LEDGER
ledgerjournal.org
ISSN 2379-5980 (online)
DOI 10.5195/LEDGER.2016.46

RESEARCH ARTICLE

A Probabilistic Analysis of the Nxt Forging Algorithm

Serguei Popov*†

Abstract. We discuss the forging algorithm of Nxt from a probabilistic point of view, and obtain explicit formulas and estimates for several important quantities, such as the probability that an account generates a block, the length of the longest sequence of consecutive blocks generated by one account, and the probability that one concurrent blockchain wins over another one. Also, we discuss some attack vectors related to splitting an account into many smaller ones.

1. Introduction

In the past three years there was a burst of popularity of Proof-of-Stake (PoS) cryptocurrencies, that used a different algorithm (from Proof-of-Work) for choosing the block creators. Unlike the Proof-of-Work (PoW) based cryptocurrencies (such as Bitcoin), where *miners* must solve complicated cryptographical puzzles in order to be able to create blocks and be rewarded for it (besides the transaction fees, there is usually a specific *block reward* that the successful miner takes), in PoS-based cryptocurrencies the creator of the next block is chosen in a deterministic (pseudo-random) way, and the chance that an account is chosen depends on its wealth (the *stake*). A note on terminology: in PoS cryptocurrencies the blocks are usually *forged* (in the blacksmith sense of this word), or *minted*. Also, usually all the coins are created in the beginning and the total number of coins never changes afterwards (although, as mentioned below, there are some other versions of PoS where new coins can be created). Therefore, in the basic version of PoS there are no block rewards as *e.g.* in Bitcoin; so, the forgers take only the transaction fees.

The main goal of this paper is to perform a probabilistic analysis of the block generation process of pure PoS cryptocurrencies, using as an example the Nxt protocol.[1] By "pure PoS" we mean that the choice of account that has the right to generate the next block is based on the number of coins in the account; the more wealthy the account is, the bigger is the probability that it will be able to generate the next block and get the corresponding transaction fees. Usually, one assumes that this probability should be exactly proportional to the account's balance, although we shall see that it is not quite the case for the Nxt. Also, we stress that most of the analysis in this paper is not Nxt-specific and can be applied in the broader context of PoS-cryptocurrencies with pseudo-random choice of the creator of the next block.

---

*

†S. Popov (popov@ime.unicamp.br) is a Professor in the Department of Statistics, Institute of Mathematics, Statistics and Scientific Computation, University of Campinas – UNICAMP, Brazil.
*1FCFYiUbL3KajNvZm4W2wTSLrUimF1Lpea

There are also other PoS-protocols with conceptually different implementations; for example, the forging probability may depend also on time the coins were in the account without being transferred (the so-called coin-age); also, there still may be some kind of block rewards for the forgers. In this paper, however, we concentrate on the "pure-PoS" algorithm implemented in the Nxt.

Some concerns were raised about the general security of the PoS-algorithms, being the "nothing-at-stake" attack the most famous of them. We do not discuss this and some other (such as the "history attack") attacks in this paper, and refer instead to a remarkable series of papers published by *Consensus Research*,[2] see.[8–11] Also, it is worth mentioning that Nxt was never successfully attacked during its three years of existence.[3]

We assume that the reader is familiar with the basic cryptocurrency-related concepts. In the next section we define a probabilistic model which effectively describes the Nxt's algorithm for choosing the next forger. In Section 3 we calculate the probability that a given account generates the next block, and discuss how this probability is affected when the account is split into several smaller ones. In Section 4 we study how many blocks in a row can be generated by an account over a long time period. (As a general principle, it is not desirable that one entity could be able to generate many blocks in a row, since cryptocurrencies are meant to be decentralised.) In Section 5 we analyse the probability that a chain forged privately by an attacker is "better" than the legitimate chain (this does not exactly correspond to the current Nxt consensus algorithm, but nevertheless gives some insight about probabilities involved). In Section 6 we analyse another attack strategy ("branching process attack") that amounts to splitting the account in many small parts, and then trying to manipulate the forging process by choosing which small accounts will forge, and which will not.

2. The Forging Algorithm

First of all, we assume that the time is discrete (say, measured in minutes), so that time intervals between the consecutive blocks are one minute long. This is *not* the case for the Nxt forging algorithm which is currently in use; however, even for the current realization the sequence of forging account is determined in the same way (only the time intervals are random). There are plans to pass to the one-block-per-minute regime in a future realization of the Nxt protocol, but, as far as the author knows, this is still being discussed.

Assume that there are N forging accounts at a given (discrete) time, $B_1,\dots,B_N$ are the corresponding balances, and we denote by

$$b_k = \frac{B_k}{B_1+\cdots+B_N}, \quad k=1,\dots,N$$

the proportion of total forging power that the k^{th} account has. In Nxt, the total number of coins is fixed (those who forge blocks receive only transaction fees), but some nodes may be offline, which effectively means that not all coins are forging. We, however, make a simplifying assumption that all accounts are active; this is not so restrictive since (as we will see below) the forging probabilities only depend on the relative stake of the account with respect to the total active stake. Then, to determine which account will generate the next block, we take i.i.d. random variables $U_1,\dots,U_N$ with uniform distribution on interval $(0,1)$, and the account which maximizes b_k/U_k generates the block; *i.e.*, the label k_0 of the generating account is determined

ISSN 2379-5980 (online)
DOI 10.5195/LEDGER.2016.46

by

$$k_0 = \underset{j\in\{1,\dots,N\}}{\arg\max} \frac{b_j}{U_j} = \underset{j\in\{1,\dots,N\}}{\arg\min} \frac{U_j}{b_j}. \tag{1}$$

We refer to the quantity b_k/U_k as the *weight* of the k^{th} account, and to b_{k_0}/U_{k_0} as the weight of the block, *i.e.*, we choose the account with the maximal weight (or, equivalently, with the minimal inverse weight) for the forging. This procedure is called the *main algorithm* (because it is actually implemented in Nxt at this time), or the *U-algorithm* (because the inverse weights have uniform distribution).

As a general rule, the probability that an account generates a block is meant to be proportional to its effective balance, but, in fact, this is only approximately true (as we shall see in Section 3). For comparison, we consider also the following rule of choosing the generating account: instead of (1), we use

$$k_0 = \underset{j\in\{1,\dots,N\}}{\arg\min} \frac{|\ln U_j|}{b_j}. \tag{2}$$

The corresponding algorithm is referred to as the *Exp-algorithm* (since the inverse weights now have exponential probability distribution).

Important note: for all the calculations in this article, we assume that all accounts are forging and all balances are effective (so that $B_1 + \cdots + B_N$ equals the total amount of NXT in existence). In the real situation when only the proportion α of all money is forging (if the $k_0{}^{\text{th}}$ account from (1) does not forge because it is offline or for any other reason, then the account with the seconds largest weight can forge that block, and so on), one can adjust the formulas in the following way. Making a reasonable assumption that *all* money of the attacker is forging and his/her (relative) stake is b', all the calculations in this article are valid with $b = \alpha^{-1}b'$.

Another important note: The Nxt forging algorithm is, in fact, *pseudo-random*: if we know the forger of the previous block and all account balances, then the forger of the next block is known too (provided it is online); this observation will be important for the discussion in Section 6. This is because the above "random" variables $U_1, \dots, U_N$ in reality are obtained deterministically as an application of a hashing function to the current block's *generating signature* and the account's public keys (no further pseudo-random number generators are involved). Nevertheless, it is still reasonable to assume "complete randomness" as a good approximation of the reality.

3. Probability of Block Generation

Observe that (see *e.g.* Example 2a of Section 10.2.1 of Ross)[15] the random variable $|\ln U_j|/b_j$ has exponential distribution with rate b_j. By (2), the inverse weight of the generated block is also an exponential random variable with rate $b_1 + \cdots + b_N = 1$ (cf. (5.6) of Ross),[16] and the probability that k^{th} account generates the block is exactly b_k (this follows *e.g.* from (5.5) of Ross).[16]

However, for the U-algorithm the calculation in the general case is not so easy. We concentrate on the following situation, which seems to be critical for assessing the security of the system: N is large, the accounts $2, \dots, N$ belong to "poor honest guys" (so $b_2, \dots, b_N$ are small), and the account 1 belongs to a "attacker", who is not necessarily poor (*i.e.*, $b := b_1$ is at least of the same order as $\max(b_2, \dots, b_N)$).

ISSN 2379-5980 (online)
DOI 10.5195/LEDGER.2016.46

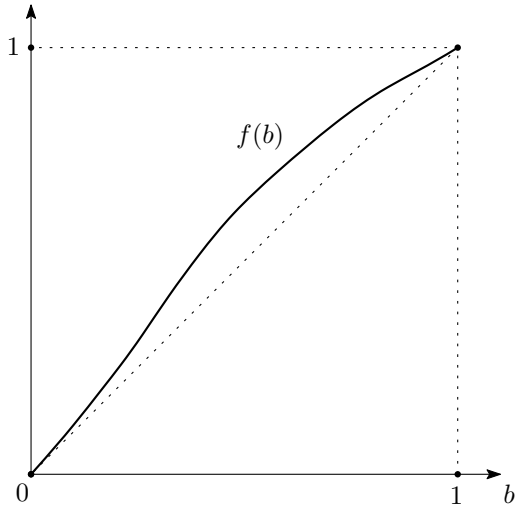

Fig. 1. The plot of $f(b)$

We first calculate the probability distribution of the largest weight among the good guys: for $x \gg \max_{k\geq 2} b_k$ let us write

$$\begin{aligned}\mathbb{P}\Big[\max_{k\geq 2}\frac{b_k}{U_k} < x\Big] &= \prod_{k\geq 2}\mathbb{P}\Big[U_k > \frac{b_k}{x}\Big]\\ &= \prod_{k\geq 2}\Big(1-\frac{b_k}{x}\Big)\\ &= \exp\sum_{k\geq 2}\ln\Big(1-\frac{b_k}{x}\Big)\\ &\approx e^{-\frac{1-b}{x}},\end{aligned} \tag{3}$$

since $\ln(1-y)\sim -y$ as $y\to 0$ and $b_2+\cdots+b_N = 1-b$. We calculate now the probability $f(b)$ that the attacker generates the block, in the following way. Let Y be a random variable with distribution (3) and independent of U_1, and we write (conditioning on U_1)

$$\begin{aligned}f(b) &:= \mathbb{P}\Big[\frac{b}{U_1} > Y\Big]\\ &= \int_0^1 \mathbb{P}\Big[Y < \frac{b}{z}\Big]\,dz\\ &= \int_0^1 e^{-\frac{1-b}{b}z}\,dz\\ &= \frac{b}{1-b}\Big(1-e^{-\frac{1-b}{b}}\Big).\end{aligned} \tag{4}$$

One can observe that $f(b) > b$ for all $b\in(0,1)$ (see also Fig. 1), and (using the Taylor expansion) $f(b) = b + b^2 + O(b^3)$ as $b\to 0$.

Let us remark also that, since $f(b)\sim b$ as $b\to 0$ and using a calculation similar to (3), if *all* relative balances are small, the situation very much resembles that under the Exp-algorithm (see also (9) below). Thus, although the U-algorithm does incentivize the users to store the money together (which, in principle, could lead to some centralization), the difference is usually not

ISSN 2379-5980 (online)
DOI 10.5195/LEDGER.2016.46

very significant (even if the account is split to many small parts) unless the account is *really* wealthy. Also, since the U-algorithm is computationally faster (no need to calculate logarithms), the author's opinion is that it is adequate enough for a PoS-currency such as Nxt.

3.1. Effect of Account Splitting on the Probability of Generating a Block—Here we examine the situation when an owner of an account splits it into two (or even several) parts, and show that, in general, this strategy is not favorable to the owner (that is, the probability that one of his/her accounts is chosen for block generation does not increase).

First of all, as discussed in the beginning of Section 3, for the Exp-algorithm, the probability that one of the new (*i.e.*, obtained after the splitting) accounts will generate the next block does not change at all. Indeed, this probability is exactly the proportion of the total balance owned by the account, and any splitting does not change this proportion (*i.e.*, all the new accounts forge *exactly* as the old one).

Now, let us consider the case of the U-algorithm. We shall prove that splitting is *always* unfavorable for a Nxt holder. Namely, we can prove the following result:

Theorem 3.1. *Assume that a person or entity controls a certain number of Nxt accounts, and let p be the probability of generating the next block (*i.e.*, the account that forges the block belongs to this person or entity). Suppose now that one of these accounts is split into two parts (while the balances of all other accounts stay intact), and let p' be the probability of block generation in this new situation. Then $p' < p$.*

By induction, one easily obtains the following:

Corollary 3.1.1. *Under the U-algorithm, in order to maximize the probability of generating the next block, all NXT that one controls should be concentrated in only one account.*

[Proof of Theorem 3.1.] Let $b_1, \dots, b_\ell$ be the relative balances of accounts controlled by that person or entity, and let $b_{\ell+1}, \dots, b_n$ be the balances of the other active accounts. Assume without restriction of generality that the first account is split into two parts with (positive) relative balances b_1', b_1'' (so that $b_1' + b_1'' = b_1$).

Let $U_1, \dots, U_n$, U_1', U_1'' be i.i.d. uniform$[0,1]$ random variables. Let

$$\begin{aligned} Y &= \min_{j=1,\dots,\ell} \frac{U_j}{b_j}, \\ Y' &= \min\Big(\frac{U_1'}{b_1'}, \frac{U_1''}{b_1''}, \min_{j=2,\dots,\ell} \frac{U_j}{b_j}\Big), \\ Z &= \min_{j=\ell+1,\dots,n} \frac{U_j}{b_j}. \end{aligned}$$

Let us denote $x^+ := \max(0,x)$ for $x \in \mathbb{R}$. Analogously *e.g.* to (3), we have for $t > 0$

$$\begin{aligned} \mathbb{P}[Y > t] &= \prod_{j=1,\dots,\ell} (1 - b_j t)^+, \\ \mathbb{P}[Y' > t] &= (1 - b_1' t)^+ (1 - b_1'' t)^+ \prod_{j=2,\dots,\ell} (1 - b_j t)^+, \end{aligned}$$

and a similar formula holds for Z; we, however, do not need the explicit form of the distribution function of Z, so we just denote this function by ζ.

ISSN 2379-5980 (online)
DOI 10.5195/LEDGER.2016.46

Observe that for $0 < t < \min\left(\frac{1}{b_1'}, \frac{1}{b_1''}\right)$ it holds that

$$\begin{aligned}(1-b_1't)(1-b_1''t) &= 1 - b_1t + b_1'b_1''t^2 \\ &> 1 - b_1t,\end{aligned}$$

so

$$(1-b_1't)^+(1-b_1''t)^+ \geq (1-b_1t)^+$$

for all $t \geq 0$ (if the left-hand side is equal to 0, then so is the right-hand side), and, moreover, the inequality is strict in the interval $\left(0, \min\left(\frac{1}{b_1'}, \frac{1}{b_1''}\right)\right)$.

Then, conditioning on Z, we obtain

$$\begin{aligned}1-p &= \mathbb{P}[Y > Z] \\ &= \int_0^\infty \prod_{j=1,\ldots,\ell} (1-b_jt)^+ \, d\zeta(t) \\ &= \int_0^\infty (1-b_1t)^+ \prod_{j=2,\ldots,\ell} (1-b_jt)^+ \, d\zeta(t) \\ &< \int_0^\infty (1-b_1't)^+(1-b_1''t)^+ \prod_{j=2,\ldots,\ell} (1-b_jt)^+ \, d\zeta(t) \\ &= \mathbb{P}[Y' > Z] \\ &= 1-p',\end{aligned}$$

and this concludes the proof of the theorem.

One should observe, however, that the disadvantage of splitting under the U-algorithm is not very significant. For example, if one person controls 5% of total active balance and has only one account, then, according to (4), the forging probability is approximately 0.0526. For *any* splitting, this probability cannot fall below 0.05 (this value corresponds to the the extreme situation when all this money is distributed between many small accounts).

4. Longest Run

In general, a situation when only one entity is able to forge many blocks in a row is considered undesirable in crypto world. Besides the conceptual issue of centralization, this can lead *e.g.* to the following kind of attack: the attackers spends in the main chain and waits for some confirmations, and then broadcasts a (long) sidechain that would be accepted by the nodes as the main one (the point is that the attacker can build this sidechain alone, without help of other forgers). Therefore, in this section we analyse how many blocks in a row can be typically generated by a given account over a long period n of time. We make a simplifying assumption that the situation is "static": there are no transactions (so that the effective balances are equal to full balances and do not change over time).

So, assume that the probability that an account generates the next block is p (see in Section 3 an explanation about how p can be calculated). It is enough to consider the following question: let R_n be the maximal number of consecutive 1's in the sequence of n Bernoulli trials with success probability p; what can be said about the properties of the random variable R_n?

The probability distribution of R_n has no tractable closed form, but is nevertheless quite well studied, see *e.g.* Schilling.[17] The following results are taken from Gordon, Schilling, and

ISSN 2379-5980 (online)
DOI 10.5195/LEDGER.2016.46

Waterman:[14] we have

$$\mathbb{E}R_n = \log_{1/p} qn + \frac{\gamma}{\ln 1/p} - \frac{1}{2} + r_1(n) + \varepsilon_1(n), \tag{5}$$

$$\mathrm{Var}R_n = \frac{\pi^2}{6\ln^2 1/p} + \frac{1}{12} + r_2(n) + \varepsilon_2(n), \tag{6}$$

where $q = 1-p$, $\gamma \approx 0.577\ldots$ is the Euler-Mascheroni constant, $\varepsilon_{1,2}(n) \to 0$ as $n \to \infty$, and $r_{1,2}(n)$ are uniformly bounded in n and very small (so, in practice, $r_{1,2}$ and $\varepsilon_{1,2}$ can be neglected).

In the same work, one can also find results on the distribution itself. Let Γ_p be a random variable with a Gumbel-type distribution: for $y \in \mathbb{R}$

$$\mathbb{P}[\Gamma_p \le y] = \exp(-p^{y+1}).$$

Then, for $x = 0, 1, 2, \ldots$ it holds that

$$\mathbb{P}[R_n = x] \approx \mathbb{P}[x - \log_{1/p} qn < \Gamma_p \le x + 1 - \log_{1/p} qn], \tag{7}$$

with the error decreasing to 0 as $n \to \infty$. So, in particular, one can obtain that

$$\begin{aligned} \mathbb{P}[R_n \ge x] &\approx 1 - \exp(-p^{x+1}qn) \\ &\approx p^{x+1}qn \end{aligned} \tag{8}$$

if $p^{x+1}qn$ is small (the last approximation follows from the Taylor expansion for the exponent).

For example, consider the situation when one account has 10% of all forging power, and the others are relatively small. Then, according to (4), the probability that this account generates a block is $p \approx 0.111125$. Take $n = 1000000$, then,[4] according to (5)–(7), we have

$$\begin{aligned} \mathbb{E}R_n &\approx 6.00273, \\ \mathrm{Var}R_n &\approx 0.424, \\ \mathbb{P}[R_n \ge 7] &\approx 0.009\,. \end{aligned}$$

Next, consider a situation where one account controls considerably more coins, say, 30% of the total. Then we have $p \approx 0.387012$, and, according to the previous discussion,

$$\begin{aligned} \mathbb{E}R_n &\approx 11.15793, \\ \mathrm{Var}R_n &\approx 1.21812, \\ \mathbb{P}[R_n \ge 13] &\approx 0.03348\,. \end{aligned}$$

That is, we see that the maximal number of blocks in a row that even a very rich account can forge usually cannot be dramatically big. See, however, the discussion in Section 6, where more elaborate strategies for forging many blocks in a row are considered.

5. "Goodness" of a Blockchain and Concurrent Blockchains

In this section we consider the following attack scenario: account 1 (the "attacker", with balance b) temporarily disconnects from the network and forges its own blockchain; (s)he then reconnects hoping that his/her blockchain would be "better" than the blockchain produced by the "good" part of the network.

ISSN 2379-5980 (online)
DOI 10.5195/LEDGER.2016.46

Now, we need to define what it means for one chain to be "better" than another. Here, we consider the following rule: if there are two chains of the same length, the one that has smaller sum of inverse weights wins. Of course, other rules may be considered, which, hopefully, can be analysed in an analogous way.[5]

First, let us look at the distribution of the inverse weight of a block. In the case of the Exp-algorithm, everything is simple: as observed in Section 3, it has the exponential distribution with rate 1. This readily implies that the expectation of the sum of weights of n blocks equals n.

As for the U-algorithm, we begin by considering the situation when all relative balances are small. Analogously to (3), W being the weight of the block, for $x \ll (\max_k b_k)^{-1}$ we calculate

$$\begin{aligned}\mathbb{P}\Big[\frac{1}{W} > x\Big] &= \mathbb{P}\Big[\max_k \frac{b_k}{U_k} < \frac{1}{x}\Big] \\ &= \prod_k \mathbb{P}\Big[U_k > x b_k\Big] \\ &= \prod_k (1 - x b_k) \\ &= \exp \sum_k \ln(1 - x b_k) \\ &\approx e^{-x}, \end{aligned} \tag{9}$$

so also in this case the distribution of the inverse weight is approximately exponential with rate 1.

We consider now the situation when all balances except the first one are small, and $b := b_1$ need not be small. For the case of the U-algorithm, similarly to (9) we obtain for $x \in (0, 1/b)$

$$\begin{aligned}\mathbb{P}\Big[\frac{1}{W} > x\Big] &= \prod_k (1 - x b_k) \\ &= (1 - bx) \exp \sum_{k \geq 2} \ln(1 - x b_k) \\ &\approx (1 - bx) e^{-(1-b)x}, \end{aligned} \tag{10}$$

so

$$\begin{aligned}\mathbb{E}\frac{1}{W} &\approx \int_0^{1/b} (1 - bx) e^{-(1-b)x} \, dx \\ &= \frac{b e^{-\frac{1-b}{b}} + 1 - 2b}{(1-b)^2}. \end{aligned} \tag{11}$$

One can observe (see Fig. 2) that the right-hand side of (11) is strictly between $1/2$ and 1 for $b \in (0, 1)$.

Now, we start analysing the attack scenario when the attacker, with balance b, temporarily disconnects from the network and forges its own blockchain. While the account 1 is disconnected, the "good" part of the network produces blocks with weights having exponential distribution with rate $1 - b$, and thus each inverse weight has expected value $\frac{1}{1-b}$.

Let $X_1, X_2, X_3, \ldots$ be the inverse weights of the blocks produced by the "good part" of the network (after the attacker disconnects), and we denote by $Y_1, Y_2, Y_3, \ldots$ the inverse weights of the blocks produced by the attacker. We are interested in controlling the probability of the following event (which means that the blockchain produced by the attacker has smaller sum of inverse

ISSN 2379-5980 (online)
DOI 10.5195/LEDGER.2016.46

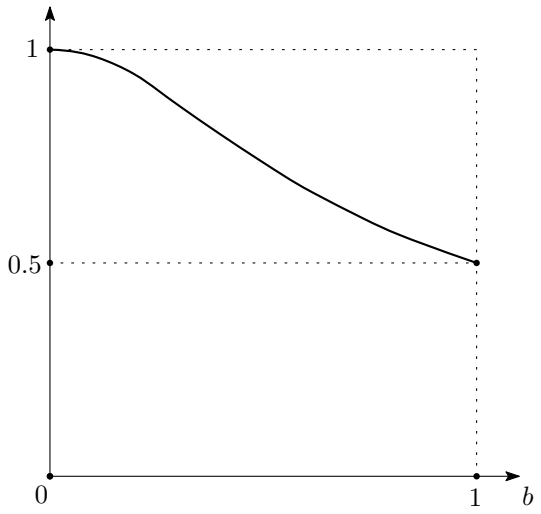

Fig. 2. Expectation of the inverse weight (as a function of b)

weights and therefore wins)

$$H_m = \{X_1 + \cdots + X_m - Y_1 - \cdots - Y_m \geq 0\}$$

for a "reasonably large" m (*e.g.*, $m = 10$ or so). If the probability of H_m is small, this means that the attacker unlikely will succeed attacking the network; on the other hand, if this probability is not small, then the system should be able to fence off the attack by other means, which we shall not discuss in this note.

We obtain an upper bound on the probability of the event H_m using Chebyshev's inequality: since the random variables are i.i.d., we write for any fixed $t > 0$

$$\begin{aligned}
\mathbb{P}[H_m] &= \mathbb{P}[X_1 + \cdots + X_m - Y_1 - \cdots - Y_m \geq 0] \\
&= \mathbb{P}\big[\exp\big(t(X_1 + \cdots + X_m - Y_1 - \cdots - Y_m)\big) \geq 1\big] \\
&\leq \mathbb{E}\exp\big(t(X_1 + \cdots + X_m - Y_1 - \cdots - Y_m)\big) \\
&= \big(\mathbb{E}e^{t(X_1 - Y_1)}\big)^m.
\end{aligned}$$

Since the above is valid for all $t > 0$, we have

$$\mathbb{P}[H_m] \leq \delta^m, \quad \text{where} \quad \delta = \inf_{t>0} \mathbb{E}e^{t(X_1 - Y_1)}. \tag{12}$$

It is important to observe that this bound is nontrivial (*i.e.*, $\delta < 1$) only in the case $\mathbb{E}X_1 < \mathbb{E}Y_1$.

For the U-algorithm, X_1 is exponentially distributed with rate $1 - b$, and Y_1 has uniform $(0, b^{-1})$ distribution. So, the condition $\mathbb{E}X_1 < \mathbb{E}Y_1$ is equivalent to $(1-b)^{-1} < (2b)^{-1}$, that is, $b < 1/3$. Then, for $b < 1/3$, the parameter δ from (12) is determined by

$$\delta = \delta(b) = b(1-b) \inf_{0<t<1-b} \frac{1 - e^{-t/b}}{t(1-b-t)} \tag{13}$$

(see the plot of $\delta(b)$ on Fig. 3), so we have

$$\mathbb{P}[H_m] \leq \delta(b)^m. \tag{14}$$

ISSN 2379-5980 (online)
DOI 10.5195/LEDGER.2016.46

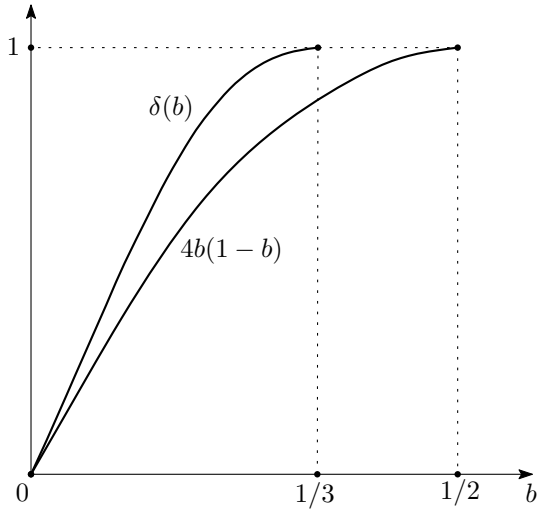

Fig. 3. The comparison of the rate functions $\delta(b)$ (for the U-algorithm) and $4b(1-b)$ (for the Exp-algorithm)

Let us mention that $\delta(b)$ is usually called the *rate function*, since it determines the rate of the exponential decay of the large deviation probabilities. For example, we have $\delta(0.1) \approx 0.439$. We have, however, $\delta(0.3) \approx 0.991$, which means that one has to take very large m in order to make the right-hand side of (14) small in this case.

For the Exp-algorithm, the attacker would produce blocks with inverse weights having exponential distribution with rate b, so each inverse weight has expected value $\frac{1}{b}$. Similarly to the above, one obtains that the condition $\mathbb{E}X_1 < \mathbb{E}Y_1$ is equivalent to $b < 1/2$, and

$$\mathbb{P}[H_m] \leq \big(4b(1-b)\big)^m \tag{15}$$

(that is, δ can be explicitly calculated in this case and equals $4b(1-b)$; observe that $4b(1-b) < 1$ for $b < 1/2$). As we see on Fig. 3, it holds that $\delta(b) > 4b(1-b)$ for $b \in (0, 1/3)$; that is, the large deviation probabilities of the "unfavorable" event decay more rapidly for the Exp-algorithm.

6. More on Account Splitting

In this section we analyze the following attack strategy: if the attacker wins the forging lottery at the current step but owns *several* accounts with inverse weights less than all the accounts of good guys (*i.e.*, these accounts of the attacker are first in the queue), then (s)he chooses which of his accounts will forge. Effectively, that means that (s)he has several independent tries for choosing the hash of the block, and so (s)he may be able to manipulate the lottery in his/her favor.[6]

We assume also that the network tries to protect itself from this kind of behaviour, in the following way: if an account was expected to forge a block (*e.g.*, was in the beginning of the forging queue) but did not forge, it loses the right to forge for some period. As far as the author knows, the Nxt's creator originally had plans to introduce this kind of rule at some stage, but, apparently, this idea was then abandoned.

Of course, following the above strategy requires that the balance of the winning accounts should be small (because of the ban for non-forging), but, as we shall see below, splitting into small parts is exactly the right strategy for maximizing the number of the best accounts in the

ISSN 2379-5980 (online)
DOI 10.5195/LEDGER.2016.46

queue.

First of all, let us estimate the probability that in the sequence of accounts ordered by the inverse weights, the first k_0 ones belong to the same person or entity, who controls the proportion b of all active balance. We will do the calculations for the case of the Exp-algorithm, since the attacker would have to split his money between many (or at least several) accounts (because the attacker's goal is to have several accounts in front of the forging queue, not just one of them), and, as observed in Section 3, in this situation both algorithms work essentially in the same way.

One obvious difficulty is that we do not know how exactly the money of the attacker is distributed between his/her accounts. We assume that the balances of the other accounts (those not belonging to the attacker) are relatively small. Also, let r be the *minimal* relative balance per account which still can forge. There may be some rule that sets the minimum balance for which the account is allowed to forge, or, in the absence of such a rule, r could be just the smallest (indivisible) unit of the currency. When all the money in an account is distributed between many accounts in such a way that each of these new accounts contains r units, we say that the (former) account is *split to dust*. Let us show the following result:

Proposition 6.1. *The best strategy for the attacker to maximize the probability that the best k_0 accounts belong to him/her (under the Exp-algorithm), is to split his/her accounts to dust.*

Recall that r is the minimal relative balance per account that is possible, and let us assume that the attackers money is held in accounts with relative balances $n_1 r, \dots, n_\ell r$, where $\ell \geq k_0$. Denote also $m = n_1 + \dots + n_\ell$, so that $b = mr$. Now, we make use of the elementary properties of the exponential probability distribution discussed in the beginning of Section 3. Consider i.i.d. exponential(r) random variables $Y_1, \dots, Y_m$, and let $Y_{(1)} < \dots < Y_{(m)}$ be the order statistics (cf. Section 6.6 of[15]) of this sample (*i.e.*, $Y_{(1)}, \dots, Y_{(m)}$ are $Y_1, \dots, Y_m$ arranged in the increasing order). Abbreviate $s_j = n_1 + \dots + n_j$, $s_0 := 0$; the idea is that $Y_{s_{j-1}+1}, \dots, Y_{s_j}$ "correspond" to the account worth $n_j r$. Then, by (5.6) of[16] we have that

$$Z_j = \min_{i=s_{j-1},\dots,s_j} Y_i, \qquad j = 1, \dots, \ell$$

are independent exponential random variables with rates $n_1 r, \dots, n_\ell r$. So, with this construction, the orders statistics of Z-variables form a subset of the order statistics of Y-variables; since the inverse weights of the attacker's accounts are the first k_0 order statistics of $Z_1, \dots, Z_\ell$, the claim follows.

The above proposition leads to a surprisingly simple (approximate) upper bound for the probability that the best k_0 accounts belong to the attacker. Assume that *all* the accounts in the network have the minimum relative balance r; then each account in the (ordered with respect to the inverse weights) sequence has probability b to belong to the attacker. Since k_0 should be typically small compared to the total number of accounts, we may assume that the ownerships of the first k_0 accounts are roughly independent, and this means that the probability that all the best k_0 accounts belong to the attacker should not exceed b^{k_0}.

Now, let us estimate the conditional probability $p^*(b)$ that the attacker wins the forging lottery on the next step, given (s)he was chosen to forge at the current step. The above analysis suggests that the number of the best accounts in the queue that belong to the attacker can be approximated by a geometric distribution with parameter b. Now, given that the attacker owns k best accounts

ISSN 2379-5980 (online)
DOI 10.5195/LEDGER.2016.46

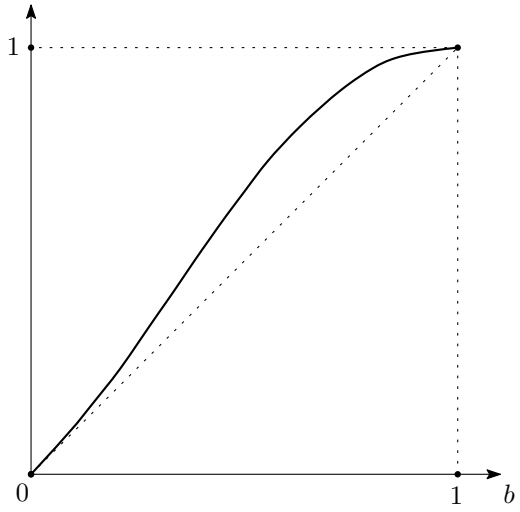

Fig. 4. The plot of $p^*(b) = 1 - \frac{(1-b)^2}{1-b(1-b)}$.

in the queue, the probability that (s)he wins the next forging lottery is $1-(1-b)^k$ (since there are k independent trials at his disposal, and the probability that all will fail is $(1-b)^k$).

Using the memoryless property of the geometric distribution (*i.e.*, as one can easily verify, $\mathbb{P}[X = k] = \mathbb{P}[X = k+n \mid X \geq n]$ if X has the geometric law) we have that, given that the winning account belongs to the attacker, he also owns the next $k-1$ ones with probability $(1-b)b^{k-1}$. So,

$$\begin{aligned} p^*(b) &= \sum_{k=1}^{\infty} (1-b)b^{k-1}(1-(1-b)^k) \\ &= (1-b)\sum_{j=0}^{\infty} b^j + (1-b)^2 \sum_{j=0}^{\infty} (b(1-b))^j \\ &= 1 - \frac{(1-b)^2}{1-b(1-b)}, \end{aligned} \tag{16}$$

see the plot of the above function on Fig. 4. The quantity $p^*(b)$ is almost b for small stakes (*e.g.*, 0.1099 for $b = 0.1$), but dramatically increases for large b. For $b = 0.9$, for instance, this probability becomes 0.989, *i.e.*, the attacker will be able to forge typically around 90 blocks in a row, instead of just 10.

Observe also that this calculation applies to the following strategy of the attacker: take the *first* of his accounts that assures that (s)he forges on the next step (so that the attacker minimizes the number of his accounts that will get banned). For this strategy, the attacker looks only one step to the future. One can consider also a more advanced strategy: since the future is deterministic, the attacker can try to calculate deeper into the future. We shall prove now that, under current implementation of the forging algorithm, *an attacker who owns more than* 50% *of all NXT can eventually forge all the blocks* (*i.e.*, at some moment (s)he starts forging and never stops).

To prove this, we need first to recall the model of *Galton-Watson* branching process (cf. *e.g.* Athreya, Ney for the general theory).[13] In general, the Galton-Watson branching process $(Z_k, k \geq 0)$ with discrete time can be described in the following way. The random variable Z_k stands for the number of *particles* in the k^{th} generation.[7] Let $p_0, p_1, p_2, \ldots \geq 0$ be such that

ISSN 2379-5980 (online)
DOI 10.5195/LEDGER.2016.46

$p_0+p_1+p_2+\cdots=1$; the vector $(p_k, k\geq 0)$ thus defines a probability distribution on nonnegative integers; we assume $p_1<1$ to rule the degenerate case out. To form the $(k+1)^{\text{th}}$ generation, each particle from the k^{th} generation is independently substituted by a random number of other particles; these random numbers are (independently) sampled from the probability distribution $(p_k, k\geq 0)$. One is then interested if the generations' sizes grow to infinity (the process *survives*) with positive probability, or it eventually happens that there are no particles at all starting from some generation (the process *dies out*). Let $\mu=p_1+2p_2+3p_3+\cdots$ be the *mean offspring* number of a particle. The branching process is called *subcritical* if $\mu<1$, *critical* if $\mu=1$, and *supercritical* if $\mu>1$. The fundamental result of the theory of branching processes is that the process dies out if and only if $\mu\leq 1$, cf. *e.g.* Theorem 1 of Chapter 1 of Athreya, Ney.[13]

Now, the goal is to apply the above to our analysis. Assume that the block ℓ_0+1 is about to be forged. Let $Z_0:=1$, and let Z_1 be the number of attacker's accounts that are first in the queue (*i.e.*, win over any account not belonging to the attacker). Now, the attacker can choose which of these Z_1 accounts will forge. Let $Z_2^{(j)}$, $j=1,\ldots,Z_1$ be the number of attacker's accounts that are first in the queue for the block ℓ_0+2, provided (s)he has chosen the j^{th} account to forge at the previous step. Let $Z_2=Z_2^{(1)}+\cdots+Z_2^{(Z_1)}$. Then, we define $Z_3, Z_4, Z_5, \ldots$ in an analogous way; it is then elementary to see that $(Z_n, n\geq 0)$ is a Galton-Watson branching process with the offspring law given by $p_k=(1-b)b^k, k\geq 0$. The mean number of offspring

$$\mu=\sum_{k=1}^{\infty} k(1-b)b^k=\frac{b}{1-b}$$

is strictly greater than 1 when $b>\frac{1}{2}$. Since a supercritical branching process survives with positive probability (in fact, one can calculate that in this case the probability of survival equals $\frac{1-b}{b}$), the attacker can choose an infinite branch in the genealogical tree of the branching process, and follow it.

The attacker can also use the same strategy with $b\leq\frac{1}{2}$; this, of course, will not permit him to forge all the blocks, but there is still a possibility to increase the number of generated blocks. Let us do the calculations. The probability generating function of the number of offspring (corresponding to the geometric distribution) is

$$g(s)=\sum_{j=0}^{\infty} s^j(1-b)b^j=\frac{1-b}{1-sb},$$

so $a_n(b):=\mathbb{P}[Z_n=0]$ satisfies the recursion

$$a_1(b)=1-b, \qquad a_{n+1}(b)=\frac{1-b}{1-ba_n(b)},$$

and the mean lifetime $h(b)$ of the branching process is

$$h(b)=\sum_{n=1}^{\infty}(1-a_n(b)).$$

Unfortunately, usually there is no closed form expression for the expected lifetime of a subcritical branching process, but, as a general fact, it holds that $h(b)\sim b$ as $b\to 0$ and $h(b)\to\infty$ as $b\to\frac{1}{2}$. Since each streak of attacker's blocks has the expected length $b^{-1}h(b)$ and the expected length of each streak of good guy's blocks is b^{-1}, the attacker is able to forge the proportion $\frac{h(b)}{1+h(b)}$ of all blocks (see Fig. 5).

ISSN 2379-5980 (online)
DOI 10.5195/LEDGER.2016.46

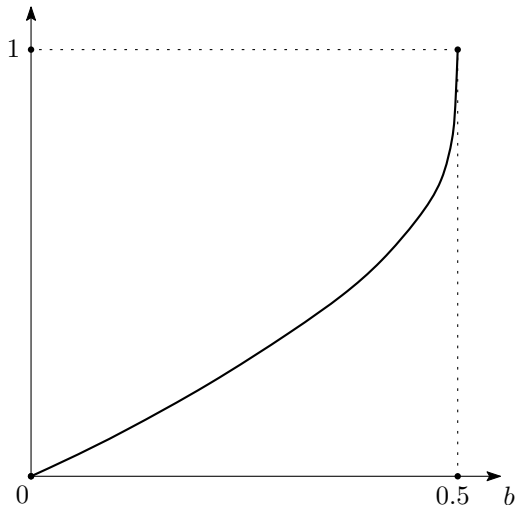

Fig. 5. The plot of $\frac{h(b)}{1+h(b)}$

Observe that, for additional network protection against this type of attack, one can consider the idea to have a *lower limit* for forging, *i.e.*, an account that has less than some fixed amount cannot forge at all.

7. Conclusions

In this paper we presented an analysis of a model of a "pure-PoS" algorithm for building a blockchain; our model is very closely related to the forging algorithm implemented in the Nxt cryptocurrency. We consider two versions of the algorithm for choosing the forger of the next block: the "completely fair" (with respect to the probability of generating the next block) Exp-algorithm, and the "slightly biased" U-algorithm (the latter is, however, implemented in the Nxt). For both of them, we analyse also the effect (on that probability) of splitting one account's wealth into several accounts. In general, we found that splitting has no effect on the (total) probability of block generation under the Exp-algorithm, and this probability always decreases under the U-algorithm, so users are incentivized to store the money together. However, the difference is usually not very significant (even if the account is split to many small parts) unless the account has quite a lot of money. Then, we analyze an attack strategy when one account (or a group of accounts) temporarily disconnects from the main network and tries to forge a "better" blockchain than the one forged by other accounts, in the situation when one (malicious) entity has proportion b of total amount of NXT, and the stakes of the others are relatively small. It should be observed that the probability that the attacker forges a better chain of length m does not tend to 0 (as $m \to \infty$) if the attacker has at least $1/3$ of all *active* balances in the network in the case of the U-algorithm (correspondingly, at least $1/2$ in the case of the Exp-algorithm). We also study the distribution of the longest run of blocks generated by one particular account (or group of accounts), and consider the "branching process" attack where an attacker who owns more than 50% of all NXT can eventually forge all the blocks (*i.e.*, at some moment (s)he starts forging and never stops).

As a general conclusion, the pure-PoS scheme turned out to be a very interesting model from the mathematical point of view; one of the important features that makes the difference is the

ISSN 2379-5980 (online)
DOI 10.5195/LEDGER.2016.46

pseudo-random nature of the algorithm for choosing the next forger (differently of the "pure randomness" of *e.g.* Bitcoin). This feature can be potentially very useful (if the next few forgers are known, they may *e.g.* "pre-approve" transactions thus effectively reducing confirmation time), but also generates some new attack vectors, as *e.g.* the "branching process" attack of Section 6. Nevertheless, our analysis shows that this scheme remains sound, as long as there is no dramatically large concentration of wealth (which is believed to be very unlikely to happen in practice).

Notes and References

[1] This paper is a revised and shortened version of MTHCL. "The math of Nxt forging." (2014) `www.docdroid.net/ecmz/forging0-5-2.pdf.html`

[2] `http://consensusresearch.org/`

[3] There are, however, some additional security measures in place, such as hard-coded checkpoints.

[4] Just for comparison, currently the Nxt blockchain has ≈ 1.02 million blocks.

[5] Currently, the Nxt uses another rule of choosing "better" chain, since the "one block each minute" regime is only in the future plans.

[6] Note that the results of this section do not contradict our previous analysis, since we did not consider this attack strategy before.

[7] "Particles" are just placeholders that can be anything in concrete applications: *e.g.* neutrons in a nuclear chain reaction, individual organisms in applications related to population dynamics in biology and/or sociology, or even English surnames (which was the subject of Galton's original research).

[8] Andruiman. "Nxt forging algorithm: simulating approach." (2014) `https://scribd.com/doc/243341106/Nxt-forging-algorithm-simulating-approach`

[9] Andruiman. "PoS forging algorithms: formal approach and multibranch forging." (2014) `https://scribd.com/doc/248208963/Multibranch-forging`

[10] Andruiman. "Multibranch forging algorithms: tails-switching effect and chain measures." (2015) `https://scribd.com/doc/256073121/Multibranch-forging-algorithms-tails-switching-effect-and-chain-measures`

[11] Andruiman. "PoS forging algorithms: multi-strategy forging and related security issues." (2015) `https://scribd.com/doc/256072839/PoS-forging-algorithms-multi-strategy-forging-and-related-security-issues`

[12] MTHCL. "The math of Nxt forging." (2014) `www.docdroid.net/ecmz/forging0-5-2.pdf.html`

[13] Athreya, K. B., Ney, P. B. *Branching Processes.* Berlin–Heidelberg–New York: Springer-Verlag (1972).

[14] Gordon, L., Schilling, M. F., Waterman, M. S. "An extreme value theory for long head runs." *Probab. Theory Relat. Fields* **72**, 279–287 (1986).

[15] Ross, S. M. *A First Course in Probability*, 8th ed. (2009).

[16] Ross, S. M. *Introduction to Probability Models.* 10th ed. (2012).

[17] Schilling, M. "The Longest Run of Heads." *The College Math J.*, **21** (3), 196–206 (1990).

ISSN 2379-5980 (online)
DOI 10.5195/LEDGER.2016.46