Hathor: An alternative towards a scalable cryptocurrency

Marcelo Salhab Brogliato
Hathor Network
Rio de Janeiro, Brazil
msbrogli@hathor.network

***Abstract*—Hathor is a transactional consensus platform comprised of an entirely novel architecture, based on concepts from both directed acyclic graph (DAG) and blockchain technologies combined. We propose a solution to the problems of scalability and decentralization maintenance among distributed ledger networks by including a chain of mined blocks inside a DAG of transactions. The blockchain ensures security when the number of transactions per second is small, whereas the DAG prevails when the number increases significantly. The primary result is that it seems to work correctly under any number of transactions per second.**

***Index Terms*—scability, dag, blockchain**

I. Introduction

The primary problem for creating digital money is how to prevent double spending. As the money is digital, and copies can be made *ad nauseam*, what can prevent counterfeiting? 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 and regulators.

The concept of transferring money using cryptography as an underlying technology was shortly presented in 1983 by Chaum [1] and was deepened in a theoretical paper in 1985 [2]. However, it was only in 1988 that Chaum et al. [3] created the term *electronic cash* and also proposed a basic and practical scheme which yielded untraceability yet allowed to trace double spendings.

According to Barber et al. [4], 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 [1, 5, 6, 7]. Some papers even propose solutions in a similar trajectory to Bitcoin, such as hash chain [8] and preventing double spending using peer-to-peer networks [9, 10]. 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.

During the research the authors received funds from the National Research Council of Brazil (CNPq www.cnpq.br). The research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest. The views expressed in this article are the author's own, and do not represent those of any institution.

Bitcoin (BTC) is a digital currency, also known as digital money, internet money, and cryptocurrency. It is the first currency based on cryptography techniques which are distributed, decentralized, and with no central bank.

Bitcoin is a computer network in which nodes act like clerks performing clearing. A transaction clearing consists of ensuring that the transaction is settled according to the rules of the system. In order to do that, every node stores a copy of Bitcoin's ledger, which records both all transactions and users' balance. When new transactions are added to the ledger, the balances are updated. It is said that Bitcoin is distributed because its ledger is public and is stored in thousands of computers. Even though the ledger is public, balances are anonymous, and no one knows who owns which funds[1]. If an attacker tries to change anything, the remaining of the network is able to detect it and ignore the change.

Bitcoin is considered decentralized because there is no authority (or government) who decides its future. Every decision must be accepted by its community, and no one can enforce their will. Every change proposal must be submitted to the community who will discuss the matter and come to a verdict. If the majority of Bitcoin's community agrees on a decision, they just have to update their clearing process accordingly, and the changes are applied.

The security of Bitcoin relies on digital signature technology and network consensus. While digital signature ensures ownership, i.e., the funds may only be spent by their owners, and nobody else; the network consensus 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. Only when all these requirements are met, the funds are transferred.

Bitcoin provides interesting incentives to all players (users and miners). On the one hand, users may have incentives to use Bitcoin because (i) the fees are small and do not depend on the amount being transferred — 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)

[1]There are some techniques which may de-anonymize transactions in specific situations, even when users are using Tor network. For further information, see ShenTu and Yu [11], Biryukov et al. [12], Jawaheri et al. [13].

and the currency issuance rate is well-known and preset in Bitcoin's rules, which makes Bitcoin's 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 collect the fees of unconfirmed transactions. It is important to highlight that anyone can become a miner, and there is no entry cost besides the mining equipment. These incentives have kept the Bitcoin network up and running since 2009 with barely any interruptions (99.99% uptime). For further information about incentives, see Ma et al. [14], Catalini and Gans [15].

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 [16] and expanded to the most important and successful cryptocurrency with a highly volatile $192 billion market capitalization, as of this writing [17]. 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 the following challenges: scaling, spamming, and centralization. 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 [18].

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.

The scaling problem is not precisely an issue of Bitcoin, but an issue of the Blockchain technology. Hence, all other Blockchain-based cryptocurrencies have the same limitations, such as Litecoin, Bitcoin Cash, and Ethereum. One may argue that increasing the maximum block size is a feasible solution to scaling, but I would say that it is just a temporary solution which buys some time until the next network saturation. Even allowing any block size would not be feasible because of spam attacks and bandwidth requirements.

Bitcoin seems to have the most decentralized network among the cryptocurrencies, even so, there are few miners and mining pools which together control over 50% of the network's computing (hash)power (for details, see Gencer et al. [19]). Hence, they have an oversized influence when it comes to changes in the Bitcoin protocol's behavior. They may also cooperate in an attack, voiding transactions which seemed confirmed. The more decentralized, the more trustworthy Bitcoin is. This centralization problem is seen as an important challenge to overcome.

Generating new transactions in Bitcoin has a tiny computational cost because one only has to generate the transaction itself, digitally sign it, and propagate it 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. This spam problem has happened several times and affects Bitcoin's trustworthy. Parker [20] reports a possible spam attack lasting at least contiguous 18 months.

After the launch of Bitcoin, more than 1,000 other cryptocurrencies have been created [21]. 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 Directed Acyclic Graph (DAG) model [22, 23, 24, 25, 26, 27]. We are especially interested in one of them: the Directed Acyclic Graph (DAG), which the most notorious implementation is 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[2] and is used to confirm other transactions, forming a directed acyclic graph of transactions. Thus, a transaction is said to be confirmed when there is enough proof-of-work from the transactions confirming it directly or indirectly. There is no other way to confirm transactions but generating new transactions.

In Iota, as transactions confirm transactions, the network benefits from a high volume of new transactions. Therefore, 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. While Iota's network has not reached this minimum number of transactions per second, it uses a central coordinator which works as a trustworthy node that stamps which transactions are valid and which are not [28].

Every transaction confirmed by the central coordinator is assumed to be valid and cannot be reverted. The remaining of the network can verify a confirmation through the central coordinator's digital signature. It is claimed that the coordinator will not be necessary anymore when the number of transactions per second reaches a minimum value, but Iota's developers cannot say precisely what is this minimum value. The presence of a coordinator just elucidates that the tangle does not seem to work properly under a low volume of transactions (and fluctuations in the number of transactions per second may severely affect Iota's trustworthiness).

[2]The mechanism that assures the immutability is the proof-of-work, which makes it computationally infeasible to tamper with transactions. It will be explained later in details.

The present work intends to propose and analyze a new architecture, named Hathor, which lies between Bitcoin and Iota and may be a viable solution to both scaling, centralization, and spam problems.

II. Hathor's architecture

This work introduces Hathor's architecture, which may be a solution to scaling, centralization, and spam issues on the Blockchain architecture. Hathor has both transactions and blocks connected forming a Directed Acyclic Graph (DAG). While transactions are only connected to other transactions, blocks are connected to both blocks and transactions forming a Blockchain inside the DAG. In Figure 1, the blocks are represented by red boxes, while transactions are represented by ovals. Both transactions and blocks have to solve a proof-of-work according to their weight. Blocks and transactions will generically be referred as vertices in this paper.

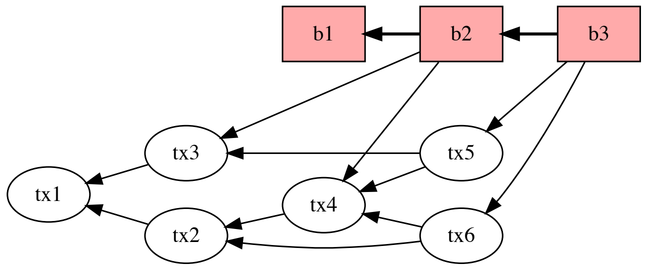


Fig. 1: Blocks are represented by red boxes, while transactions are represented by ovals.

The weight of blocks and transactions is the difficulty of mining them, i.e., the higher the weight, the harder it is to solve the proof-of-work. Let w_A be the weight of a vertex A, let the random variable X_A be the number of trials to solve the proof-of-work of this vertex, and $\mathbf{E}[X_A]$ be the average number of trials to solve the proof-of-work. Then,

$$w_A = \log_2 \mathbf{E}[X_A]$$

Blocks and transactions are connected to their parents, which form a directed acyclic graph called the DAG of verifications. When A is a parent of B, we say that B verifies A, in other words, B checks that A is a valid vertex. As A is also verifying its parents, we may use a recursive argument and say that B is verifying the whole subDAG behind its parents.

Blocks and transactions are also connected to their inputs, which form another directed acyclic graph called the DAG of funds. When A has an input pointing to an output of B, we say that A is spending an output of B, or simply A spends B. As blocks do not have inputs, they are dead-ends in the DAG of funds, i.e., if one chooses an arbitrary transaction and walks backwards through the DAG of funds, one will always ends in a block.

The genesis is the initial state of the DAG, and the state changes every new block or transaction connected to the DAG. This new vertex is valid when (i) its parents already exist and are valid as well, (ii) its timestamp is greater than their parents', and (iii) their funds are valid. The validity of the funds depends on the rules of the network and will be discussed later.

In Hathor, there are two difficulty levels: (i) one for new transactions which are just moving tokens around, and (ii) another one for "blocks" which are generating new tokens. The first may be adjusted to prevent spammers, which would spend too many resources to generate a great number of new transactions, whereas the latter is adjusted to keep the pace of new blocks constant.

Both Hathor's miners and users solve proof-of-works, decentralizing even more the network's hash rate. Even though the users' difficulty is less than the miners', the network's total hash rate will increase every new transaction. The more transactions arrive, the higher the total hash rate.

In a low load scenario, there is a small number of new transactions coming into the network, which means they give a minor contribution to confirmations. In this case, the confirmation is held mostly by blocks. On the other hand, in the high load scenarios, there is a large number of new transactions giving a major contribution to confirmations. In this case, the blocks strengthen the confirmations, but many of them will have already been confirmed before the next blocks are found. The higher the number of new transactions, the faster the transactions are confirmed. The blocks assure a "maximum confirmation time".

The incentive scheme which keeps the network running is the same as Bitcoin's. Miners go towards newly generate coins, whereas users just want to exchange their tokens. When there is no new transaction to be confirmed, the miners keep the network up and running while they find new blocks.

A. Transactions

There is a trade-off in the difficulty of mining new transactions. On the one hand, the higher the weight, the harder it is to generate new transactions, preventing spammers and increasing the confirmation of previous vertices. But, on the other hand, it is worse for microtransactions, and it is harder and slower for IoT and mobile devices to generate new transactions. An alternative is that IoT and mobile devices may only sign their transactions and send them to another device that will solve their proof-of-work and propagate it into the network. In the future, the weight of transactions may even be dynamic, increasing when a spam attack is in course and reducing when it is gone.

A transaction's weight depends only on the transaction's size (in bytes) and on the total amount being moved. The idea here is to allow small amounts to be easily moved, while big amounts will take longer to the moved. This allows microtransaction to quickly get into the network. Regarding the transaction's size, requiring more work for larger transactions makes sense because it prevents abuses, such as a denial-of-service attack using enormous transactions and consuming a lot of bandwidth and disk space.

B. *Conflicting transactions*

When two or more transactions spend the same output, we say they are conflicting transactions. In this case, at most one of them will be executed while all others will be voided. The executed transaction is the one with the highest accumulated weight, i.e., the one most verified by other transactions. In case of tie, all conflicting transactions will be voided. It is easier to notice that all conflicting transactions have the same outgoing neighbors in the DAG of funds.

When two transactions have exactly the same inputs and outputs, but different hashes, we say they are twin transactions. This is a special case of conflicting transactions because the funds are going from the same origins to the same destinations. All conflicting transactions but twin transactions are double spending attempts.

Every transaction has an accumulated weight, which is the amount of work that is verifying it. Let A be a transaction, w_A be the weight of A, and a_A be the accumulated weight of A. Let V be any vertex in the DAG such that there is a path from V to A, and w_V be the weight of V. Then,

$$a_A = \log_2\left(2^{w_A} + \sum_{V \rightsquigarrow A} 2^{w_V}\right)$$

When two or more transactions are in conflict, i.e., they all spend the same output, at most one of them will be executed, while all other will be voided. We say that the winner of a transaction is the transaction that has the highest accumulated weight. If there are more than one transaction with the highest accumulated weight, they are all voided.

C. *Block*

A block is like a regular transaction with no inputs which confirms exactly one previous block and at least two transactions. There may be any number of outputs provided that they sum up to the number of newly generated tokens. The blocks are ordered according to their timestamp.

Every block has a score, which is the amount of work that is verified by it. Let B be a block, w_B be the weight of B, and s_B be the score of B. Let V be any vertex in the DAG such that there is a path from B to V, and w_V be the weight of V. Then,

$$s_B = \log_2\left(2^{w_B} + \sum_{B \rightsquigarrow V} 2^{w_V}\right)$$

As blocks must have one, and only one, block as a parent, they form one or more blockchains inside the DAG of verifications. The blockchain with the highest score is called the best blockchain, which is the one that contains the highest work behind it. All blocks outside the best blockchain are voided, which means that all transactions spending their outputs are voided as well. In other words, only transactions spending outputs of blocks in the best blockchain may be executed.

D. *Transaction confirmation*

Transactions are classified into three groups: (i) confirmed transactions, (ii) in-progress transactions, and (iii) unverified transactions. The confirmed transactions are the ones which have already been settled, i.e., their accumulated weights have reached a minimum level. The unverified transactions are the brand new transactions which have not been verified even once yet, i.e., their accumulated weights equal their weights. The in-progress transactions are in the middle between confirmed and unverified transactions, i.e., they have already been verified a few times, but not enough to reach the minimum level required to be a confirmed transaction. For simplicity, the pending transactions encompass both in-progress and unverified transactions.

In Bitcoin, it is well-accepted that one should wait at least “six confirmations” before accepting a transaction, which means that at least six blocks in the best blockchain must confirm the transaction. This Bitcoin’s criteria is based on some results presented in Satoshi’s seminal work [16] and derived here in more detail in Appendix VIII. Adopting six confirmations is the same as saying that the probability of a successful attack is less than 0.1%.

Therefore, in order to have the same level of security as Bitcoin, a transaction is said to be confirmed when the probability of a successful attack is less than 0.1%. This probability depends only on the network hash rate and on the accumulated weight of the transaction. Thus, a transaction A is confirmed when:

$$a_A \geq \log_2(6 \cdot \Delta t \cdot H)$$

Where a_A is the accumulated weight of A, Δt is the average time between blocks, and H is the network hash rate.

E. *Transaction validation*

A transaction will be considered valid when it complies with the following rules: (i) it spends only unspent outputs; (ii) the sum of the inputs is greater than or equal to the sum of the outputs; (iii) it confirms at least two pending transactions; and (iv) it solves the proof-of-work with a valid weight.

Each transaction has a timestamp field which is used to record when it was generated. This timestamp field must be in UTC time to prevent timezone issues.

The digital signature is used to ensure that only the owners may spend their tokens. It will be calculated signing the transaction’s input and output only. This allows the transaction to be signed in one device and to be sent to another device that will choose which transaction will be confirmed and will solve the proof-of-work.

Services of solving proof-of-work may also be offered by companies. They give their customers a wallet address and they send the payment inside of the transaction itself. This allows IoT devices to save energy, delegating the task of solving the proof-of-work.

In case of transaction conflict, in which two transactions try to spend the same tokens, the one with higher accumulated weight is chosen and the other is invalidated. Although it is not a possible policy in Iota because of the submarine attack, Hathor does not have the same problem. In Hathor, like in Bitcoin, the submarine attack is only possible if the attacker has a hash rate higher than the whole network, including the miners. In other words, when analysing the double spending attack, Hathor is as safe as Bitcoin.

F. Orphan blocks

Differently from Bitcoin, orphan blocks are rare in Hathor. As blocks are just simple vertices in the DAG, they are light and are quickly propagated through the network. This reduces significantly the probability of orphan blocks.

G. Governance

In general, cryptocurrencies are decentralized, which means there is no central authority who decides its future, i.e., no one can enforce their will. Every decision must be accepted by its community, which means the community must agree. But what happens if they do not agree? When a consensus is not reached, the rules remain the same and the cryptocurrency may stall. The lack of a central authority may generate long debates, split the community, slow down strategic decision-making, and, ultimately, come to a "civil war"—it is precisely what happened between Bitcoin Core (BTC) and Bitcoin Cash (BCH) in 2017 [30, 31].

Governance is an important part of a cryptocurrency because it must evolve, which means its community must agree into changing the rules. Governance is an agreement of how the community will proceed to change the rules.

Despite the large literature available about governance, what separates Blockchain-based cryptocurrencies from them is the decentralization (against the hierarchical model). The number of papers about governance in decentralized cryptocurrencies has been growing, but it still lacks a solution. Hacker [32] resorts to the theory of complex systems and proposes a governance framework for decentralized cryptocurrencies, which is, in summary, a centralized coordination entity. Hsieh et al. [33] has analyzed the effects of governance in returns using panel data on several cryptocurrencies. They present a deeper discussion about the parts of a governance mechanism and concludes that

> "... on the one hand, investors value cryptocurrencies' core value proposition, rooted in decentralization; but on the other hand, are suspicious of decentralized governance at higher levels in the organization because they could slow down strategic decision-making (e.g., regarding the introduction of new innovations) or create information asymmetries between investors and technologists."

I believe that the solution to a good governance will come from financial incentives to all players to find a common ground. In Bitcoin, users have less bargaining power than miners because they do not contribute with work, whereas, in Hathor, both miners and users are working together. So, when it comes to changing the rules, the bargaining power is more distributed than in Bitcoin. The distribution depends on the ratio of the miners' hashpower and the users' hashpower. The higher the ratio, the closer to Bitcoin's governance. The lower the ratio, the higher the bargaining power of the users.

III. Consensus & Synchronization

A critical part of Hathor is its consensus and synchronization algorithms. The synchronization algorithm's goal is to keep all nodes in the peer-to-peer network synchronized. Two nodes are synced if, and only if, they have precisely the same set of blocks and transactions. The consensus algorithm's goal is to make all peers in the network agree on which blocks and transactions are executed and which are voided.

In this section, we are going to describe the algorithms that keep the peers in the network synced and in agreement.

List of Symbols

V_t	set of transactions
V_b	set of blocks
V	$V = V_t \cup V_b$, i.e., set of all blocks and transactions
Z	$Z \subseteq V$ is the set of voided transactions
Z_v	set of blocks or transactions that voids $v \in V$
E_v	$E_v = \{(v_i, v_j) \mid v_i \text{ verifies } v_j\}$, i.e., edges of verification
E_f	$E_f = \{(v_i, v_j) \mid v_i \text{ spends an output of } v_j\}$, i.e., edges of funds
E	$E = E_f \cup E_v$, i.e., edges of funds and verification
G_f	$G_f = (V, E_f)$, i.e., DAG of funds
G_v	$G_v = (V, E_v)$, i.e., DAG of verifications
G	$G = G_f \cup G_v$, i.e., complete DAG with both funds and verifications
B	$B = \{b \in V_b \mid \nexists x \in V_b, (x, b) \in E\}$, i.e., the head of the blockchains
w_v	weight of a block or transaction $v \in V$
a_v	accumulated weight of a transaction $v \in V_t$
s_b	score of a block $b \in V_b$
t_v	timestamp of a block or transaction $v \in V$
$\pi(b)$	parent of block $b \in V_b$ in the blockchain

IV. Consensus algorithm

The consensus algorithm's goal is to make all peers agree on which blocks and transactions are executed and which are voided. Since the synchronization algorithm is continuously running, we need to guarantee that peers will agree assuming they have the same DAG, i.e., $G_1 = G_2$. This way, the peers will eventually get synced and in agreement.

A. Conflicting transactions

Every transaction has inputs and outputs. The inputs point to other transaction's outputs, and we say that they are spending those outputs. We also say that a transaction is spending other transaction's funds when there is at least one input point to another transaction's output. We say that a transaction is in conflict with another transaction when they are both trying to spend the same output. Conflicts may have different topologies because transactions may conflict in different ways.

In Figure 2a, we can see the simplest conflict: both tx2 and tx3 are trying to spend output 0 of tx1. In this case, the one with highest accumulated weight will be executed, while the other will be voided. In case of a tie, both will be voided. Figure 2b has a similar situation, but there is one more transaction (tx4) trying to spend output 0 of tx1. The conflict resolution in this case is the same: only the transaction with highest accumulated weight will be executed, and the others will be voided.

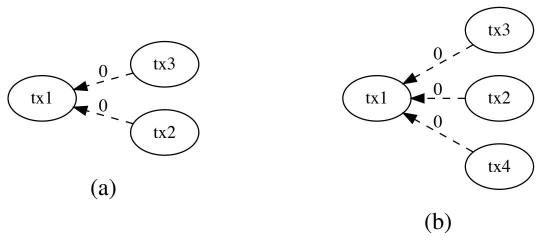


Fig. 2: In both examples, all transactions are trying to spend the output 0 of tx1, so, only one of them will be executed, while the others will be voided.

In Figure 3, tx4 has two groups of conflicts. It has a conflict with tx3 because both are trying to spend the output 0 of tx1, while it also has a conflict with tx5 because both are trying to spend the output 1 of tx2. Even though tx3 and tx5 are not in conflict between themselves, they will be both voided if tx4 is executed. If tx4 is voided, they are independent.

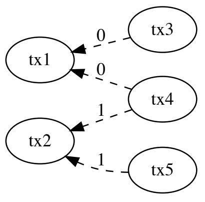


Fig. 3: tx4 has multiple conflicts. It conflicts with tx3 because of output 0 of tx1, and it also conflicts with tx5 because of output 1 of tx2. In this case, either tx4 is voided, or both tx3 and tx5 are voided.

The case presented in Figure 4 is more complex. While tx4 has conflicts with tx3 and tx5, there are two other transactions (tx6 and tx7) that depends on them. The situation of tx6 and tx7 are different because tx6 may be executed but tx7 may not. Since tx7 would only be executed when both tx4 and tx5 are executed, and tx4 and tx5 are in conflict, tx7 will always be a voided transaction. There are only three possible outcomes to the conflict resolution of Figure 4: (i) tx3, tx4, tx5, tx6, and tx7 are voided because tx3, tx4, and tx5 have the same accumulated weight; (ii) tx4 has the highest accumulated weight, so it is executed, and tx3, tx5, tx6, and tx7 are voided; or (iii) tx4 does not have the highest accumulated weight, so it is voided, and tx3, tx5, and tx6 are executed; but tx7 is also voided because it verifies tx4.

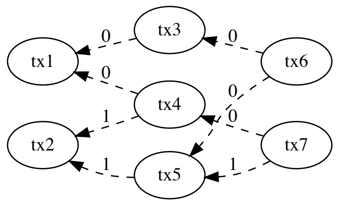


Fig. 4: tx4 has multiple conflicts, while tx6 and tx7 depends on the conflict resolution of tx3, tx4, and tx5. In this case, tx7 will always be voided, and tx6 will only be executed if tx4 is voided and both tx3 and tx5 are executed.

There are other interesting cases, such as in Figure 5, where tx3 is in conflict with tx2 while it is trying to spend an output of tx2. The conflict resolution in this case is simple because tx3 will always be voided. So, tx2 does not have a conflict in fact and is executed.

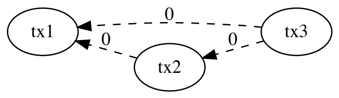


Fig. 5: tx3 is in conflict with tx2 because both are trying to spend output 0 of tx1. But, at the same time, tx3 tries to spend an output of tx2. In this case, tx2 will be executed, while tx3 will be voided.

B. Blockchain and the best chain

Besides transactions, the DAG also have blocks that make blockchains. A blockchain is a sequence of blocks such that each block verifies its previous block. Every blockchain can be uniquely identified by its head, i.e., the right-most block of the blockchain. In fact, every block is the head of a blockchain. In Figure 6, we can see several blockchains, for instance (i) blockchain **[b5]** = (b1, b2, b3, b4, b5), (ii) blockchain **[b6]** = (b1, b2, b6), (iii) blockchain **[b4]** = (b1, b2, b3, b4), and so on. Different blockchains always have at least one block in common—in this case, **[b5]** ∩ **[b6]** = (b1, b2). Notice that **[b5]** is the best blockchain, which means all blocks not in **[b5]** are voided; in this case, only b6.

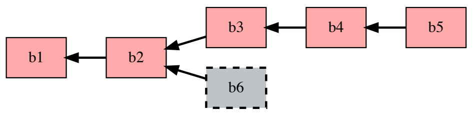


Fig. 6: Simple blockchain in which each block verifies its previous block.

In Figure 7, the blockchains **[b6]** and **[b11]** are candidates for the best blockchain. The next block will define which one will be the best blockchain. While a new block has not been found, **[b2]** is the temporary best blockchain. All miners are supposed to be mining from blocks b6 and b11.

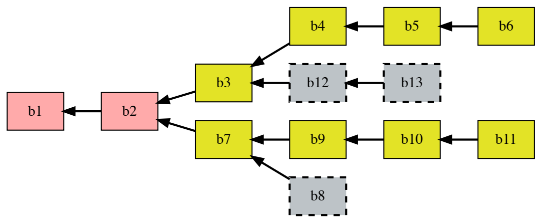


Fig. 7: Blockchains **[b6]** and **[b11]** are candidates for the best blockchain.

In fact, the criteria to choose the best blockchain is not the longest chain, but the one with highest score. The score of a blockchain is a measure of the work that is being verified by the head of the blockchain.

The score of the block is also affected by the transactions that are being verified, so, miners have the incentive to choose the most recent transactions instead of old ones. In Figure 8, **[b6]**'s score is higher than **[b7]**'s because the former directly verifies both tx7 and tx8 while the latter verifies tx7 and tx5.

Let s_{b6} and s_{b7} be the scores of b6 and b7, respectively, and w_x be the score of $x \in V$. Then,

$$s_{\text{b6}} = w_{\text{b6}} + \sum_{i=1}^{5} w_{b_i} + \sum_{i=1}^{8} w_{tx_i}$$
$$s_{\text{b7}} = w_{\text{b7}} + \sum_{i=1}^{5} w_{b_i} + \sum_{i=1}^{7} w_{tx_i}$$

Thus, assuming that $w_{\text{b6}} = w_{\text{b7}}$, we have $s_{\text{b6}} - s_{\text{b7}} = w_{\text{tx8}} \Rightarrow s_{\text{b6}} > s_{\text{b7}}$.

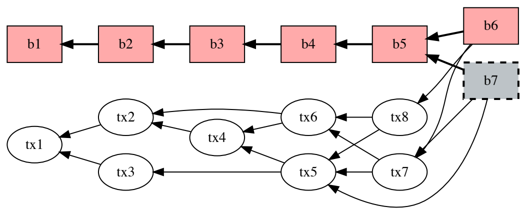


Fig. 8: **[b6]** is the best blockchain because its score is higher than the score of **[b7]**.

1) Dilemma: Who wins?: In Figure 9a, tx2 and tx3 are conflicting transactions, and $w_{tx3} > w_{tx2}$. So, $s_{b5} > s_{b4}$, which means **[b5]** would be the best blockchain. But, because of the many transactions verifying tx2, $a_{tx2} > a_{tx3}$, which means tx2 will be executed and tx3 will be voided. It is a dilemma. On the one side, the best blockchain should be **[b5]**. But, on the other side, tx2 should be executed and tx3 should be voided.

Which criteria will prevail? Either the score of the blockchain or the accumulated weight of the transactions? In this case, we say that the work done by b5 will be beaten by the work done by the "many other transactions". So, in the right outcome, tx3 and b5 are voided, while tx4 is executed and **[b4]** is the best blockchain. It is shown in Figure 9b.

This dilemma shows that it does not matter where the work comes from. It may come both from blocks and transactions, and no one is more important than the other. It is the same principle used by Blockchain-based architectures: the chain with the highest work prevails.

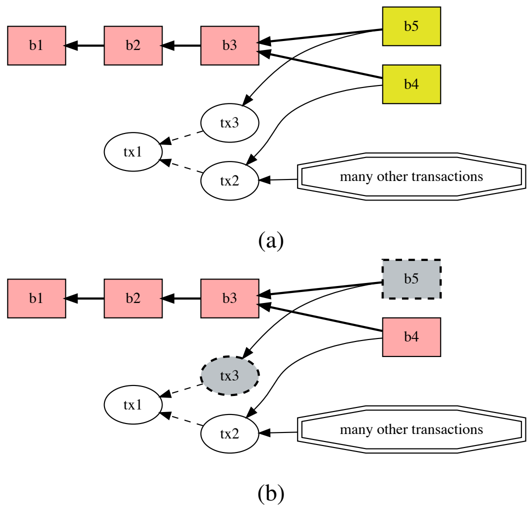


Fig. 9: tx2 and tx3 are conflicting transactions, and $w_{tx3} > w_{tx2}$, which implies that $s_{b5} > s_{b4}$. But, because of the main transactions verifying tx2, $a_{tx2} > a_{tx3}$. The dilemma is that, on the one hand, **[b5]** should be the best blockchain, but, on the other hand, tx3 should be voided. The solution in this case is presented in 9b, where tx3 is voided and **[b4]** is the best blockchain.

C. Consensus definition

Let $G = (V, E)$ be a DAG of blocks and transactions. Let $Z_v \subseteq \mathcal{P}(V)$ be the set of blocks and transactions that voids $v \in V$. Let $Z \subseteq V$ be the set of voided blocks and transactions, i.e., $Z = \{x \in V \mid Z_x \neq \emptyset\}$.

A consensus is formally a function $f : v \mapsto Z_v$ that maps a block or transaction to the set of blocks and transactions that voids it. Notice that both the accumulated weights and the scores are invariant with the consensus. We say that a consensus is valid if, and only if, it satisfies all of the following rules:

General rules

(i) If $x \in V$ is voided, then all its descendants are voided as well, i.e., $x \in Z, y \rightsquigarrow x \Rightarrow y \in Z \wedge Z_x \subseteq Z_y$.

(ii) If $x \in V$ is executed, than all its ancestors are executed as well, i.e., $x \notin Z, x \rightsquigarrow y \Rightarrow y \notin Z$.

(iii) Blocks or transactions may only void their descendants, i.e., let $Z_x \neq \emptyset$, then $\forall y \neq x \in Z_x$, $x \rightsquigarrow y$.

Blockchain rules

(iv) The head of the best blockchain has the highest score among all blocks that are not voided by an ancestor, i.e., $[b^*]$ is the best blockchain $\Leftrightarrow \{b^*\} = \{b \in V_b \mid s_b \geq s_{b^*} \wedge Z_b - \{b\} = \emptyset\}$.

(v) All blocks in the best blockchain are executed, and all blocks out of the best blockchain are voided, i.e., let b be a block, then $b \in [b^*] \Leftrightarrow Z_b = \emptyset$.

(vi) All blocks out of the best blockchain voids themselves, i.e., let b be a block and $[b^*]$ be the best blockchain, then $b \notin [b^*] \Leftrightarrow b \in Z_b$.

Transaction rules

(vii) v is a transaction with no conflicts $\Rightarrow v \notin Z_v$.

(viii) The winner transaction of a conflict has the highest accumulated weight among all conflicting transactions that are not voided by an ancestor. In case of a tie, they will all be voided. Let $C = \{v_0, v_1, \dots, v_n\}$ be conflicting transactions, then $v_k \notin Z_{v_k} \Leftrightarrow \{v_k\} = \{x \in C \mid a_x \geq a_{v_k} \wedge Z_x - \{x\} = \emptyset\}$.

(ix) Only one transaction of a conflict may be non-voided, i.e., let $\{v_0, v_1, \dots, v_n\}$ be transactions in conflict, then $v_k \notin Z_{v_k} \Rightarrow v_i \in Z_{v_i}, \forall\, i \neq k$.

We conjecture that there is one, and only one, consensus that is valid, i.e., if f and g are valid consensus, then $f = g$.

1) Analysis of the dilemma: Let's use the rules to analyse possible consensus for the dilemma of Figure 9a.

Let $f_1(v) = \emptyset$ be a consensus, i.e., all blocks and transactions are executed. It is clearly invalid because several rules would be violated. From rule (iv), we know that **[b5]** is the best blockchain. A first violation is that, from rule (vi), b4 must be voided but it is not. A second violation is that, from rule (ix), either tx2 or tx3 must be voided, but both are executed.

V. Synchronization algorithm

The primary goal of the synchronization algorithm is to let all peers in the peer-to-peer network have the same blocks and transactions. The assumptions is that anyone may join or leave the network at their discretion. In the network, each connection has exactly two peers who can exchange messages in both ways. It is assumed that the messages are reliable, ordered, error-checked, and cannot be tampered with. In practice, the messages are exchanged using a Transport Layer Security Security (TLS) protocol.

Another assumptions is that a new block or transaction may arrive at any time. These new blocks and transactions may have any timestamp, including old ones. This may happen during the merge of a split brain, for instance.

To establish a connection, one of the peers must try to connect to another peer. The peer that opens the connection is called a client, while the other peer is called a server. In the peer-to-peer network, there are two types of peers: (i) the ones that can be both clients and servers, (ii) and the ones that can only be clients. It is important because two client-only peers cannot communicate unless they both connect to a third-party proxy.

A. Split brain

A split brain in a peer-to-peer network occurs when the peers are partitioned in two or more parts such that the parts cannot communicate between them, in other words, each part will be by itself, without receiving new blocks and transactions from the other parts. The split brain can last any longer, from minutes to days. The challenge is to merge the DAGs after the network has recovered and the parts can communicate again. This is a critical problem for every distributed system, and we need to be clear about how we want to deal with this cases.

When a split brain occurs, each part will share a common past (the DAG before the split brain) and will start their own future (the new blocks and transactions arriving during the split brain). So, after the end of the split brain, the merging process will have to deal with many blockchains and maybe conflicting transactions.

A submarine attack is similar to a split brain, because attackers mine their own blocks and transactions separately, and then they propagate their blocks and transactions to the network in the right moment. It is like a planned split brain.

The synchronization algorithm is the easiest part. It has to copy the new blocks and transactions between all peers as fast as possible. Then, the consensus algorithm must decide which blocks and transactions are executed and which are voided. As multiple blockchains will exist, probably many blocks will be voided, and this voidance will be propagated to all transactions spending these blocks' outputs and their descendants. All voided transactions that are not conflicting transactions may be recovered, i.e., reconnected to non-voided parents and sent again to the network.

We may reduce this problem rejecting all blocks and transactions with timestamp smaller than a threshold. This is similar to a checkpoint, where some reorgs are prohibited in the protocol level. This approach would create hard forks when split brains occur for a long time, would allow some attack vectors against new peers joining the network, and would make the consensus algorithm be path dependent, i.e., it would not be enough to download the transactions in topological order, since the order the transactions are downloaded would impact on the final consensus. This is why such rule does not exists in Hathor.

I don't think there is a simple way to detect whether a split brain was caused intentionally by an attacker or not. We can say that a split brain without conflicting transactions between the parts is a sign of a non-intentional split brain.

B. Solution

Let t_v be the timestamp of $v \in V$. As the timestamp must be strictly increasing, $(a, b) \in E \Rightarrow t_a > t_b$. Hence, any sequence $(v_0, v_1, \dots, v_n, \dots)$ such that $t_{v_i} \leq t_{v_j}\ \forall\, i < j$ is a topological sorting, i.e., if a peer download the transactions in this order, all transactions' parents and inputs will have already been downloaded before it.

Let $H_x \subseteq G$ be the subDAG of descendants of $x \in V$, i.e., $v_i \in H_x \Leftrightarrow v_i \rightsquigarrow x$.

Definition 1. *Unverified transactions are the transactions which have not been verified yet, i.e., v_i is an unverified transaction when $\nexists\, v_j \mid (v_j, v_i) \in E_t$.*

Lemma 1. $v \in G \wedge v \rightsquigarrow x \Rightarrow x \in G$

Proof. Suppose $x \notin G$. Since $v \rightsquigarrow x$, take any path from v to x, namely $(v, a_1, \ldots, a_k, x)$. From this path, x is a parent of a_k. As $x \notin G$, a_k would be invalid, therefore the whole path would be invalid, including v_0, which is a contradiction. $\square$

Theorem 2. *Let $G_1 \subseteq G$ and $G_2 \subseteq G$ be two subDAGs of G, and $U_1 \subseteq V_1$ and $U_2 \subseteq V_2$ be their unverified transactions, respectively. Then, $G_1 = G_2 \Leftrightarrow U_1 = U_2$.*

Proof. ($\Rightarrow$) Suppose $U_1 \neq U_2$. Then, $\exists\, u \in U_1 \wedge u \notin U_2$, which implies either $(u \notin V_2)$ or $(u \in V_2 - U_2)$. If $u \notin V_2$, then $u \notin V_2 \Rightarrow u \notin V_1 \Rightarrow u \notin U_1$, which is a contradiction. If $u \in V_2 - U_2$, it means that $\exists\, x \in G_2$ such that $(x, u) \in E_2$. But, $u \in U_1 \Rightarrow (x, u) \notin E_1 \Rightarrow (x, u) \notin E_2$, which is a contradiction as well. The case of $u \notin U_1 \wedge u \in U_2$ is analogous.

($\Leftarrow$) Suppose $G_1 \neq G_2$. Then, $\exists\, x \in G_1 \wedge x \notin G_2$. Hence, there must exists a path from $u \in U_1 = U_2$ such that $u \rightsquigarrow x$. But, from Lemma 1, $u \in U_2 \wedge u \rightsquigarrow x \Rightarrow x \in G_2$, which is a contradiction. The case of $x \notin G_1 \wedge x \in G_2$ is analogous. $\square$

The above theorem is the base of the synchronization algorithm. It says that two DAGs are exactly the same if, and only if, their unverified transactions are equal. This means that it is enough to compare the unverified transactions of two peers to know whether they are synced or not.

But, as new transactions and blocks arrive, the DAG G changes over time. Let $G(t) \in G$ be the subDAG at time t, i.e., it contains only the blocks and transactions with timestamp smaller than or equal to t, i.e., $V(t) = \{x \in V \mid t_x \leq t\}$ and $E(t) = \{(x, y) \in E \mid t_x \leq t \wedge t_y \leq t\}$. It is easy to notice that $G(t_0) \subseteq G(t_1)$ for $t_0 \leq t_1$, since $V_0(t_0) = \{v_i \mid t_i \leq t_0\}$ is a subset of $V_1(t_1) = \{v_i \mid t_i \leq t_1\}$.

Theorem 3. *Let $G_1(t)$ and $G_2(t)$ be DAGs of two peers. Then, $G_1(t) = G_2(t) \Rightarrow G_1(i) = G_2(i)\, \forall\, i \leq t$.*

Proof. Since $i \leq t$, we have $G_1(i) \subseteq G_1(t)$ and $G_2(i) \subseteq G_2(t)$. Thus, $G_1(t) = G_2(t)$ implies $V_1(t) = V_2(t)$, which implies $V_1(i) = V_2(i)$, which implies $G_1(i) = G_2(i)$. $\square$

Theorem 4. *Let $G_1(t)$ and $G_2(t)$ be DAGs of two peers. Then, $G_1(t) \neq G_2(t) \Rightarrow G_1(i) \neq G_2(i)\, \forall\, i > t$.*

Proof. Suppose $G_1(i) = G_2(i)$. Since $t < i$, $G_1(t) \subseteq G_1(i)$ and $G_2(t) \subseteq G_2(i)$. From Theorem 3, we would have $G_1(t) = G_2(t)$, which is a contradiction. $\square$

Theorems 3 and 4 allows us to run an exponential search followed by a binary search to find the largest t^* such that both peers are synced, i.e., $G_1(t^*) = G_2(t^*)$ and $G_1(i) \neq G_2(i)\, \forall\, i > t^*$. In each step of the search, the peers check whether their unverified transactions are equal or not, which is sufficient due to Theorem 2.

After t^* has been found, the peers download the transactions with timestamp greater than t^* ordered by their timestamp. The download algorithm request the transactions, and checks the unverified transactions every new timestamp. If the unverified transactions of the peers are the same, both restarts the search for a new t^*.

C. Synchronizing in a dynamic DAG

The presented algorithm works for any static DAG G. After a finite number of messages, the peers are always synced. But how does it behave when a new block or transaction x arrives? In this case, x will be unverified regardless of its timestamp. Thus, $x \in U(t)\,, \forall\, t \geq t_x$. In this case, it will eventually be download after a search algorithm execution.

But, to quickly process a high number of transactions per second, the peers must synchronize in near real-time, which means we cannot just wait for the next search. Thus, when a peer receives x for the first time, it adds x to its DAG and propagates x to its neighbors.

1) Receiving old transactions: When an old transaction x arrive, it is unverified transactions and will be sent immediately to all synced peers. The syncing peers will find x when $t > t_x$ in the synchronization algorithm.

2) Propagation policies: The propagation policy is an algorithm that decides to which neighbors a new block or transaction will be send to. The flood algorithm is the simplest propagation policy and just send x to all neighbors. On one hand, it consumes more bandwidth, but, on the other hand, it is robust against sybil attacks.

Peers must prioritize blocks over transactions, which means they should propagate blocks as soon as possible, even if there are some transactions waiting in the queue.

We can simplify and say that a peer has two types of neighbor: synced and syncing. An optimized version of the flood algorithm is to send x to all peers that are either (i) synced or (ii) syncing with $t_x < t^*$. Another optimization is that a peer should never propagate x when it has requested to download x, because x has already been propagated.

Let $t_{max} = \max_{v \in V} t_v$ be the maximum timestamp of all blocks and transactions known by a peer. This peer classifies a neighbor as synced when $t_{max} - t_* \leq L$. The L threshold is important because, in a network propagating a high number of transactions per second from different places, two high-latency neighbors will be sending new transactions among them, which means they will never be totally synced. This analysis is not taking into account the blocks because there will be one block per minute, on average.

Let's say two peers have latency d (seconds) between messages with bandwidth B (bits/second). Let r be the number of transactions per second generated by one of them, while the other is not generating any transaction. Let s be the average size (bytes) of a transaction sent through the network. Suppose that new transactions are immediately sent to the other peer. Then, if $8 \cdot s \cdot r \geq B$ for more than L seconds, peers will not be synced anymore. Hence, $r_{max} = B/(8 \cdot s)$ is the maximum number of transactions per seconds a peer can countinuously withstand.

The size of the transactions depends on both the type of transaction and its number of inputs and outputs. A typical P2PKH transaction, with one input and two outputs, has around 300 bytes. Thus, let's add a network overhead and consider $s = 512$ bytes. In this case, a peer with bandwidth of 1Mbps ($B = 2^{20}$) would be able to receive 256 tps ($r_{max} = 256$).

Let n be the number of new transactions in a window of L seconds. Then, if the time interval between transactions follows an exponential distribution, we have that n follows a Poisson distribution with parameter $\lambda = r \cdot L$, i.e, $P(n = k) = e^{-\lambda} \frac{\lambda^k}{k!}$. Let $n_{max} = r_{max} \cdot L = \frac{B \cdot L}{8 \cdot s}$. Thus, the probability of a peer gets out-of-sync is:

$$P(n > n_{max}) = 1 - \sum_{k=0}^{\lfloor n_{max} \rfloor} P(n = k)$$

When $\lambda > 1000$, we may approximate the Poisson distribution by the Normal distribution with $\mu = \lambda$ and $\sigma^2 = \lambda$. Let Φ be the standard normal distribution, then:

$$\begin{aligned} P(n > n_{max}) &= 1 - \Phi\left(\frac{n_{max} - \lambda}{\sqrt{\lambda}}\right) \\ &= 1 - \Phi\left(\sqrt{\frac{L}{r}} \cdot (r_{max} - r)\right) \end{aligned}$$

It is interesting that increasing L reduces the probability of $P(n > n_{max})$ when $r < r_{max}$. But it also increases the probability when $r > r_{max}$. Generally speaking, if $r > r_{max}$ for more than L seconds, peers will get out-of-sync. Notice that r_{max} is a parameter of the peer, so, it will happen only for those which $r > r_{max}$.

3) Mining criteria: A full node is considered sync'ed and is able to mine new blocks when it is sync'ed with at least one peer.

D. Algorithm to compare $U_1(t)$ and $U_2(t)$

As $U(t)$ is the set of unverified transaction at time t, we need an efficient algorithm to find them.

Theorem 5. *Let x be a transaction and $V_f^x = \{y \mid (y, x) \in E\}$ be the set of transactions that directly verifies x in the DAG of verifications. Thus,*

$$x \in U(t) \Leftrightarrow t_x \leq t < \min_{y \in V_f^x} t_y$$

If $V_f^x = \emptyset$, $x \in U(t) \Leftrightarrow t \geq t_x$.

Proof. ($\Rightarrow$) Suppose that $\exists\, y \in V_x$ such that $t_y \leq t$. In this case, $(y, x) \in E_f(t)$, which implies $x \notin U(t)$, which is a contradiction.

($\Leftarrow$) $t < \min_{y \in V_f^x} t_y$ implies that $\nexists\, y$ such that $(y, x) \in E_f(t)$, which implies that $x \in U(t)$. □

Let I be an interval tree containing exactly one interval per transaction. So, for each $x \in V$, if $V_f^x \neq \emptyset$, then $[t_x, \min_{y \in V_f^x} t_y) \in I$, otherwise, $[t_x, \infty) \in I$.

Hence, using Theorem 5 and the interval tree I, we may calculate $U(t)$ looking for all intervals that intersect with t. This operation runs in $O(\log |V|)$.

The maintenance of I is also efficient, because add and delete operations runs in $O(\log |V|)$ as well.

VI. Conclusion

Bitcoin's underlying technology, blockchain, has been called by many as a major invention, even 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, I proposed and analysed a new architecture named Hathor, which seems a scalable alternative to Bitcoin.

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. In order to increase Bitcoin's capacity, its community has first proposed and implemented segregated witness, which improved scalability yet was not enough. Finally, they proposed the lightning network, which is in development and should be available in the next months. I believe these proposals relieve the network—a temporary solution—, but do not solve the scalability problem.

Hathor's architecture allows a great number of transactions per second, since new transactions verify previous ones (and there is no such thing as "maximum block size"). The more transactions are coming, the faster previous transactions will be confirmed. It is the opposite of Bitcoin because the network benefits from high volume scenarios. As I have shown, Hathor seems to solve the scalability problem present in Blockchain-based cryptocurrencies without affecting security.

As the transactions also have a proof-of-work, it becomes harder to perform a spam attack. The attacker would spend a considerable amount of computational resources to solve the proof-of-work of every transaction, and the amount of work depends on the transaction's weight parameter. Future work may explore automatic adjustments in transaction's weight to improve spam prevention. For instance, the network can detect a higher number of new transactions coming and increase the transaction's weight for a while. Or else, the transaction's weight may be a function of the time between an output being spent and its spending transaction, so, transferring the same tokens over and over in a small window of time would require more work. Anyway, the transaction's weight seems to tackle the spam issue. The new challenge is to set a proper transaction's weight which would prevent spam without impairing IoT devices.

The last, but not least, challenge is the hashpower centralization. Although Bitcoin seems to have the most decentralized network among cryptocurrencies, there are few miners and mining pools which *together* control over 50% of the network's computing (hash)power [19]. Hence, they have an oversized influence when it comes to changes in the Bitcoin protocol's behavior. Hathor's architecture splits the hashpower among miners and users. Even if miners have more individual hashpower than users, because they would have rigs with appropriate cooling and energy supply, I believe their aggregate hashpower will not surpass users' aggregate hashpower when millions of devices are generating transactions. Future IoT devices may even come with an application-specific integrated circuit (asic) designed to solve Hathor's proof-of-work without spending too much battery. Future work may check common IoT processors' hashpower, which would allows us to estimate how many devices would be necessary to surpass miners' hashpower.

References

[1] D. Chaum, "Blind signatures for untraceable payments," in *Advances in cryptology*. Springer, 1983, pp. 199–203.

[2] ——, "Security without identification: Transaction systems to make big brother obsolete," *Communications of the ACM*, vol. 28, no. 10, pp. 1030–1044, 1985.

[3] D. Chaum, A. Fiat, and M. Naor, "Untraceable electronic cash," in *Conference on the Theory and Application of Cryptography*. Springer, 1988, pp. 319–327.

[4] S. Barber, X. Boyen, E. Shi, and E. Uzun, "Bitter to better—how to make bitcoin a better currency," in *International Conference on Financial Cryptography and Data Security*. Springer, 2012, pp. 399–414.

[5] T. Okamoto, "An efficient divisible electronic cash scheme," in *Annual International Cryptology Conference*. Springer, 1995, pp. 438–451.

[6] J. Camenisch, S. Hohenberger, and A. Lysyanskaya, "Compact e-cash," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2005, pp. 302–321.

[7] S. Canard and A. Gouget, "Divisible e-cash systems can be truly anonymous," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2007, pp. 482–497.

[8] Y. Zongkai, L. Weimin, and T. Yunmeng, "A new fair micropayment system based on hash chain," in *e-Technology, e-Commerce and e-Service, 2004. EEE'04. 2004 IEEE International Conference on*. IEEE, 2004, pp. 139–145.

[9] I. Osipkov, E. Y. Vasserman, N. Hopper, and Y. Kim, "Combating double-spending using cooperative p2p systems," in *Distributed Computing Systems, 2007. ICDCS'07. 27th International Conference on*. IEEE, 2007, pp. 41–41.

[10] J.-H. Hoepman, "Distributed double spending prevention," in *International Workshop on Security Protocols*. Springer, 2007, pp. 152–165.

[11] Q. ShenTu and J. Yu, "Research on anonymization and de-anonymization in the bitcoin system," *arXiv preprint arXiv:1510.07782*, 2015.

[12] A. Biryukov, D. Khovratovich, and I. Pustogarov, "Deanonymisation of clients in bitcoin p2p network," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2014, pp. 15–29.

[13] H. A. Jawaheri, M. A. Sabah, Y. Boshmaf, and A. Erbad, "When a small leak sinks a great ship: Deanonymizing tor hidden service users through bitcoin transactions analysis," *arXiv preprint arXiv:1801.07501*, 2018.

[14] J. Ma, J. S. Gans, and R. Tourky, "Market structure in bitcoin mining," National Bureau of Economic Research, Tech. Rep., 2018.

[15] C. Catalini and J. S. Gans, "Some simple economics of the blockchain," National Bureau of Economic Research, Tech. Rep., 2016.

[16] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008, available at https://bitcoin.org/bitcoin.pdf.

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

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

[19] A. E. Gencer, S. Basu, I. Eyal, R. van Renesse, and E. G. Sirer, "Decentralization in bitcoin and ethereum networks," *arXiv preprint arXiv:1801.03998*, 2018.

[20] L. Parker, "Bitcoin 'spam attack' stressed network for at least 18 months, claims software developer," 2017, https://bravenewcoin.com/news/bitcoin-spam-attack-stressed-network-for-at-least-18-months-claims-software-developer/.

[21] CoinMarketCap, "Cryptocurrency market capitalizations," https://coinmarketcap.com/. Last accessed on July 14, 2017.

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

[23] S. Popov and J. Labs, "The tangle," 2016, available at https://iota.org/IOTA_Whitepaper.pdf.

[24] S. D. Lerner, "Dagcoin: a cryptocurrency without blocks," 2015, available at https://bitslog.wordpress.com/2015/09/11/dagcoin/.

[25] Y. Sompolinsky and A. Zohar, "Accelerating bitcoin's transaction processing. fast money grows on trees, not chains," 2013, available at https://eprint.iacr.org/2013/881.pdf.

[26] Y. Lewenberg, Y. Sompolinsky, and A. Zohar, "Inclusive block chain protocols," 2015, available at http://www.cs.huji.ac.il/~avivz/pubs/15/inclusivebtc.pdf.

[27] D. Vorick, "Getting rid of blocks," 2015, available at slides.com/davidvorick/braids.

[28] D. Sønstebø, "The transparency compendium," https://blog.iota.org/the-transparency-compendium-26aa5bb8e260. Last accessed on July 20, 2017.

[29] BitcoinWiki, "Controlled supply," 2017, https://en.bitcoin.it/wiki/Controlled_supply.

[30] L. Y. Chen and Y. Nakamura, "Bitcoin is having a civil war right as it enters a critical month," 2017, https://www.bloomberg.com/news/articles/2017-07-10/bitcoin-risks-splintering-as-civil-war-enters-critical-month.

[31] L. Shin, "Bitcoin cash skyrockets, bitcoin price drops as civil war continues," 2017, https://www.forbes.com/sites/laurashin/2017/11/12/bitcoin-cash-skyrockets-bitcoin-price-drops-as-civil-war-continues/#67ffeed635b5.

[32] P. Hacker, "Corporate governance for complex cryptocurrencies? a framework for stability and decision making in blockchain-based monetary systems," 2017.

[33] Y.-Y. Hsieh, J.-P. Vergne, and S. Wang, "The internal and external governance of blockchain-based organizations: Evidence from cryptocurrencies," 2017.

[34] H. Gilbert and H. Handschuh, "Security analysis of sha-256 and sisters," in *International workshop on selected areas in cryptography*. Springer, 2003, pp. 175–193.

[35] H. Dobbertin, A. Bosselaers, and B. Preneel, "Ripemd-160: A strengthened version of ripemd," in *Fast Software Encryption*. Springer, 1996, pp. 71–82.

[36] A. P. Ozisik, G. Bissias, and B. N. Levine, "Estimation of miner hash rates and consensus on blockchains," Tech. rep., PDF available from arxiv. org and https://www. cs. umass. edu/ brian/status-reports. pdf (June 2017), Tech. Rep.

[37] C. Decker and R. Wattenhofer, "Information propagation in the bitcoin network," in *Peer-to-Peer Computing (P2P), 2013 IEEE Thirteenth International Conference on*. IEEE, 2013, pp. 1–10.

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

[39] M. Babaioff, S. Dobzinski, S. Oren, and A. Zohar, "On bitcoin and red balloons," in *Proceedings of the 13th ACM conference on electronic commerce*. ACM, 2012, pp. 56–73.

[40] D. Shah *et al.*, "Gossip algorithms," *Foundations and Trends® in Networking*, vol. 3, no. 1, pp. 1–125, 2009.

[41] G. Karame, E. Androulaki, and S. Capkun, "Two bitcoins at the price of one? double-spending attacks on fast payments in bitcoin." *IACR Cryptology ePrint Archive*, vol. 2012, no. 248, 2012.

[42] E. Heilman, A. Kendler, A. Zohar, and S. Goldberg, "Eclipse attacks on bitcoin's peer-to-peer network." in *USENIX Security Symposium*, 2015, pp. 129–144.

[43] L. Bahack, "Theoretical bitcoin attacks with less than half of the computational power (draft)," *arXiv preprint arXiv:1312.7013*, 2013.

[44] J. Bonneau, E. W. Felten, S. Goldfeder, J. A. Kroll, and A. Narayanan, "Why buy when you can rent? bribery attacks on bitcoin consensus," 2016.

[45] T. Neudecker, P. Andelfinger, and H. Hartenstein, "A simulation model for analysis of attacks on the bitcoin peer-to-peer network," in *Integrated Network Management (IM), 2015 IFIP/IEEE International Symposium on*. IEEE, 2015, pp. 1327–1332.

[46] N. T. Bailey, "On queueing processes with bulk service," *Journal of the Royal Statistical Society. Series B (Methodological)*, pp. 80–87, 1954.

VII. Appendix: Consensus Analysis

A. Algorithm to find a consensus

The proposed algorithm is iterative and processes the blocks and transactions in topological sort, which means that, when a block or transaction is being processed, all their parents and inputs have already been processed.

For each block or transaction v, we store Z_v as a metadata called `voided_by`. The algorithm updates the sets Z_v when processing new blocks or transactions.

The algorithm assumes that the consensus is valid before starting to process a block or transaction. Under this assumption, it guarantees that the consensus will also be valid after processing. As we start with only the genesis, the initial consensus is always valid.

The algorithm for blocks is different from the algorithm for transactions. While the algorithm for transactions focus on avoiding double spendings, the algorithm for blocks focus on finding the best blockchain.

1) Calculation of the accumulated weight a_v: The accumulated weight is a measure of the work that has been done to verify a transaction. It is calculated as the sum of the weight of all descendants of v. Thus, every new block or transaction affects the accumulated weight of all its ancestors.

Let v be a transaction, w_v be the weight of v, a_v be the accumulated weight of v, and $A = \{x \in V \mid x \rightsquigarrow v\}$. Then,

$$a_v = \log_2 \left(2^{w_v} + \sum_{x \in A} 2^{w_x} \right)$$

The calculation of a_v runs in $O(|V|)$, which may be too expensive. So, we store previous calculations in a metadata called `acc_weight`. As we will see in the algorithm, we do not have to know the exact value of a_v. We just need to compare it with another value. So, we may stop the calculation as soon as we trespass this other value.

2) Calculation of the score (s_b): The score is a measure of the work that is being verified by a block. It is calculated as the sum of the weight all ancestors of b. As the ancestors of a block never changes, new blocks or transactions do not affect the score of the blocks.

Let b be a block, w_b be the weight of b, s_b be the score of b, and $D = \{x \in V \mid b \rightsquigarrow x\}$. Then,

$$s_b = \log_2 \left(2^{w_b} + \sum_{x \in D} 2^{w_x} \right)$$

The calculation of s_b runs in $O(|V|)$, which may be too expensive. To calculate it faster, we store a metadata called `first_block` for each transaction. This metadata points to the first block that verifies it. So, it is empty when it has not been verified by any block yet.

Let $E = \{v \in V_t \mid b \rightsquigarrow v \wedge v \text{ has not been verified by any block yet}\}$. Thus, when a new block is added to the DAG, we calculate:

$$s_b = \log_2 \left(2^{w_b} + 2^{s_{\pi(b)}} + \sum_{x \in E} 2^{w_x} \right)$$

This new calculation runs in $O(|E|)$.

B. Algorithm for new transactions

This algorithm must detect conflicting transactions, and decide which one of them will be executed, while the remaining will be voided. Then, it has to guarantee that all descendants of a voided transaction will also be voided.

The detection of conflicting transactions is simple. We store a set of transactions spending every transaction's output in metadata, called `spent_outputs`. So, if there are two or more transactions in the same set, they are all conflicting transactions. The maintenance of this list is done in constant time. It is updated when the transaction is added to G.

If a new transaction v has a conflict, we need to resolve its conflict. The resolution is done comparing the accumulated weight of v with the accumulated weight of every conflict x such that $Z_x - \{x\} = \emptyset$. There are some optimizations to make it faster, but we will not discuss them here.

When a v, we can fastly calculate Z_v as the union of its parents' and inputs'. As v affects the accumulated weight of all its ancestors, we need to resolve the conflicts of all transactions in Z_v.

The algorithm is detailed in Algorithms 1, 2, 3, and 4.

There are some optimizations in Algorithm 2 regarding the calculation of the accumulated weight. Although the calculation runs in $O(|V|)$, most of the times we don't have to actually calculate it. As $a_v < m$ is enough to finish the conflict resolution, we can first compare a_v to the accumulated weight of the voided transactions in B. As the accumulated weight of voided transactions is always updated, we can do it in $O(1)$. Then, only if a_v is the highest among them, we compare a_v to the accumulated weight of the executed transactions. Even so, before calculating the values, we can compare a_v to $s_{\text{head}} - s_{\text{first-block}_i} + w_i$.

C. Algorithm for new blocks

This algorithm must find the best blockchain and ensure that all blocks out of the best blockchain is voided.

When a new block b is added to G, it may be connected to either the head of the best blockchain or to any other block. If it is connected to the head of the best blockchain, it will probably be the next head of the best blockchain, depending only on the transactions it verifies. If all its parents are executed, then b will be the next head of the best blockchain. Otherwise, b will be voided.

When b is connected to any other block, we need

D. Lemmas & Theorems

Now, let's work on some logical conclusions from this set of rules.

Lemma 6. *Let $b \in V_b$ be a block. Then, $s_b > s_x$, $\forall\, x \in [b]$.*

Proof. By the definition, $s_x = \log_2 \left(2^{w_x} + \sum_{y \mid x \rightsquigarrow y} 2^{w_y} \right)$, and $s_b = \log_2 \left(2^{w_b} + \sum_{y \mid b \rightsquigarrow y} 2^{w_y} \right)$. As $b \rightsquigarrow x$, we have that $x \rightsquigarrow y \Rightarrow b \rightsquigarrow y$. So, $\{y \mid x \rightsquigarrow y\} \subset \{y \mid b \rightsquigarrow y\}$. Hence,

$$\begin{aligned}\sum_{y|b\rightsquigarrow y} 2^{w_y} &= 2^{w_x} + \sum_{y\neq x|b\rightsquigarrow y} 2^{w_y} \\ &\geq 2^{w_x} + \sum_{y|v\rightsquigarrow y} 2^{w_y} \\ &= 2^{s_x}\end{aligned}$$

Finally, $2^{w_b} + \sum_{y|b\rightsquigarrow y} 2^{w_y} > \sum_{y|b\rightsquigarrow y} 2^{w_y} \geq 2^{s_x}$, which implies $s_b > s_x$. □

Lemma 7. $b \in B \wedge |B| = 1 \Rightarrow [b]$ *is the best blockchain*

Proof. From Lemma 6, b has the highest score in **[b]**. As $|B| = 1$, it is the only blockchain, and b has the highest score of all blocks. From rule (iv), **[b]** is the best blockchain. □

Lemma 8. $Z \neq \emptyset \wedge |B| = 1 \Rightarrow \exists\, v \in V\,,\, Z_v = \{v\}$

Proof. Suppose that $\nexists\, v \in V\,,\, Z_v = \{v\}$. In this case, let $v \in Z$ be a voided block or transaction. But, v cannot be a block because $|B| = 1$, and, from Lemma 7 and rule (v), all blocks in the best blockchain are executed.

So, v can only be a transaction. We also know that $v \in Z \Rightarrow Z_v \neq \emptyset$. Let $x \in Z_v$ be a block or a transaction. Because of rule (v), x cannot be a block, so x must be a voided transaction with conflicts. Thus, from rule (ix), $x \in Z_x$, which is a contradiction. □

Lemma 9. *Let v be a transaction, then $\exists\, Z_x\,,\, v \in Z_x \Rightarrow v$ has a conflict $\wedge v \in Z_v$.*

Proof. Suppose that v is a transaction with no conflicts. Then, from rule (vii), $v \notin Z_v$. Let's partition V in two subsets: descendants and non-descendants of v.

For the descendants, from rule (i), $x \notin Z_y \,\forall\, y\,,\, y \rightsquigarrow x$.

For the non-descendants, suppose that $z \in V$ such that $z \not\rightsquigarrow x$ and $v \in Z_z$. Thus, □

Theorem 10. *Let G be a DAG with no conflicting transactions and a single blockchain. Then $f(v) = \emptyset$ is the only valid consensus.*

Proof. Suppose that f is a valid consensus, and $\exists\, v \in V\,,\, f(v) = Z_v \neq \emptyset$. It implies that $Z \neq \emptyset$. As there is a single blockchain, we have $|B| = 1$. So, from Lemma 8, $\exists\, v \in V\,,\, Z_v = \{v\}$. But, from rule (vii), $v \in Z_v \Rightarrow v$ has a conflict, which is a contradiction. □

Conjecture 11. *For any G, there is only one valid consensus* **S**.

VIII. Appendix: Bitcoin Analysis

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

A. Hash function

Hash functions has been widely studied in computer science. In short, a hash function $h : \{0,1\}^\infty \rightarrow \{0,1\}^n$ 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 [34], Dobbertin et al. [35].

B. Mining one block

Let $\mathbb{B}$ be the set of Bitcoin blocks and $h : \mathbb{B} \rightarrow \{0,1\}^{256}$ 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 $(x_1, x_2, \dots, x_k)$ until they find a solution, i.e., all attempts will fail ($h(x_i) \geq A$ 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 $\mathbf{P}(X \leq k) = 1 - (1-p)^k$. 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 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 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 $\mathbf{P}(T \leq t) = \mathbf{P}(X/H \leq t) = \mathbf{P}(X \leq tH) = 1 - (1-p)^{tH} = 1 - \left(1 - \frac{1}{\eta H}\right)^{tH}$. But, as the Bitcoin network hash rate is really large, we may approximate the CDF of T by $\lim_{H \to \infty} \mathbf{P}(T \leq t) = 1 - e^{-\frac{t}{\eta}}$, which is equal to the CDF of the exponential distribution with parameter $\lambda = \frac{1}{\eta}$.

Theorem 12. *When $H \to +\infty$, the time between blocks follows an exponential distribution with parameter $\lambda = \frac{1}{\eta}$, i.e., $\lim_{H \to +\infty} \mathbf{P}(T \leq t) = 1 - e^{-\frac{t}{\eta}}$.*

Proof.

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

Replacing $u = \eta H$,

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

□

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

Theorem 13. $x > M \Rightarrow |(1 + 1/x)^x - e| < e/M$.

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 $\lambda = 1/\eta = pH$ may be used to estimate the total network's hash rate (or a miner's hash rate). For further information, see Ozisik et al. [36].

C. 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 distribution 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 10, 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.

D. 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

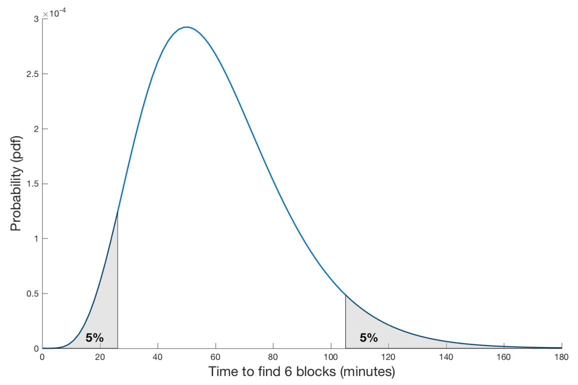


Fig. 10: Probability density function of Y_6, i.e., probability of finding 6 blocks after time t. The shaded areas shows the lower 5% and upper 5% of the pdf.

$T_\alpha = \frac{X}{\alpha H}$ 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 14. *When the miner with α percent of the network's total hash rate is part of the mining network, $\mathbf{P}$(next block is from T_α) $= \alpha$.*

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 15. *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.

E. 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 [37], 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 [38] 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 the 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. [39] 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. [40].

F. 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, $\mathbf{P}(T \leq t) = \mathbf{P}(X \leq H \int_0^t u(t)dt)$. When $H \to +\infty$, $\mathbf{P}(T \leq t) = 1 - e^{-\frac{1}{\eta}\int_0^t u(t)dt}$, and the pdf of T is $\frac{u(t)}{\eta} \cdot e^{-\frac{1}{\eta}\int_0^t u(t)dt}$.

1) Hash rate suddenly changing: 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.

2) 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 $\frac{1+abx}{1+bx} = \frac{1-a}{1+bx} + a$, which yields $\int_0^t u(x)dx = at + \frac{1-a}{b}\log(1+bt)$. 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:

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

Let $\mathcal{L}$ be the Laplace Transform. 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 $\mathcal{L}\{f_T(t)\} = s\mathcal{L}\{F_T(t)\}$,

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

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, $b \to +\infty$ yields $n \to 0$, $u(t) \to a$, and $\mathbf{E}[T] = \frac{1}{\lambda a}$.

Theorem 16.

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

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 $\lambda = 1/600$, $a = 2$, $M = 1$ week $= 3600\cdot 24\cdot 7 = 604,800$, b can be calculated using $\epsilon = \frac{a-1}{1+bM} < 0.01$, which yields $b > 0.000163690$ and $n < 10.1818$. 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) 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, $\mathbf{P}(T \le t) = 1 - e^{-\frac{bt + a^2t^2/2}{\eta}}$.

It is well known that $\mathbf{E}(T) = \int_0^\infty 1 - \mathbf{P}(T \le t)dt$. 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:

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

4) Comparison of the models: In order to compare the hash rate change models, namely (i) suddenly changing, (ii) smoothly changing, and (iii) linearly changing, I have applied each of them to the same scenarios. In the first scenario, the hashrate will double in the next week, whereas, in the second scenario, it will halve in the next week.

In both the smoothly change model and the linear change model, I could have calculated each model's average time between blocks during one week. But, it would not give us much information, because the estimated average time between models would be increasing (or decreasing) more and more as the days goes by. And we are really interested in the average time between blocks throughout the days, and not the average of one week.

Thus, I have analyzed a piecewise hash rate change, i.e., I have calculated the average time between blocks for each hour throughout the week. First, I split the whole week into $24\cdot 7$ intervals, $(t_0, t_1, t_2, \ldots, t_168)$, where $t_i = 3600i$. Then, I calculated the average for each interval (t_k, t_{k+1}). Let H_k^0 and H_k^1 be the initial and final hash rate of the (t_k, t_{k+1}) interval. So, I also ensured the continuity of the hash rate between consecutive intervals, i.e., $H_k^1 = H_{k+1}^0$.

I compared both the smoothly change model and the linear change model with the suddenly changing model. The difference between them is negligible. Let ϵ be the maximum absolute error between the models, than $\epsilon < 0.8$ and $\epsilon/H < 0.2\%$, for all intervals. The maximum absolute error between the linear and the suddenly changing models can be seen in Figure 11.

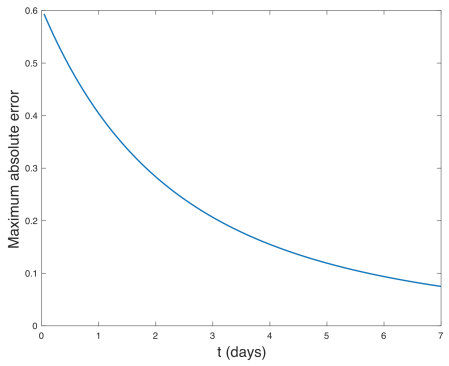


Fig. 11: Maximum absolute error between the linear and the suddently change models.

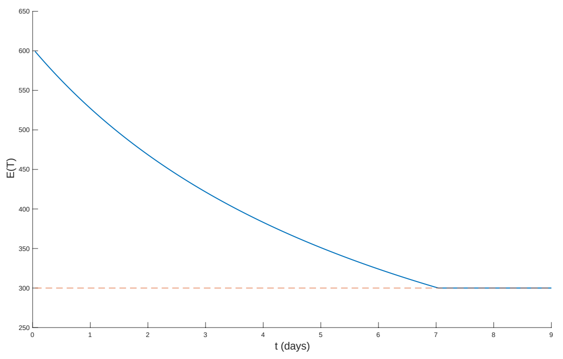
Therefore, we may conclude that it is reasonable to approximate the average time between blocks using only the suddenly changing model in each interval of one hour.

The average time between blocks throughout the days can be seen in Figure 12. It was calculated using the suddenly changing model with the hash rate changing linearly during the week.

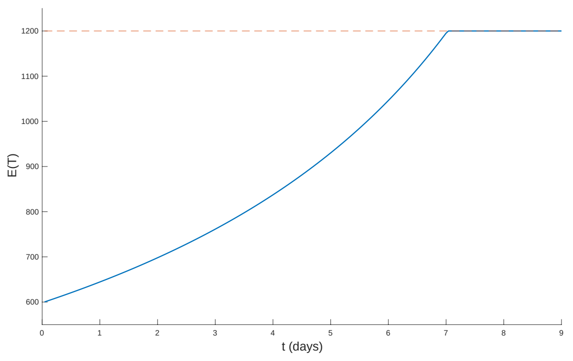
G. Attack in the Bitcoin network

There are many possible ways to attack the Bitcoin network [41, 42, 43, 44, 45]. 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



(a) Doubling the hash rate



(b) Halving the hash rate

Fig. 12: The average time between blocks when the hash rate changes over time.

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 [16].

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

As Nakamoto [16] 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 $m \to +\infty$,

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

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, $p = \frac{\lambda_{\text{network}}}{\lambda_{\text{network}} + \lambda_{\text{attacker}}} = \frac{\gamma}{\beta+\gamma}$, 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 Nakamoto [16].

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 17. *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 13. 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 $\binom{k+s-1}{s}(1-p)^s p^k$.

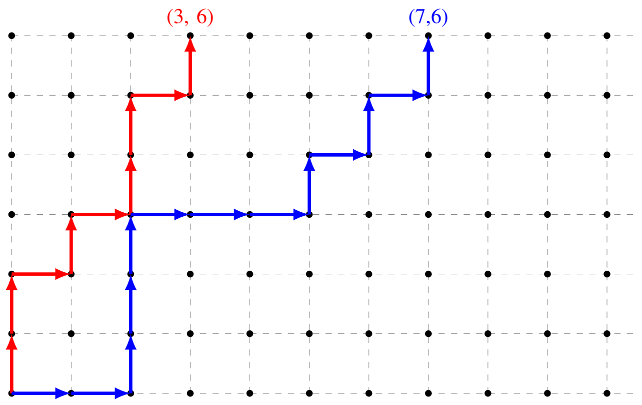


Fig. 13: Both the attacker and the network are mining. Each step up is a new block found by the network with probability p. Each step right is a new block found by the attacker with probability $1-p$. It ends when the network finds k blocks — in this example, $k = 6$. The red path has probability $p^6(1-p)^3$, 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 $\mathbf{P}(S \geq k) = \sum_{s=k}^{\infty} \binom{k+s-1}{s}(1-p)^s p^k$.

Theorem 18.

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

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

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

Proof.

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

□

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

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

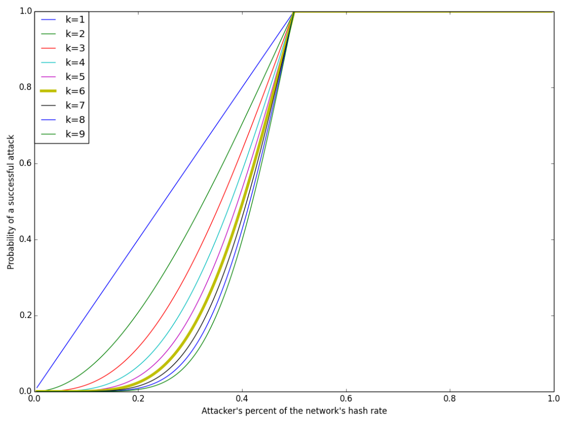


Fig. 14: Probability of a successful attack according to the network's hash rate of the attacker (β).

H. 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 [46], 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 $m = \frac{\lambda_{\text{TX}}}{\lambda_{\text{blocks}}}$ and z_s is the single root of the polynomial $z^s(1 + m(1-z)) - 1$ with $|z_s| > 1$. In this case, the average size of the unconfirmed transaction pool is $\mathbf{E}(\pi) = \frac{1}{z_s - 1}$.

When $s > m$, the probabilities π_n form a simple geometric series with common ratio smaller than one, which means the probabilities are exponentially decreasing. Since $\pi_n \to 0$ when

$n \to \infty$, we may interpret it as a stable system, i.e., the unconfirmed transactions pool size is finite.

When $s \leq m$, the system is unstable, which means the unconfirmed transactions pool size keeps growing towards infinity. In this case, the system is not capable of processing the demand for a long period of time.

Using the fact that $m = \frac{\lambda_{\text{TX}}}{\lambda_{\text{blocks}}}$ and $\lambda_{\text{blocks}} = 1/\eta$, the stability condition $s > m$ is reached when $\lambda_{\text{TX}} < s/\eta$.

In the Bitcoin network, the average number of transactions per block is $s = 2,250$, so, the system is stable when $\lambda_{\text{TX}} < 2,250/600 = 3.75$ tx/s. Therefore, 3.75 is the maximum number of new transactions per second that the Bitcoin network may handle. When $\lambda_{\text{TX}} > 3.75$ tx/s, the unconfirmed transaction pool starts to grow indefinitely.

When the system is stable, the average waiting time of a transaction to be confirmed is $\mathbf{E}(w) = \frac{1}{\lambda_{\text{TX}}(z_s - 1)}$.

$m \ll s$ yields $z_s \to 1 + 1/m$. Thus, the average number of unconfirmed transactions $\mathbf{E}(\pi) \to m$ and the average waiting time $\mathbf{E}(w) \to \frac{1}{\lambda_{\text{blocks}}} = \eta = 600$ seconds. In the Bitcoin network, $m \ll s$ is reached when $\lambda_{\text{TX}} \ll 3.75$ tx/s. In other words, when the number of new transactions per second is way smaller than 3.75 tx/s, the average waiting time of a transaction to be confirmed is 600 seconds, which means, on average, all transactions will be confirmed in the next block.

But, $\lambda_{\text{TX}} \to s/\eta$ yields $z_s \to 1$. Hence, $\mathbf{E}(\pi) \to +\infty$, which means the system is going towards instability.

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

For instance, in order to be a stable system and process 15 transactions per second, each block would have to confirm, on average, 9,000 transactions. Bitcoin's network is really far from this point.

IX. Notes on Scalability

Many projects have been claiming to process dozens of thousands of transactions per second, even hundreds of thousands of transactions per second. I do not know whether they can really process these numbers of transactions per second, but I would like to make a few comments about it.

Before going into the limit of the number of transactions per second, I think we should answer the following question: Whom are we developing Hathor to? Should it run either in a home computer or in a data center? I think both of these questions are important to understanding how far we can go in the number of transactions per second.

In Hathor, a transaction is made of inputs, outputs, parents, and some other fields. As part of the verification of a transaction validity, we need to verify the digital signatures of the inputs. The inputs may be linked to different types of outputs. For a P2PKH output, we need to verify one digital signature. For a P2SH output using multisig, we need to verify n digital signatures, according to the multisig configuration. Anyway, we need to verify at least one digital signature per transaction.

Hence, we run a performance test to check how many digital signatures a home computer is able to verify, and the results are surprising. We chose to test the secp256k1, which is an elliptic curve digital signature algorithm defined in Standards for Efficient Cryptography (SEC) and used by Bitcoin and many other tokens.

A. Methodology

The performance test of the secp256k1 algorithm was developed in Python 3.6 calling the libssl library, while the tests of the ecdsap256 and the rsa2048 were directly tested in the openssl command-line interface.

We used Python 3.6 because it is easy for anyone to run the test as well. The overhead of the language seems negligible because it is just calling the libssl library. Even so, maybe it is a good idea to run the performance tests again but entirely developed in C language.

B. Results

A 2.7 GHz Core i7 (I7-8559U) processor can verify 2,700 digital signatures per second per core. As it has four physical cores and eight virtual cores, if we assume no overhead, it will be able to verify up to 10,800 digital signatures per second using all four cores.

We have tested with several other processors, but all of them had poorer performance than the i7 processor above.

In 2012, Peter Wulle ran a similar test using a 2.2 GHz Core i7- (I7-670QM) and could verify 1,735 digital signatures per second per core. See https://bitcointalk.org/index.php?topic=103172.msg1131983#msg1131983.

C. Other algorithms

We have also tested ecdsap256 and rsa2048 in the same processor. The i7 was the fastest again and can verify 464 digital signatures of ecdsap256 per second per core, and 854 digital signatures of the rsa2048 per second per core.

D. Discussion

I wonder what processors those projects are using to be able to process that number of transactions per second. Either they are using other digital signatures algorithms, or they are processing the transactions in a data center with several servers distributing the verifications among them. If they are using a different digital signature algorithm, we need to check whether it may affect the security of the tokens.

Another possibility is to use a better processor. The Xeon processors may have up to 64 cores, but they seem too expensive for home users. In Brazil, a Xeon processor with 18 cores costs around $ 1,800.

Thus, it does not seem fair to compare Hathor with those projects, since they have very different specifications. While Hathor is being developed to be executed in any home computer, they seem to be developed to be executed in data centers.

As transactions may have multiple inputs, reducing, even more, the number of transactions per second, and Hathor has also to handle many more things than just verifying the digital signatures, I believe that the estimated limit to the number of transactions per second should be around 2,000 and 4,000 for home computers.

Currently, Hathor implementation can handle 300 tps in a single core of the 2.7 GHz Core i7 (I7-8559U) processor. Thus, we aim to reach between 1,200 and 1,500 after distributing the tasks among the four physical cores.

X. Hash rate estimation

A. Introduction

We would like to update the block's weight every new block. This would make the network adjust faster to changes in the network's hash rate.

Let X_i be the number of hashes to solve the ith block. From my thesis, X_i follows a geometric distribution with parameter $p_i = 2^{-w_i}$. Let H be the hash rate (hashes/second), and $T_i = X_i/H$ be the time taken to calculate the X_i hashes. Finally, let θ the target average time between blocks.

Let w_i be the weight of the ith block, and t_i be the time the network took to solve the ith block. The time interval t_i is the difference between the timestamp of the blocks $i+1$ and i. An attacker may try to tamper with a block's timestamp to obtain an advantage when mining.

From the blockchain, we calculate $\{t_1, t_2, \dots, t_n\}$, which are samples of T_i with a known w_i (which means X_i is well defined).

We know that H may change over time. The problems are: how do we calculate the next block's weight? What is the best estimator for the average of H in the last minutes? The optimal number of minutes is part of the problem because it affects the variance of the estimator.

B. Naïve estimator

From the definition we know that $\mathbf{E}[T_i] = 2^{w_i}/H$ and $\mathbf{V}[T_i] = 2^{w_i}(2^{w_i}-1)/H^2$.

Then,

$$\mathbf{E}\left[\sum\nolimits_i T_i\right] = \frac{\sum_i 2^{w_i}}{H}$$

As $\sum_i t_i$ is a sample of $\sum_i T_i$, the naïve estimator is given by the following equation:

$$\hat{H} = \frac{\sum_i 2^{w_i}}{\sum_i t_i}$$

Finally, if we would like to find a new block every θ seconds, the next block's weight should be $w_{n+1} = \log_2(H \cdot \theta)$, since $\mathbf{E}[X_{n+1}] = 1/p_{n+1} = 2^{w_{n+1}} = H \cdot \theta \Rightarrow \mathbf{E}[T] = \mathbf{E}[X]/H = \theta$.

1) *Study case: w_i is constant:* Let's analyse a scenario with w_i constant, i.e., $\forall i \in \{1, 2, \dots, n\}, w_i = w$. This means we will update the weight every n blocks instead of every block.

Using the central limit theorem, when n is big enough, we can use the variance to calculate the confidence interval of $\sum_i t_i/n$. Hence,

$$\frac{\sum_i t_i}{n} \in \frac{2^w}{H} \cdot \left(1 \pm z_\alpha \sqrt{\frac{1-2^{-w}}{n}}\right)$$

Where z_α is the well-known z-score and n must be large enough to the central limit theorem. For a 95% confidence interval, $z_\alpha = 1.96$; for a 99% confidence interval, $z_\alpha = 2.576$.

Let w^* be the optimal weight where the average time between blocks is θ. Then, $2^{w^*} = H \cdot \theta$. Replacing in the confidence interval, we have:

$$\frac{\sum_i t_i}{n} \in \theta \cdot 2^{w-w^*} \cdot \left(1 \pm z_\alpha \sqrt{\frac{1-2^{-w}}{n}}\right) \tag{2}$$

In this scenario, $\hat{H} = (n \cdot 2^w)/\sum_i t_i$. Replacing in Equation 2, we obtain:

$$\hat{H} \in \frac{2^{w^*}}{\theta} \cdot \left(1 \pm z_\alpha \sqrt{\frac{1-2^{-w}}{n}}\right)^{-1} \tag{3}$$

In the real network, w is usually way bigger than 25. Thus, $\sqrt{1-2^{-w}} \approx 1$, and we get:

$$\hat{H} \in \frac{2^{w^*}}{\theta} \cdot \left(1 \pm \frac{z_\alpha}{\sqrt{n}}\right)^{-1} \tag{4}$$

As $2^{w^*}/\theta$ is multiplying the interval, we can analyze the interval percentually, i.e., only the $1 \pm z_\alpha/\sqrt{n}$ part. In Table I we can see some confidence intervals for $\theta = 60$ seconds.

Frequency	**n**	**CI 99%**
Every 30 minutes	30	(0.5510, 5.3938)
Every hour	60	(0.7504, 1.4982)
Every day	1,440	(0.9364, 1.0728)
Every week	10,080	(0.9749, 1.0263)
Every two weeks	20,160	(0.9821, 1.0184)

TABLE I: 99% confidence interval ($z_\alpha = 2.576$) for $\theta = 60$ seconds

One compelling advantage of this estimation method is that it only depends on the difference between the timestamp of the first and last blocks since $\sum_i t_i$ is equal to this difference. So, if an attacker is tampering with the timestamps, but most blocks are mined by honest miners, the attacker may affect the next weight only if their blocks are exactly the first or the last block among the last n blocks. When the attacker's blocks are only in the middle of the chain, they won't affect the next weight at all. Even when the first or the last blocks belong to the attacker, other blocks in between won't, and the timestamp of the blocks must be strictly increasing. This reduces the odds of a successful attack.

2) *Study case: $|w_{i+1} - w_i| \le A$:* In this subsection, we assume that the weight will be updated every block, but limited to maximum change A, i.e., $\forall i \in \{1, 2, \dots, n\}, |w_{i+1} - w_i| \le A$. In this case, the calculation of w_{n+1} will use the distance between the previous n blocks.

Let $\overline{w} = \log_2\left(\sum_i 2^{w_i}\right) - \log_2 n$ be the average weight. Then, after some algebra, we got:

$$\mathbf{E}\left[\sum_i T_i\right] = \frac{n \cdot 2^{\overline{w}}}{H}$$
$$\mathbf{V}\left[\sum_i T_i\right] = \frac{n \cdot 2^{2\overline{w}}}{H^2}\left(\frac{\sum_i 2^{2(w_i-\overline{w})}}{n} - 2^{-\overline{w}}\right)$$

We can prove by induction that $|w_{i+1} - w_i| \leq A \Rightarrow |w_1 - w_n| \leq (n-1) \cdot A$ just applying the triangle inequality ($|w_i - w_j| \leq |w_i - w_k| + |w_k - w_j|$). The equality happens when $w_i = w_1 + (i-1)A$ or $w_i = w_1 - (i-1)A$.

When $w_i = w_1 + (i-1)A$, we have the maximum value of $\overline{w}$. Hence,

$$\overline{w} \leq w_1 + A(n-1) + \log_2\left(1 - 2^{-A\cdot n}\right) - \log_2\left(1 - 2^{-A}\right) - \log_2 n$$

Then,

$$2^{2(w_i-\overline{w})} \leq \left(2^{-2A(n-i)}\right) \cdot n^2 \cdot \frac{\left(1-2^{-A}\right)^2}{\left(1-2^{-A\cdot n}\right)^2}$$

Finally,

$$\sum_i 2^{2(w_i-\overline{w})} \leq n^2 \cdot \frac{\left(1-2^{-2A\cdot n}\right)}{\left(1-2^{-2A}\right)} \cdot \frac{\left(1-2^{-A}\right)^2}{\left(1-2^{-A\cdot n}\right)^2}$$

To simplify the analysis, let's define $\gamma_1(n)$ and $\gamma_2(n)$ as:

$$\gamma_1(n) = \frac{\left(1-2^{-2A\cdot n}\right)}{\left(1-2^{-2A}\right)} \cdot \frac{\left(1-2^{-A}\right)^2}{\left(1-2^{-A\cdot n}\right)^2}$$
$$\gamma_2(n) = 2^{-[w_1+(n-1)A]} \cdot \left(\frac{1-2^{-A}}{1-2^{-A\cdot n}}\right)$$

After applying the central limit theorem and some algebra, we got:

$$\frac{\sum_i t_i}{n} \in \frac{2^{\overline{w}}}{H} \cdot \left(1 \pm z_\alpha \sqrt{\gamma_1(n) - \gamma_2(n)}\right)$$

Let w^* be the optimal weight where the average time between blocks is θ. Then, $2^{w^*} = H \cdot \theta$. Replacing in the confidence interval, we have:

$$\frac{\sum_i t_i}{n} \in \theta \cdot 2^{\overline{w}-w^*} \cdot \left(1 \pm z_\alpha \sqrt{\gamma_1(n) - \gamma_2(n)}\right) \tag{5}$$

In this scenario, $\hat{H} = (n \cdot 2^{\overline{w}}) / \sum_i t_i$. Replacing in Equation 5, we obtain:

$$\hat{H} \in \frac{2^{w^*}}{\theta} \cdot \left(1 \pm z_\alpha \sqrt{\gamma_1(n) - \gamma_2(n)}\right)^{-1} \tag{6}$$

We can apply two approximations: (i) as $\overline{w} \geq \min_i w_i$, and w_i is usually way bigger than 25, $2^{-\overline{w}} \approx 0$; and (ii) notice that $\gamma_1(n)$ is strictly decreasing as n increases, and it converges to a fixed value. In fact, $\gamma_1(n) \approx \left(1-2^{-A}\right)^2 / \left(1-2^{-2A}\right)$ for $n \geq 10$. As we used the central limit theorem, we already have to choose $n \geq 30$.

Applying both approximations, we obtain:

$$\hat{H} \in \frac{2^{w^*}}{\theta} \cdot \left(1 \pm z_\alpha \left(\frac{1-2^{-A}}{\sqrt{1-2^{-2A}}}\right)\right)^{-1} \tag{7}$$

It is interesting that n does not affect the confidence interval, which depends only on the value of A. This confidence interval assumes the worst case, in which the $|w_{i+1} - w_i| = A$.

A	**CI 99%**
0.001	(0.9542, 1.0503)
0.010	(0.8683, 1.1787)
0.100	(0.6759, 1.9210)
0.200	(0.5960, 3.1023)
0.300	(0.5467, 5.8524)
0.400	(0.5112, 22.7623)

TABLE II: 99% confidence interval ($z_\alpha = 2.576$) for $|w_{i+1} - w_i| \leq A$ and $n \geq 10$

Table II shows the confidence interval for some values of A. We can notice that it is very imprecise even for $A = 0.4$. In order to have a small interval, we need to use small values of A. For instance, $A = 0.001$ means that, after the hash rate doubled, the weight will only be fully adjusted after 1,000 blocks.

C. Least squares

There are at least three ways to fit the data with minimum least squares: by t_i, by w_i, or by $\mathbf{E}[X_i] = 2^{w_i}$. From the theoretical model, $T_i = X_i/H$, which means $\mathbf{E}[T_i] = \mathbf{E}[X_i]/H = 2^{w_i}/H$.

1) Fit by t_i: The estimation error is $\epsilon_1^2 = \sum_i (t_i - 2^{w_i}/H)^2$. Solving $d\epsilon_1^2/dH = 0$, we have:

$$\hat{H} = \frac{\sum_i 2^{2w_i}}{\sum_i t_i \cdot 2^{w_i}}$$

Notice that if w is constant, i.e., $w_i = w$, then $\hat{H} = n \cdot 2^w / \sum_i t_i$ which is the same as the Naïve estimator.

2) Fit by $\mathbf{E}[X_i]$: The estimation error is $\epsilon_2^2 = \sum_i (2^{w_i} - H \cdot t_i)^2$.

Solving $d\epsilon_2^2/dH = 0$, we have $\sum_i 2(2^{w_i} - H \cdot t_i)(-t_i) = 0$. Then,

$$\hat{H} = \frac{\sum_i t_i \cdot 2^{w_i}}{\sum_i t_i^2}$$

3) Fit by w_i: The estimation error is $\epsilon_3^2 = \sum_i (w_i - \log_2 t_i - \log_2 H)^2$.

Solving $d\epsilon_3^2/dH = 0$, we have $\sum_i 2(w_i - \log_2 t_i - \log_2 H)(-1/H) = 0$. Then,

$$\log_2 \hat{H} = \frac{\sum_i (w_i - \log_2 t_i)}{n}$$

Removing the $\log_2$, we obtain:

$$\hat{H} = \left(\prod_i t_i \cdot 2^{w_i}\right)^{1/n} = 2^{\overline{w}} \cdot \sqrt[n]{t_1 t_2 \cdots t_n}, \quad \text{where } \overline{w} = \frac{\sum_i w_i}{n}$$

D. Maximum likelihood estimator

As $P(X_i = k) = (1 - p_i)^{k-1} \cdot p_i$, then:

$$\begin{aligned} P(T_i = t_i \mid w_i) &= P(X_i = H \cdot t_i \mid w_i) \\ &= (1 - p_i)^{H \cdot t_i - 1} \cdot p_i \end{aligned}$$

Hence,

$$\mathcal{L} = P(t_1, t_2, \dots, t_n | w_1, w_2, \dots, w_n) = \prod_{i=1}^{n} P(T_i = t_i | w_i)$$

Finally,

$$\log \mathcal{L} = \sum_{i=1}^{n} \left[(H \cdot t_i - 1) \log(1 - 2^{-w_i}) - w_i \log 2 \right]$$

Unfortunately, $\log \mathcal{L}$ does not have any local maximum. So, it seems that this method does not work.

XI. Appendix: Width of the DAG

Problem: Let m people randomly choose A different numbers from the set $[n] = \{1, 2, \dots, n\}$. Let X_i be the chosen number whereas $|X_i| = A$. What is the probabilty of exactly k numbers won't be chosen by anyone, i.e., $P(|[n] - \cup_{i=1}^{m} X_i| = k)$?

Let $P(n, m, k)$ be the probability in question for a fixed A. Then,

Theorem 20.

$$P(n, m, k) = P(n - 1, m, k - 1) \cdot \frac{n}{k} \cdot \left(\frac{n - A}{n} \right)^m$$

Proof. Let #(n, m, k) be the number of ways m people choose A numbers such that $|[n] - \cup_{i=1}^{m} X_i| = k$, and $\#_n$ be the number of ways m people choose A numbers. So,

$$P(n, m, k) = \frac{\#(n, m, k)}{\#_n}$$

Choose an arbitrary number u from $[n]$, which will be one of the k non-chosen numbers. Then, remove u from $[n]$ and let people choose their A numbers such that $k - 1$ number won't be chosen by anyone. Finally, we will have k non-chosen number—one (u) plus the $k - 1$ non-chosen by people. Hence, as people have chosen from $n - 1$ numbers, we have:

$$\#(n, m, k) = \frac{1}{k} \cdot \sum_{i=1}^{n} \#(n-1, m, k-1) = \frac{n}{k} \cdot \#(n-1, m, k-1)$$

The factor $1/k$ is necessary because we are counting the same set of non-chosen numbers k times. Let $\{u_1, u_2, \dots, u_k\}$ be the set of non-chosen numbers. In the sum, we are counting this set exactly k times, one for each u_i removed from $[n]$.

Hence,

$$\begin{aligned} P(n, m, k) &= \frac{\#(n, m, k)}{\#_n} && (8) \\ &= \frac{n}{k} \cdot \frac{\#(n - 1, m, k - 1)}{\#_n} && (9) \\ &= \frac{n}{k} \cdot \frac{\#(n - 1, m, k - 1)}{\#_{n-1}} \cdot \frac{\#_{n-1}}{\#_n} && (10) \\ &= \frac{n}{k} \cdot P(n - 1, m, k - 1) \cdot \frac{\#_{n-1}}{\#_n} && (11) \end{aligned}$$

Finally, $\#_n = \binom{n}{A}^m$, and we have:

$$\frac{\#_{n-1}}{\#_n} = \left[\frac{\binom{n-1}{A}}{\binom{n}{A}} \right]^m = \left(\frac{n - A}{n} \right)^m$$

□

Corollary.

$$P(n, m, k) = P(n - k, m, 0) \cdot \binom{n}{k} \cdot \left[\frac{\binom{n-A}{k}}{\binom{n}{k}} \right]^m$$

Proof. Apply Theorem 20 k times. □

Theorem 21.

$$P(n, m, 0) = \sum_{i=0}^{A} P(n, m - 1, i) \cdot \frac{\binom{n-i}{A-i}}{\binom{n}{A}}$$

Proof. Let's say that $m - 1$ people have already chosen their numbers and exactly i numbers have not been chosen by anyone. Thus, in order to have all numbers chosen at least once, the last person must choose all these i numbers among their A choices. So, applying the law of total probability, we have:

$$P(n, m, 0) = \sum_{i=0}^{A} P(n, m-1, i) \cdot \Pr(m\text{th person chooses all } i \text{ non-chosen}$$

Finally, as the last person must choose all i non-chosen numbers, the last person will choose $A - i$ numbers from the $n - i$ remaining numbers. Hence,

$$\Pr(m\text{th person chooses all } i \text{ non-chosen numbers}) = \frac{\binom{n-i}{A-i}}{\binom{n}{A}}$$

□

Corollary.

$$P(n, m, 0) = \sum_{i=0}^{A} P(n - i, m - 1, 0) \cdot \binom{A}{i} \cdot \left[\frac{\binom{n-A}{i}}{\binom{n}{i}} \right]^{m-1}$$

Therefore, in order to calculate $P(n, m, k)$, we just have to apply Corollary XI and Theorem 21 multiple times, reducing to one of the following boundary cases:

- $P(n, m, 0) = 1$, if $n = A$ and $m \geq 1$.
- $P(n, m, 0) = 0$, if $n = A$ and $m = 0$.
- $P(n, m, k) = 0$, if $n - k > A \cdot m$ or $n - k < A$.

Algorithm 1 Add transaction v to G

```
function AddTransactionToGraph($v$)
    $Z_v \leftarrow \bigcup_{x|(v,x)\in E} Z_x$
    if $v$ has conflicts then
        $Z_v \leftarrow Z_v \cup \{v\}$
    end if
    for $x \in Z_v - \{v\}$ do
        ResolveConflict($x$)
    end for
    if $Z_v = \{v\}$ then
        ResolveConflict($v$)
    end if
end function
```

Algorithm 2 Resolve conflict of v

```
function ResolveConflict($v$)
    if $Z_v \neq \{v\}$ then
        return
    end if
    $A \leftarrow \{x \mid x$ conflicts with $v \wedge Z_x - \{x\} = \emptyset\}$
    if $A = \emptyset$ then
        MarkAsExecuted($v$)
        return
    end if
    $m \leftarrow \max_{y\in A}\{a_y\}$
    $B \leftarrow \{x \in A \mid a_x = m\}$
    if $a_v < m$ then
        return
    end if
    for $b \in B$ do
        MarkAsVoided($b$)
    end for
    if $a_v > m$ then
        MarkAsExecuted($v$)
    end if
end function
```

Algorithm 3 Mark as voided

```
function MarkAsVoided($v$)
    if $v \in Z_v$ then
        return
    end if
    $Z_v \leftarrow \{v\}$
    $A \leftarrow \{x \in V \mid x \rightsquigarrow v\}$
    $B \leftarrow \emptyset$
    for $x \in A$ do
        if $x$ has conflicts then
            if $Z_x = \emptyset$ then
                $B \leftarrow B \cup \{y \mid y$ has a conflict with $x\}$
            end if
            $Z_x \leftarrow Z_x \cup \{x\}$
        end if
        $Z_x \leftarrow Z_x \cup \{v\}$
    end for
    for $x \in B$ do
        ResolveConflict($x$)
    end for
end function
```

Algorithm 4 Mark as executed

```
function MarkAsExecuted($v$)
    if $v \notin Z_v$ then
        return
    end if
    $Z_v \leftarrow Z_v - \{v\}$
    $A \leftarrow \{x \in V \mid x \rightsquigarrow v\}$
    for $x \in A$ do
        $Z_x \leftarrow Z_x - \{v\}$
    end for
    for $x \in A \land Z_x = \{x\}$ do
        ResolveConflict($x$)
    end for
end function
```

Algorithm 5 Add block b to G

```
function AddBlockToGraph($b$)
    $Z_b \leftarrow \bigcup_{(b,x)\in E} Z_x$
    for $x \in Z_b$ do
        $a_x \leftarrow \log_2(2^{a_x} + 2^{w_b})$
    end for
    if $b$ is not connected to the head of the best chain then
        $Z_b \leftarrow Z_b \cup \{b\}$
    end if
    for $x \in Z_b - \{b\}$ do
        ResolveConflict($x$)
    end for
    if $Z_b = \{b\}$ then
        $s_b \leftarrow$ CalculateScore($b$)
        if $s_b < s_{head}$ then
            return
        end if
        UnmarkAsBestChain($b_{head}$, $b$)
        if $s_b > s_{head}$ then
            MarkAsBestChain($b$)
        end if
    end if
end function
```