MARCELO SALHAB BROGLIATO

ESSAYS IN COMPUTATIONAL MANAGEMENT SCIENCE

ESSAYS IN COMPUTATIONAL MANAGEMENT SCIENCE

MARCELO SALHAB BROGLIATO

Escola Brasileira de Administração Pública e de Empresas
Fundação Getulio Vargas

January 2018

Marcelo Salhab Brogliato: *Essays in Computational Management Science,* © January 2018

SUPERVISORS:
Alexandre Linhares LOCATION:

Rio de Janeiro

CONTENTS

LIST OF FIGURES

LIST OF TABLES

Part I

AN ALTERNATIVE STRUCTURE TO A BLOCKCHAIN: CONFIRMATIONS WITHOUT BLOCKS

1

INTRODUCTION

Once I understood the logic behind change addresses...
`ripemd160(SHA256(pubkey))`... the inneficiency...
the wasted space; the wasted energy... something clicked.

This is military-grade cryptography and full-blown paranoia.
Whoever did it hears voices whispering 'they know everything.'

Other than that, it's a get-rich-quick-scheme
that looks exactly like a get-rich-quick-scheme.

— Alex Linhares, EBAPE/FGV

We are watching History being made.
Or History being repeated.

— David Collum, Cornell University, 2013

The main problem with digital money is how to prevent double spending. As the money is digital, and copies can be made *ad nauseam*, what can prevent counterfeiting? That is, what would prevent users from sending copies of the same money to two (or more) people? That is precisely the problem solved by Bitcoin and its underlying blockchain technology. The current solution behind fiat money is having a single issuer, a central bank — then trusting the financial institutions.

Bitcoin (BTC) is the first digital currency, also known as digital money, internet money, and cryptocurrency. It is the first currency based on cryptography techniques, distributed and decentralized, and with no central bank. Bitcoin is distributed since its ledger is public and is stored in thousands of computers. It is decentralized because there is no authority (or government) who decides its future — any decision must be accepted by its community. The security of Bitcoin relies on digital signature technology and network agreement. While digital signature ensures ownership, i.e., the funds may only be spent by their owners, and nobody else; the network agreement both prevents double spending and ensures that all processed transactions have sufficient funds. In short, every

transaction must spend only unspent funds, must have enough funds available, and must be signed by its owners, authorizing its execution. When all these requirements are met, the funds are transferred.

According to Barber et al. [3], despite the 30-year literature on e-cash, most of the proposed schemes requires a central authority which controls the currency issuance and prevents double spending. The no central point of trust and predictable money supply together with a clever solution to the double-spending problem is what separates Bitcoin from the previous e-cash philosophies.

Bitcoin provides interesting incentives to all players, namely the users and the miners. On the one hand, users may have incentives to use Bitcoin because of the following: (i) the fees are small and does not depend on the amount being transfered — but only in the size (in bytes) of the transaction — (ii) the transfers will be confirmed in a well-known period; (iii) it is not possible to revert an already confirmed transfer, not even with a judicial order; and (iv) and the currency issuance rate is well-known and preset in Bitcoin's rules, which makes the Bitcoin supply predictable and trustworthy, different from fiat currencies which depends on decisions of their central banks — i.e., it would be virtually impossible to face a hyper inflation in Bitcoin due to currency issuance. On the other hand, miners have incentive to mine Bitcoin because new Bitcoins are found every ten minutes, and they may also receive the fees of unconfirmed transactions. These incentives have kept the Bitcoin network up and running since 2009 with barely any interruptions (99.99% uptime).

Since 2009, Bitcoin has been growing and becoming more and more used all around the world. It started as an experiment based on a seminal work by Nakamoto [15] and expanded to the most important and successful cryptocurrency with a highly volatile $192 billion market capitalization, as of this writing [6]. There are hundreds of companies investing in different uses of the technology, from exchanges to debit cards, and billions of dollars being invested in the new markets based on Bitcoin's technology.

Despite Bitcoin's huge success, there are still many challenges to be overcome. We will focus on a specific subset of those challenges, namely the scaling, decentralization, and spam challenges. One important challenge that we will skip is to reduce the size of the ledger (or blockchain), which today is around 125GB and is growing at a rate of 4.5GB per month [5].

The network must scale to support hundreds of transactions per second, while its capacity is around only eight transactions per second. Thus, the more Bitcoin becomes popular, the more saturated the network is. Network saturation has many side effects and may affect the players' incentive to keep the network running. The

transaction fees have to be increased to compete for faster confirmation. The pool of unconfirmed transactions grows indefinitely, which may cause some transactions to be discarded due to low memory space available, as the previously predictable confirmation time of transactions becomes unpredictable.

Bitcoin seems to have the most decentralized network between the cryptocurrencies, even so, there are few miners and mining pools which together control over 50% of the network's computing (hash)power. Thus, they have an oversized influence when it comes to changes in the Bitcoin protocol's behavior, and it is also seen as a problem to be solved. The more decentralized, the more trustworthy Bitcoin is.

Generating new transactions in Bitcoin has a tiny cost because one only has to generate the transaction itself, digitally sign it, and propagate in the Bitcoin network. On the one hand, it means that any device is capable of generating new transactions, but, on the other hand, it makes Bitcoin susceptible to spam attacks. One may generate hundreds of thousands of new valid transactions, overloading the unconfirmed transactions pool and saturating the network.

The number of ideas and publications focusing on improving Bitcoin's design and overcoming those challenges is increasing every day. Many of these proposals are organized into BIPs (Bitcoin Improvement Proposals) which are discussed and implemented by the community; while others come in the form of whitepapers and alternative software forks (which would include the need of a protocol upgrade). Other proposals are published in blogs and forums, describing new cryptocurrencies. Bitcoin's community hardly ever publishes their ideas in academic journals, preferring instead, of BIPs, white papers, and web discussions.

After the launch of Bitcoin, more than 1,000 other cryptocurrencies have been created (REF). In general, they are Bitcoin-like, which means they use similar technologies, including the blockchain. Some cryptocurrencies differs a lot from Bitcoin, like the ones which use the DAG model [9, 17, 13, 19, 14, 20]. We are especially interested in one of them: Iota.

Iota uses a DAG model, called tangle, which has a different design than Bitcoin's blockchain. It has neither mining nor confirmation blocks and transaction fees. Each transaction has its own proof-of-work and is used to confirm other transactions, forming a directed acyclic graph. In Iota, as transactions confirm transactions, the network benefits from a high volume of new transactions. Hence, theoretically, it scales to any large number of transactions per second. The scaling problem of tangle is exactly the opposite of Bitcoin's. It must have at least a given number of transaction per seconds;

otherwise, the transactions are not confirmed, and the cryptocurrency does not work.

The present work intends to analyze a new architecture which lies between Bitcoin and Iota and may be a viable solution to both Bitcoin's scaling, centralization, and spam problems. We also present a mathematical analysis of Bitcoin's mining, forking, and safety.

2

BITCOIN & BLOCKCHAIN

In Nakamoto's (2009) seminal paper, there is no distinction between bitcoin and blockchain. They are just one thing which solves an important theoretical problem: how to create a distributed and decentralized digital form of hard money on the internet, in which all users can agree as to whom is entitled to which funds.

But, in practice, it is interesting to separate these concepts. Bitcoin uses the blockchain technology to create a distributed ledger, while the blockchain is a technology which allows information to be stored in an immutable and distributed way.

The blockchain technology works through the creation of new blocks. Each new block confirms that all the previous blocks are valid and have not been tampered with. The mechanism that assures the immutability is proof-of-work, which makes it computationally infeasible to tamper with previous transaction records without having to recalculate all the previous proof-of-works faster than all of the remaining machines of the network. The network agrees that work should be done in the block at the longest chain in the blockchain.

The proof-of-work is a mathematical problem with the following characteristics: (i) it is hard to find a solution; (ii) this hardness level may be adjusted; and (iii) it is fast to check whether the proposed solution is correct.

Bitcoin's blockchain uses the mathematical problem of finding a random number which, after being applied to the hash function SHA-256 twice, results in a number smaller than a given threshold A. As SHA-256 is a pseudo-random function, its output is uniformly distributed between 0 and $2^{256} - 1$ [11]. Thus, if the given number is $A = 2^{255}$, one has probability 50% of finding a solution (just the most significant bit of the hash needs to be zero). But if the given number is $A = 2^{240}$, one has probability 0.0015% of finding a solution (as the 16 most significant bits of the hash must equal zero). Hence, finding a solution is a hard problem which difficulty depends on the given number A. The lesser the given threshold A, the higher the difficulty. On the other side, checking whether a solution is correct is fast, one just has to apply the SHA-256 twice and compare.

By design, Bitcoin's blockchain proof-of-work difficulty is dynamically adjusted every 2016 blocks to keep an average pace of 10 minutes between block creation. Thus, the goal is to adjust the difficulty every 14 days. If it takes less than 14 days to find 2016 blocks, that means the network's hash power has increased; thus the

difficulty is increased. If it takes more than 14 days to find 2016 blocks, it means the network's hash power has decreased; thus the difficulty is decreased.

When miners are finding a solution to a new block, they are mining or working in the new block. A block is found when a solution to the proof-of-work is found. When a new block is found, it indirectly confirms all the previous blocks in the chain and their transactions. It may happen that two miners find two different blocks in a small interval of time. In this case, both miners will propagate their blocks, and the network will randomly choose one of them as the next block. This phenomenon is called a fork. Thus, when the next block is found, one of those blocks will be confirmed, and the other will be ignored and referred to as an orphan block. As blocks confirm transactions, all the transactions in the orphan block which have not been confirmed by another block will return to the unconfirmed transaction pool. Hence, it is not safe to accept a transaction when it is confirmed by only one block. It is the idea behind the rule of thumb of waiting for at least six confirmations before accepting a transaction.

Newly propagated blocks are validated by the Bitcoin's network. If the solution of the proof-of-work is incorrect or if any transaction included in the block has any issue, then the block is discarded. In order to work properly, the whole network must agree in what is allowed and what is not. Should one think that something should be allowed and accept it in their blocks, the remaining of the network will discard their newly propagated blocks. That is why Bitcoin's network is distributed and decentralized. Everything depends on the agreement of the network, or, precisely, the agreement of the owners of at least 50% of the hash power. Even if the remaining 49% disagrees, the 50% or more who agree will generate, on average, more blocks than the remaining of the network and their rules will prevail on the longest chain. If a disagreement between miners' rules happens, that is referred as either a hard-fork or a soft-fork (in general, a hard-fork relaxes the constraints, while the latter hardens them).

A practical example of a network disagreement is the increase of the block size. No group with more than 50% of the network's hash power agreed into increasing the maximum block size to increase the number of transactions confirmed by a block and thus increasing the network's capacity. Hence, the capacity remains the same, and the community has been discussing the issue in search of a consensus.

Bitcoin uses the blockchain technology to create a distributed ledger. It allows every new block to generate new bitcoins and also to collect the fees from the confirmed transactions within the block. Bitcoin's transactions have two main parts: (i) inputs, and (ii) outputs. Each transaction sends bitcoins from one or more input

addresses to one or more output addresses. In order to prove that one is the owner of the input bitcoins, one must digitally sign the transaction proving such ownership.

The digital signature scheme used by Bitcoin is based on a pair of private and public keys. The private key is used to sign the transaction, while the public key is used to check whether the signature is valid. Thus, the owner of some Bitcoin funds is, in fact, the owner of a pair of private and public keys. The private key must never be publicly published, as whoever has access to the private key is able to spend its funds. In other words, in order to protect their funds, the owners must protect the private key. If one loses their private key, unfortunately, access to their funds will be lost forever. The public key may be used to a proof-of-ownership, i.e., one may publish the public key with some message digitally signed by the private key, proving that he/she is the owner of the funds.

Bitcoins (BTCs) owned by someone are, in fact, unspent outputs in one or more transactions. For instance, one may have 6 BTCs spread between three transactions' unspent output: the first with 1 BTC, the second with 2 BTCs, and the third with 3 BTCs.

In a transaction, the inputs are pointers to other transactions' outputs (which they are spending). A transaction output may only be spent once and thus may not be partially spent. For instance, when one has 3 BTCs in one transaction output and would like to send 1 BTC to a friend, they have to create a transaction with one input spending the 3 BTCs and two outputs, one for the friend with 1 BTC and one's change with 2 BTCs.

Each transaction's output has a script that is executed by the miners to check whether one has or has not permission to spend that output. In other words, whether one has the ownership of that output. In order to execute these scripts, the miners also need some data. This data is given by the transaction which is spending the output.

The output's scripts usually checks whether the public key is valid and whether the digital signature was signed by the private key associated to that public key. Although there are only 3 commonly used scripts, one may create a custom script using the Bitcoin's script language [1].

The input contains the data which prove that the sender is the owner of the referred outputs, i.e., the input which must be accepted by the output scripts being spent. Usually, each input has the public key of the sender and a digital signature.

Those users accustomed to block explorers may have been misled by the transaction information that these websites provide. For example, suppose a miner receives a transaction with "one input

1 Your courageous author once tried to make a transaction with custom script to try to double spend a deposit to an exchange, only to learn through this intrepid adventure that Bitcoin allows only 3 script patterns and the others are treated as invalid.

from address A1". This "one input" actually consists of a pointer to a previous unspent output (i.e., there is no "input" address, as is displayed, but only a pointer). This pointer reference allows lookup to be executed in $O(1)$ time.

After lookup, the miner knows how many BTC tokens are available at that unspent output. But, in order to certify ownership of that output, the miner receives instructions in the form of a script with the rules that lead to the desired unspent output address (and this one is displayed as the input address by those websites). Because this process requires a digital signature, only the holder of the corresponding private key is able to sign such transaction.

Next, we will do a mathematical analysis of Bitcoin in order to better understand its minings properties, how a fork would affect the network and its security against attackers.

3

ANALYSIS OF BITCOIN

The primary objective of this chapter is to increase our understanding of the Bitcoin through mathematical tools.

3.1 HASH FUNCTION

Hash functions has been widely studied in computer science. In short, a hash function $h : \{0,1\}^\infty \to \{0,1\}^n$ has the following properties:

1. $x = y \Rightarrow h(x) = h(y)$
2. $h(x) \sim \mathcal{U}(0, 2^n - 1)$, where $\mathcal{U}$ is the uniform distribution, i.e., $\forall a \in [0, 2^n - 1], \mathbf{P}(h(x) = a) = \frac{1}{2^n}$

In other words, when two inputs are the same, they have the same output. But, when the inputs are different, their outputs are uniformly distributed. Clearly, the hash functions are surjective but not injective. They are not injective because the image of h has only 2^n elements and the domain has infinite elements. When $x \neq y$ and $h(x) = h(y)$, we say that x and y are a collision. A hash function is considered to be safe when it is unknown how to quickly find a collision of a given hash, i.e., one has to check all possible values until the correct one is found (known as the brute-force attack).

Bitcoin uses two hash functions: HASH-160 and HASH-256. The first has $n = 160$ and consists of the composition of *SHA-256* and *RIPEMD-160*. The latter has $n = 256$ and applies *SHA-256* twice. The first is used in transactions' scripts and the latter in the mining algorithm. For both hash functions, it is infeasible to run a brute-force attack because it would demand, on average, either 2^{160} or 2^{256} trials, and those would take a tremendous amount of time even for the fastest known processors.

For further information about hash functions, see Gilbert and Handschuh [11], Dobbertin et al. [10].

3.2 MINING ONE BLOCK

Let $\mathbb{B}$ be the set of Bitcoin blocks and $h : \mathbb{B} \to \{0,1\}^{256}$ be the Bitcoin *HASH-256* function. The mining process consists of finding $x \in \mathbb{B}$ such as $h(x) < A$, where A is a given threshold. The smaller the A, the harder to find a new block. In fact, $\mathbf{P}(h(x) < A) = \frac{A}{2^{256}}$.

Hence, in order to find a new block, one must try different inputs $(x_1, x_2, \ldots, x_k)$ until they find a solution, i.e., all attempts will fail

($h(x_i) \geqslant A$ for $i < k$) but the last ($h(x_k) < A$). The probability of finding a solution exactly in the k^{th} attempt follows a geometric distribution. Let X be the number of attempts until a success, then $\mathbf{P}(X = k) = (1-p)^{k-1}p$, where $p = \frac{A}{2^{256}}$. Also, we have $\mathbf{P}(X \leqslant k) = 1-(1-p)^k$. The average number of attempts is $\mathbf{E}(X) = 1/p$ and the variance is $\mathbf{V}(X) = \frac{1-p}{p^2}$.

In the Bitcoin's protocol, the given number A is adjusted so that the network would find a new a block every 10 minutes, on average. Suppose that the Bitcoin's network is able to calculate H hashes per second — H is the total hash rate of the network. The time required to find a solution would be $T = X/H$, and $\mathbf{E}(T) = \mathbf{E}(X)/H$ would be the average number of seconds to find a new block. So, the rule of finding a new block every 10 minutes ($\eta = 600$ seconds) — on average — leads to the following equation: $\mathbf{E}(T) = \eta = 600$. So, $\mathbf{E}(T) = \mathbf{E}(X)/H = \frac{1}{pH} = \eta = 600 \Rightarrow p = \frac{1}{\eta H}$. Finally, $\mathbf{E}(X) = \eta H$, $\mathbf{E}(T) = \eta$, $\mathbf{V}(X) = (\eta H)^2 - \eta H$, and $\mathbf{V}(T) = \eta^2 - \eta/H$.

The cumulative distribution function (CDF) of T is $\mathbf{P}(T \leqslant t) = \mathbf{P}(X/H \leqslant t) = \mathbf{P}(X \leqslant tH) = 1-(1-p)^{tH} = 1-\left(1-\frac{1}{\eta H}\right)^{tH}$. But, as the Bitcoin's network hash rate is really large, we may approximate the CDF of T by $\lim_{H\to\infty} \mathbf{P}(T \leqslant t) = 1-e^{-\frac{t}{\eta}}$, which is equal to the CDF of the exponential distribution with parameter $\lambda = \frac{1}{\eta}$.

Theorem 1. *When $H \to +\infty$, the time between blocks follows an exponential distribution with parameter $\lambda = \frac{1}{\eta}$, i.e., $\lim_{H\to+\infty} \mathbf{P}(T \leqslant t) = 1-e^{-\frac{t}{\eta}}$.*

Proof.

$$\begin{aligned}\mathbf{P}(T \leqslant t) &= 1-(1-p)^{tH} \\ &= 1-\left(1-\frac{1}{\eta H}\right)^{tH}\end{aligned}$$

Replacing $u = \eta H$,

$$\begin{aligned}\lim_{H\to+\infty} \mathbf{P}(T \leqslant t) &= \lim_{u\to+\infty} 1-\left(1-\frac{1}{u}\right)^{\frac{tu}{\eta}} \\ &= \lim_{u\to+\infty} 1-\left[\left(1-\frac{1}{u}\right)^{u}\right]^{\frac{t}{\eta}} \\ &= 1-(1/e)^{\frac{t}{\eta}} \\ &= 1-e^{-\frac{t}{\eta}}\end{aligned}$$

□

Now, we would like to understand from which value of H it is reasonable to assume that T follows an exponential distribution.

Theorem 2. $x > M \Rightarrow |(1+1/x)^x - e| < e/M$.

Proof. Let's use the classical inequality $\frac{x}{1+x} < \log(1+x) < x$ for $x > -1$. So, $\frac{1/x}{1+1/x} < \log(1+x) < 1/x$. Simplifying, $\frac{1/x}{1+1/x} = 1/(1+x)$. Thus, $1/(1+x) < \log(1+1/x) < 1/x \Rightarrow x/(1+x) < x\log(1+1/x) < 1$.

As $\log(1+\frac{1}{M}) > 0$ and $1 < 1+\log(1+\frac{1}{M})$.

$x > M \Rightarrow 1/x < 1/M \Rightarrow 1+1/x < 1+1/M \Rightarrow 1/(1+1/x) > 1/(1+1/M) \Rightarrow x/(1+x) > M/(1+M)$.

Again, $\log(1+x) < x \Rightarrow \log(1-1/M) < -1/M \Rightarrow 1+\log(1-1/M) < (M-1)/M < M/(1+M)$, since $(x-1)/x < x/(x+1)$.

Hence, $1+\log(1-1/M) < M/(1+M) < x/(1+x) < x\log(1+1/x)$, and $x\log(1+1/x) < 1 < 1+\log(1+\frac{1}{M})$.

Finally,

$$
\begin{aligned}
1+\log(1-1/M) < x\log(1+1/x) &< 1+\log(1+\frac{1}{M}) \\
e^{1+\log(1-1/M)} < e^{x\log(1+1/x)} &< e^{1+\log(1+\frac{1}{M})} \\
e\cdot e^{\log(1-1/M)} < e^{\log((1+1/x)^x)} &< e\cdot e^{\log(1+\frac{1}{M})} \\
e(1-1/M) < (1+1/x)^x &< e(1+\frac{1}{M}) \\
e-e/M < (1+1/x)^x &< e+e/M \\
-e/M < (1+1/x)^x - e &< e/M
\end{aligned}
$$

Therefore, $|(1+1/x)^x - e| < e/M$. □

We may consider H big enough to say that T follows an exponential distribution when $e/H < \epsilon$, where ϵ is the maximum approximation error. When $\epsilon = 10^{-6} \Rightarrow H > e\cdot 10^6$. So, when $H > 2.6$Mh/s, our approximation is good enough.

The symmetrical confidence interval with level α would be $[t_0, t_1]$, where $\mathbf{P}(t_0 < T < t_1) = 1-\alpha$, $\mathbf{P}(T < t_0) = \alpha/2$, and $\mathbf{P}(T > t_1) = \alpha/2$. These conditions give the following equations: $1-e^{-t_0/\eta} = \alpha/2$, and $e^{-t_1/\eta} = \alpha/2$. Solving these equations, we have $t_0 = -\eta\ln(1-\alpha/2)$, and $t_1 = -\eta\ln(\alpha/2)$.

For instance, if $\alpha = 10\%$, then $t_0 = 30.77$ and $t_1 = 1797.44$ (or $[0.51, 30.76]$ in minutes). Thus, 90% of the time the intervals between blocks are between 30 seconds and 30 minutes, with average of 10 minutes.

The fact that the time between blocks follows an exponential distribution with $\lambda = 1/\eta = p$H may be used to estimate the total network's hash rate (or a miner's hash rate). For further information, see [16].

3.3 MINING SEVERAL BLOCKS

Let $T_1, T_2, T_3, \ldots, T_n$ be the time to find the first block (T_1), then the time to find the second block (T_2), and so on. Let's analyze the dis-

tribution of $Y_n = \sum_{i=1}^{n} T_i$ which is the total time to find the next n blocks. As Y_n is the sum of random variables which follow an exponential distribution with same $\lambda = \frac{1}{\eta}$, then $Y_n \sim \text{Erlang}(n, \frac{1}{\eta})$. Thus, the CDF of Y would be $\mathbf{P}(Y_n < t) = 1 - \sum_{k=0}^{n-1} \frac{1}{k!} e^{-\lambda t} (\lambda t)^k$.

Many exchanges require at least six confirmations in order to accept a deposit in Bitcoin. So, for $n = 6$, $\mathbf{P}(Y_6 < 1 \text{ hour}) = \mathbf{P}(Y_6 < 3600) = 0.5543$, i.e., only 55% of the deposits will be accepted in one hour. The symmetrical confidence interval with $\alpha = 10\%$ is $[27, 105]$ in minutes. Thus, 90% of the times, it will take between 27 minutes and 1 hour and 45 minutes to have your deposit accepted — assuming that your transaction will be confirmed in the very next block. The pdf of Y_6 is shown in Figure 1, in which the 10% symmetrical confidence interval is shown in the white area. The average total time of six confirmations is $\mathbf{E}(Y_6) = 6 \cdot 600 = 3600 = 60$ minutes.

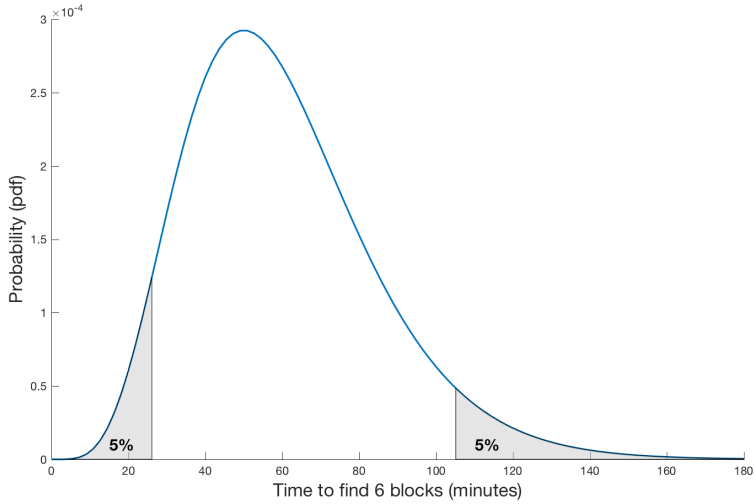

Figure 1: Probability density function of Y_6, i.e., probability of finding 6 blocks after time t. The shaded areas shows the lower 5% and upper 5% of the pdf.

3.4 MINING FOR A MINER

Let's analyze the probability of finding a new block for a miner who has α percent of the network's total hash rate. Let $T_\alpha = \frac{X}{\alpha H}$ be the time required for the miner to find a new block. As $T_\alpha = \left(\frac{1}{\alpha}\right) T$, when $H \to +\infty$, T_α also follows an exponential with parameter $\lambda_\alpha = \frac{\alpha}{\eta}$. Hence, we confirm the intuition that the miner with α percent of the network's total hash power will find α percent of the blocks.

Theorem 3. *When the miner with α percent of the network's total hash rate is part of the mining network, $\mathbf{P}(\text{next block is from } T_\alpha) = \alpha$.*

Proof.

$$\begin{aligned}\mathbf{P}(\text{next block is from } T_\alpha) &= P\left(T_\alpha = \min\{T_\alpha, T_{1-\alpha}\}\right)\\ &= \frac{\lambda_\alpha}{\lambda_\alpha + \lambda_{1-\alpha}}\\ &= \frac{\alpha/\eta}{\alpha/\eta + (1-\alpha)/\eta}\\ &= \frac{\alpha}{\alpha + 1 - \alpha}\\ &= \alpha.\end{aligned}$$

□

Theorem 4. *When one miner with α percent of the network's total hash rate multiplies their hash rate by m, the probability of this miner find the next block is multiplied by $\frac{m}{m\alpha+1-\alpha}$.*

Proof. When miners increase their hash rate, they also increase the network's total hash rate. Let H be the network's hash rate before the increase. Thus, the network's total hash rate after the increase is $H + (m-1)\alpha H = (1 - \alpha + m\alpha)H$. So,

$$\begin{aligned}\mathbf{P}(\text{next block is from } T_{m\alpha}) &= P\left(T_{m\alpha} = \min\{T_{m\alpha}, T_{1-\alpha}\}\right)\\ &= \frac{\lambda_{m\alpha}}{\lambda_{m\alpha} + \lambda_{1-\alpha}}\\ &= \frac{m\alpha/\eta}{m\alpha/\eta + (1-\alpha)/\eta}\\ &= \frac{m\alpha}{m\alpha + 1 - \alpha}\\ &= \alpha\left(\frac{m}{m\alpha + 1 - \alpha}\right).\end{aligned}$$

□

Corollary. *If one miner has a really tiny percent of the network's total hash rate, then multiplying their hash rate by m approximately multiplies their probability of finding the next block by m.*

Proof.

$$\lim_{\alpha\to 0} \mathbf{P}(\text{next block is from } T_{m\alpha}) = \lim_{\alpha\to 0} \frac{m}{m\alpha + 1 - \alpha} = m.$$

□

That way, it is not exactly correct to say that when one doubles their hash rate, their probability will double as well. It is only true for small miners.

3.5 ORPHAN BLOCKS

An orphan block would be created if a new block is found during the propagation time of a new block. Let α be the percentage of the total hash rate of the node which is outdated, and Δt the propagation time in seconds. Thus, $\mathbf{P}(\text{new orphan}) = \mathbf{P}(T < \Delta t) = 1 - e^{-\frac{\alpha \Delta t}{\eta}}$.

Bitcoin peer-to-peer network is a gossip network, where miners are semi-randomly connected to each other, and each miner sends all information it receives to all its peers. According to Decker and Wattenhofer [8], the average time for a new block propagate over the network is 12.6 seconds, while the 95% percentile is 40 seconds, which indicates a long-tail distribution. BitcoinStats [4] has measured the propagation time between 2013 and 2017. During 2017, the worst daily 90% percentile was 21 seconds. Notice that both results may not be contradictory because Bitcoin network is continuously evolving.

For instance, if a node has 10% of the total hash rate and it takes 30 seconds to receive the update, then $\mathbf{P}(\text{new orphan}) = 1 - e^{-\frac{0.1 \cdot 30}{600}} = 0.004987$, which is almost 0.5%. I would say that a node with 10% of the total hash rate would be well connected and it would take less time to receive the update, so, the probability would be even smaller than 0.5%.

Another important factor is that, as Bitcoin is open-source, miners are free to change the gossip algorithm, which leads to the network incentives. See Babaioff et al. [1] for an analysis of the incentives to miners forward new blocks and transactions in the network.

For further information about gossip algorithms, see Shah et al. [18].

3.6 ANALYSIS OF NETWORK'S HASH RATE CHANGE

The difficulty, given by the number A, is adjusted every 2016 blocks. As, $\mathbf{P}(13 \text{ days} < Y_{2016} < 15 \text{ days}) = \mathbf{P}(13 \cdot 24 \cdot 3600 < Y_{2016} < 14 \cdot 24 \cdot 3600) = 0.9986$, it is expected that the total time to find 2016 blocks will be between 13 and 15 days, assuming that the network's hash rate remains constant. If it takes less than the expected time, it means that the network's total hash rate has increased. While if it takes more than the expected time, it means that the network's total hash rate has decreased. So, let's analyze what happens when the network's hash rate changes significantly.

Let $H \cdot u(t)$ be the network's total hash rate over time. So, the number of hashes calculated in t seconds is $H \int_0^t u(t)dt$. Hence, $\mathbf{P}(T \leqslant t) = \mathbf{P}(X \leqslant H \int_0^t u(t)dt)$. When $H \to +\infty$, $\mathbf{P}(T \leqslant t) = 1 - e^{-\frac{1}{\eta} \int_0^t u(t)dt}$, and the pdf of T is $\frac{u(t)}{\eta} \cdot e^{-\frac{1}{\eta} \int_0^t u(t)dt}$.

Let's say that the network's total hash rate has suddenly multiplied by α. So, $u(t) = \alpha$, $\int_0^t u(t)dt = \alpha t$, and T also follows an exponential distribution, but with $\lambda = \frac{\alpha}{\eta}$. Thus, $Y_n^\alpha = \sum_{i=1}^n T_i^\alpha \sim \text{Erlang}(n, \frac{\alpha}{\eta})$.

Thus, $\mathbf{E}[Y_n^\alpha] = \frac{\mathbf{E}[Y_n]}{\alpha}$, i.e., the average total time required to find n blocks will be divided by α, while $\mathbf{V}[Y_n^\alpha] = \frac{\mathbf{V}[Y_n]}{\alpha^2}$ and the variance will be divided by α^2. Hence, on one hand, when the network's hash rate increases ($\alpha > 1$), the 2016 blocks will be found earlier. On the other hand, when the network's hash rate decreases ($\alpha < 1$), the 2016 blocks will be found later.

For example, if the network's total hash rate suddenly doubles ($\alpha = 2$), then $\mathbf{P}(6.5 \text{ days} < Y_{2016} < 7.5 \text{ days}) = 0.9986$, and the time required to find 2016 blocks halved. On the other side, if the network's total hash rate suddenly halves ($\alpha = 0.5$), then $\mathbf{P}(27 \text{ days} < Y_{2016} < 29 \text{ days}) = 0.9469$, and the time required to find 2016 blocks doubled. It is an important conclusion, since it shows that even if half of the network stops mining, it will only double the time to the next difficulty adjustment, i.e., the time between blocks will be 20 minutes for, at most, the next 29 days, at which point the adjustment will occur and everything will be back to the normal 10 minutes between blocks.

3.6.1 *Hash rate smoothly changing*

Let $u(t) = \frac{1+abx}{1+bx}$. It is an useful function because $u(0) = 1$ and $\lim_{t\to\infty} u(t) = a$. The bigger the b, the faster $u(t) \to a$. For example, if $a = 2$, it means H would be smoothly doubling. If $a = 0.5$, it means H would be smoothly halving.

It is easy to integrate $u(t)$ because $\frac{1+abx}{1+bx} = \frac{1-a}{1+bx} + a \Rightarrow \int_0^t u(x)dx = at + \frac{1-a}{b}\log(1+bt)$. So,

$$F_T(t) = 1 - (1+bt)^{\frac{\lambda(a-1)}{b}} e^{-\lambda a t}.$$

$$f_T(t) = \lambda\left(\frac{1+abt}{1+bt}\right)(1+bt)^{\frac{\lambda(a-1)}{b}} e^{-\lambda a t}.$$

Assuming that $n = \frac{\lambda(a-1)}{b}$ is integer, we have:

$$F_T(t) = 1 - (1+bt)^n e^{-\lambda a t}$$

Thus,

$$\begin{aligned}
\mathcal{L}\{F_T(t)\} &= \mathcal{L}\{1 - (1+bt)^n e^{-\lambda a t}\} \\
&= \mathcal{L}\{1\} - \mathcal{L}\{(1+bt)^n e^{-\lambda a t}\} \qquad (\mathcal{L} \text{ is a linear operator}) \\
&= \frac{1}{s} - \mathcal{L}\{(1+bt)^n e^{-\lambda a t}\} \\
&= \frac{1}{s} - \sum_{k=0}^{n} \binom{n}{k} b^k \mathcal{L}\{t^k e^{-\lambda a t}\} \\
&= \frac{1}{s} - \sum_{k=0}^{n} \binom{n}{k} b^k \frac{k!}{(s+\lambda a)^{k+1}}
\end{aligned}$$

Hence, as $\mathcal{L}\{f_T(t)\} = s\mathcal{L}\{F_T(t)\}$,

$$\mathcal{L}\{f_T(t)\} = 1 - \sum_{k=0}^{n} \binom{n}{k} \frac{s b^k k!}{(s+\lambda a)^{k+1}}$$

Then,

$$\begin{aligned}
\frac{d}{ds}\mathcal{L}\{f_T(t)\} &= -\sum_{k=0}^{n} \binom{n}{k} b^k k! \frac{d}{ds} \frac{s}{(s+\lambda a)^{k+1}} \\
&= -\sum_{k=0}^{n} \binom{n}{k} b^k k! \left[\frac{1}{(s+a\lambda)^{k+1}} - \frac{s(k+1)}{(s+a\lambda)^{k+1}} \right] \\
\frac{d}{ds}\mathcal{L}\{f_T(t)\}|_{s=0} &= -\sum_{k=0}^{n} \binom{n}{k} b^k k! \frac{1}{(\lambda a)^{k+1}} \\
&= -\frac{1}{a\lambda} \sum_{k=0}^{n} \binom{n}{k} k! \left(\frac{b}{\lambda a}\right)^k \\
&= -\frac{1}{a\lambda} \sum_{k=0}^{n} \frac{n!}{(n-k)!} \left(\frac{b}{\lambda a}\right)^k \\
&= -\frac{1}{a\lambda} \left[n! \sum_{k=0}^{n} \frac{1}{(n-k)!} \left(\frac{b}{\lambda a}\right)^k \right] \\
&= -\frac{1}{a\lambda} \left[n! \sum_{k=0}^{n} \frac{1}{k!} \left(\frac{b}{\lambda a}\right)^{n-k} \right] \qquad (k \to n-k) \\
&= -\frac{1}{a\lambda} \left[n! \left(\frac{b}{\lambda a}\right)^n \sum_{k=0}^{n} \frac{1}{k!} \left(\frac{b}{\lambda a}\right)^{-k} \right] \\
&= -\frac{1}{a\lambda} \left[n! \left(\frac{b}{\lambda a}\right)^n \sum_{k=0}^{n} \frac{1}{k!} \left(\frac{\lambda a}{b}\right)^k \right]
\end{aligned}$$

Finally, as $\mathbf{E}[T] = -\mathcal{L}\{f_T(t)\}|_{s=0}$,

$$\mathbf{E}[T] = \frac{1}{\lambda a} \left[n! \left(\frac{b}{\lambda a}\right)^n \sum_{k=0}^{n} \frac{1}{k!} \left(\frac{\lambda a}{b}\right)^k \right], \text{ where } n = \frac{\lambda(a-1)}{b}$$

Let's check this equation for already known scenarios. When $a = 1$, then $n = 0$ and $\mathbf{E}[T] = 1/\lambda$. When $b \to +\infty$, it reduces to the case in which the hash rate is multiplied by a, which we have already studied. In fact, $n \to 0$, $u(t) \to a$, and $\mathbf{E}[T] = \frac{1}{\lambda a}$.

Theorem 5.

$$a > 1 \textit{ and } x > M \Rightarrow \left| \frac{1+abx}{1+bx} - a \right| < \frac{a-1}{1+bM}$$

Proof. $x > M \Rightarrow \frac{1}{1+bx} < \frac{1}{1+bM}$. As $1 - a < 0$, $\frac{1-a}{1+bx} > \frac{1-a}{1+bM}$. Thus, $\frac{1-a}{1+bM} < \frac{1-a}{1+bx} + a - a = \frac{1+abx}{1+bx} - a < 0 < \frac{a-1}{1+bM}$. Hence, $-\frac{a-1}{1+bM} < \frac{1+abx}{1+bx} - a < \frac{a-1}{1+bM}$. □

For instance, if we would like to know the impact of smoothly double the hash rate in the next week, then the parameters would be $\lambda = 1/600$, $a = 2$, $M = 1\,\text{week} = 3600 \cdot 24 \cdot 7 = 604,800$, b can be calculated using $\epsilon = \frac{a-1}{1+bM} < 0.01 \Leftarrow b > 0.000163690 \Leftarrow n < 10.1818$. So, for $n = 10$, then $b = 0.000166666$ and $\epsilon = 0.009823 < 0.01$, as expected. Finally, $\mathbf{E}[T] = 557.65$. In other words, during the next week, the average time between blocks will be 9 minutes and 17 seconds, instead of the normal 10 minutes. If the hash rate had suddenly doubled, the average time between blocks would be 5 minutes.

3.6.2 *Piecewise linear model of hash rate change*

Let's analyze what would happen if the network's hash rate is growing linearly with angular coefficient a^2, i.e., $u(a, b, t) = a^2 t + b$. Thus, $\mathbf{P}(T \leqslant t) = 1 - e^{-\frac{bt + a^2t^2/2}{\eta}}$.

It is well known that $\mathbf{E}(T) = \int_0^\infty 1 - \mathbf{P}(T \leqslant t)dt$. Thus, replacing $y = \frac{a^2t+b}{a\sqrt{2\eta}}$, and using the fact that $\int_0^\infty e^{-x^2}dx = \frac{\sqrt{\pi}}{2}\operatorname{erf}(x)$, we have:

$$\begin{aligned}
\mathbf{E}(T)|_{t_1}^{t_2} &= \int_{t_1}^{t_2} \exp\left(-\frac{bt + a^2t^2/2}{\eta}\right) dt \\
&= \frac{\sqrt{2\eta}}{a} \exp\left(\frac{b^2}{2a^2\eta}\right) \int_{y_1}^{y_2} \exp(-y^2)dy \\
&= \frac{\sqrt{2\eta}}{a} \exp\left(\frac{b^2}{2a^2\eta}\right) \frac{\sqrt{\pi}}{2}[\operatorname{erf}(y_1) - \operatorname{erf}(y_2)] \\
&= \frac{\sqrt{2\pi\eta}}{2a} \exp\left(\frac{b^2}{2a^2\eta}\right) [\operatorname{erf}(y_2) - \operatorname{erf}(y_1)] \qquad (1)
\end{aligned}$$

Where $y_1 = \frac{a^2t_1+b}{a\sqrt{2\eta}}$ and $y_2 = \frac{a^2t_2+b}{a\sqrt{2\eta}}$.

Thus, $\mathbf{E}(T) = \mathbf{E}(T)|_0^\infty$. When $t_1 = 0 \Rightarrow y_1 = \frac{b^2}{2\sqrt{2\eta}}$ and $t_2 \to \infty \Rightarrow y_2 \to \infty \Rightarrow \operatorname{erf}(y_2) = 1$, then:

$$\mathbf{E}(T) = \frac{\sqrt{2\pi\eta}}{2a} \exp\left(\frac{b^2}{2a^2\eta}\right) \left[1 - \operatorname{erf}\left(\frac{1}{a\sqrt{2\eta}}\right)\right]$$

The function $\mathbf{E}(T)|_{t_1}^{t_2}$ may be used to a piecewise linear analysis of any hash rate change. Let's analyze the hash doubling in one week. Then, $u(1\text{week}) = u(604800) = 2$, thus $a^2 = \frac{1}{604800} \Rightarrow a = \frac{1}{120\sqrt{42}}$. Let's sample the interval $[0, 1\text{week}]$ every hour, i.e., $(t_0, t_1, t_2, \ldots, t_168)$, where $t_i = i \cdot 1\text{hour} = i \cdot 3600$.

For each point t_i, let $g_i(t) = [H(t - t_i) - H(t - 604800)]u(t) + 2H(t - 604800)$, then $\mathbf{E}(T) = \mathbf{E}(T)|_{t_i}^{t_{i+1}} + 1 - e^{\frac{t}{\eta}}$. The result is presented in Figure .

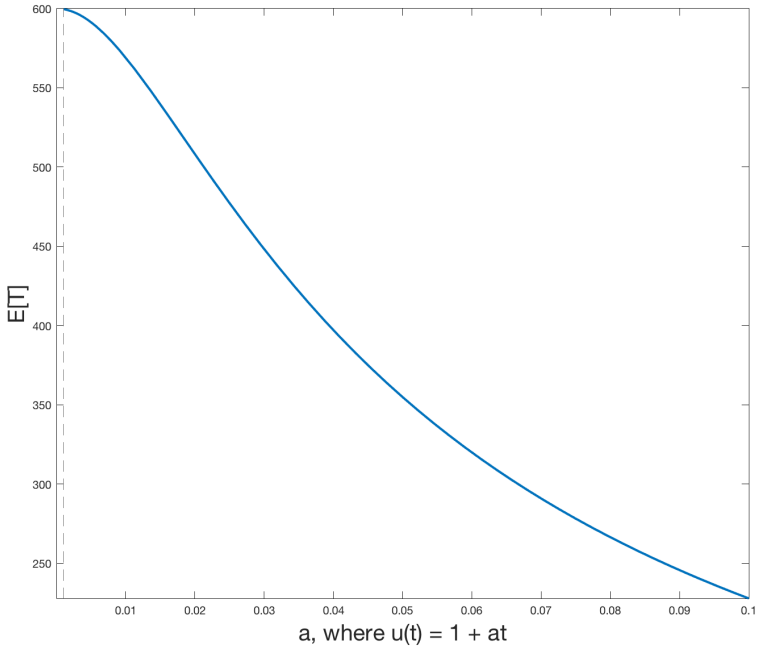

Figure 2: $\mathbf{E}[T]$ when H increases linearly with $u(t) = 1 + at$.

3.7 ATTACK IN THE BITCOIN NETWORK

Karame et al. [12]

There are many possible ways to attack the Bitcoin (REF). In this section, we are interested in a particular attack: the double spending attack.

In the double spending attack, the attacker's send some funds to the victim, let's say a merchant. They wait for k confirmations of the transaction, and the victim delivers the good or the service to the attacker. Then, the attacker mine enough blocks with a conflicting transaction, double spending the funds which was sent to the victim. If the attacker is successful, the original transaction will be *erased* and the victim will be left with no funds at all. In order to be successful, the attacker must propagate more blocks than the network in the same period, propagating a chain longer than the main chain. Hence, we would like to understand what the odds are that the attacker will be successful. This attack was originally discussed by Nakamoto [15].

In order to maximize their odds, the attacker must start to mine the new blocks as soon as they send the funds to the victim. In this moment, it starts to mine in the head of the blockchain, just like the rest of the network. So, in the beginning, the attacker and the network are in exactly the same point.

Let βH be the hash rate of the attackers, and γH be the network's hash rate without the attackers. Thus, when $H \to +\infty$, we already

know that $T_{attackers}$ and $T_{network}$ follow exponential distributions with parameters $\lambda_{attacker} = \frac{\beta}{\eta}$ and $\lambda_{network} = \frac{\gamma}{\eta}$, respectively.

As [15] has done, we will also model the attack using the Gambler's Ruin. In this game, a gambler wins \$1 at each round, with probability p, and loses \$1, with probability $1-p$. The rounds are independent. The gambler starts with \$$k$ plays continuously until he either accumulates a target amount of \$$m$, or loses all his money. Let $\rho = \frac{1-p}{p}$, then the probability of losing his fortune is:

$$\mathbf{P}(\text{losing his fortune}) = \begin{cases} \frac{\rho^k - \rho^m}{1-\rho^m}, \text{ if } \rho \neq 1, \\ \frac{m-k}{m}, \text{ if } \rho = 1. \end{cases}$$

When $m \to +\infty$,

$$\mathbf{P}(\text{losing his fortune}) = \begin{cases} \rho^k, \text{ if } \rho < 1, \\ 1, \text{ if } \rho \geqslant 1. \end{cases}$$

The gambler winning \$1 is the same as the network finding a new block, the gambler losing \$1 is the same as the attacker finding a new block. The initial \$$k$ is the same as the number of blocks the attacker is behind the network. Thus, the gambler loses his fortune is the same as the attacker successfully finds k or more blocks than the network, i.e., losing his fortune means that the attack was successful.

In our case, $p = \frac{\lambda_{network}}{\lambda_{network}+\lambda_{attacker}} = \frac{\gamma}{\beta+\gamma}$, thus $\rho = \frac{\beta}{\gamma}$. Hence, $\rho < 1 \Leftrightarrow \beta < \gamma$.

Suppose that the attacker is mining with the network. Suddently, he stops mining with the network and starts attacking, i.e., starts to mine in another chain. In this scenario, since the attacker's hash rate is not mining with the network anymore, $\gamma = 1 - \beta$. Thus, $\beta < \gamma \Rightarrow \beta < 0.5 \Leftrightarrow \rho < 1$. Here comes the conclusion that, if the attacker has 50% or more of the network's hash rate, then his attack will be certainly successful. We got exactly the same equations and conclusions as [15].

But this scenario seems not to be the optimal attack, because the attacker has waited k confirmations before starting the attack. A better approach would be to start attacking just after propagating the transaction. In this case, our previous model is not good, because even if the attacker have found more blocks than the network, he cannot propagate those blocks before the network has found k confirmations. So, we have to model the probabilities before the network has found the k block. Then, if the attacker has more blocks than the network, he has successfully attacked. Otherwise, we return to the previous model, in which the attacker must still find more blocks.

Theorem 6. *Assuming that the attacker starts the attack just after publishing the transaction, the probability of the attacker has already found*

exactly s *blocks while it waits the network to find* k *blocks is* $\mathbf{P}(S = s) = \binom{k+s-1}{s}(1-p)^s p^k$.

Proof. The attacker must find exactly s blocks while the network must find exactly k blocks. It is as they would be walking the grid from the point $(0,0)$ to (s,k), where it is only allowed to go up or right, like in Figure 3. When the attacker finds a block, it would be a movement to the right. When the network finds a block, it would be an upward movement. No matter the order which the blocks are found, all the paths occur with probability $(1-p)^s p^k$.

The walking ends when $(\cdot, k)$ is reached, i.e., when the network finds k blocks, regardless of how many blocks the attacker has found – i.e., it is not allowed to walk above the line $(\cdot, k)$. Thus, the number of paths between $(0,0)$ and (s,k) moving only upward or to the right, without going into the line $(\cdot, k)$ is exactly the number of paths between $(0,0)$ and $(s, k-1)$, which is equal to the number of permutations of the sequence $(u, u, \ldots, u, r, r, \ldots, r)$ in which there are s movements to the right (r) and $k-1$ upward movements (u). This number of permutations is $\frac{(k-1+s)!}{s!(k-1)!} = \binom{(k-1)+(s)}{s}$ because there are s repetitions of the element r and $k-1$ repetitions of the element u.

Finally, the probability is $\binom{k+s-1}{s}(1-p)^s p^k$.

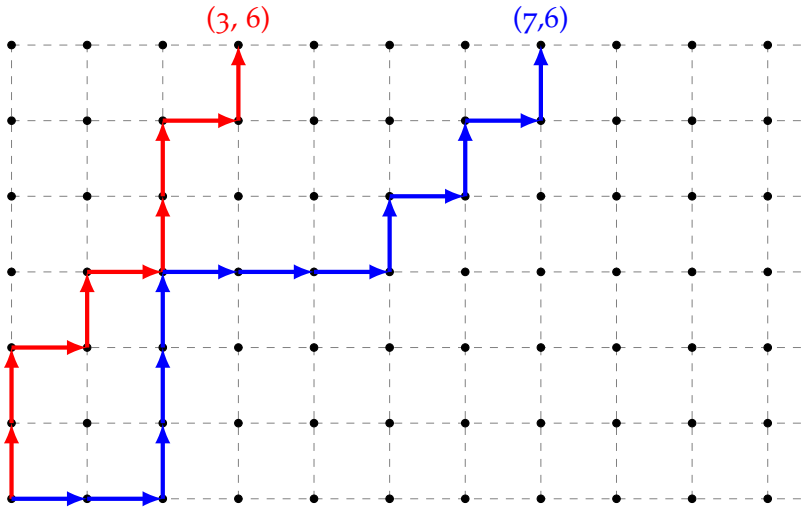

Figure 3: Both the attacker and the network are mining. Each step up is a new block found by the network with probability p. Each step right is a new block found by the attacker with probability $1-p$. It ends when the network finds k blocks — in this example, $k = 6$. The red path has probability $p^6(1-p)^3$, while the blue path has probability $p^6(1-p)^7$. Notice that the blue path is a successfull attack, because the attacker has found more blocks than the network. In the red path, the attacker still have to catch up 3 blocks to have a successful attack, which happens with probability ρ^3, if $p < 0.5$.

□

Assuming that the attacker starts mining just after publishing the victim's transaction, the probability of the attacker will have found more than k blocks while it waits the network to find k blocks is $\mathbf{P}(S \geqslant k) = \sum_{s=k}^{\infty} \binom{k+s-1}{s}(1-p)^s p^k$.

Theorem 7.

$$\mathbf{P}(S \geqslant k) = 1 - \sum_{s=0}^{k-1} \binom{k+s-1}{s}(1-p)^s p^k.$$

Proof. Let's use the following identity:

$$\frac{1}{(1-z)^{a+1}} = \sum_{i=0}^{\infty} \binom{i+a}{i} z^i, \text{ for } |z| < 1$$

Thus, replacing $z = 1-p$, $i = s$, and $a = k-1$, we have:

$$\frac{1}{p^k} = \sum_{s=0}^{\infty} \binom{s+k-1}{s}(1-p)^s$$

$$1 = \sum_{s=0}^{\infty} \binom{s+k-1}{s}(1-p)^s p^k.$$

Now, just split $\sum_{s=0}^{\infty} = \sum_{s=0}^{k-1} + \sum_{s=k}^{\infty}$ and it is done. □

Using this last theorem, we moved from an infinity sum to a finity sum.

Theorem 8. *Let* $p = \frac{\gamma}{\beta+\gamma}$.

$$\mathbf{P}(\textit{successful attack}) = \begin{cases} 1 - \sum_{s=0}^{k-1} \binom{k+s-1}{s} \left((1-p)^s p^k - (1-p)^k p^s\right), p \geqslant 0.5 \\ 1, p < 0.5. \end{cases}$$

Proof.

$$\mathbf{P}(\text{successful attack}) = \mathbf{P}(S \geqslant k) + \sum_{i=0}^{k-1} \mathbf{P}(s = i)\rho^{k-i}$$

□

For $k = 6$, $p = 0.9$, $\mathbf{P}(\text{successful attack}) = 0.0005914121600000266$.
For $k = 6$, $p = 0.7$, $\mathbf{P}(\text{successful attack}) = 0.15644958192000014$.

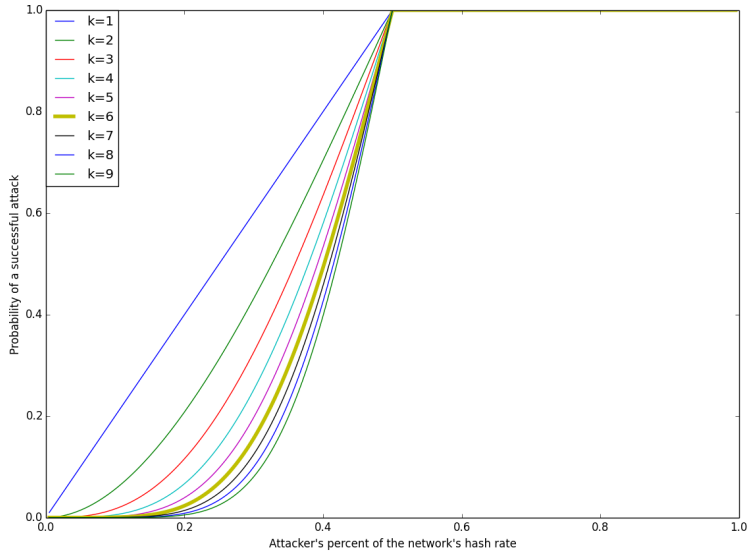

Figure 4: Probability of a successful attack according to the network's hash rate of the attacker (β).

3.8 CONFIRMATION TIME AND NETWORK CAPACITY

Let's say that when a new transaction is propagated it is enqueued in the unconfirmed transaction queue. Then, when a new block is found, some of these transactions in the queue are confirmed. We are interested in some measures of the queue, like the expected time to confirm a transaction and the queue's length.

Let's assume that all transactions have exactly the same size S and pay exactly the same fee. If the Bitcoin block's maximum size is M, there would be room for $s = \lfloor M/S \rfloor$ transactions in each block. Using the results from Bailey [2], we have found that $\pi_n = \frac{z_s - 1}{z_s^{n+1}}$ is the probability of having n unconfirmed transactions in the pool subjected to $s > m$, where $m = \frac{\lambda_{TX}}{\lambda_{blocks}}$ and z_s is the single root of the polynomial $z^s(1 + m(1 - z)) - 1$ with $|z_s| > 1$. The average size of the unconfirmed transaction pool is $\mathbf{E}(\pi) = \frac{1}{z_s - 1}$. Thus, $s > m \Rightarrow \lambda_{TX} < s\lambda_{blocks}$. The probabilities form a simple geometric series, and so they are exponentially decreasing. We may interpret it as a stable system, i.e., the unconfirmed transactions pool size is finite. In the Bitcoin's network, $\lambda_{blocks} = 1/\eta$ and the average number of transactions per block is $s = 2,250$, so, this solution is valid when $\lambda_{TX} < 2,250/600 = 3.75$. Hence, 3.75 is the maximum number of new transactions per second that the Bitcoin's network may handle. When $\lambda_{TX} > 3.75$, the unconfirmed transaction pool starts to grow indefinitely.

The average waiting time of a transaction to be confirmed, when $m < s$, is $\mathbf{E}(w) = \frac{1}{\lambda_{TX}(z_s - 1)}$.

When $m \ll s$, $z_s \to 1 + 1/m$. So, the average number of unconfirmed transactions is $\mathbf{E}(\pi) \to m$ and the average waiting time $\mathbf{E}(w) \to \frac{1}{\lambda_{\text{blocks}}} = \eta = 600$ seconds. In the Bitcoin's network, $s \gg m \Rightarrow \lambda_{TX} \ll 3.75$.

When $\lambda_{TX} \to s$, $z_s \to 1$. So, $\mathbf{E}(\pi) \to +\infty$.

Finally, we conclude that the Bitcoin's network capacity is $\lambda_{\text{blocks}} s = s/\eta = s/600$ transactions per second, where s is the average number of transactions per block.

3.9 FORK ANALYSIS

When a disagreement between miners' rules happens, that is referred as either a hard-fork or a soft-fork. It is said to be a soft-fork when the rules are backward compatible, and a hard-fork when the rules are not backward compatible. In general, a hard-fork relaxes the constraints, while the latter hardens them.

Suppose that the miners' are split in two groups, G_1 and G_2, with different rules. Let's say H_1 and H_2 are their hash rate, respectively. We have two different scenarios to analyze: (i) when neither of them accepts other's blocks; and (ii) when G_2 accepts G_1's blocks, but not the other way around.

Scenario (i) is easy to analyze, because the network would just split and, after a while, both difficulties will be adjusted. Then, they will be just like two different Bitcoin networks.

Scenario (ii) is more trick. As G_2 accepts G_1 blocks, their hashrate will matter. If $H_1 > H_2$, then G_2 will frequently skip their blockchain, because G_1's blockchain will be longer most of the time. If $H_1 < H_2$, then G_2 may have its own blockchain after a while — a true fork. But what would happen if H_1 keeps increasing and eventually gets larger than H_2? It if happens for sufficient time, the whole G_2's blockchain may be discarded in order to move to G_1's blockchain when G_1's gets longer.

Theorem 9. *When $H_1 > H_2$, the*

4

DAG MODEL

DAG model proposes a whole different approach to confirmations. It proposes that there is no need for a block to confirm transactions, as transactions can confirm themselves. Here, each transaction has its own proof-of-work, named weight, and they must confirm two other, previous, transactions. Hence, each transaction has an accumulated weight, which is the accumulated proof-of-work that has confirmed it so far. In this sense, instead of a chain of blocks, the transactions and their confirmations form a directed and acyclic graph, as in Fig. 5.

Like the Blockchain, the DAG model is another technology to store immutable data and may be the underlying technology to different applications, such as cryptocurrencies, digital contracts (Ethereum-like), digital notaries, and so forth. The main question which this work proposes to answer is: Is it reliable to use DAG model? In which conditions?

The transactions may be either confirmed or unconfirmed. The confirmed transactions have been already confirmed by at least one more transaction. It does not mean they are already irreversible and protected against a double spend attack — it just means at least one transaction has done some work to confirm it. The unconfirmed transactions are called tips and they are eager to be confirmed. Usually a new transaction selects two tips to confirm, but this rule may not be followed.

Transactions may have any format, including Bitcoin-like transactions—with inputs, outputs, and scripts. But it also may be completely different, just like in the cryptocurrency Iota [17]. This work will not discuss details of the transactions' format, because we understand that it does not affect exactly the network scalability and security.

The accumulated weight of a transaction A is the sum of all weights of the transactions which confirm A, including A itself, i.e., $w_A + \sum_{A \rightsquigarrow P} w_P$. For example, in Fig. 5, the accumulated weight of transaction 3 is the sum of the weights of the transactions 5, 6, 7, and 8. The accumulated weight may be interpreted as how hard it is to rollback a transaction. It is analogous to the number of confirmations of a block in Bitcoin.

The score of a transaction A is the sum of all weights of the transactions which are being confirmed by A, including A itself, i.e., $w_A + \sum_{P \rightsquigarrow A} w_P$. For example, in Fig. 5, the score of transaction 3 is the sum of the weights of the transactions 1 and 2. It is a measure of

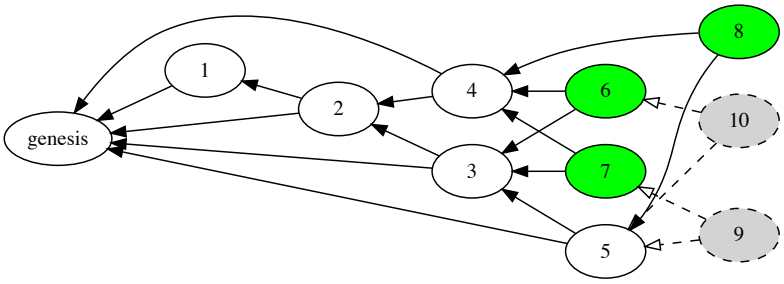

Figure 5: White nodes represent transactions that have been confirmed at least once. Green circles represent unconfirmed transactions (tips). Gray and dashed nodes are the transactions currently solving the proof-of-work in order to be propagated.

how much proof-of-work has been done before confirming this transaction. It may indicate whether the confirmed transactions have received enough attention (and hashpower) or whether it may have received tangential attention.

The height of a transaction A is the length of the longest path from transaction A to the genesis transaction. For example, in Fig. 5, the height of transaction 5 is four ($5 \rightarrow 3 \rightarrow 2 \rightarrow 1 \rightarrow$ genesis). It may be interpreted as the "'age" of the transaction. The lower the height, the older the transaction.

The depth of a transaction A is the length of the longest path in the inverted graph from transaction A to any unconfirmed transaction (tip). For example, in Fig. 5, the depth of transaction 2 is three ($2 \rightarrow 3 \rightarrow 5 \rightarrow 8$). It is the opposite of the height. It may be interpreted as the youth of the transaction. The lower the depth, the younger the transaction. When a new transaction is confirming two transactions with high depth, it is referred as lazy transaction.

An important factor of DAG model is how the new transactions choose which transactions they will confirm. There are several possible approaches, such as randomly selecting two of the unconfirmed transactions (tips). In Fig 5, the reader may have noticed that transaction 8 will confirm transaction 4, which has already been confirmed by transaction 7. It may be on purpose, or maybe transaction 4 was unconfirmed when it was chosen, but it got confirmed during the calculation of the proof-of-work or the network propagation of the transaction. The selection algorithm seems to be important to protect the network against double spend attacks.

The higher the volume of new transactions, the more unconfirmed transactions will appear. In Fig. 6, the reader can notice that the number of new transactions was increased for a while, and then

decreased back to the original value. The DAG has behaved well when exposed to a high load scenario, since it reduced the number of tips to only three. It is like a moving swarm which gets wider when the number of new transactions increases and gets thinner when the number of new transactions decreases.

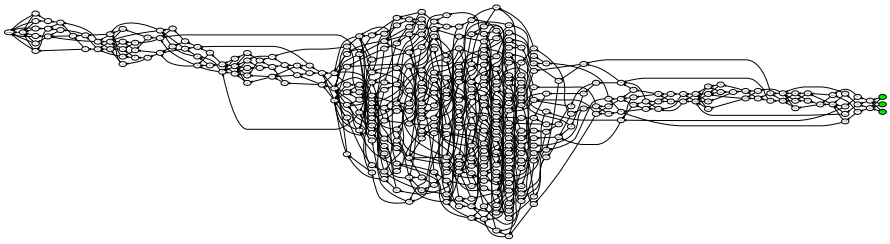

Figure 6: Suddenly the number of transactions per second increases and the width of the swarm grows. After a while, the number of transactions per second decreases and the width of the swarm shrinks.

Conflicting transactions may happen when two or more transactions try to spend the "same money" — or, in the Bitcoin's transaction format, try to spend the same output. In this case, the network must choose which of the transactions will be accepted and the other one will be invalidated, even when both have already been confirmed. In fact, when one transaction is invalidated, the whole sub-DAG which confirms it is also invalidated. In this case, it may happen to reverse some transactions.

Intuitively, when there is a conflict, the network should accept the transaction which has greater accumulated weight, invalidating the others (see Fig. 7). But it may be not enough to prevent a nuclear submarine attack.

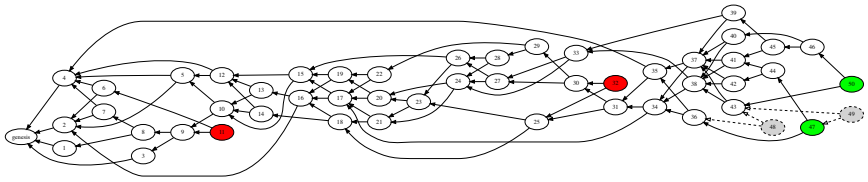

Figure 7: The red nodes are transactions which had some conflict with previous transaction and were invalidated by the network. Notice that none of them have been confirmed.

4.1 NUCLEAR SUBMARINE ATTACK

The nuclear submarine attack, also known as the parasite chain attack, is when the attacker generates a separate DAG (or a side DAG), with many transactions and a lot of proof-of-work. This side DAG is off the network, i.e., its transactions have not been propagated. Then, at a convenient moment, the attacker suddenly

propagates these transactions. The whole network needs to decide how to handle these transactions.

If the transactions have no conflict with any transaction of the main DAG, i.e., there is no transaction spending the "same money", then it seems easy to handle the transactions. But, as it is an attack, probably there will be some conflicts, and it is not easy to choose which transaction should be invalidated. As the attacker has been generating a separate DAG, the conflicting transaction may have an accumulated weight similar or greater than the transaction in the main DAG. Thus, using only the accumulated weight may not be enough to prevent this attack.

For example, the attacker may generate a transaction which transfers all their funds to another address. Then, they start to generate many new transactions which confirms themselves and even confirms some of the transactions in the main DAG. But none of these transactions are propagated to the network. Then, the attacker buys something in the real world, pays with cryptocurrency, and wait until the payment gets the accumulated weight demanded by the merchant. Finally, the attacker suddenly propagates all the transactions to the network in a small window of time. This may cause the network to accept the attacker's original transaction instead of the one used to pay the merchant. Hence, the merchant transaction is invalidated, and the double spend attack has succeeded.

4.2 PROPOSAL: QUESTIONS TO BE EXPLORED

Given these preliminaries, these are some questions with which we will be concerned.

In this paper, we will analyze the network scaling and security. How a DAG network scales, simulating different loads and measuring the bandwidth, computational effort, and storage space necessary to handle all the transactions in time. What is the minimum transaction's aggregated weight which it may be considered unlikely to be reversed?

We are interested in how the rate of new transactions affects how long it takes to a new transaction to be confirmed for the first time. We had already run some simulations under normal load (Fig. 8) and high load (Fig. 9).

Besides the time to the first confirmation, it is also important to measure how long it takes to a new transaction reach a specific accumulated weight. It is useful for exchanges and merchants to set their minimum requirements. Bitcoin's exchanges and merchants usually requires a minimum of 6 confirmations blocks.

As the DAG network is distributed, its users may use different parameters — they cannot violate the network's rules, but they may

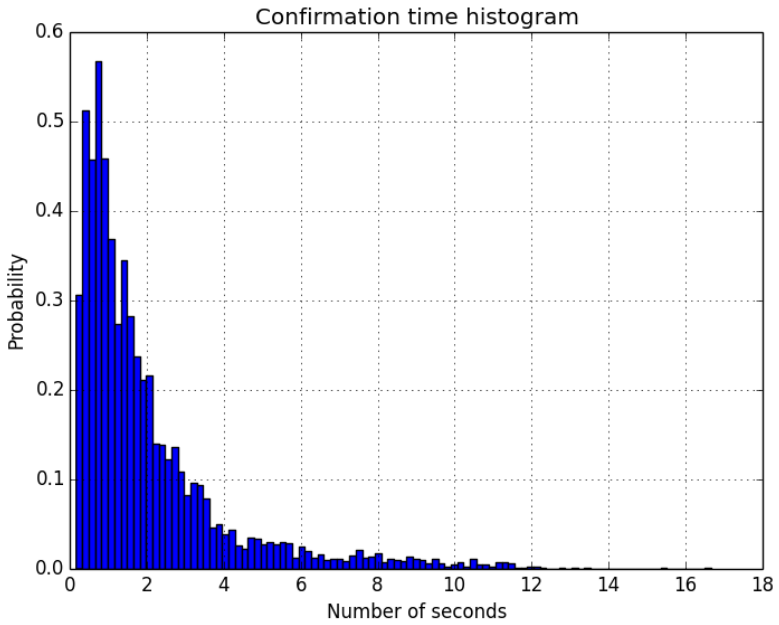

Figure 8: Histogram of how long has been a transaction waiting until its first confirmation. It was a simulation of 15 minutes with new transactions rate changing between 1 and 15 tx/s.

use different tip selection algorithms, for instance. We would like to explore how different users' parameters may affect the network, how much is necessary to cause either a hard-fork or a soft-fork.

[17] suggests that constraining transactions' weight in a range would both prevent spam, because it would be necessary a minimum work to propagate a new transaction, and attacks, because an attacker would not be allowed to propagate a transaction with very high weight. We agree with the spam argument, and would like how a minimum weight would affect mobile devices's new transactions. We partially agree with the upper bound, because an attacker would be able to generate many transaction with lower weight. We will analyze if constraining transactions' weight would be effective against nuclear submarine attacks.

We have another suggestion to prevent nuclear submarine attacks: set a maximum depth for the transactions confirmed by a new transaction. So, new transactions would have to confirming newer transactions instead of old transactions, and a nuclear submarine attack would be controlled because the attacker would be not be able to create a large separate DAG. But the maximum depth rule would possibly also affect transaction created by low (hash)power devices, because it would take a longer time to solve the proof-of-work and, if the confirmations are going fast, the transaction would possibly be invalidated. This raises another important question: what would be an optimal maximum depth allowed?

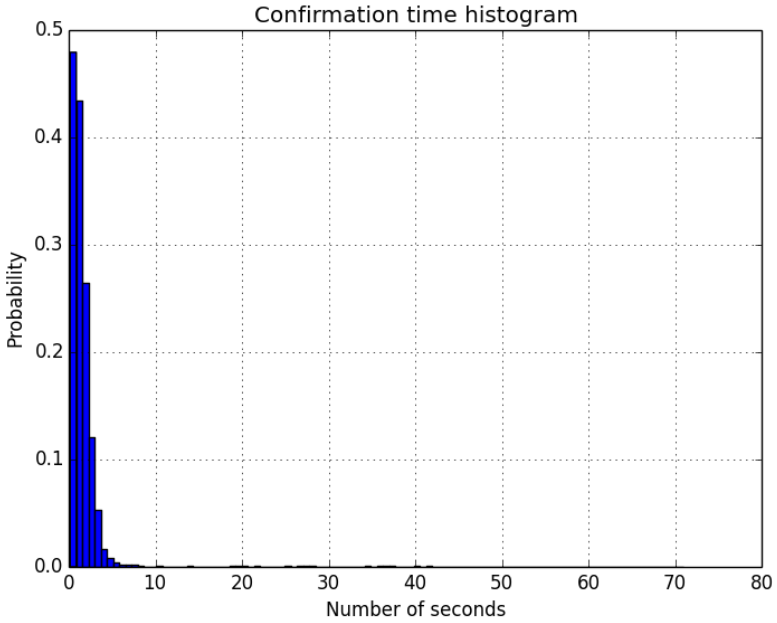

Figure 9: Histogram of how long has been a transaction waiting until its first confirmation. It was a simulation of 5 minutes with new transactions rate of 50 tx/s, i.e., very high load.

If we would like to include fee transactions, how would it be distributed between confirmations? As transactions may have multiple direct confirmations, it is not obvious who would receive the transaction fee. An intuitive suggestion would be to give the fee to the first confirmation with a minimum weight, but it might have some propagation problems, because it takes a while to propagate a transaction in the network and there would be conflict.

In order to solve the fee distribution problem, we will simulate how the network would behave if we include blocks. These blocks would be like Bitcoin's, with an adjustable proof-of-work according to the network's hashpower. They would receive the fees from all confirmed transactions which had not been confirmed by any block before, and they would be able to generate new coins. These blocks would also help solving other problems.

5

METHODOLOGY

The methodology we have been using is computer simulation. Through the simulation of many scenarios of DAG model with different parameters, we will understand how the network behaves in complex scenarios, mainly when the load is suddenly increased, and when the network is under attack. We will first test our hypothesis through simulation and than prove them mathematically.

The simulator has been developed using an event-based design which is capable of running hours of simulation in just a few minutes. It creates agents who decide to make a transaction, then they select which transactions will be confirmed, then they spend some time working in the proof-of-work, and, finally, they propagate the transaction to the network. The other agents receive the transaction and may accept or deny it. The agents may use different parameters among themselves.

When a new transaction is added to the DAG, it uses a depth-first search [7] (i) to update the aggregated weight of the directly and indirectly confirmed transactions, (ii) to calculate its own score, and (iii) to generate a topological sort [7]. The topological sort is used to calculate the longest path from all other transactions to the new transaction, and this longest path is used to update the depth of the whole DAG. If the new transaction confirms only transactions which has already been confirmed, the depth update is skipped thanks to Theorem 10.

The necessary time to generate the proof-of-work is estimated by the function $k \cdot 2^{weight}$, where k is a constant related to the computing power of the agent and weight is randomly chosen.

The time between two new transactions is sampled from the exponential distribution with parameter λ. The value of λ is changed over time, causing changes in the load of the network. We consider 1 tx/s to be a low load, and 15 tx/s to be a high load. Even though we have already tested up to 100 tx/s.

The simulator will output different reports: (i) snapshots of the DAG at interesting moments, such as Fig. 5, 6, and 7; (ii) histogram of confirmation time, such as Fig. 8 and 9; and many others.

6

ANALYSIS OF DAG MODEL

In order to a further understanding of the DAG Model, we will analyze the mathematical model behind it.

The DAG Model's mining starts just like Bitcoin's, i.e., $\mathbf{P}(h(x) < A) = \frac{A}{2^{256}}$. Then, it follows that solving a transaction proof-of-work follows a geometric distribution. Let X be the number of attempts until a success, then $\mathbf{P}(X = k) = (1-p)^{k-1}p$, where $p = \frac{A}{2^{256}}$. Also, $\mathbf{P}(X \leqslant k) = 1-(1-p)^k$. The average number of attempts is $\mathbf{E}(X) = 1/p$.

Let w be the transaction's weight, i.e., $\mathbf{E}(X) = w \Rightarrow w = 1/p$. Let $T = X/H$ be the time required to solve the transaction's proof-of-work, where H is the hash rate of the transaction's owner. Then, $\mathbf{P}(T < t) = \mathbf{P}(X < tH) = 1-(1-p)^{tH} = 1-\left(1-\frac{1}{w}\right)^{tH} = 1-e^{tH\log\left(1-\frac{1}{w}\right)}$, i.e., T follows an exponential distribution with $\lambda = -H\log\left(1-\frac{1}{w}\right)$.

The number of transactions solving the proof-of-work may be modeled by a Birth and Death process, since if we have k transactions solving the proof-of-work only two things may happen: either a new transaction will be created or one the transactions will finish solving the proof-of-work. Let the time between new transactions follows an exponential distribution with μ parameter, then the probability of a new transaction emerges before any transaction finishes its proof-of-work is $q_k = \mathbf{P}(T_{\text{new tx}} = \min\{T_{\text{new tx}}, T_1, T_2, \ldots, T_k\}) = \frac{\mu}{\mu+\sum_{i=1}^{k}\lambda_i}$. Hence, the probability of any transaction finishes its proof-of-work before a new transaction emerges is $p_k = 1-q_k = \frac{\sum_{i=1}^{k}\lambda_i}{\mu+\sum_{i=1}^{k}\lambda_i}$.

Every time a new transaction emerges, it chooses two tips to confirm before solving the proof-of-work. When it finishes solving the proof-of-work, it is propagated and becomes a tip. So, two new transactions may choose the same tips to confirm. If there are t tips, a new transaction will randomly choose 2 out of these t tips, even if they have already been chosen by other new transactions — in fact, they do not know which have been chosen because these new transactions have not being propagated yet.

The following theorems will be mathematically proved. They have already been used in the implementation of the simulator.

Theorem 10. *When a new transaction confirms an already confirmed transaction, the depth of all transactions remains the same, i.e., the depth of the transactions are only changed when a new transaction confirm an unconfirmed transaction.*

Theorem 11. *The height of a transaction is equal to the maximum height of its parents plus one.*

Theorem 12. *If new transactions choose randomly the unconfirmed transactions to be confirmed, then no unconfirmed transaction will be left behind, i.e., all transactions will be confirmed in due time.*

7 CONCLUSION

Bitcoin's underlying technology blockchain has been called by many as a major invention comparable to the invention of the internet. But it is unlikely that Bitcoin and blockchain have achieved the final or most optimal design for a secure and scalable electronic transaction system. In this work, we will investigate whether DAG model is suitable to be a real alternative to blockchain.

Preliminary results obtained from our simulator have shown that, using a random tip selection strategy, the network seems to support up to 50 transactions per second, with no tip abandoned (Fig. 6) or taking too long to have at least one confirmation (Fig. 9). Today, Bitcoin network can barely handle 8 transactions per second without increasing the unconfirmed transaction list to hundreds of thousands — several transactions take days to be confirmed.

We have not yet tested the security of the network against double spend attacks. It will be one of the next steps and we expect to block a nuclear submarine attack including a rule for invalidate new conflicting transactions which confirm transactions with depth greater than a given threshold. We are unsure whether this rule would invalidate legitimate transactions on a high load scenario.

We also plan to improve the analysis of the confirmation time. So far, we have the histograms of how long it takes to a transaction to be confirmed for the first time (Fig. 8 and 9). We will also generate the histograms of how long it takes to a transaction to have an accumulated weight of at least a given number. It is an important histogram because it helps merchants and users to decide how long they should wait to consider the transaction almost irreversible.

We will also work on the mathematical modeling of DAG networks, and the equations will help us to understand the limits of these networks.

BIBLIOGRAPHY

[1] Moshe Babaioff, Shahar Dobzinski, Sigal Oren, and Aviv Zohar. On bitcoin and red balloons. In *Proceedings of the 13th ACM conference on electronic commerce*, pages 56–73. ACM, 2012.

[2] Norman TJ Bailey. On queueing processes with bulk service. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 80–87, 1954.

[3] Simon Barber, Xavier Boyen, Elaine Shi, and Ersin Uzun. Bitter to better—how to make bitcoin a better currency. In *International Conference on Financial Cryptography and Data Security*, pages 399–414. Springer, 2012.

[4] BitcoinStats. Data propagation, 2013-2017. http://bitcoinstats.com/network/propagation/.

[5] Blockchain.info. Bitcoin blockchain size. https://blockchain.info/charts/blocks-size. Last accessed on July 14, 2017.

[6] CoinMarketCap. Bitcoin market capitalizations. http://coinmarketcap.com/currencies/bitcoin/. Last accessed on July 14, 2017.

[7] Thomas H Cormen. *Introduction to algorithms*. MIT press, 2009.

[8] Christian Decker and Roger Wattenhofer. Information propagation in the bitcoin network. In *Peer-to-Peer Computing (P2P), 2013 IEEE Thirteenth International Conference on*, pages 1–10. IEEE, 2013.

[9] Discussion. Dag, a generalized blockchain, 2014. https://nxtforum.org/proof-of-stake-algorithm/dag-a-generalized-blockchain/ (registration at nxtforum.org required).

[10] Hans Dobbertin, Antoon Bosselaers, and Bart Preneel. Ripemd-160: A strengthened version of ripemd. In *Fast Software Encryption*, pages 71–82. Springer, 1996.

[11] Henri Gilbert and Helena Handschuh. Security analysis of sha-256 and sisters. In *International workshop on selected areas in cryptography*, pages 175–193. Springer, 2003.

[12] Ghassan Karame, Elli Androulaki, and Srdjan Capkun. Two bitcoins at the price of one? double-spending attacks on fast payments in bitcoin. *IACR Cryptology ePrint Archive*, 2012(248), 2012.

[13] Sergio Demian Lerner. Dagcoin: a cryptocurrency without blocks, 2015. Available at https://bitslog.wordpress.com/2015/09/11/dagcoin/.

[14] Yoad Lewenberg, Yonatan Sompolinsky, and Aviv Zohar. Inclusive block chain protocols, 2015. Available at http://www.cs.huji.ac.il/~avivz/pubs/15/inclusivebtc.pdf.

[15] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2008. Available at https://bitcoin.org/bitcoin.pdf.

[16] A Pinar Ozisik, George Bissias, and Brian N Levine. Estimation of miner hash rates and consensus on blockchains. Technical report, Tech. rep., PDF available from arxiv. org and https://www. cs. umass. edu/ brian/status-reports. pdf (June 2017).

[17] Serguei Popov and Jinn Labs. The tangle. 2016. Available at https://iota.org/IOTA_Whitepaper.pdf.

[18] Devavrat Shah et al. Gossip algorithms. *Foundations and Trends® in Networking*, 3(1):1–125, 2009.

[19] Yonatan Sompolinsky and Aviv Zohar. Accelerating bitcoin's transaction processing. fast money grows on trees, not chains, 2013. Available at https://eprint.iacr.org/2013/881.pdf.

[20] David Vorick. Getting rid of blocks, 2015. Available at slides.com/davidvorick/braids.