodes are honest. The only comparison between a block timestamp and the system clock is made when trying to approximate if the block is in the future. Propagation delay can only reduce the difference, so it does not increase the risk of dropping a legitimate block.

Choosing the Parameters

The proposed difficulty adjustment method relies on the following parameters:

- N - The window size
- λ - The target block delay
- δ - The permitted difficulty deviation
- $\mathcal{F}$ - Timestamp deviation tolerance, in block delays
- e - Maximal synchronization discrepancy between two nodes

The relations above tell us how to set N and $\mathcal{F}$ given the rest of the parameters.

Assuming $\lambda = 1$ and $e = 120$ (in units of seconds) and assuming we allow $\delta = 10\%$ we need $\mathcal{F} \geq (1+\delta)e/\lambda = 132$.

Choosing $\mathcal{F} = 132$ we need $N \geq 2\mathcal{F}/\delta = 2640$.

Decreasing N Further

Assuming an attacker has a fraction of p of the hash rate, and assuming constant hash rate during the attack, we get on average that $\alpha = (1-p) + p(1+\delta) = 1 + \delta p$.

Assuming that $p < 1/2$ we get that on average $\alpha < 1 + \delta/2$. Further analysis is required, but it seams reasonable that $\alpha > 1 + 3\delta/4$ is a rare occurrence even during an attack.

This hints that taking $N \geq 2\mathcal{F}/(4\delta/3)$ suffices to ensure that $\alpha < 1 + \delta$ most of the time.

Further Questions

Is there an approximation for e?

Does decreasing N allows for some cascading difficulty decrease attack?

What probabilistic model is suitable to analyse the distribution of α?

What about a combination of a hash rate attack and a difficulty decrease attack?

How is all of the above affected assuming exponential hashrate growth?