# Block statistics for real-time-targeted difficulty adjustments, with absolutely scheduled exponentially rising targets (RTT-ASERT)

Mark B. Lundeberg

November 6, 2019

Consider a real time difficulty algorithm, where the block target ($=$ $2^{224}$/block difficulty) for the block with height $N$ and timestamp $t_N$ is defined as:

$$\mathrm{target}_N = \mathrm{target}_0 \exp([t_N - \tilde{t}_N]/\lambda), \tag{1}$$

where $\tilde{t}_N = t_0 + NT$ and where $T$ is the targeted average block interval (e.g., $T \approx 10$ minutes). The parameter $\lambda$ is a relaxation time and practically could be for instance $6T$ or $12T$. The base target $\mathrm{target}_0$ and genesis time $t_0$ are not particularly important in this analysis since they just produce offsets in time.

## 1 Statistics of the block-finding process

At a given time $t$, a hashrate $H(t)$ is allocated to mining blocks, and furthermore let's assume that all miners have highly synchronized clocks, and are trying to mine a block $N$ with timestamp $t_N = t$ and they immediately publish any found blocks (whether these actually happen in reality is a separate question!). Then the rate probability of producing block $N$ at timestamp $t$ (supposing it has not already been found yet) is given by:

$$\lambda(t) = H(t) \cdot 2^{-256}\mathrm{target(t)}.$$

Following wikipedia's forumulas on survival analysis, you just need to integrate $\lambda(t)$ to find the "survival function" $S(t)$, which is the complementary cumulative distribution function of the time when the block is found. Let's say that at some start time $t_{\mathrm{s}}$, the miners haven't found the block yet, and you want to know the statistics of the predicted *solve time* $\delta = t_N - t_{\mathrm{s}}$. This is

$$S(\delta) = \exp\left(-\int_{t_{\mathrm{s}}}^{t_{\mathrm{s}}+\delta} \lambda(t)\mathrm{d}t\right).$$

Now, let's consider the case of $H(t) = H$ constant, and plug in our exponentially varying target Eq. 1. This gives:

$$S(\delta) = \exp\left(-\lambda H 2^{-256}\mathrm{target}_0 \exp([t_{\mathrm{s}} - \tilde{t}_N]/\lambda)(\exp(\delta/\lambda) - 1)\right) \tag{2}$$

This looks a bit hairy, but let's separate the big constant factor as $X$:

$$X = \lambda H 2^{-256}\mathrm{target}_0 \exp([t_{\mathrm{s}} - \tilde{t}_N]/\lambda), \tag{3}$$

$$S(\delta) = \exp\left(-X(\exp(\delta/\lambda) - 1)\right). \tag{4}$$

This dimensionless $X$ can be thought of as the "shape parameter". For large $X \gg 1$, this will be the familiar $S(\delta) \approx \exp(-X\delta/\lambda)$ with a maximum initial slope $-X/\lambda$ occuring at $\delta = 0$, i.e., as if the target is independent of time, since the block will be found "fast" relative to time $\lambda$. With small $X \ll 1$ in contrast, $S(\delta) \approx \exp[-\exp(\delta/\lambda + \ln X)]$ which has a more sigmoid shape, with a crossing near $\delta \approx \lambda \ln(1/X)$, a maximum slope of around $-0.37/\lambda$, but a rather hard cutoff for $2\lambda$ beyond this crossing.

# 2 Jump-forward simulation

In general, mining simulations can be naively done in a 'real time' nature by sampling $\lambda(t)$ at very small time intervals, however these simulations will be slow and thus in a given time will not generate as much useful information. When $H(t)$ is known ahead of time, it is *much* more efficient to do 'jump-forward' simulations by simply calculating a random $\delta$ with appropriate distribution. Since $S$ is the complementary cumulative distribution function of the block event, you just need to set $S = x$, where $x$ is a random number chosen uniformly between 0 and 1. Then solve for $\delta$.

Taking the constant-$H$ case considered above in Eq. 4, this yields

$$\delta = \lambda \ln(1 - (\ln x)/X) \tag{5}$$

where again the $X$ is a constant defined above. This is slightly more complex than the formula used in static target simulations, but not much more so.

Switch mining simulations typically consider the case where $H(t)$ is piecewise-constant: a low value $H(t) = H_a$ when the target is below some critical value, $\mathrm{target}(t) < \mathrm{target}_{\mathrm{crit}}$, then jumping to a high value $H(t) = H_b$ afterwards. In simulations this is easiest to achieve by calculating $\delta_{\mathrm{crit}}$, the time of switch, then generating a $\delta$ assuming $H_a$. If $\delta < \delta_{\mathrm{crit}}$ then the block was found, otherwise jump the simulation forward in time by $\delta_{\mathrm{crit}}$ (i.e., $t_{\mathrm{s}} \to t_{\mathrm{s}} + \delta_{\mathrm{crit}}$), then calculate a new $X$ for the new $t_{\mathrm{s}}$ and $H_b$, and calculate a new random $\delta$ measured from the new $t_0$. Arbitrary piecewise-constant $H(t)$ can be done in a jump-forward simulation using this trick, though of course for each jump it requires knowing the exact time of the next change in hashrate.

# 3 Stochastic state properties

In practice, the feedback nature of difficulty targeting means that $\delta$ will seek towards a typical value of $T$, the target block interval. Since slow relaxation with $\lambda > T$ ought to be chosen, this means the chain will primarily be operating in the $X \approx \lambda/T > 1$ regime. Statistical fluctuations and temporary high, sustained hashpower increases can push down $X$, but the system quickly recovers on a $\lambda$ timescale.

Since the targeting equation is so simple (it doesn't depend on prior blocks' timestamps nor their targets), at any given time the state of the system can be represented by a single variable: the current target. The target exponentially rises and every time a block is found, the target changes by a factor of $\exp(-T/\lambda)$.

An even simpler picture of dynamics arises when we take logarithm of target as our variable. For convience let's work in $b = \ln(2^{256}/\mathrm{target})$ space, which is basically the log of difficulty (taking difficulty=1 to mean a block can be solved in 1 hash). So, this means the $b$ ramps *linearly* down over time, at a rate of $1/\lambda$. Then for every block found, $b$ jumps up by an offset $+T/\lambda$. Note that at any given time, the target can only take on specific discrete values (depending on the height $N$), and those values recur every time $T$. To get a picture of the 'steady state', it is necessary to average the dynamics over a time $T$ period.

I'm not sure how to solve this equation, not even in the steady state (where it becomes a retarded functional differential equation). However the behaviour is well suited to numerical simulation, where for each time step the $p(b)$ shifts left by one pixel, and then some fraction is shifted right. The steady state distributions are rather interesting looking, for example below with $\lambda = 12T$, where the "steady" hashrate for $b > 0$ is only $0.2/T$, but then for $b < 0$ some switch miners come in and the hashrate is $5/T$. Note, however, that real time block difficulty algorithms don't even need a steady hashrate, since there is no hazard of 'freezing up'!