Fundação Getulio Vargas
Escola de Matemática Aplicada

Athos Cotta Couto

A blockchain-based consensus algorithm for DAG DLTs

Rio de Janeiro
2021

Athos Cotta Couto

A blockchain-based consensus algorithm for DAG DLTs

Dissertation submitted to the Escola de Matemática Aplicada as partial requirementfor obtaining a Master's degree in Mathematical Modeling .

Supervisor: Flavio Codeço Coelho

Rio de Janeiro
2021

Dados Internacionais de Catalogação na Publicação (CIP)
Ficha catalográfica elaborada pelo Sistema de Bibliotecas/FGV

Couto, Athos Cotta
A blockchain-based consensus algorithm for DAG DLTs / Athos Cotta Couto. – 2020.
46 f.

Dissertação (mestrado) -Fundação Getulio Vargas, Escola de Matemática Aplicada.
Orientador: Flavio Codeço Coelho.
Inclui bibliografia.

1. Blockchains (Base de dados). 2. Bitcoin. 3. Sistema de transação (Sistemas de computação). 4. Transferência eletrônica de fundos. I. Coelho, Flávio Codeço, 1969-. II. Fundação Getulio Vargas. Escola de Matemática Aplicada. III. Título.

CDD – 005.75

Elaborada por Márcia Nunes Bacha – CRB-7/4403

ATHOS COTTA COUTO

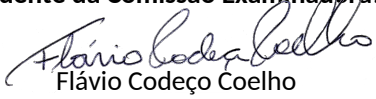
“A BLOCKCHAIN-BASED CONSENSUS ALGORITHM FOR DAG DLTS’’.

Dissertação apresentado(a) ao Curso de Mestrado em Modelagem Matemática do(a) Escola de Matemática Aplicada para obtenção do grau de Mestre(a) em Modelagem Matemática.

Data da defesa: 29/09/20

ASSINATURA DOS MEMBROS DA BANCA EXAMINADORA

Presidente da Comissão Examinadora: Profº **Flávio Codeço Coelho**

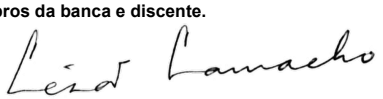


Flávio Codeço Coelho
Orientador

Renato Rocha Souza
Membro

Marcelo Salhab Brogliato
Membro

Nos termos da Lei nº 13.979 de 06/02/20 - DOU nº 27 de 07/02/20 e Portaria MEC nº 544 de 16/06/20 - DOU nº 114 de 17/06/20 que dispõem sobre a suspensão temporária das atividades acadêmicas presenciais e a utilização de recursos tecnológicos face ao COVID-19, as apresentações das defesas de Tese e Dissertação, de forma excepcional, serão realizadas de forma remota e síncrona, incluindo-se nessa modalidade membros da banca e discente.



César Leopoldo Camacho Manco
Diretor

Antonio de Araujo Freitas Junior
Pró-Reitor de Ensino, Pesquisa e Pós-Graduação FGV

Antonio Freitas, PhD
Pró-Reitor de Ensino, Pesquisa e Pós-Graduação
Fundação Getúlio Vargas

Instrução Normativa n° 01/19, de 09/07/19 - Pró-Reitoria FGV
Em caso de participação de Membro(s) da Banca Examinadora de forma não-presencial*, o Presidente da Comissão Examinadora assinará o documento como representante legal, delegado por esta I.N.
*Skype, Videoconferência, Apps de vídeo etc

Acknowledgements

First and foremost, I thank all the teachers and professors that I had the pleasure to have. They have taught me everything that I know and have taken an essential part in making me the person I am today.

I thank Flavio and Renato for all the assistance and willingness to see this through. I thank Marcelo for all the professional mentorship and academic inspiration.

I thank FGV/EMAp and all its staff, in particular Cirlei and Elisângela, for all the patience and support.

I thank my family for encouraging me every step along the way. I thank my parents for enabling me to achieve everything that I have. I thank my wife Jessica for allowing me to accomplish even more every day — while putting up with the whole nights and weekends of dedication. This work is as mine as it is yours.

Lastly, I dedicate this work to the memory of Lygia and Lucia. Your kindness and joy will be greatly missed.

Abstract

Since Bitcoin and the blockchain were proposed in 2008, distributed ledger technologies and cryptocurrencies have spanned several areas of research. One of the biggest challenges of the field is how to scale the distributed system to support a higher transaction throughput, keeping a safe, distributed, and permissionless network.

This work builds on top of the current blockchain and directed acyclic graph distributed ledger technologies literature and proposes a simple, fast and predictable algorithm to reach network consensus. It builds a model to formalize consensus on a DAG DLT and performs experiments to validate the feasibility of the proposed algorithm.
Key-words: distributed ledger technology, DAG, scalability

List of Figures

List of Tables

Contents

1 Introduction

The problem of creating a digital currency has been studied for at least three decades. [Chaum, 1983] introduced the idea of using cryptographic primitives to create a payment system that preserved users' privacy. [Chaum et al., 1988], a following work from the same author, introduced the concept of *electronic cash*.

[Szabo, 1998] was one of the works that expanded the research in the field, proposing a protocol that, not only allowed the creation of a digital currency — known as *Bit Gold*[1], but also made possible the implementation of a secure title database that allowed global rights transference. Szabo stated that money depends on our trust at third parties — who act on their own behalf and are not predictable. He proposed a way to minimize the dependency on these parties by using a protocol that would produce a predictable — and scarce – amount of digital tokens, that could be used as cash.

Bitcoin was introduced at [Nakamoto, 2008] as a peer-to-peer electronic cash system. Nakamoto proposed a fully decentralized system that relied on *proof of work* to build — with a security model that requires minimal trust — an append-only structure that could solve the double-spending problem. Shortly after the publication of Bitcoin's article, a functional implementation of Bitcoin was released by the same author.

[Barber et al., 2012] surveyed several digital cash schemes proposed on the decades before Bitcoin. They list several reasons why Bitcoin was the model that prevailed, the main ones being no central point of trust, economic incentives, and predictable money supply. They argue that while Bitcoin predecessors had some a combination of these properties, none of them had them all. Bitcoin was especially successful because it created incentives for the ones responsible for keeping the network running and secure.

Since Nakamoto's seminal work, the concept of Bitcoin has been separated from the underlying technology that that powers it — the *blockchain*. Research in the field has advanced with several breakthroughs such as Turing-complete *smart contracts* proposed at [Wood, 2014] and *proof of stake*, which was introduced in King and Nadal [2012]. Several distributed systems implemented digital currencies — now mainly referred to as *cryptocurrencies* — using these and other new technologies.

Cryptocurrencies and blockchain have spanned several research areas such

[1]The work was deepened in [Szabo, 2008]

as privacy and fungibility [Van Saberhagen, 2013, Ben-Sasson et al., 2014, Jedusor, 2016, Kosba et al., 2016], scalability [Gencer et al., 2017, Gazi et al., 2018, Croman et al., 2016, Poon and Dryja, 2016], security [Heilman et al., 2015, Atzei et al., 2017], incentives, economics and game theory [Kroll et al., 2013, Johnson et al., 2014] and many others.

Not only has the research among cryptocurrencies and blockchains grown, but the market around it has surpassed the 100 billion USD mark by June 2017 [ElBahrawy et al., 2017] and at the time of writing is valued at over 200 billion USD, according to CoinMarketCap.

1.1 Challenges

The cryptocurrency industry seems promising and is receiving a lot of focus and investment [Dixon and Haun], but it still has a myriad of challenges to solve. There are countless business and usability [Narayanan et al., 2016] issues that prevent mass adoption.

When we look at the technical problems surrounding cryptocurrencies, one of the main challenges is scalability. Bitcoin network supports an average of seven to eight transactions per second [Valdeolmillos et al., 2019]. The digital payment company Visa, as of June 2019, allegedly had the capacity to process over sixty thousand transactions per second [Visa, 2019] — making its capacity almost a thousand times bigger than Bitcoin's.

There has been active research on how to scale cryptocurrencies in several different directions in the last years. Sidechains such as the *Lightning Network* [Poon and Dryja, 2016] use cryptographic primitives and Bitcoin's scripting capabilities to create a off-chain secure payment channels. It allows parties to agree on instant transactions and to unilaterally exit the channel.

The Lightning Network has its limitations. First, it is an interactive protocol, this means that all parties must be online to operate. Second, if two parties are not directly connected in the Lightning Network, they must create a transaction in the Bitcoin network to connect, or find a route of connections in the side network that links them. For a transaction to be executed in that route, funds must be allocated all over the path, this means that if a single node in the path does not have the necessary funds, the transaction will fail — that is why Lightning Network is more suitable for small payments.

A second scaling technique that has been widely researched is *sharding* [Luu et al., 2016, Gencer et al., 2017]. Sharding allows a blockchain to be

broken down in several subchains. A subchain may contain a single service running inside the blockchain, but it also can contain all blockchain traffic from a geographical location — what matters is that it can be used to isolate a part of the blockchain.

When a blockchain is sharded, only a fraction of its nodes — the ones that mine new blocks — need to keep track of the whole blockchain state. Most of the network nodes can sync with only the sub-chains that they have interest in. This reduces the node hardware entry barrier, making it possible for more nodes to join the blockchain and also saves a lot of network bandwidth since most nodes don't need to be kept in sync with the whole blockchain history.

Buterin [2019] discusses sharding implementation on Ethereum and argues that there is a trilemma involved in every blockchain system. Buterin abstractly defines c and n to state the situation. The amount of available computational resources for an average network node is given by c. n is some value that represents the size of the system — e.g. transaction load, number of agents, market cap and the system's state size should all be proportional to n. The trilemma claims that any blockchain system can have at most two out of the following three properties:

1. Decentralization: nodes have at most $O(c)$ resources.
2. Scalability: $O(n) \gg O(c)$.
3. Security: the network is secure against attackers with $O(n)$ resources.

Buterin [2019] also has several considerations and possible flaws of simple sharding strategies. The document history shows that there are discussions about adding sharding to the Ethereum network since 2016 and they still have no release date for a functional implementation. Sharding may be a step towards scaling cryptocurrencies, but they are not mutually exclusive with other solutions and it may incur a lot of the complexity that may slow the system evolution or be a hidden door for bugs or security breaches.

The blockchain technology *per se* may be a bottleneck for cryptocurrencies to scale. Since it can only be updated concurrently by one node in the network at a time, its structure is a natural bottleneck to scale the network.

One solution to the concurrency problem is to allow forks on the chain to be subsequently merged. This approach wields a *directed acyclic graph*, also known as *DAG*, and has been used as base for several cryptocurrency architectures [Popov, 2016, Sompolinsky et al., 2016, Li et al., 2018, Sompolinsky and Zohar, 2020, Churyumov, 2016].

As it happens with sharding or the lightning network approaches, the DAG comes with its own set of upsides and downsides. Even though it can be used to enhance network scalability, it also increases the system complexity.

IOTA, for instance, models its network — called *tangle* — as a stochastic process. To derive properties of the tangle, Popov [2016] makes assumptions about the network evolution that may not hold all the time — especially on attack situations. Both their *tip selection* and consensus algorithms are built on top of these assumptions.

It may not be possible to predict that the network will behave well under the most diverse scenarios — being it heterogeneous load from fair users or burst of loads from attackers. That is why the protocols that dictate how the network behaves should rely as little as possible on assumptions of how the users behave. They should also be as simple as possible to prevent that unpredicted use of the network causes any kind of undesired behavior that can be used as an attack vector.

1.2 Objectives

This work builds on top of the current blockchain and directed acyclic graph distributed ledger technologies literature and aims to provide a simple, fast and predictable algorithm to reach network consensus.

To achieve the objective stated above, this work:

1. Proposes a set of rules for consensus on a DAG-based DLT.
2. Introduces an algorithm that can efficiently and continuously evaluate these rules on an ever growing DAG.
3. Proves that there is only a unique consensus that satisfies the proposed rules.
4. Analyzes the computational complexity of the proposed algorithm.
5. Simulates complex scenarios and analyze the feasibility of the proposed algorithm.

This paper is based on Hathor [Brogliato, 2018] and proposes an alternative set of consensus rules for DAG-based DLTs. As stated above, it also introduces a consensus algorithm that can efficiently determine if a transaction is valid or not, based on the proposed rules. Both the algorithm and rules are designed to better position Hathor to deal with the scalability challenge.

2 Literature Review

In the decade that followed the seminal work of Nakamoto [2008] a lot of research has been done in the field. There have been cryptography advances that allow anonymous transactions and fungible coins. Distributed Turing-complete machines were implemented on peer-to-peer networks to run distributed applications. Different consensus mechanisms allow the creation of energy-efficient blockchains or even reuse of proof-of-work between blockchains.

The work that introduced Bitcoin was the same that marked the origin of the blockchain. The same way that people have disassociated the blockchain technology from Bitcoin, they've realized that blockchain — a distributed, append-only structure — is only one of the ways of implementing a distributed ledger with eventual consensus.

In this section, we go over the different technologies that are used to implement distributed ledgers, also known as *distributed ledger technologies* or *DLT*s. We also introduce the concept of consensus and finish with some example of DLTs that use directed acyclic graphs as their underlying structure and explain their consensus mechanism.

2.1 Distributed Ledger Technologies

A distributed ledger technology is composed of a set of protocols and the underlying data structures that allow peers to communicate over a network to concurrently maintain a ledger and reach eventual consensus about its content.

Bitcoin is an example of a currency implemented over a DLT. A blockchain may be used to implement a DLT, but not all blockchains are DLTs. Permissioned blockchains, for instance, are centralized and thus can't be used to implement a DLT.

2.1.1 Blockchain and Bitcoin

Satoshi Nakamoto introduced Bitcoin on his seminal work Nakamoto [2008]. Even though he did not name it, he also created the concept of Blockchain, the underlying technology used to implement Bitcoin.

Satoshi's breakthrough was not only creating a digital cash system with minimal fees that prevented the double-spending problem. He created an

economic-sound system that gave financial incentives for those responsible for running the network.

According to Antonopoulos [2017] Bitcoin consists of:

- A decentralized peer-to-peer network
- A public ledger
- A set of rules for validating new blocks and currency issuance
- A mechanism for reaching decentralized consensus on the valid blockchain

For our purposes, we can see the network as a black box. Any peer in the network receives data as new blocks, and sends new blocks through it. When a network peer sends a block, we can expect that all peers in the network will receive it, if the data is valid.

The public ledger is the set of blocks that a network peer has received. It is public because the node will share any values on it with peers that aren't synced with it.

The block validation rules are used to check all transactions on the new block, validating if who spends the funds actually owns them and if they are only spending them once. Those rules also allow blocks to emit Bitcoin as a reward for the one who creates it.

Bitcoin's mechanism to create blocks and reach consensus will be detailed at section 2.2.1.

2.1.2 Other relevant cryptocurrencies

Several cryptocurrencies were developed to overcome some of Bitcoin's shortcomings.

Privacy coins were created to provide anonymous transactions and fungibility. van Saberhagen [2013] introduces CryptoNote, the technology on which Monero is based. It uses ring signatures and one time addresses to provide these functionalities. Sasson et al. [2014] takes the advances in zero-knowledge Succinct Non-interactive Arguments of Knowledge, also known as zk-SNARKs, and proposes ZeroCash, which is the technological base of ZCash.

More recently Jedusor [2016] proposed Mimblewimble, a simpler cryptographic scheme based on Pedersen commitments and an iterative protocol that was aimed at reducing the storage and network overhead of Monero and

ZCash. Mimblewimble was implemented in BEAM and Grim cryptocurrencies and is also being integrated into Litecoin.

Wood [2014] introduced Ethereum, a platform for decentralized applications. EOS, NEO, and TRON are other examples of blockchains created to run distributed applications on top of them.

2.2 Consensus

Consensus is a fundamental problem in the study of distributed systems. As defined in an early work by Lamport et al. [1980], the problem concerns how a set of processes, where each one holds a private value and pairs of processes only communicate on a two-way channel, should agree on the same set of values — even when there are faulty processes among them.

Faulty here may indicate that a process or user of the system simply made a mistake. But it also includes the possibility that an attacker is manipulating that user or process. Systems which can withstand such faulty actors and can keep operating properly after them are known as Byzantine fault-tolerant [Lamport et al., 1982].

On a digital currency network, which users operate on an asynchronous fashion, consensus is the mechanism by which all parties agree on which information should be considered true, and which information should be ignored at a given moment.

2.2.1 Proof of Work and Nakamoto's Consensus

Nakamoto [2008] introduced with Bitcoin an architecture for reaching decentralized consensus. It was a novel mechanism because it allowed a permissionless network, where users don't trust each other, to propagate information concurrently and agree on the final result.

At the core of Nakamoto's proposal, lies the concept of proof of work [Dwork and Naor, 1992]. A proof of work is a mathematical problem which is hard to perform — require resources such as processing time — and can be easily verified once completed.

Bitcoin uses a proof of work problem with adjustable difficulty to set the rate at which new blocks of transactions are are mine, or added to the network. Once every 2016 blocks, the difficulty of the problem is adjusted, so the block mining rate remains as close as possible to 10 minutes per block.

Each block has a reference to the previous one. The chain of blocks from the latest block mined to the first block created contains all the information submitted to the network. But blocks may be added concurrently, so many chains exist. The mechanism through which users agree on the right chain is the consensus. And on Bitcoin, the chain chosen is the one with the most proof of work.

This explains how users agree on the data the system has but doesn't explain why users mine blocks. Since proof of work requires computational resources, it costs money. Nakamoto novel architecture also included a monetary incentive miners.

2.3 Directed Acyclic Graph DLTs

2.3.1 Consensus on DAGs

The blockchain may generate several possible valid chains. But, because of Nakamoto consensus, picking the right one is straightforward, and once one does it, there is no conflict on the information of the picked chain.

The same is not necessarily true for directed acyclic graphs. Some DAG DLTs required that every added transaction must not confirm conflicting data, making every subgraph induced from a transaction a valid history. But that isn't always the case and when conflicts are allowed as part of the DAG we must have a mechanism that can help users to agree on the correct ledger history.

In this section we'll go through the main directed acyclic graph cryptocurrencies, describing how they work, focusing on how they reach consensus on the ledger history. Since we're going to focus on Hathor, it will have it own 2.4

2.3.2 IOTA and the Tangle

IOTA is a cryptocurrency designed for the Internet-of-Things industry [Popov, 2016]. It isn't based on the blockchain, it uses a directed acyclic graph of transactions it calls *tangle*. IOTA was made to make micro-payments, so it doesn't have fees. Like Bitcoin, it uses proof-of-work to prevent spam, requiring that valid transactions added to the tangle provide some proof that the transaction has value.

All new transactions must confirm two previous non-conflicting transactions — which may be the same one. By the rules of IOTA, two transactions conflict not only if they use the same tokens, but if they directly or indirectly confirm transactions that use the same tokens. If a vertex confirms a conflicting transaction it risks being ignored by incoming transactions that detect its transgression.

IOTA accepts conflicting transactions in the tangle, but once they happen users must be able to agree on which one should be approved and which one should be orphaned. This is done by running repeatedly the same algorithm that is used to select confirmed transactions.

The algorithm is a Markov Chain Monte Carlo one, which is based on a random walk from the genesis transactions towards the tips. The goal of the algorithm its to prevent lazy tips to get approved. Lazy tips are the ones that approve old transactions, ones that have sufficient approval when the tip was added. To do so it chooses the next step in a way that vertices with higher cumulative weight — the total number of transactions which confirm them — have a substantially higher probability of being chosen than the ones with lower cumulative weight.

The security model of IOTA queries a constant number of transactions being added to the network to make transactions irreversible. To go around this shortcoming, it introduces a coordinator, a centralized component on the network which is responsible for preventing foul play while the new transaction rate allows users to users to double-spend by issuing a conflicting transaction and them issuing several seemingly fair transactions approving it.

There are plans to remove the centralized coordinator [Popov et al., 2020], but as of this writing, the implementation is delayed and some of the required milestones are still being researched.

2.3.3 Byteball

Byteball [Churyumov, 2016] is a decentralized system proposed in 2016 and which is the base of the OByte cryptocurrency. The system is composed of storage units that have data and references to previous storage units. To be accepted by the system a storage unit must be signed and pay a fee that is proportional to the number of bytes that unit store. The fee can be collected by following units which confirm the later one.

Byteball accepts conflicting storage units that don't have a relative order

in the DAG. It has some rules to ensure users can serially create new units without a relative order on the DAG, but it handles double-spending by defining the main chain.

The main chain is picked by choosing any tip of the DAG. Then the best parent selection algorithm is used to select the next unit in the chain. Doing it all the way to the genesis we get a special chain that can be used to establish a total order in the DAG. This order is defined by the height of the first element in the chain which confirms the storage unit, called the main chain index. If after the main chain is defined there are still conflicting storage units with the same main chain index, then their hash is compared to see which one should be confirmed.

Byteball uses special users of the network, also known as witnesses, to define the main chain. They sign newly added transactions — hence the name — and as long as most of the witnesses of the network behave fairly, the network is able to prevent double-spending attempts.

2.4 Hathor

Hathor was introduced in [Brogliato, 2018] as scalable and fully decentralized cryptocurrency. It borrowed concepts from Bitcoin and from IOTA, allowing Hathor to overcome the shortcomings of both networks. The novel architecture is based on a directed acyclic graph in which each network vertex verifies older vertices and has its own proof-of-work verification. Vertices can be transactions or blocks.

Transactions are vertices that move tokens on the network by consuming a non-empty set of previous outputs and generating some number of outputs. Blocks do not move tokens like transactions, but they create a special output that generates tokens. They also verify a previous vertex, but they must confirm one, and only one, a previous block of the network. These sequential block confirmations create a blockchain inside the graph.

Hathor chose to use different mining difficulty adjustment algorithms for blocks and transactions, decoupling both mining processes. This choice came from the insight that transaction mining difficulty can be set to prevent spam, but also should allow Internet of Things devices — which have small processing power — to independently mine small transactions. These requirements are essentially different from blocks mining difficulty that, like Bitcoin, should be set so blocks are emitted in a predefined rate — what is essential for the economical incentive mechanism to be sound.

Since there are two different mining mechanisms, Brogliato argues that Hathor has the potential to be more decentralized than Bitcoin, because even if few agents concentrate most of block mining capability, "[...] their aggregate hash power will not surpass users' aggregate hash power when millions of devices are generating transactions" [Brogliato, 2018, p. 7A]. Compared with current IOTA implementation, this approach also offers advantages, since it does not require a centralized coordinator.

Hathor differs from previous approaches to distributed ledgers that were based on the DAG, because it doesn't use the directed graph as an evolution of the blockchain. It uses the DAG as a ledger and keeps the blockchain as an incentive mechanism, token issuance, and a base for consensus.

2.4.1 Architecture

Hathor architecture [Brogliato, 2019], like most cryptocurrencies, can be described in terms of two different directed acyclic graphs:

- DAG of verifications
- DAG of funds

The DAG of verifications contains the transactions and blocks as vertices and its edges are based on verifications — or confirmations – relations between vertices. The DAG of funds has the same vertices as the DAG of verifications, but it connects vertex u to vertex v if u uses an output from v as input.

The initial state of the verifications DAG, the genesis, contains a single block and two transactions. The state is modified by each subsequent transaction or block which is connected to the graph.

A transaction is considered valid if it follows the rules:

- It spends only unspent outputs.
- It generates no more funds than it spends.
- Confirms at least two other transactions.
- Solves the proof-of-work problem with valid weight.

A block differs from a transaction since it has no inputs and has to solve a different proof-of-work problem. As Bitcoin's, a block must confirm a previous block and can generate outputs that emit a fixed amount of tokens. Hathor also demands that blocks, like transactions, confirm at least two transactions.

Since there are no vertices before the genesis, they are excepted from the confirmation rules.

One particularity of Hathor's architecture is that a transaction is not required to confirm directly or transitively the transactions that generated its inputs. This decouples the DAG of verifications from the DAG of funds. It makes it easier to add new transactions to the graph — since there is no need to look for a transaction that generated the inputs. But it also can add a burden to synchronization — since the partial order defined by the DAG of verifications must be complemented by the one of the DAG of funds.

2.4.2 Consensus

Hathor consensus [Brogliato, 2019] is defined based on nine rules. It formally defines the consensus as a function $f : v \mapsto Z_v$ which maps a vertex to the set of vertices that void it. If $Z_v \neq \emptyset$ then v said to be *voided*. Otherwise if $Z_v = \emptyset$, v is said to be *executed*.

Brogliato defines three kinds of rules, ones that apply to all kinds of vertices, the ones that apply solely to blocks, and ones that are exclusive to transactions. The General rules can be stated as:

- If $v \in V$ is voided, then all u such that $u \rightsquigarrow v$ is also voided.
- If $v \in V$ is executed, than all u such that $v \rightsquigarrow u$ is also executed.
- Blocks or transactions may only void their descendants, $u \in Z_v$ implies that $v \rightsquigarrow u$.

The rules that rule the blockchain can be declared as:

- The head of the best blockchain $b*$ has the highest score among all blocks that are not voided by an ancestor, i.e., $b^* = b \mid s_b \geq s_b^* \wedge Z_b - b = \emptyset$
- All blocks in the best blockchain are executed, and all blocks out of the best blockchain are voided, i.e., $b^* \rightsquigarrow b \Leftrightarrow Z_b = \emptyset$.

- All blocks out of the best blockchain voids themselves, i.e., $b^* \not\rightsquigarrow b \Leftrightarrow b \in Z_b$.

Last, but not least, the rules that apply only to transaction vertices can be defined as:

- If a transaction v has no conflicts, then $v \notin Z_v$
- The transaction with the highest accumulated weight among all conflicting transactions that are not voided by an ancestor is the winner of the conflict. In case of a tie, all conflicting transactions are voided.
- At most one transaction of a conflict may be non-voided.

3 Methodology

To create a solid basis for understanding the proposed algorithm we have performed two different kinds of analysis.

The first one is based on a mathematical model of distributed ledger technologies as graphs. This model lets us describe, formalize, and derive results that give us guarantees over the behavior of the network.

The second one is based on computational simulation. They allow us to understand how the network behaves under complex scenarios that would be complex to obtain strong mathematical results without oversimplifying assumptions.

3.1 Mathematical model

On section 4 we've defined the basic graph models and results that let us treat distributed ledger technologies as acyclic directed graphs. This laid the foundations that allowed us to analyze current consensus algorithms on blockchains and DAGs.

On section 6 we've extended the definitions laid on the section before to allow us to create a novel consensus function on a DAG distributed ledger. With those definitions in hand, we were able to develop an algorithm to compute the consensus function. This algorithm was subsequently analyzed under the optics of computational complexity theory and by the end of the sectioan, we argue why we consider it efficient.

3.2 Simulation

Even tough mathematical models are extremely powerful and let us derive strong guarantees over the behavior of systems, they become difficult to handle when the complexity of these systems grows. That is why we've taken a simulation approach to understand how the proposed algorithm works under different scenarios.

We've followed the steps of [Brogliato, 2018] and chose a event-based design for the simulations. The simulation is composed of several agents who create and propagate transactions through the network, each one with their local state of the DLT.

The simulations are executed to validate that the proposed algorithm works well, making sure it has sensible confirmation times and rates. Since

nodes of the network may use different strategies to add transactions and blocks, testing different approaches to tip selection is paramount to ensure that the algorithm is well behaved.

4 Graphs and distributed ledger technologies

Distributed ledger technologies are, as the name implies, distributed systems. Directed acyclic graph-based DLTs, such as IOTA and Hathor, support concurrent addition of data across the network. This ability to add different bits of data concurrently and asynchronously may generate a network that has sets of data that conflict among themselves.

The DAG that contains the block and transactions may be used to define a partial chronological ordering among the vertices on that graph. This partial order may be used to create strict validation rules — to prevent data that will never be considered valid from being added to the network.

Since the data was added concurrently on possibly different parts of the network, we can't immediately define an absolute ordering between the data. At least not a chronological one. That is why it is important to not only validate data that is added to the DLT, but also have which allow the peers in the network to determine which data should be discarded, or *voided*, in face of conflicts. Data that is not voided is said to be *executed* by the network. The set of rules which control this protocol is called *consensus rules*. These rules yield a function, the *consensus function*, which maps a vertex in the DLT to a boolean value — which indicates if the data in the vertex is executed or voided. Finally, there should also be a *consensus algorithm*, which is a procedure that allows peers in the network to calculate the consensus function.

4.1 DLTs and Consensus

Definition 4.1 *A* distributed ledger technology $\mathcal{D} = \langle G, \mathcal{R}, f_i, f_o \rangle$ *is composed by a directed acyclic graph* $G = (V, E)$*, a set* $\mathcal{R}$ *of rules, or axioms, and two functions* $f_i : V \mapsto \mathcal{P}(\mathcal{T})$ *and* $f_o : V \mapsto \mathcal{P}(\mathcal{T})$*, which map a vertex* v *to its set of inputs* I_v *and its set of outputs* O_v*, respectively.*

The vertices of the graph can represent blocks — as in Bitcoin or Hathor — or transactions — as in IOTA or Hathor. Rules dictate which types of vertices exist, what kind of data they store when they are valid or not. The rules also tell when data in the DLT is conflicting and how to solve the conflict.

While it is possible to formally define inputs, outputs, and tokens, there is no need to do so for this study. But it is worth noting that they are

mathematical objects of the same kind.

To define consensus in the DLT we don't need all elements of the previous definition as they are.

Definition 4.2 *Let $G = (V, E)$ be a DAG and $\mathcal{D} = \langle G, \mathcal{R}, f_i, f_o \rangle$ a DLT. Its* graph of conflicts *is given by $C = (V, E_c)$, where $E_c = \{(u, v) \mid I_u \cap I_v \neq \emptyset\}$. We say that two vertices u, v are in conflict if $(u, v) \in E_c$.*

Note that this last definition narrows the definition of conflict. It states that conflicts happen in pairs, there can never be a group of $n > 2$ vertices where there is conflict among all of them, but there isn't among any group of $n - 1$ vertices. This may seem to be limiting at first, but it makes sense if we consider that conflicts arise when transactions share inputs, which are atomic entities.

Definition 4.2 may also appear to ignore conflicting, or competing, blocks. But if we consider that blocks have special inputs that are bound to their height, for instance, all different blockchains will conflict among themselves. We'll assume this is true from now on.

Definition 4.3 *Let $\langle G, \mathcal{R}, f_i, f_o \rangle$ be a DLT. Its* graph of funds *is given by $F = (V, E_f)$, where $E_f = \{(u, v) \mid I_u \cap O_v \neq \emptyset\}$.*

The graph of funds connects vertices of the DLT which consume the funds to the vertex that generated the funds.

With this in hand, we can define a consensus as:

Definition 4.4 *A* consensus *over a distributed ledger technology $\mathcal{D} = \langle G, \mathcal{R}, f_i, f_o \rangle$ is a function $\kappa : \mathcal{V} \mapsto \mathcal{B}$ such that:*

Condition 1 *For every pair of vertices $(u, v) \in E_c$:*

$$\kappa(u) = 1 \implies \kappa(v) = 0$$

Condition 2 *For every pair of vertices $(u, v) \in E_f$:*

$$\kappa(v) = 0 \implies \kappa(u) = 0$$

When $\kappa(u) = 0$ we say that the vertex is voided, and when $\kappa(u) = 1$ we say that the vertex is executed, which formalizes the same concepts that were introduced at the beginning of the section.

The second condition of the previous definition implies that if a vertex is voided, all other vertices that reach it through the DAG of funds will also be voided. This is an expected result, since an executed transaction on a ledger must only consume outputs of executed transactions.

There is no need to extend the same constraint to the DAG of verifications. Suppose that we do. In networks that have a high transaction rate, a transaction u could be confirmed by several others, such as v, before it gets voided. Once it does, every of those confirming transactions would also be voided.

From the ledger — referring here to the history of transactions — perspective, if u and v are not connected by the graph of funds, there is no reason to void v once u is voided. The confirming transaction v has no intrinsic relation to the funds of u.

Keeping the transitivity of voided transactions only in the graph of funds can also simplify the network operation and stability. It is simpler to operate because one does not have to take in consideration how likely a transaction is to be voided when picking tips to be confirmed. The network gets more stable because the impact that a single transaction can have on the whole network is reduced. By reducing unnecessary side effects of a void transaction we also diminish the surface area of attacks that try to sploit specific network configurations in order to destabilize network consensus — diminishing trust on network consensus or even creating denial of service attacks.

4.2 Total order on the DLT

So far there is nothing that allows us to solve the conflicts we find. The approach Nakamoto took, assigning a proof of work based weight for each block, is one of the keys aspects that make blockchain so unique. We'll follow his steps, and define a Nakamoto distributed ledger technology as:

Definition 4.5 *A distributed ledger technology is a* $\mathcal{N} = \langle G, \mathcal{R}, f_i, f_o \rangle$ *is a* Nakamoto *DLT if it has an associated* weight function $w : G \mapsto \mathbb{R}^+$*, which maps a vertex* u *to a positive real weight.*

Note that the Nakamoto DLT definition is not a specific Bitcoin model.

It represents any DLT that uses an accumulated weight to determine network consensus — just like Nakamoto consensus does .

Definition 4.6 *Given a Nakamoto distributed ledger technology $\mathcal{N}$ and a vertex $u \in \mathcal{N}$, the* forward weight $w_f(u)$ *of u is defined as:*

$$w_f(u) = w(u) + \sum_{u \rightsquigarrow v} w(v)$$

Definition 4.7 *Given a Nakamoto distributed ledger technology $\mathcal{N}$ and a vertex $u \in \mathcal{N}$, the* backward weight $w_f(u)$ *of u is defined as:*

$$w_b(u) = w(u) + \sum_{v \rightsquigarrow u} w(v)$$

Definition 4.8 *Given a Nakamoto distributed ledger technology $\mathcal{N}$ and a vertex $u \in \mathcal{N}$, the* accumulated weight $w_c(u)$ *of u is defined as:*

$$w_c(u) = \begin{cases} w_f(u), & \textit{if } u \textit{ is a block} \\ w_b(u), & \textit{otherwise} \end{cases}$$

This definition may seem confusing at first. When we compare conflicting blocks we want to get the one on the chain with the most work — which means we want the sum of the work done throughout the whole chain. But when we compare two conflicting transactions, we want the one which was "verified the most". That is why we sum the weight of every vertex that verifies this transaction.

Definition 4.9 *The* best block b *is the block such that, for every other block u, we have $w_c(b) \geq w_c(u)$.*

The best block may not be well defined when there are multiple blocks with the same accumulated weight. Since every vertex must have a unique ID in a DLT, we could use this ID as a tiebreaker. But in practice, it doesn't matter because subsequent blocks in the DLT would determine which one should be voided, ignoring any tiebreaker used instants before. Because of this, to keep the theory simple, we'll assume that no two blocks can have the same accumulated weight.

Definition 4.10 *A* best chain*, or* longest chain*, is the chain that has the best block as its head.*

Definition 4.11 *The* height $h(b)$ *of a block* b *is the size of the set of executed blocks reachable by it.*

Definition 4.12 *The* parent block *of a vertex* v *is the block* $b = p(v)$ *on the best chain with smallest height such that* $b \rightsquigarrow v$*.*

Definition 4.13 *The* height $h(t)$ *of a transaction* t *is the height of its parent block. Formally,* $h(t) = h(p(t))$*. If there is no block* b *such that* $b = p(t)$*, then* $h(t) = \infty$*.*

The definitions above allow us to compare two transactions. Specially two conflicting ones. Let u, v be conflicting transactions, if $h(u) < h(v)$ then u was verified by $p(u)$ and $p(v)$ also verifies $p(u)$, which means $p(v)$ verifies u. But $p(u)$ does not verify v because $h(v) > h(p(u))$. v should be the voided transaction because it was the one with less verification.

What if $h(u) = h(v)$? We can still use the accumulated weight to compare transactions. Then we can say that if $w_c(u) > w_c(v)$ then u is executed and v is voided. This makes sense because the transaction with higher accumulated weight would be the transaction with more verifications. But this approach has two drawbacks:

- w_c grows with the network, which can lead to changes on consensus while vertices are added to the network. We'd like to keep these changes as small as possible.
- w_c may be expensive to compute accurately. Accumulated weight calculations on general DAGs may take effort that is proportional to the DAG size.

There is a way to keep the benefits of using a function like w_c, and eliminate most of its downsides: calculate $w_c(u)$ on the subgraph induced by $p(u)$. While this yields a total order with bounded computational complexity, there are also simpler options.

One of the simplest ways to define an absolute order on a directed acyclic graph, which respects the partial order defined by its edges, is to traverse it in post order. When we look at the whole DAG, there is no immediate

start point to do so. But by taking the k-th block as starting point we can define an order among all vertices u such that $h(u) = k$. Since all children of a transaction are stored in an ordered fashion, there is only one possible traversal post-order traversal which respects that order.

Definition 4.14 *Let $\mathcal{D}$ be a DLT. We define $\phi : \mathcal{D} \mapsto \mathbb{I}$, which maps $t \in \mathcal{D}$ to $\phi(t)$, the index of the position the transaction t appears on the post order traversal of the subgraph that contains all vertex of height $h(t)$. If there is no block b such that $b = p(t)$, then $\phi(t) = 0$.*

Definition 4.15 *Let $\mathcal{N}$ be a Nakamoto distributed ledger technology. For $u, v \in \mathcal{N}$, the total order $\mathcal{R}'$ is defined as:*

$$u < v \iff \begin{cases} h(u) < h(v) \\ h(u) = h(v) \text{ and } \phi(u) < \phi(v) \end{cases}$$

5 Proposed Consensus

5.1 Rules

Definition 5.1 (Proposed consensus rules) *Let $\mathcal{D}$ be a Nakamoto DLT and $\mathcal{R}$ be a total order defined over $\mathcal{D}$. We define a function $\kappa_{\mathcal{R}}$, over $\mathcal{D}$ by the rules:*

Rule 0 *Genesis vertices are always executed.*

Rule 1 *A block is executed if, and only if, it is on the best chain.*

Rule 2 *Let t be a transaction. If t uses as input any output from a voided block or transaction, then t must be voided.*

Rule 3 *Let t be a transaction. If $h(t) = \infty$, then t must be voided.*

Rule 4 *Let t_1 and t_2 be conflicting transactions such that $t_1 < t_2$. If no rule above voids t_1, then t_2 must be voided.*

Rule 5 *Let t be a transaction. If no rule above voids t, then t is executed.*

It isn't enough to define a set of rules, we have to prove that there is a function which satisfies those rules. This will be done in section 5.2. Also, the function should be unique, otherwise different users may end up calculating different functions that follow the rules, thus not reaching a consensus.

Theorem 5.1 *For a given $\mathcal{R}$, if $\kappa_{\mathcal{R}}$ exists, it is unique.*

Proof. The proof is by contradiction. Let's suppose that κ' also follows all rules from 5.1 and that $\kappa_{\mathcal{R}} \neq \kappa'$. This means there exists a DLT $\mathcal{D}$ and at least a vertex $v \in \mathcal{D}$ such that $\kappa_{\mathcal{R}}(v) \neq \kappa'(v)$.

Let's suppose v is a block. Then v is either on the best chain or not. Either way, one of the functions is breaking rules 1. Since this cannot happen, v must be a transaction.

Since $\mathcal{D}$ contain a finite set of vertices and $\mathcal{R}$ defines a total order on $\mathcal{D}$, we can use the well-ordering principle to find the smallest v such that $\kappa_{\mathcal{R}}(v) \neq \kappa'(v)$.

Let's suppose that rule 3 voids v, then the function which executes v is breaking it. The other way around also yields a conflict.

Let's suppose that rule 2 voids v, then the function which executes v is breaking it. Then v cannot be voided by rule 2. Let's supposed there is u such that u isn't void, u and v are in conflict and $h^*(u) < h^*(v)$. Then v must be void, which contradicts the result of one of the functions. Then rule 5 applies and v is executed, which cannot happen, because it implies one of the functions didn't follow rule 5.

This concludes the contradiction, which proves that κ^* must be unique. ■

It is also important to demonstrate that the function that satisfies the rules is a consensus as we've defined.

Theorem 5.2 *For a given $\mathcal{R}$, if $\kappa_{\mathcal{R}}$ exists, it is a consensus.*

Proof. Since the rule 2 is equivalent to condition 2 of 4.4, we must only prove that condition 1 holds.

Let's suppose that there are two executed vertices u, v that conflict. Both must be blocks, or both must be transactions.

First, let's suppose both are blocks. Since they are conflicting, they are on different chains. Let's also suppose they break the first condition. This means that both are executed. This would break rule 1, thus cannot happen.

Since they cannot be blocks and break the first condition, they must be transactions to break it. Once again, both must be executed to break the condition. If they have the same priority, then both break rule 3 and both must be voided. If they have different ones, then they break 4.

Thus both can't be transactions and, by contradiction, $\kappa_{\mathcal{R}}$ is a consensus function. ■

5.2 Existence

We haven't so far proved that there exists any function $\kappa_{\mathcal{R}}$ which satisfies the 5.1. Depending on the DAG structure rule 2 may conflict with 4, preventing any function that satisfies both rules to exist.

One way to ensure κ^* exists for the given DLT, is to prevent such DAG configurations to occur. For instance, to prevent the order defined by the graph of funds to conflict with the one defined by $\mathcal{R}'$, we can make sure that the former is a subset of the order defined by the graph of verifications. Since the order defined by the graph of verifications is a subset of $\mathcal{R}'$, by transitivity, there will be no conflicts. From the DLT perspective, this is equivalent to

only accepting transactions which indirectly or directly verify all its inputs — which is an approach very similar to the one Byteball [Churyumov, 2016] takes.

Another alternative, inspired by Hathor [Brogliato, 2018], is to use the G^* to calculate h, p, ϕ for each transaction. G^*, the *DAG of dependencies*, is defined by $G^* = G \cup C$, the union between the DAG of verifications and the DAG of funds. By using G^* to calculate the height, parent, and order of a transaction, we are defining new functions, which can be denoted by h^*, p^*, ϕ^*, respectively. With these functions in hand, we can also redefine $\mathcal{R}'$, which we will denote by $\mathcal{R}^*$, the proposed consensus function.

Theorem 5.3 *There exists a function $\kappa_{\mathcal{R}^*}$ which follows the rules 5.1.*

This statement can be proven by construction, and we will do it on section 6, by introducing an algorithm that calculates. From now on, we will denote $\kappa_{\mathcal{R}^*}$ by κ^*.

6 Consensus Algorithm

The proposed algorithm is broken into four major functions. It has one function to add a block to the DAG, and one to add a transaction. Algorithm 1 is straightforward, it adds a transaction, sets its height and priority, and marks it as void.

Algorithm 1: Adds the transaction t to the DLT $\mathcal{D}$.

```
1  function AddTransaction(t):
2      h*(t) ← ∞
3      φ*(t) ← 0
4      κ*(c) ← 0
```

Algorithm 2, the one which adds blocks to the DAG, is more intricate. When the new block is not the one with the most accumulated weight, it just marks the block as void and returns. But when it changes the best chain, it rolls the best-chain back to the point where the oldest block is an ancestor of the new block. Then we have to add all other ancestors of the new block on the best-chain, wrapping it up by adding the new block itself.

Algorithm 2: Adds the block b to the DLT $\mathcal{D}$.

```
1  function AddBlock(b):
2      if w_c(b) ≤ w_c(BestBlock()):
3          κ*(b) ← 0
4          return
5
6      while b ↛ BestBlock():
7          PopFromBestChain()
8
9      toAdd ← {b} ∪ {b' | IsBlock(b'), b ⇝ b', b' ⇝ BestBlock()}
10     for b' in toAdd from oldest:
11         AddToBestChain(b')
```

Two functions that expand and shrink the best-chain, by adding and removing a block to it. Algorithm 3 describes the former function, while algorithm 4 describes the later.

To append a block b to the best chain, algorithm 3 marks it as executed and visits all transactions t of G^* which are reachable from b such that

$h^*(t) = \infty$ in post-order. Since all other transactions that can void t have already been visited at this point — and are voided or executed —, it can void or execute t based on its conflicts and inputs. While visiting a transaction t, the algorithm also calculates the value of h^*, ϕ^*, and κ^* for t.

Algorithm 3: Appends b to the best chain.

```
1  function AddToBestChain(b):
2      κ*(b) ← 1
3      k ← 1
4      toVisit ← {t | IsTransaction(t), b ⇝ t, h*(t) = ∞}
5      for t in toVisit using post-order from b:
6          h*(t) ← h*(b)
7          φ*(t) ← k
8          k ← k + 1
9          κ*(t) ← 1
10         if t has any void input:
11             κ*(t) ← 0
12         if t has any executed conflict c with h*(c) ≠ ∞:
13             κ*(t) ← 0
```

Algorithm 4 illustrates how to remove the last block b from the best chain. First, it voids the block. Then it visits all transactions t that are reachable from it such that $h^*(b) = h^*(t)$ and voids these transactions, setting $h^*(t) = \infty$ and $\phi^*(t) = 0$. This last step allows other blocks to visit them while being added to the best chain.

Algorithm 4: Pops the head of the best chain.

```
1  function PopFromBestChain():
2      κ*(b) ← 0
3
4      toVisit ← {t | IsTransaction(t), b ⇝ t, h*(t) = h*(b)}
5      for t in toVisit:
6          h*(t) ← ∞
7          φ*(t) ← 0
8          κ*(t) ← 0
```

6.1 Asymptotical Analysis

When analyzing the algorithm, we want to validate if it yields a calculation of κ^* which follows 5.1, thus proving κ^* exists. But we also want to check its asymptotic running time in the worst scenario, and check if its worst-case scenario performance is adequate.

Let $T(\texttt{f})$ denote the running time of the function `f`. Then considering that the values of h^*, ϕ^* and κ^* are stored in a structure that allows constant time access, then $T(\texttt{AddTransaction}) = O(1)$.

By the same reasoning as above, we can argue that for each loop iteration, `PopFromBestChain` does $O(1)$ work. And it does it exactly $|toVisit|$ times. Lets consider that τ is an upper bound to $|toVisit|$, then we can safely say that $T(\texttt{PopFromBestChain}) = O(\tau)$.

Analysis of `AddToBestChain` is a bit more intricate. By the same reasoning used above, the first four operations of the **for** loop can be done in $O(1)$. Checking if a single input of t is voided, can be also done in constant time. Thus, checking if any of them are void in a naive fashion requires time that is proportional to the number of inputs of the transaction, which is in turn proportional to a measure of the size of the transaction, which we will denote by σ_t. To check if t has any executed conflicts is equivalent to check if any inputs of it have been used, which is also proportional to σ_t.

Let's consider σ an upper bound for σ_t and, once again, consider that τ is an upper bound to $|toVisit|$. If we store the graph as an adjacency list, then a post-order traversal on all vertices from $toVisit$ can be done in $O(\tau * \sigma)$.

Finally, if we have to rewind n blocks from the best chain and add back another m when running `AddBlock`, the complexity of running it would be given by $T(\texttt{AddBlock}) = O(n\tau + m\tau\sigma)$. Since n and m may be arbitrarily large, it seems that we don't have a good bound for `AddBlock`.

But instead of looking at the running time by itself, we should compare the cost of calculating `AddBlock` to the cost of mining all the blocks that entered in the new chain, $T(\texttt{MineBlocks})$. This way we are comparing the computational overhead of a chain swap in every node in the network with the cost of mining the new chain. In other words, we're comparing the computational power every node should spend to deal with an attack with the cost of the attacker to launch its attempt. If we consider ω the work to mine a block:

$$\frac{T(\texttt{AddBlock})}{T(\texttt{MineBlocks})} = O\left(\frac{n\tau + m\tau\sigma}{m\omega}\right) = O\left(\frac{n\tau}{m\omega} + \frac{\tau\sigma}{\omega}\right)$$

$\frac{\tau\sigma}{\omega}$ compares a measure of the total size of transactions between a block, to the effort put in creating that block. We can use proof of work to prevent transaction spam, so we can assume that $\tau\sigma$ is bounded. Assuming that blocks have some sort of incentive like Bitcoin's, the measure of the work needed to mine a block is proportional to the whole network hash power.

The only factor that is apparently not manageable here is the $\frac{n}{m}$, since n may be arbitrarily larger than m. To go around it, we can use aggregate analysis and compute the cost of rewinding a block on the call of `AddBlock` which creates that block. Assuming that a 51% is not possible, we can say that there is a constant k such that no block is rewind more than k times, then:

$$\frac{T(\texttt{AddBlock})}{T(\texttt{MineBlocks})} = O\left(\frac{m\tau\sigma + k\tau}{m\omega}\right) = O\left(\frac{\tau(\sigma + \frac{k}{m})}{\omega}\right)$$

Since k is a constant, we may simplify the expression, yielding:

$$\frac{T(\texttt{AddBlock})}{T(\texttt{MineBlocks})} = O\left(\frac{\tau\sigma}{\omega}\right)$$

Which proves that, in the worst-case scenario of the algorithm, the cost to calculate consensus is proportional to the cost of adding a new chain which beats the current one.

6.2 Correctness

To prove the correctness of the algorithm, first, we must prove it works for the smallest DLT possible — which is done by rule 0. Then we can start with a version of the DLT which has a consensus function that follows all the rules in definition 5.1 and simulate the addition of a block and transaction, going through all possible cases of the algorithm. We'll limit ourselves to an informal outline of the proof here.

First note that `AddTransaction` calculates κ^* correctly for a new transaction t, since newly added transactions don't have a parent block, which makes them have $h^*(t) = \infty$ and by rule 3 they must always be voided.

When adding a block b, `AddBlock` first checks if b should be the new best block. If that isn't the case, then b is voided per rule 1. Then if any changes in the best chain are made, the chain is rewind, and $h*$ is recalculated for every transaction that lost or received a parent block. This part still follows rule 3.

All transactions which have a parent are then visited from smallest to highest height, in post-order fashion. This means that if u can void v — either because u is in conflict and is executed or because it creates an input v and is voided — then u is visited before v. When visiting v, we make sure rules 2 and 4 are followed. Then, if the transaction isn't voided by the later rules, rule 5 is applied and the transaction is executed. This shows the procedure calculates k^* for all nodes, making sure all rules are enforced. Since the algorithm is correct, it calculates k^*, demonstrating it exists.

7 Experiments

The algorithm design and analysis show that it performs well, but it imposes on harsh restriction on the DLT: only transactions verified by blocks are executed. To make sure this restriction doesn't get in the way of a well functioning DLT, we create a stochastic process that simulates several agents adding vertices to the network and analyze two variables:

- The rate of orphaned transactions, given by ω.
- The time that it takes for a transaction to be executed, given by δ.

The generation of blocks and transactions is modeled by a Poisson process each. The block process has λ_B as its parameter, while the transaction one has λ_T. If any of them generate a vertex at time t, it is only added to the DLT at the time $t + \epsilon$. This is done to simulate delays in network propagation.

The parent of a block is always picked from the best-chain. If at a given moment, there is more than one best-chain, then one is picked uniformly.

Since agents in the network may choose different strategies to pick the verified transactions by a new vertex, we test three different strategies and compare how their impact on the variables mentioned above.

All tests were performed with $\lambda_B^{-1} = 10$ and $\epsilon = 1$, with λ_T asumming several different values. Each run simulated 1000 units of time, blocks on the first and last 100 units were not analyzed to prevent data from transient states to interfere with the results.

7.1 Weighted chosen tips

With load varying from one transaction per unit of time on figure 7.1 to 100 transactions per unit of time on figure 7.1 we see very encouraging results.

The horizontal axis shows how long it took for a transaction to be first verified by a block as a function of λ_B. On the vertical it displays the density of transactions that take that amount of time to be verified, yielding a histogram. If we look at the plots, we can notice that the average time to confirm a transaction slightly increases when λ_T gets higher. But with 100 transactions per unit of time — or 1000 transactions per block — we see that more than half of the transactions are executed before $2\lambda_B$, most are by $6\lambda_B$ and all by $8\lambda_B$.

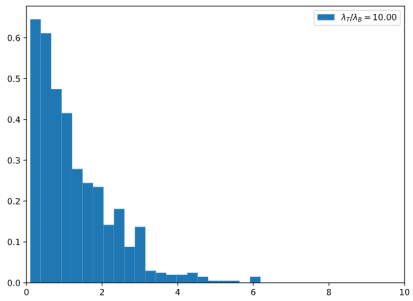


Figure 1: Distribution of time taken to confirm transactions in one run with $\lambda_T = 1$

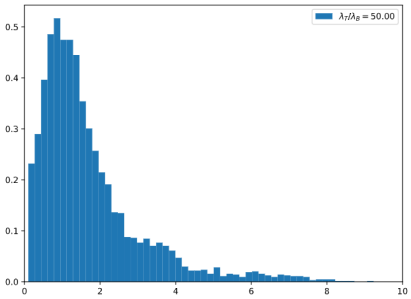


Figure 2: Distribution of time taken to confirm transactions in one run with $\lambda_T = 5$

Figure 7.1 shows the histogram of the average time of a transaction execution in 100 experiment runs. Most of the averages are below $2\lambda_B$ and the total average of the experiments was $1.7136\lambda_B$.

λ_T	ω
1	1.89%
5	4.26%
10	4.56%
20	5.50%
100	5.52%

Table 1: Orphan transaction percentage by λ_T for one experiment run

Table 1 let us know that the rate of orphan transactions is low when

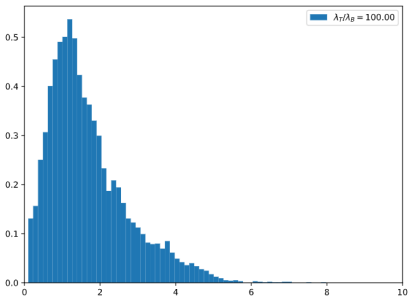


Figure 3: Distribution of time taken to confirm transactions in one run with $\lambda_T = 10$

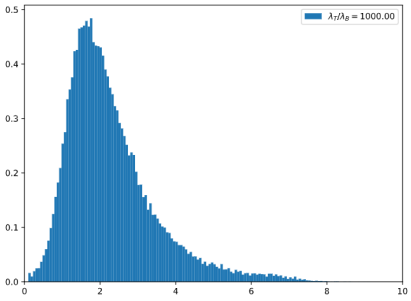


Figure 4: Distribution of time taken to confirm transactions in one run with $\lambda_T = 100$

throughput is low and grows steadily, stabilizing around $5,5\%$. For $\lambda_T = 10$ we've also ran 100 experiments, getting an average ω of $5,39\%$.

7.2 Uniformly chosen tips

When picking tips uniformly, there were no orphan transactions, which means that $\omega = 0$. But when we look at the average time distribution, it was very close to the one from the weighted chosen tips trial.

7.3 Recent tips only

When transaction producers only pick the most recent transactions as tips, we still see the same latency profile. But when we only restrict our search to the most recent λ_T tips, we see a lot of orphan transactions, as table 2 shows

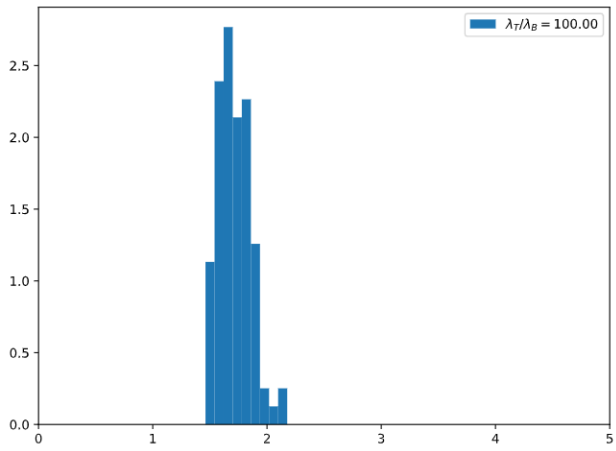


Figure 5: Distribution of average time taken to confirm transactions in 100 runs with $\lambda_T = 10$

us. When we fix $\lambda_T = 100$ and run 100 experiments, we get an average of 18.29% orphan transactions.

λ_T	ω
1	37.88%
5	19.97%
10	18.48%
20	17.11%
100	16.52%

Table 2: Orphan transaction percentage with transaction throughput λ_T, looking at the last λ_T tips, for one experiment run

But if we look at $2\lambda_T$ last tips and repeat the last experiments, the number of orphan transactions goes to 0 as soon as λ_T grows. Repeating 100 runs with $\lambda_T = 100$ results in ω as low as $0,2980\%$.

7.4 Results

Experiments show that even under high load and different tip picking methods, average confirmation time takes less than $2\lambda_B$, getting as high as, but not higher than, $8\lambda_B$ in some cases.

λ_T	ω
1	7.35%
5	0.08%
10	0%
20	0%
100	0%

Table 3: Orphan transaction percentage with transaction throughput λ_T, looking at the last $2\lambda_T$ tips, for one experiment run

When we look at not executed — or orphaned — transactions, we see that the strategy that users pick tips may influence a lot on how many transactions are left behind. If users are too lazy to look for older transactions, the DLT as a whole may leave more than 15% of transactions behind. This means that about 2% of transactions will be left behind after a retry. But I argue that this won't be the case unless the network is under attack, because uneven network delays, users with different strategies, and other incentives — i.e. tips in transactions — may diminish the chances that transactions are left behind.

8 Final Considerations

Cryptocurrencies and distributed ledger technologies are a very recent and vibrant research field, with many challenges. Scalability is one of the biggest ones. Recent research has been proposing novel architectures, such as Hathor, to tackle this problem. This work follows this line, proposing a simple and performant way of using a blockchain to find consensus and a distributed DAG.

We've defined consensus formally and proved that the proposed rules formed a unique consensus, which gives guarantees that the derived algorithm won't generate two possible different consensuses for the same graph. We've also analyzed the algorithm asymptotically, giving guarantees that the amount of effort a node of the network will spend calculating the consensus is bound by the amount of work performed to modify the network.

Finally, we've also performed experiments to validate the feasibility of the presented algorithm, asserting it validates most propagated transactions in a reasonable time frame.

8.1 Further research directions

Further experiments can be conducted to generate better insights on the behavior of the DLT under different situations — i.e. high load, intermittent load and hash-rate, split brains —, especially when malicious users are involved.

One particularly interesting direction is to study the implications of different restrictions — i.e. requiring transactions to verify directly or indirectly all their inputs, like Byteball — on the DAG structure. Another possible research direction would be understanding the trade-offs of allowing transactions and blocks to verify more than two previous transactions.

Another path to follow is through game-theoretic analysis on tip selection strategies, looking for optimal ways of adding a transaction to be verified as fast as possible and as certainly as possible by a block.

It is also possible to study possible difficulty adjustment algorithms for transactions, which would make transaction cheaper on low traffic settings and more expensive on higher traffic, preventing spam.

9 References

Andreas M Antonopoulos. *Mastering bitcoin: Programming the open blockchain.* " O'Reilly Media, Inc.", 2017.

Nicola Atzei, Massimo Bartoletti, and Tiziana Cimoli. A survey of attacks on ethereum smart contracts (sok). In *International conference on principles of security and trust*, pages 164–186. Springer, 2017.

Simon Barber, Xavier Boyen, Elaine Shi, and Ersin Uzun. Bitter to better—how to make bitcoin a better currency. In *International conference on financial cryptography and data security*, pages 399–414. Springer, 2012.

Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. Succinct non-interactive zero knowledge for a von neumann architecture. In *23rd {USENIX} Security Symposium ({USENIX} Security 14)*, pages 781–796, 2014.

Marcelo Salhab Brogliato. *Essays in computational management science*. PhD thesis, FGV EBAPE, 2018. URL https://bibliotecadigital.fgv.br/dspace/handle/10438/24615.

Marcelo Salhab Brogliato. Hathor: An alternative towards a scalable cryptocurrency. Technical report, 2019.

Vitalik et al. Buterin. Sharding faq, 2019. URL https://github.com/ethereum/wiki/wiki/Sharding-FAQ.

David Chaum. Blind signatures for untraceable payments. In *Advances in cryptology*, pages 199–203. Springer, 1983.

David Chaum, Amos Fiat, and Moni Naor. Untraceable electronic cash. In *Conference on the Theory and Application of Cryptography*, pages 319–327. Springer, 1988.

Anton Churyumov. Byteball: A decentralized system for storage and transfer of value. *URL https://byteball. org/Byteball. pdf*, 2016.

CoinMarketCap. Top 100 cryptocurrencies by market capitalization. URL https://coinmarketcap.com/.

Kyle Croman, Christian Decker, Ittay Eyal, Adem Efe Gencer, Ari Juels, Ahmed Kosba, Andrew Miller, Prateek Saxena, Elaine Shi, Emin Gün Sirer, et al. On scaling decentralized blockchains. In *International conference on financial cryptography and data security*, pages 106–125. Springer, 2016.

Chris Dixon and Katie Haun. Crypto fund ii. URL https://a16z.com/2020/04/30/crypto-fund-ii/.

Cynthia Dwork and Moni Naor. Pricing via processing or combatting junk mail. In *Annual International Cryptology Conference*, pages 139–147. Springer, 1992.

Abeer ElBahrawy, Laura Alessandretti, Anne Kandler, Romualdo Pastor-Satorras, and Andrea Baronchelli. Evolutionary dynamics of the cryptocurrency market. *Royal Society open science*, 4(11):170623, 2017.

Peter Gazi, Aggelos Kiayias, and Dionysis Zindros. Proof-of-stake sidechains. *IACR Cryptology ePrint Archive*, 2018:1239, 2018.

Adem Efe Gencer, Robbert van Renesse, and Emin Gün Sirer. Short paper: Service-oriented sharding for blockchains. In *International Conference on Financial Cryptography and Data Security*, pages 393–401. Springer, 2017.

Ethan Heilman, Alison Kendler, Aviv Zohar, and Sharon Goldberg. Eclipse attacks on bitcoin's peer-to-peer network. In *24th {USENIX} Security Symposium ({USENIX} Security 15)*, pages 129–144, 2015.

Tom Elvis Jedusor. Mimblewimble, 2016. URL https://scalingbitcoin.org/papers/mimblewimble.txt.

Benjamin Johnson, Aron Laszka, Jens Grossklags, Marie Vasek, and Tyler Moore. Game-theoretic analysis of ddos attacks against bitcoin mining pools. In *International Conference on Financial Cryptography and Data Security*, pages 72–86. Springer, 2014.

Sunny King and Scott Nadal. Ppcoin: Peer-to-peer crypto-currency with proof-of-stake. *self-published paper, August*, 19, 2012.

Ahmed Kosba, Andrew Miller, Elaine Shi, Zikai Wen, and Charalampos Papamanthou. Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In *2016 IEEE symposium on security and privacy (SP)*, pages 839–858. IEEE, 2016.

Joshua A Kroll, Ian C Davey, and Edward W Felten. The economics of bitcoin mining, or bitcoin in the presence of adversaries. In *Proceedings of WEIS*, volume 2013, page 11, 2013.

Leslie Lamport, Robert Shostak, and Marshall Pease. Reaching agreement in the presence of faults. *Journal of the ACM (JACM)*, 27(2):228–234, 1980.

Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, pages 382–401, July 1982. URL https://www.microsoft.com/en-us/research/publication/byzantine-generals-problem/.

Chenxing Li, Peilun Li, Dong Zhou, Wei Xu, Fan Long, and Andrew Yao. Scaling nakamoto consensus to thousands of transactions per second. *arXiv preprint arXiv:1805.03870*, 2018.

Loi Luu, Viswesh Narayanan, Chaodong Zheng, Kunal Baweja, Seth Gilbert, and Prateek Saxena. A secure sharding protocol for open blockchains. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 17–30, 2016.

Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system.(2008). *Bitcoin.–URL: https://bitcoin. org/bitcoin. pdf*, 2008.

Arvind Narayanan, Joseph Bonneau, Edward Felten, Andrew Miller, and Steven Goldfeder. *Bitcoin and cryptocurrency technologies: a comprehensive introduction*. Princeton University Press, 2016.

Joseph Poon and Thaddeus Dryja. The bitcoin lightning network: Scalable off-chain instant payments, 2016.

Serguei Popov. The tangle. *cit. on*, page 131, 2016.

Serguei Popov, Hans Moog, Darcy Camargo, Angelo Capossele, Vassil Dimitrov, Alon Gal, Andrew Greve, Bartosz Kusmierz, Sebastian Mueller, Andreas Penzkofer, et al. The coordicide, 2020.

Eli Ben Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE Symposium on Security and Privacy*, pages 459–474. IEEE, 2014.

Yonatan Sompolinsky and Aviv Zohar. Phantom, ghostdag, 2020.

Yonatan Sompolinsky, Yoad Lewenberg, and Aviv Zohar. Spectre: A fast and scalable cryptocurrency protocol. *IACR Cryptology ePrint Archive*, 2016:1159, 2016.

Nick Szabo. Secure property titles with owner authority. *https://nakamotoinstitute.org/secure-property-titles/*, 1998.

Nick Szabo. Bit gold. *Website/Blog*, 2008.

Diego Valdeolmillos, Yeray Mezquita, Alfonso González-Briones, Javier Prieto, and Juan Manuel Corchado. Blockchain technology: A review of the current challenges of cryptocurrency. In *International Congress on Blockchain and Applications*, pages 153–160. Springer, 2019.

Nicolas van Saberhagen. Cryptonote v2.0, 2013. URL https://cryptonote.org/whitepaper.pdf.

Nicolas Van Saberhagen. Cryptonote v 2.0, 2013.

Visa. Visa fact sheet, 2019. URL https://usa.visa.com/dam/VCOM/global/about-visa/documents/visa-fact-sheet-july-2019.pdf.

Gavin et al. Wood. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151(2014):1–32, 2014.