Tangle 2.0
Leaderless Nakamoto Consensus on the Heaviest DAG

Sebastian Müller*[†], Andreas Penzkofer[†], Nikita Polyanskii[†], Jonas Theis[†], William Sanders[†], Hans Moog[†]

*Aix Marseille Université, CNRS, Centrale Marseille, I2M - UMR 7373, 13453 Marseille, France
[†]IOTA Foundation, Berlin, Germany

Abstract—We introduce the theoretical foundations of the Tangle 2.0, a probabilistic leaderless consensus protocol based on a directed acyclic graph (DAG) called the Tangle. The Tangle naturally succeeds the blockchain as its next evolutionary step as it offers features suited to establish more efficient and scalable distributed ledger solutions.

Consensus is no longer found in the longest chain but on the heaviest DAG, where PoW is replaced by a stake- or reputation-based weight function. The DAG structure and the underlying Reality-based UTXO Ledger allow parallel validation of transactions without the need for total ordering. Moreover, it enables the removal of the intermediary of miners and validators, allowing a pure two-step process that follows the *propose-vote* paradigm at the node level and not at the validator level.

We propose a framework to analyse liveness and safety under different communication and adversary models. This allows providing impossibility results in some edge cases and in the asynchronous communication model. We provide formal proof of the security of the protocol assuming a common random coin.

Keywords—consensus protocol, leaderless, asynchronous, fault-tolerance, directed acyclic graph, security

1. Introduction

In distributed systems, different events may happen *at the same time*, but participants may perceive them in different orders. In contrast, distributed ledger technologies (DLTs) such as Bitcoin [1] typically use a totally ordered data structure, a blockchain, to record the transactions that define the state of the ledger. This design creates a bottleneck, e.g. a miner or validator, through which each transaction must pass. The creation of blocks can also happen concurrently at different parts of the network, leading to bifurcations of the chain that must be resolved. This is typically done by the longest–chain rule [1] or some variant of the heaviest sub-tree [2]. To guarantee the security of the system, the throughput of the system is artificially suppressed so that each block propagates fully before the next block is created, and very few "orphan blocks" spontaneously split the chain. Another effect that limits scalability is that the transactions are handled in batches. The miners create these batches or blocks of transactions and the blockchain can be seen as a three-step process. In the first step, a client sends a transaction to the block producers, then some block producer proposes the block containing a batch of transactions, and in the last step, validators validate the block.

A more novel approach that addresses the asynchronous setting of the distributed system has been taken by IOTA [3]. This approach eliminates the need for clustered transactions and uses a directed acyclic graph (DAG) (as the underlying data structure) to express simultaneous events. In this model, individual transactions are added to the ledger, and each transaction refers to at least two previous transactions. This property reduces the update of the ledger to two steps: One node proposes a transaction to the ledger and waits for the other nodes to validate it. The removal of the intermediary of miners or validators promises to solve (or at least mitigate) several problems associated with them, e.g. mining races [4], centralisation [5], miner extractable value [6], and negative externalities [7] and allows for a fee-less architecture. However, the parallelism involved in adding new transactions to the ledger means that consensus must be found on a "wider" subgraph than just the longest chain or the heaviest sub-tree.

1.1. Results

Two main problems of Nakamoto's "longest-chain rule" are the severely limited scalability and the lack of parallelisability. The lack of parallelisability results in the underlying communication network requiring strong assumptions about synchronicity. We propose a consensus protocol that works efficiently and fast in an asynchronous model and allows a high degree of parallelisation. This is achieved by replacing the "longest-chain rule" with the "heaviest-DAG rule". As the resulting consensus is not based on a total ordering of the transactions, it enables the transactions to be stream processed. An optimization that becomes more and more relevant in the validation of smart contract updates and optional sharding solutions.

Another disadvantage in blockchains, which is perhaps not so well known, is the need for intermediaries in the form of miners or validators. By enabling leaderless writing access to the ledger we remove this dependency and reduce the system to a dichotomy of fund owners and nodes, where nodes take additional roles akin to validators. Nodes propose new blocks, which contain transactions from fund owners,

and append them to the Tangle. Nodes utilise the append process to validate and vote on previous blocks in a highly efficient implicit voting scheme.

We propose a generalisation of the voting power of nodes in form of a generalised weight function. This generalisation allows for a high level of configurability of our protocol, making it adaptable to the needs and security requirements of the system in which it should be implemented, such as permissionless or permissioned.

We introduce an asynchronous leaderless protocol that employs a weight-based voting scheme on the Tangle. In this scheme, the supporters of transactions, which are the nodes, are tracked through implicit votes. The confirmation status of transactions can be determined using threshold criteria. We provide the algorithms for the various core components. More specifically, we describe how the supporter lists are updated through the implicit voting scheme and how nodes should attach their blocks to the Tangle. We provide theorems for the convergence, as well as the liveness and safety of the system. First, given a random, unpredictable influx of blocks, Theorem 7.1 gives guarantees that the system will converge eventually on a consensus state if an adversary has less than 50% of the weight, however, no safety guarantees are given in this case. Second, we give safety and liveness guarantees by extending the protocol and incorporating the capability to synchronise the nodes at certain intervals with the help of a common coin. The security guarantees for this extended protocol are given in Theorem 10.1. Finally, we provide an overview of simulation results that display the performance of the protocol.

1.2. Structure of the Paper

The document is structured as follows. In Section 1.3 we give an overview of essential aspects relevant to the design of a DLT solution. In Section 1.4 we provide an overview of other recent DAG-based protocols and highlight the differences to our proposal. Section 1.5 provides an overview of used symbols, acronyms and glossary. Section 2 gives an overview of some of the graph-theoretical preliminaries used in this paper. In Section 3 we provide a basic network setting within which the proposed Sybil protection mechanism operates. Section 4 describes the functionality of the Tangle data structure and how it is utilised to confirm blocks. Section 5 introduces an overview of the Reality-based UTXO Ledger, which forms a central component in our approach that helps with tracking the opinions of honest nodes about conflicting transactions. In Section 6, we describe the voting protocol and confirmation of transactions. In Section 7 we define the communication and adversary models and address the liveness and security of the system in Sections 8 and 9. In particular, we show that certain attacks that attempt to create a "metastable" situation, could become problematic under specific circumstances and strong assumptions about the adversary. In Section 10 we provide a solution to this by introducing a synchronization of nodes at larger time intervals. In Section 11, to showcase the performance of the protocol, we provide results from simulation studies. Finally,

we conclude the paper with Section 12, where we describe future research directions.

1.3. Background

Consensus protocols in general and even DLTs, in particular, are such a large research area that we have to refer to some review articles for a more detailed introduction, e.g. [8]–[10]. Although a consensus protocol depends on many different aspects, we focus, in the remaining part of the introduction, on those that are most relevant for the design choices of our proposed protocol.

Ledger Model. Distributed ledgers (DLs) generally arrive in two flavours of balance keeping: an account-based model, where funds are directly associated with the account of a user, such as is the case with Ethereum [11]; and an unspent transaction output (UTXO) model, where tokens are linked to a so-called output, and users own the keys to the output, as is the case with Bitcoin [1] and many of its derivatives, as well as Cardano [12], Avalanche [13], and IOTA [14]. As an important observation in the latter case, the UTXOs form a DAG themselves. A total ordering of the transactions is unnecessary for many use cases and situations, as most of them are parallelisable. However, the append-only nature of the UTXO ledger hinders this advantage of parallelisation in the presence of conflicting transactions. In [15] we propose an augmented UTXO ledger model that optimistically updates the ledger and tracks the dependencies of the possible conflicts. We construct a consensus protocol that utilises this ledger model to enable fast and parallelisable conflict resolution.

The Tangle and Partial Order. The Tangle is the DAG that stores all transactions of the distributed ledger (DL). Every DAG induces a partial order on the set of vertices, the collection of transactions in our setting. This property contrasts with a blockchain where a total order of transactions is established. As in systems with crash failures, atomic broadcast and consensus are equivalent problems, see [16], the partial order of the DAG induces additional "difficulties" in the consensus protocol. More precisely, there have been serious limitations concerning the security of a DAG-based DLT. In the original proposal of the Tangle, [3], the longest chain rule was replaced by the "heaviest sub-graph", i.e. the sub-DAG containing the most transactions. However, it turned out that this design is vulnerable to various types of attacks and would rely too much on the Proof-of-Work necessary to issue a transaction, e.g. [17]. Another critical element of the design that is common to many other DAG-based proposals is that it suffers a liveness problem. Honest transactions that refer to transactions that turn out to be malicious in the future can not be added to the ledger state. The protocol we propose in this paper solves the security problems by relying on a weight function for nodes and by using the Reality-based Ledger. It also treats the problems of liveness by separating transactions from their containers,

which are blocks,[1] and by applying a new block referencing scheme. In particular, this batch-less architecture enables a stream process-oriented design of the DLT.

Sybil Protection. Sybil protection plays a crucial role in a "permissionless environment" where everyone can participate. By leveraging Proof-of-Work (PoW), Bitcoin's Nakamoto consensus was the first to achieve consensus in such an open environment. As PoW leads to enormous energy waste and many negative externalities, a lot of effort has been put into proposing more sustainable alternatives. The most prominent of them is called Proof-of-Stake (PoS), where the validators' voting power is proportional to their stake (i.e. in terms of the underlying cryptocurrency) in the system.

The Sybil protection used in this paper is based on node identities. We describe it generically as a function of a scarce resource or an abstract reputation function. This function, called *weight* assigns every node identity a positive number. For example, this weight can correspond to an amount of staked tokens, delegated tokens, or the "mana" described in [14]. We want to note that the weight does not have to be connected to the underlying token but can be replaced by any other "weight" serving as a good Sybil protection. In particular, our framework can also be used in a permissioned setting, where only the pre-defined validators would have a positive weight and can apply to the situation with dynamic committee selections.

Nakamoto Consensus. Distributed consensus allows participants to agree on a constantly growing log of transactions. It has been an important research topic in recent decades, and its importance in computer science has never been disputed. There are many ways to categorize consensus protocols. For instance, there are the classical landmark results on PAXOS and BFTs, and the newer *Nakamoto* type consensus mechanisms.

We understand as Nakamoto consensus the rule to select the longest sub-chain, e.g. see [10], and as a variant also the heaviest weighted sub-chain. We extend this concept to the heaviest sub-DAG. More precisely we consider, a Nakamoto blockchain consensus to follow the *propose-vote* paradigm and that it can be described as follows. The time is divided into epochs, and for each epoch, there is an "elected" leader. This leader batches transactions into a new block and proposes this block. Then the other participants vote on the proposed block, e.g. by extending the chain to which the proposed block is attached. Once the number of votes reaches a certain threshold, the proposed block is considered part of the ledger. The specific definition of the various elements mentioned above may vary and lead to different variants of the Nakamoto Consensus. To some extent, the above paradigm reduces to the necessity to agree on a unique leader in each epoch. Once the participants have a consensus on the leader, the linearity of the blockchain

implies consensus on the ledger state. However, the fact that only a leader can advance the ledger state creates an obvious bottleneck with well-known performance limitations. In our proposal, we remove the role of the "leader" entirely and allow the participants to propose their blocks and the contained transactions concurrently. Once a block is proposed, all participants can vote and participate in the consensus finding. The weight of the vote is proportional to the *weight* of the node, introduced above, such that the protocol adapts to different weight distributions. The protocol is also classified as a non-binary consensus protocol since it can decide on several transactions simultaneously and is an ever-ongoing voting procedure forming a progressively-growing history.[2] It also relates to a probabilistic consensus in the sense that the more supporting nodes a transaction accumulated the more likely it is that this transaction is eventually confirmed and added to the ledger.

Voting. In our non-linear architecture, each new block references at least two existing blocks. This results in a DAG structure as mentioned above. As with a blockchain, a new block not only votes on its direct references but also on its past cone. Although this is an efficient voting scheme, there is the problem of orphanage or liveness. If a block contains an invalid block in its past cone, it can no longer be voted for and, thus, the contained transaction cannot be included in the ledger. We solve this problem by introducing two different references. The first reference is to the Tangle structure and the second is to the DAG structure originating from the UTXO ledger. The last reference allows voting for transactions that were originally orphaned and also to change previously issued votes. Eventually, both types of votes accumulate in a voting weight, which we call the Approval Weight (AW). The higher this AW the higher the probability that the transaction is eventually included in the ledger. We refer to Figure 1 for an example of the voting mechanism.

Generally, the voting mechanism can be applied to any DAG-based data structure with an append process that allows for referencing previous blocks. It requires three main ingredients: the first essential ingredient is a reference scheme that efficiently casts and propagates votes. The second necessary ingredient is the construction of a generalised invariant data structure that allows conflicts to coexist (see Section 5). This feature allows to treat transactions "optimistically"; every new incoming transaction is considered "honest" unless it conflicts with another transaction. Consequently, nodes may start to build on top of every new transaction, even though this transaction may turn out to be conflicting. The third ingredient is a voting mechanism, dubbed On Tangle Voting (OTV), that efficiently votes on a possible unbounded number of transactions simultaneously. The efficiency is achieved by maintaining a low block overhead since votes of other nodes can be piggy-backed

---

1. Unlike many blockchain protocols, we require each block to contain precisely one transaction. However, in principle, the protocol can be adapted such that blocks contain more than one transaction.

2. Both the ledger DAG structure and the Tangle are technically transient data structures since they can be pruned in theory. Thus, the voting is also transient knowledge. However, for simplicity, we assume that both data structures are not pruned.
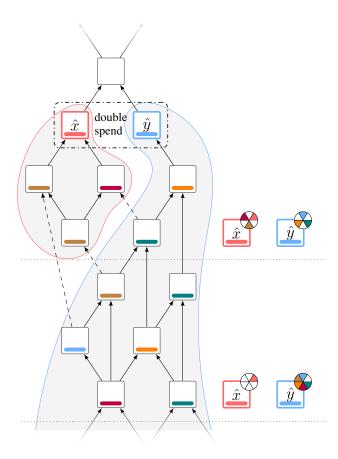
Figure 1: The Tangle is utilised as a voting layer for nodes to reach a consensus about the outcome of a conflict. Nodes agree on the winner between conflicting transactions $\hat{x}$ and $\hat{y}$ using a leaderless protocol. Different colours represent signatures of different nodes. The number of supporting nodes, shown on the right, increases for transaction $\hat{y}$ with time. The dashed references are so-called transaction references and allow to "rescue" transactions that voted for the "losing part".

through the implicit voting mechanism. Also in contrast to classical Byzantine fault tolerance, nodes don't have to be monitored for activity since the issuance of transactions (casting of votes) is a clear sign of being functional.

Security. Since the beginning of research on consensus protocols, the concept of security has been at the centre of attention. Any consensus protocol aims to reach consensus on a data. Some of the participants may be faulty or even active in preventing a consensus, and one is interested in the conditions under which consensus can be achieved.

The security of a propose-vote consensus protocol is usually divided into two points; liveness and safety. Liveness means that any correct transaction is finally accepted by all honest participants, and safety means that all participants finally agree on the same set of transactions. The question

of whether a given consensus protocol fulfils these properties depends largely on the model assumptions. Roughly, these can be divided into the communication model and the attacker model.

In the most restrictive communication model, the synchronous model, many different solutions are known since the landmark result [18]. However, this is not the case under the most general communication model, the asynchronous model, which does not assume any bounds on the transmission delay of block; commonly denoted by Δ. One of the most famous results on consensus protocols is the FLP impossibility result [19] stating that in an asynchronous communication model, a single faulty participant can hinder the consensus finding. As the FLP impossibility result relies on specific configurations of block delays, many practitioners argued that it does not apply to real-world implementations as these particular situations are very unlikely to occur. In between these two extreme communications models, several intermediary models have been proposed, and many positive results have been obtained under stronger assumptions on the network delay, e.g. the partially synchronous model [20], the timed asynchronous model [21], and the asynchronous model with failure detection [16].

Besides the communication model, the adversary model plays an important role, especially in the security analysis of Nakamoto protocols. The protocol's security is commonly expressed in the amount of scarce resources, e.g. energy or computing power, that is necessary to attack the protocol and revert already confirmed transactions. Nakamoto [1] analyzed this property by considering a specific attack, the so-called private-double spend attack. Note that here the classic communication model is the partial synchronous one. Over the last decade, a pertinent research question was the search for worst-case attacker strategies and the identification of the security threshold in terms of the percentage of the scarce resource controlled by an adversary. Tight consistency bounds were recently given in [22] and [23] for several classes of longest-chain type protocols. While these security thresholds do hold in the partial synchronous situation, they fail in the asynchronous setting, e.g. [24].

There is also a line of research that studies how an attacker can compensate its lower weight with more influence on the communication level. The most prominent of such an attack is the balance attack, [25], which consists of delaying network communications between multiple subgroups of nodes with balanced mining power.

This discussion is of particular interest to us because we propose a framework for modelling the communication level and adversarial level jointly. Unsurprisingly, we obtain impossibility results in the asynchronous communication model. Still, under further synchronicity assumptions, we prove that the protocol guarantees liveness and safety (with a very high probability) if the adversarial weight does not exceed certain thresholds. The obtained security bounds are established for any possible attack strategy and are configurable by the protocol.

The situations that lead to the impossibility results in the asynchronous model are frequently considered irrelevant

for practicable purposes, e.g. [26], [27]. The argument for this is that in real-world applications, the randomness in the block delays is so great that the particular situation cannot occur. While we partly agree with this reasoning concerning our OTV, we added a second synchronicity level to our core voting protocol to obtain a rigorous security threshold. For this reason, we see our consensus protocol as a two-layer solution. The first layer works in an asynchronous setting and allows fast and secure confirmation under normal network conditions. The second layer is based on an optional synchronization of the nodes that allows consensus finding under worst-case scenarios. The synchronized level relies on a decentralised random beacon or common coin that makes the protocol robust against attacks similar to the balanced attack described above. Randomization of consensus protocols to circumvent the impossibility results are known since [28], which introduces local randomness. A common coin was introduced [29] and is used in several approaches to increase the security in the asynchronous setting.

Performance. Defining a measure for the efficiency of a consensus protocol is not an easy task since it relies on many different aspects. Natural choices are the number of blocks sent between the participants and, in synchronous models, the number of communication steps. In DLTs, common measures are the number of transactions per second and the time to confirmation. As our protocol uses implicit voting and no direct blocks are exchanged between the nodes, it is optimal in block complexity (if votes are cast through blocks that would have been sent anyway). We present estimates for the time to confirmation and show their dependence on the distribution of the weights. We do not evaluate quantitative performance measures such as throughput and energy consumption in this work. This type of study will be addressed in follow up research.

A common misunderstanding is that asynchronous consensus protocols are not appropriate for time-critical applications [26]. The fallacy is that synchronous protocols assume strong synchronicity assumptions; however, the security is harmed once these assumptions are not satisfied. We argue that it is even the converse and that asynchronous protocols might be better suited for time-critical applications. Under a good communication situation, transactions are approved much faster than in synchronous models based on network delay estimations with an essential security margin.

One main drawback of the leader-based architecture of blockchains is its lack of scalability capability. To make this more precise, let Δ be the network latency, λ the block issuing rate, and q the weight of the adversary. Then, following [2], [22], the condition for the security of the protocol is expressed as

$$q < \frac{1-q}{1+(1-q)\lambda\Delta}. \tag{1}$$

For the design of a system that should support resilience against a maximum adversary weight q, this equation informs about the bound on the maximum rate at which blocks

can be issued safely. A safety violation can occur, for example, if there is disagreement about the recent leader. These disagreements can be caused by blocks being produced in parallel [30] or due to certain attack scenarios [31], [32]. As a consequence re-organisations of the blockchain may occur, in particular for DLTs, where the block production rate is high [33].

In our case there is no theoretical upper limit for the throughput of the protocol in this paper; however, the limits of scalability of our protocol still need to be investigated in future work.

1.4. Related Work on DAG-Based Protocols

We already mentioned various related works in the general introduction. This section focuses on the general architecture and mention previous proposals that use DAGs in the underlying data structures. Blockchain-based protocols rely on a chain or "linearisation" of blocks that create a total order of transactions. These blocks have three purposes: *leader election*, *data transmission* and *voting* for ancestor blocks through the chain structure, see [34]. Each of these aspects can be, individually or combined, addressed through a DAG-based component. The various proposals differ in how and which of these components are replaced by a DAG.

The most common approach is to use a DAG structure for the data transmission. This is the most natural approach since if blocks are created at a high rate compared to their propagation time, many competing or even conflicting blocks are created, leading to frequent bifurcation points of the chain. As this results in a performance loss, a natural proposal is to include not only the "main chain" but also bifurcations using additional references, e.g. [2], [35]–[38].

Protocols can also achieve a higher degree of parallelisation of the data transmission or writing access if all participants can write and propose blocks. This concurrent writing access removes considerably performance limitations of traditional blockchains. In blockchains where only a tiny proportion of participants can write to the ledger, and these participants are randomly chosen, e.g. by PoW or PoS, participants need to communicate the set of pending transactions to all their peers. This *memory pool* is a considerable performance limitation as nodes must broadcast transactions twice. Several interesting proposals allow participants to add concurrent blocks to the ledger and to construct a *distributed memory pool* in the form of a DAG. In the following, we give two approaches that differ in how consensus is achieved and in the underlying Sybil protection. More specifically the first utilises a permissioned setting, while the second employs a permissionless setting.

In the permissioned setting there is the following interesting line of research. The aim is to construct an atomic broadcast protocol based on a combined encoding of the data transmission history and voting on "leader blocks". Such protocols allow the network participants to reach a consensus on a total ordering of the received transactions, and this linearised output forms the ledger. The most robust protocols achieve Byzantine fault tolerance in asynchronous

settings and reach optimal communication complexity, see Honeybadger [39] and [40]. Improvements are proposed, for example, in Hashgraph [41] and Aleph [42] and more recently in Narwhal [43] based on the encoding of the "communication history" in the form of a DAG. These protocols remove the bottleneck of data dissemination of the classical Nakamoto consensus by decoupling the data dissemination from the consensus finding. Promising improvements for the consensus finding on top of the DAG-based memory pool were recently made in DAG Rider [44] and Bullshark [45]. We also want to mention [46] that analyses and discusses this kind of protocol from a more abstract and general point of view.

There is a common point with our approach to mention here. A DAG structure serves as a "testimony" of the communication among the nodes, and new blocks are used for (implicit) voting on previous blocks. In other words, the DAG is used for the two purposes of data transmission and voting. However, voting is done only over so-called "anchor blocks", leading to an *a posteriori* leader election and total ordering of the transactions. Furthermore, and as mentioned above, these DAG-based broadcast protocols are designed for permissioned networks, which leads to similar safety-liveness properties to standard BFT protocols. A difference is, thus, that our protocol is designed for an asynchronous network environment and is not round-based as these proposals above.

In the permissionless setting, another route is taken by Prism [34]. This approach explicitly decomposes the three purposes of blocks into three types: proposer blocks, transaction blocks and voter blocks. Having separate transaction blocks allows participants to issue transactions and removes the need for a memory pool. The three types of blocks form a structured DAG that allows a very efficient way to vote on "leader blocks" that eventually give consensus via total ordering. Our approach is orthogonal in that we do not distinguish between different kinds of blocks but that the underlying DAG delivers consensus without an additional tool. In an implementation [47] of Prism, another DAG was used to increase the performance of the execution of the transaction. More precisely, [47] used a scoreboarding technique to execute the (totally) ordered UTXO transactions in parallel. In our approach, we actively construct a DAG, called the Ledger DAG, that encodes the dependencies of the transactions. This DAG is created before reaching consensus and allows tracking dependencies between pending or conflicting transactions. It was demonstrated in [48] that Prism can also support smart contract platforms and that in their implementation, the bottleneck is no longer the consensus but the execution of the smart contracts.

The main difference of our proposal to all the aforementioned protocols is that consensus is found on the heaviest DAG without the need for a "linearisation" using any leader selection. This reduces the purposes of blocks to data transmission and voting.

We want to mention another class of DAG-based and leaderless consensus protocols. However, it is conceptually different from the proposals above and our proposal. In this kind of protocol, e.g. [13], [49], the voting is performed via direct queries between the peers and hence necessities an additional communication layer. A DAG structure is used in Avalanche [13] to "transitively" vote on several blocks at once. We note, however, that the authors of [13] fail to analyze their proposed protocol properly, and the question of whether it has the desired properties remains unclear, e.g. [50, Section 2.3].

Finally, let us note that the above is only a selection of previous work on DAG-based DLTs and refer the reader to [10] for a more detailed summary.

1.5. List of Acronyms and Symbols

For the reader's convenience, in this section, we summarize important notations and acronyms that are used throughout the paper. Furthermore, in Appendix C we provide a glossary of the terms in use in this paper.

Symbols

Set Symbols

$\mathcal{B}$	set of branches
$\mathcal{C}$	set of conflicts
$\mathcal{N}$	set of nodes in network
$\mathcal{L}$	ledger or set of transactions
$\mathcal{T}$	set of blocks

DAGs

$D_\mathcal{L}$	Ledger DAG
$D_\mathcal{T}$	Tangle DAG
$D_\mathcal{V}$	Voting DAG
$\text{child}_V(x)$	set of children of vertex x in DAG $D=(V,E)$
$\text{cone}_V^{(f)}(x)$	future cone of vertex x in DAG $D=(V,E)$
$\text{cone}_V^{(p)}(x)$	past cone of vertex x in DAG $D=(V,E)$
$D=(V,E)$	directed acyclic graph (DAG) with vertex set V and edge set E
ρ	genesis or vertex with out-degree zero
$\max_V(S)$	set of maximal elements in set S (maximal according to DAG $D=(V,E)$)
$\min_V(S)$	set of minimal elements in set S (minimal according to DAG $D=(V,E)$)
$N_V(x)$	set of neighbours of a vertex x in graph $G=(V,E)$
$\leq_V$	partial order on set V (usually induced by a given DAG $D=(V,E)$)
$\text{par}_V(x)$	set of parent of vertex x in DAG $D=(V,E)$
$\text{sprt}_V(x)$	supporters of x in DAG $D=(V,E)$

Time Symbols

$\tau_f(\cdot)$	time to confirmation defined on $\mathcal{T}$
$\tau_{cf}(\cdot)$	confluence time defined on $\mathcal{T}$
$\tau_s(\cdot)$	solidification time defined on $\mathcal{T}$

Weight Functions

$\mathbf{w}(\cdot)$	weight function defined on $\mathcal{N}$
$\mathbf{AW}(\cdot)$	Approval Weight defined on $\mathcal{L}$
$\mathbf{WW}(\cdot)$	Witness Weight defined on $\mathcal{T}$

Acronyms

AW   Approval Weight
dRNG   Distributed Random Number Generator
DAG   Directed Acyclic Graph
DLT   Distributed Ledger Technology
OTV   On Tangle Voting
P2P   Peer-to-Peer
PoW   Proof-of-Work
PoVP   Proof-of-Voting-Power
TSA   Tip Selection Algorithm
TTC   Time to Confirmation
UTXO   Unspent Transaction Output
WW   Witness Weight

Graph structures

We employ several graph structures as a base for the consensus protocol. Table 1 gives an overview of the utilised graphs.

DAGs	Vertices	Edges
Tangle	blocks	references
Ledger DAG	transactions	spending relations
Voting DAG	blocks, transactions	voting references

TABLE 1: Overview of DAGs

2. Graph Theoretical Preliminaries

In this section, we summarize basic graph theoretical notations that are used in the remaining part of the paper.

The set of integers between 1 and m is denoted by $[m]$. A *graph* G is a pair (V, E), where V denotes the set of vertices and E denotes the set of edges. A graph is called *directed* if every edge has its direction, e.g. for an edge (u, v), the direction goes from u to v.

Definition 2.1 (DAG). A *directed acyclic graph (DAG)* is a directed graph with no directed cycles, i.e. by following the directions of edges, we never form a closed loop.

A vertex v in a graph $G = (V, E)$ is called *adjacent* to a vertex u if $(u, v) \in E$. An edge $e \in E$ is said to be adjacent to a vertex $v \in V$ if e contains v. The *out-degree* and *in-degree* of a vertex v in a directed graph $G = (V, E)$ is the number of adjacent edges of the form (v, u) and, respectively, (u, v). A vertex in a graph is called *isolated* if there is no edge adjacent to it.

Definition 2.2 (Neighbours in a graph). Let $G = (V, E)$ be a graph. For a vertex $v \in V$, define the *set of neighbours* (or G-neighbours), written as $N_V(v)$[3], to be the vertices adjacent to v.

Definition 2.3 (Parents, children and leaves in a DAG). Let $D = (V, E)$ be a DAG. For a vertex $v \in V$, define the

---

3. In the remainder of the paper, we will often identify the graph with its vertex set since for a given set of vertices V, we will have only one DAG $D = (V, E)$. Thereby, the set of neighbours $N_V(v)$ and other concepts that use V as a subscript will be clear from the context.

---

set of *parents*, written as $\mathrm{par}_V(v)$, to be the set of vertices $u \in V$ such that $(v, u) \in E$. Similarly, we define the set of *children*, written as $\mathrm{child}_V(v)$, to be the set of vertices $u \in V$ such that $(u, v) \in E$. A vertex $v \in V$ with in-degree zero is called a *leaf*.

Definition 2.4 (Partial order induced by a DAG). Let $D = (V, E)$ be a DAG. We write $u \leq_V v$ for some $u, v \in V$ if and only if there exists a directed path from u to v, i.e. there are some vertices $w_0 = u, w_1, \ldots, w_{s-1}, w_s = v$ such that $(w_{i-1}, w_i) \in E$ for all $i \in [s]$. Furthermore, we note $u <_V v$ if $u \leq_V v$ and $u \neq v$.

Note that there could be different DAGs producing the same partial order. The DAG with the fewest number of edges that gives the partial order $\leq_V$ is usually called the *transitive reduction* of D or the *Hasse diagram* of $\leq_V$.

Definition 2.5 (Minimal subDAG induced by a set of vertices). Let $D = (V, E)$ be a DAG. For a subset of vertices $S \subseteq V$, we define the *minimal subDAG* of D induced by S to be the DAG $D' = (V', E')$ whose vertex set is $V' = S$ and there is an edge $(v, u) \in E'$ if and only if $u, v \in S$, $v <_V u$ and there is no $w \in S \setminus \{u, v\}$ such that $v <_V w <_V u$.

Definition 2.6 (Maximal and minimal elements). Let $D = (V, E)$ be a DAG and let $\leq_V$ be the partial order induced by D. For a subset of vertices $S \subseteq V$, an element $u \in S$ is called *D-maximal* (*D-minimal*) in S if there is no $v \in S \setminus \{u\}$ such that $u \leq_V v$ ($v \leq_V u$). Define $\max_V(S)$ and $\min_V(S)$ to be the set of D-maximal and, respectively, D-minimal elements in S.

Definition 2.7 (Future and past cones). Let $D = (V, E)$ be a DAG. For $x \in V$, define the *past cone* of x in D, written as $\mathrm{cone}_V^{(p)}(x)$ to be the set of all vertices $y \in V$ such that $x \leq_V y$. Similarly, define the *future cone* of x in D, written as $\mathrm{cone}_V^{(f)}(x)$ to be the set of all vertices $y \in V$ such that $y \leq_V x$.

Definition 2.8 (Past-closed sets). Let $D = (V, E)$ be a DAG. A subset $S \subset V$ is called *D-past-closed* if and only if for every $u \in S$, the past cone $\mathrm{cone}_V^{(p)}(u)$ is contained in S.

3. Nodes and Participation

At a high level, DLTs can be divided into permissioned and permissionless networks. In a permissioned setting, only selected parties can participate, while in the permissionless setting, anyone can join the network at any time. In a permissioned network, participants have either reading access or writing (validation) rights. A "fully" permissioned (or private) DLT selects the participants in advance and restricts any activity in the network to these only. This is in contrast to a permissionless network where anybody can participate in the network and validate the ledger. Our protocol can work in both settings using a generic weight function on the participating nodes. In the permissionless

setting, this weight function serves as a Sybil protection, and in the permissioned setting, this weight function regulates the participant's influence.

In Section 3.1, we introduce the network participants called nodes. In Section 3.2 we describe a Sybil protection mechanism based on assigning specific weights to nodes. Finally, in Section 3.3 we discuss how the writing ability of nodes is controlled by their weight.

3.1. Network

The network participants in the DLT are called *nodes*, and we denote the set of all nodes by $\mathcal{N} := \{1, \ldots, N\}$, where N is the total number of nodes. *A priori*, different nodes may have different perceptions of the set of nodes. For example, in a permissionless setting, for a node to join the network, the knowledge of a single node entrance point is sufficient. For the sake of a better presentation, we assume that every node is aware of every other node. Nodes directly communicate with a subset of other nodes, i.e. its neighbours, via bidirectional channels. Thus, together all nodes create a peer-to-peer (P2P) overlay network. Nodes use public-key cryptography for their identification. Their unique node ID is derived from the public key, and all their blocks are signed with their private keys.

In contrast to other DLTs, where nodes can be divided into separate functional classes, we assume all nodes behave in the same way. Specifically, all nodes have two main roles. First, they propagate specific blocks through the network by receiving and sending these from and to their neighbours. Second, by creating new blocks and appending them to the data structure, nodes implicitly vote on the state of the previous blocks and their contained transactions; this procedure is called On Tangle Voting (OTV), see Section 6. For the voting part, we assume a scarce resource, see Section 3.2. This resource endows every node with a certain *weight* that is used for the implicit voting procedure.

3.2. Sybil Protection

A common problem in permissionless distributed systems is that it is easy to spawn a significant number of nodes, also known as the Sybil attack. Thus, any critical component must ensure that the action of nodes is limited, otherwise, it would be trivial for an attacker to gain a disproportionately large influence and corrupt the protocol.

To limit or prevent Sybil attacks, we assume that each node can be associated with a particular reputation or *weight* attributing them an equivalent proportion of voting power in the applied voting mechanism.

Definition 3.1 (Weight)**.** For a given node $i \in \mathcal{N}$ there is an associated weight $\mathbf{w}(i)$, given by a function $\mathbf{w} : \mathcal{N} \to [0, 1]$. The weights are assumed to be normalised, i.e.

$$\sum_{i \in \mathcal{N}} \mathbf{w}(i) = 1.$$

The above weight function plays a crucial role in the validation process, see Sections 4.4-6.4.

Remark 3.1. *We make use of the same weights as a control for the writing access in Section 3.3. Note, however, that the weight for writing and validation could be different.*

A common way to implement such a weight is the so-called resource testing, where each identity has to prove the ownership of specific difficult-to-obtain resources. Since in the cryptocurrency world, users own a certain amount of a scarce resource, i.e. tokens, a practical Sybil protection mechanism can be based on proving the ownership of tokens and, thus, a certain amount of collateral.

Another way of implementing the weights is through delegation methods. The owners of source tokens, from which the weights are derived, can then delegate these weights to any node of their choosing. This brings several key advantages. For example, fund owners can delegate weight to nodes that provide good service or revoke it when the node does not behave as expected, thus enabling the implementation of a "reputation" system. In the extreme case, this even allows decoupling the weights from the token distribution and incorporate real-world trust models.

Generally, the weight distribution in our system may change over time due to changes in the weights or inevitable churns (nodes join and leave). Due to the asynchronous nature of the protocol, the perception of the weights may then differ from node to node. The protocol design considers this effect and allows a certain divergence in the weight vector. This tolerance to different perceptions provides for some additional features of the protocol. However, a more detailed discussion of a divergence in the nodes' view on the weight vector is out of the scope of this paper. Thus, for simplicity, we make the following assumption.

Assumption 3.1 (Agreement on stability of weights)**.** All nodes in the network perceive the weight of node i to be precisely $\mathbf{w}(i)$. This weight is assumed to remain constant over time.

3.3. Writing Access

The distributed nature of the protocol and the Byzantine environment within which it operates puts several constraints on the writing access. These constraints are even more critical for our protocol since it is not leader-based and does not rely on the intermediary of miners and block creators. Similar to [51] we require the following conditions:

1) **Consistency:** if a block that is issued by an honest node is written to the (distributed) database by one honest node, it should eventually be written by all honest nodes.
2) **Fairness:** given a weight function and a maximum bandwidth, nodes can issue blocks at a rate proportional to their weight.
3) **Security:** the above constraints are guaranteed in a Byzantine environment.

Consequently, the protocol should ensure that in congested scenarios only a limited amount of blocks are propagated, i.e. the block rate is capped by a certain throughput.

Furthermore, this should happen fairly. These requirements prevent nodes from becoming overloaded and from inconsistencies in the ledger being created. In principle, this could be enabled through fees and PoW, or more novel alternatives as the access control algorithm presented in [51].

For the safe operation of the consensus mechanism, we assume the availability of such a mechanism. The required tool should provide guarantees on the constraints mentioned above. We make the following assumption.

Assumption 3.2 (Writing access). The writing access is controlled such that consistency, security, and fairness in writing access are guaranteed for a given weight function **w**.

4. Block Structure and Witness Weight

In this section, we introduce our protocol's data structure concepts. To replicate a certain content over the distributed network, a node must wrap this content in a block.[4] However, when the content is simply transactions, we require a block to contain only one transaction in its payload. This assumption is made for sake of a better presentation and can be relaxed, such that blocks contain more than one transaction. Moreover, each block has to refer to at least two blocks issued in the past. The latter requirement is motivated by the leaderless architecture of our protocol, in which each node can issue blocks independently of others. In addition, we discuss a particular metric on blocks, called the Witness Weight, that allows nodes to reliably understand when a significant fraction of the network has seen a given block.

In Section 4.1, we formally define a block. Section 4.2 discusses the Tangle, a DAG formed by blocks and their references. The local version of the Tangle seen by a specific node is introduced in Section 4.3. Using the weight function for nodes introduced in Section 3.2, we formally define the Witness Weight of a given block in the local Tangle in Section 4.4 and show how to use this metric as a confirmation rule for blocks in Section 4.5. The analysis of the growth of the Witness Weight is provided in Section 4.6.

4.1. Blocks

The protocol's goal is to replicate certain content between the nodes in the network reliably. For example, this content could be the atomic updates of balances of fund owners.

This content is wrapped into an object that we call *block*. A node that would like to initiate the addition of certain content to the Tangle across the network assembles such a block, which includes the content, k references to previous blocks and the signature of the node (see Figure 2). We call the process of assembling and initial broadcasting the *issuance* of a block. Each node that receives a new block forwards it to its neighbours.

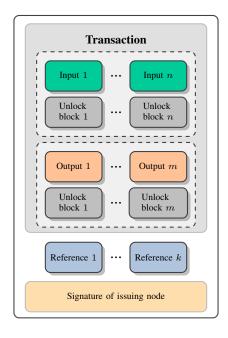4. In prior works, we refer to this object as a message.

Figure 2: Simplified block layout with a transaction as content. The fund owner provides the node with the transaction. The node wraps the transaction into a block and signs the block.

Definition 4.1 (Block). A *reference* $\mathrm{ref}(x)$ *of block* x is a pair (r_y, v), where $r_y = \mathrm{hash}(y)$ is a unique value that corresponds to a previously issued block y and v is the value of a label. We define a block x as an object with content

$$x = (\{\mathrm{ref}_1(x), \dots, \mathrm{ref}_k(x)\}, \hat{x}, \mathrm{nodeID(x)}),$$

where the $\mathrm{ref}_i(x)$'s are references, $\hat{x}$ is a transaction and $\mathrm{nodeID(x)}$ identifies the issuing node.

Remark 4.1. *A collision-resistant hash function is used to map data of arbitrary size to a fixed-size binary sequence, i.e.* $\mathrm{hash} : \{0, 1\}^* \to \{0, 1\}^h$. *Moreover, it is required that it is practically impossible to find for a given sequence x another sequence x' such that $\mathrm{hash}(x) = \mathrm{hash}(x')$. Throughout the remainder of the paper, we assume that a particular hash function is fixed and used by all participants.*

Remark 4.2. *The label v indicates the reference or voting type, as we will see later in Section 6.3.*

The issuing node obtains the content through a service-client relationship with the issuer of the content, which can be facilitated through an application programming interface (API) call. Alternatively, the node itself may also be the issuer of the content. An essential application for the content is the transfer of funds, i.e. the consumption and creation of outputs. We call this type of content a *transaction*. In this paper, for the sake of presentation, we will assume that each block contains exactly one transaction in its payload. However, in general, blocks are not limited to this use case.

As blocks will also be used to propagate votes, keeping track of the issuing nodes is crucial.

Definition 4.2 (Issuer of a block). For a block x, the node that issued x is denoted as **issue**(x), where **issue**$(x) \in \mathcal{N}$.

4.2. The Tangle

The Tangle is a data structure built in accordance with the following rule as stated in the original paper [3] of the Tangle: "*In order to issue a [block]* [5]*, a node chooses two other [blocks] to approve*".

More generally, we modify this by allowing a block to reference up to k existing blocks. The data structure takes the form of a DAG, where the blocks correspond to the vertices, and the references form the edges.

Let us define this data structure more formally. We denote the set of blocks by $\mathcal{T}$. There is a special block, called the *genesis* and denoted by ρ. This block does not contain any references. Any other block has to directly refer to at least two (not necessarily distinct) blocks. Thereby, the reference relationship can be encoded into a DAG.

Definition 4.3 (The Tangle). The Tangle $D_{\mathcal{T}}$ is a DAG whose vertex set is the set of blocks $\mathcal{T}$. There is a directed edge from y and x in $D_{\mathcal{T}}$ if and only if y directly refers to x.

Using the notation from Section 2, we write $\leq_{\mathcal{T}}$ to denote the partial order on the set of blocks induced by $D_{\mathcal{T}}$. For a block $x \in \mathcal{T}$, the Tangle past and future cone of x are denoted as $\mathrm{cone}_{\mathcal{T}}^{(p)}(x)$ and $\mathrm{cone}_{\mathcal{T}}^{(f)}(x)$, respectively. The parents and children of x are written as $\mathrm{par}_{\mathcal{T}}(x)$ and $\mathrm{child}_{\mathcal{T}}(x)$. If $x <_{\mathcal{T}} y$ we say that block x *approves* or *references* block y. Specifically, if $x \in \mathrm{child}_{\mathcal{T}}(y)$, then x *directly references* y; if $x \notin \mathrm{child}_{\mathcal{T}}(y)$ and $x <_{\mathcal{T}}$, then x *indirectly references* y. A leaf in the Tangle DAG is said to be a *tip*.

Example 4.1. We refer to Figure 3 for an illustration of the Tangle and the Tangle future and past cones of block x.

4.3. Local Tangles

Due to the distributed nature of the network, nodes can receive blocks at differing times or even out of order. The time at which a node first receives a block is called *arrival time*.

Blocks can also be lost during their broadcast. While, generally, this could be problematic, the Tangle DAG allows for an elegant solution to remedy the loss by a process called *solidification*. If a node receives a block for which the parents are unknown, it requests the missing block from its peers. Upon receipt of the missing parent block, the past cone is now complete (unless their parents are missing - in which case the node has to repeat this procedure recursively). Once a block's past cone is completed, the node flags the block as *solid*. The time of solidification of a block x in node i is denoted by $\tau_{s,i}(x)$. We only consider blocks included in the Tangle after they are flagged solid.

5. The term used in the original whitepaper is *transaction*, however, in this work we distinguish between the block and its contained transaction.
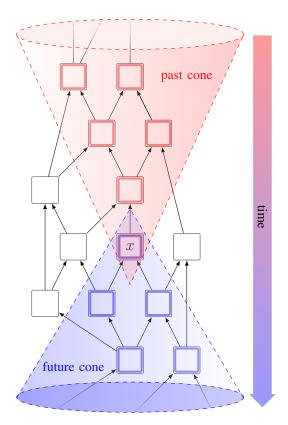

Figure 3: Future and past cones of a block x in the Tangle

As a consequence of the above, we can argue that there is no such thing as one Tangle in the network, as every node may have a different perception of it. Hence, at time t a node i is aware only of the block x that satisfy $\tau_{s,i}(x) \leq t$. We denote by $\mathcal{T}_{i,t}$ and $D_{\mathcal{T}_{i,t}}$ the local perception of the block set and the Tangle DAG perceived from node i at (local) time t. Past and future cones then are also given in their *local* forms $\mathrm{cone}_{\mathcal{T}_{i,t}}^{(f)}(x)$ and $\mathrm{cone}_{\mathcal{T}_{i,t}}^{(p)}(x)$. We omit subscripts and simply write $D_{\mathcal{T}} = D_{\mathcal{T}_{i,t}}$ if the dependence on i and t is clear from the context.

4.4. Witness Weight and Weighted Local Tangles

In the original Tangle whitepaper [3] the cumulative weight of a block plays a crucial role in the consensus finding. This cumulative weight is the number of blocks referencing a given block. In case of a conflict, nodes follow the part of the Tangle that contains the largest cumulative weight.

We adopt this fundamental idea to the setting where each node carries some weight. In this way, the nodes' weight replaces the PoW in the block creation as a Sybil protection mechanism. The nodes' signature in each block links the issuing node to the block (see Section 4.1). Thus, a node can be associated with the set of blocks on the Tangle issued by that node, and the node's weight can be mapped to the blocks.

Definition 4.4 (Block Supporter and Witness Weight). Let $x \in \mathcal{T}_{i,t}$ be a block. Denote by $\mathrm{sprt}_{\mathcal{T}_{i,t}}(x)$ the set of nodes that issues a block in the future cone of x:

$$\mathrm{sprt}_{\mathcal{T}_{i,t}}(x) = \left\{ j \in \mathcal{N} : \ \exists y \in \mathrm{cone}^{(f)}_{\mathcal{T}_{i,t}}(x), \ j = \mathbf{issue}(y) \right\}.$$

We call nodes from $\mathrm{sprt}_{\mathcal{T}_{i,t}}(x)$ *supporters* of x. We define the function $\mathbf{WW}_{i,t} : \mathcal{T}_{i,t} \to [0,1]$ which is called the *Witness Weight (WW)* of a block seen by node i at time t as follows

$$\mathbf{WW}_{i,t}(x) := \sum_{j \in \mathrm{sprt}_{\mathcal{T}_{i,t}}(x)} \mathbf{w}(j). \tag{2}$$

As the total weight is normalised to 1 the WW describes the percentage of weight approving a given block. Whenever it is clear from the context, we omit indices i and t.

Example 4.2. In Figure 4, we give an example of the set of nodes approving given blocks x, y and z. We use unique colours in the bottom of blocks to represent signatures of different issuing nodes. One can readily check that $\mathrm{sprt}_{\mathcal{T}}(x)$ consists of nodes corresponding to brown, cyan and gray colours.
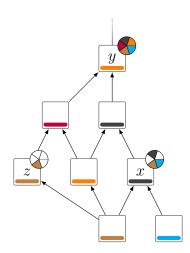

Figure 4: A Tangle DAG, where the issuing node of a block can be identified with a unique colour shown in the bottom of the block. The colors of the supporters of blocks x, y, z are depicted in the top-right corners.

We proceed with two trivial statements saying that the WWs of blocks are monotonically increasing toward the genesis and the WW of a block can only grow over time.

Lemma 4.1 (Monotonicity of the WW). *For any two blocks $x, y \in \mathcal{T}$ such that $x \leq_{\mathcal{T}} y$, it holds that $\mathrm{sprt}_{\mathcal{T}}(x) \subseteq \mathrm{sprt}_{\mathcal{T}}(y)$ and, hence, $\mathbf{WW}(x) \leq \mathbf{WW}(y)$.*

Lemma 4.2 (Growth of the WW). *For any block $x \in \mathcal{T}$, node $i \in \mathcal{N}$ and time instants t_1 and t_2 such that $t_1 < t_2$, it holds that $\mathrm{sprt}_{\mathcal{T}_{i,t_1}}(x) \subseteq \mathrm{sprt}_{\mathcal{T}_{i,t_2}}(x)$ and, hence, $\mathbf{WW}_{i,t_1}(x) \leq \mathbf{WW}_{i,t_2}(x)$.*

A more delicate analysis of the growth of the WW under certain assumptions is provided in Section 4.6.

4.5. Confirmation Rule for Blocks

The block stream is controlled by the writing access control, see Section 3.3. A priori, this control alone may not be sufficient to guarantee that all nodes see all blocks in the network. However, to guarantee the safety of the system, nodes must have consensus on which blocks should permanently be accepted in the data set $\mathcal{T}$, otherwise, inconsistencies between the nodes could arise. If such a consensus is achieved, we consider a block *confirmed*. Furthermore, to maintain consistency in the data structure $D_{\mathcal{T}}$, a block x can only be confirmed if all blocks in $\mathrm{cone}^{(p)}_{\mathcal{T}}(x)$ are confirmed.

Tools that provide information about the confirmation status of blocks, with specific safety and liveness considerations, are generally referred to as *confirmation rule*. We design such a tool based on the concept of WWs of the blocks. The WW allows the nodes and users to create their subjective confirmation criterion. The larger the WW of a block, the higher the probability that the block will be in the ledger forever. This idea is similar to the "depth" of a transaction in a blockchain. Therefore, the actual confirmation criterion may depend on the protocol environment and the underlying use case.

Definition 4.5 (Confirmed block). Let $\theta \in (0.5, 1]$ be a fixed threshold. We say that a block $x \in \mathcal{T}$ is *confirmed* for a node $i \in \mathcal{N}$ at time t if $\mathbf{WW}_{i,s}(x) \geq \theta$, for some $s \leq t$.

Once a block is confirmed for a node, it remains confirmed forever. This irreversibility of the confirmation status places some strong requirements on the convergence of this status. More specifically, once a single node reaches the threshold for a given block, all nodes should reach this threshold eventually with a very high probability.

In an honest scenario, this assumption can be easily satisfied since a high WW also represents that a large proportion of nodes have "seen" a given block and issued a block approving it. If the default tip selection algorithm is suitably chosen and followed by sufficiently many nodes all nodes will attach blocks eventually to the future cone of that block with a very high probability (for more details, see Section 4.6). In Section 8 we discuss the liveness and safety of the protocol in detail.

4.6. Growth of Witness Weight

In this section, we model the block issuance and discuss the growth of the WW and its dependencies on the protocol environment.

We consider the following assumption.

Assumption 4.1 (Issuing rate). Each node $i \in \mathcal{N}$ issues blocks at a Poisson rate λ_i (per second). The rate λ_i is proportional to the corresponding weights $\mathbf{w}(i)$ (see Definition 3.1), i.e. $\lambda_i = \lambda \mathbf{w}(i)$ for some constant $\lambda > 0$. We assume that every node issues blocks independently of the other nodes. The rate of issuance for all nodes is then

$$\lambda = \sum_{i \in \mathcal{N}} \lambda_i.$$

Remark 4.3. *Under Assumption 4.1 the times between two successive blocks from a node $i \in \mathcal{N}$ are independent and exponentially distributed with parameter λ_i.*

To develop a heuristic for the WW we use the following approach. We assume that there is an "omniscient observer", that is instantly aware of all blocks issued by all nodes. The observer's perception of the state may differ from the perception of a given node, however, these differences have no substantial influence on the heuristic result. We refer to [52], [53] where this method has already been proven to lead to good heuristics. This view is reflected in the notation by omitting the index i. For instance, $\mathcal{T}_t$ denotes the set of blocks perceived by this omniscient observer at time t and $\mathbf{WW}_t(x)$ denotes the corresponding WW of a block x at time t.

Let x be a block issued at time t_0 and denote by $E_i(\delta, x)$ the event that node i issues a block in the time interval $[t_0, t_0 + \delta]$ in the future cone of x. We write $\mathbf{1}\{E_i(\delta, x)\}$ for the indicator function of this event; it is equal to 1 if the event occurred and 0 otherwise.

For $t = t_0 + \delta$, the WW of block x perceived by the omniscient observer satisfies

$$\mathbf{WW}_t(x) = \sum_{i=1}^{N} \mathbf{w}(i) \mathbf{1}\{E_i(\delta, x)\}. \qquad (3)$$

Node i issues blocks with rate $\lambda \mathbf{w}(i)$ and, thus, we have that

$$\mathbb{P}(E_i(\delta, x)) \leq 1 - \exp(-\delta \lambda \mathbf{w}(i)). \qquad (4)$$

Note that the equality does not necessarily hold since not all new incoming blocks have to witness block x. Taking the expectation in Equation (3) and applying Inequality (4) we obtain

$$\mathbb{E}[\mathbf{WW}_t(x)] \leq \sum_{i=1}^{N} \mathbf{w}(i) \left(1 - \exp(-\delta \lambda \mathbf{w}(i))\right). \qquad (5)$$

The formula given in (3) holds in the very general setting. For the analysis of the protocol, it is, however, important to consider a specific weight distribution. Probably the most appropriate modelings of weight distributions rely on universality phenomena. The most famous example of this universality phenomenon is the central limit theorem. While the central limit theorem is suited to describe statistics where values are of the same order of magnitude, it is not appropriate to model more heterogeneous situations where the values might differ in several orders of magnitude. These heterogeneous situations are frequently described by a Zipf law and appear in many fields; e.g. city populations, internet traffic data, the formation of P2P communities, company sizes, and science citations. We refer to [54] for a brief introduction and more references, and to [55]–[57] for the appearance of Zipf's law on the internet, computer networks, and DLTs.

We consider a situation with N elements or nodes. Zipf's law predicts that the (normalised) weight of the node of rank r is given by

$$\mathbf{w}(r) = \frac{r^{-s}}{\sum_{j=1}^{N} j^{-s}}, \qquad (6)$$

where $s \in [0, \infty)$ is the Zipf parameter. Since the weights $\mathbf{w}(\cdot)$ in (6) only depends on two parameters, s and N, this provides a convenient model to investigate the performance of the protocol in a wide range of network situations. For instance, a homogeneous network with N nodes having equal weight can be modeled by choosing $s = 0$. With increasing value of s the network becomes increasingly centralised.

Example 4.3. We refer to Figure 5. The growth of the WW depends on several factors, notably the issuing rate λ and the distribution of the nodes' weight. In the case of a Zipf distribution the weight depends on two parameters, the number of nodes N and the Zipf parameter s. The upper bound (5) is a convex monotone function in δ and λ. The dependence on the parameters N and s is not so obvious. For this reason, we perform some Monte-Carlo simulations for $N \in \{100, 1000, 10000\}$ and $s \in \{0, 0.2, 0.4, 0.6, 0.8, 1, 1.2\}$, and $\lambda = 1000$.[6] For a given t_0 we approximate the WW at time $t_0 + t$ to be the sum of the weights of all nodes having issued a block during the time interval $[t_0, t_0 + t]$. This provides a lower bound estimate for the WW of a block that is issued at t_0. In Figure 5 every line corresponds to one realisation of the growth of the issued WW (for $t_0 = 0$).

4.7. Estimates on Time to Confirmation

As discussed in Section 4.5, a confirmation rule is essential for many use cases, and time to confirmation (TTC) is undoubtedly a vital performance measure of every consensus protocol. As a thorough analysis of the TTC is out of the scope of this paper, we give a first "heuristic" upper bound in this section.

Definition 4.6 (Time to confirmation). We define the time to confirmation of a block x (at level θ) by a node i as

$$\tau_{f,i} = \tau_{f,i}(x) := \inf\{t > 0 : \mathbf{WW}_{i,t}(x) \geq \theta\} - \tau_{s,i}(x), \qquad (7)$$

where $\tau_{s,i}(x)$ is the solidification time of x (see Section 4.3).

In the remainder of this section, we omit index $i \in \mathcal{N}$ since the provided analysis is relevant to all nodes. We divide the TTC into two periods. During the first period, we wait for the confluence time $\tau_c = \tau_c(x)$ until a given block x is contained in the past cone of (almost) all current tips. During the second period, the issuance time τ_{iss}, we let the WW grow until it reaches the threshold θ. The TTC τ_f is then bounded above by

$$\tau_f \leq \tau_c + \tau_{iss}. \qquad (8)$$

---

6. Typical values for the Zipf parameter found in popular cryptocurrencies for the top 100 addresses are in the range between $s = 0.7$ and $s = 1.2$ [57].
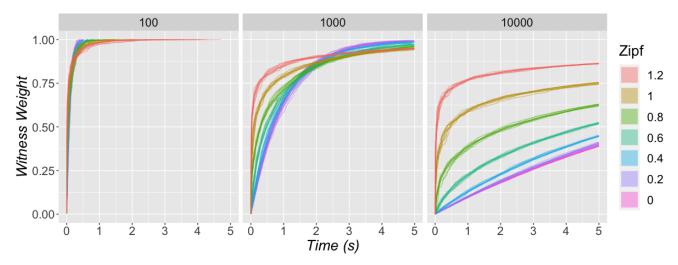
Figure 5: Growth of the issued WW in Example 4.3 with 1000 blocks per second. We see the different behaviour for 100 nodes (left), 1000 nodes (middle) and 10.000 nodes (right). The growth depends essentially on the chosen Zipf parameter s (in colour) and the number of nodes.

Estimates for τ_{iss} are obtained from (3) and this formula can be simplified for specific choices of the weights (see Example 4.5).

Example 4.4. We demonstrate the confluence time and the issuance time with the help of Figure 6. Blocks with a solid frame are in the future cone of block x. After the confluence time, all blocks approve x. The yellow, green and purple colours represent blocks by nodes that hold significant weight, i.e. the nodes have a large influence on the confirmation. Once the cumulative weight of the nodes issued in the future cone of x reaches the threshold θ, the block becomes confirmed.

With some additional assumptions, we can obtain estimates for the confluence time τ_c similarly to [3]. Our first assumption is that the delay between block creation and the moment that other nodes in the network receive this block is constant.

Assumption 4.2 (Constant network delay). We assume that the time between the block creation and until any other node receives this block equals some constant h.

Definition 4.7 (Number of tips). Let $L(t)$ be the total number of tips of the Tangle at time t.

As mentioned in Section 4.3, there is no "objective Tangle," and every node has its own perception. Nevertheless, previous work [52] showed that the approximation made in this section leads to reasonable approximations for some quantitative properties of the Tangle, such as the number of tips and confluence times. For this reason, we omit the subscript "i" and work with a unique objective Tangle in this section. Similar to [3] we assume the number of tips to be in a stationary regime.

Assumption 4.3 (Constant Tangle width). We assume that the number of tips $L(t)$ of the Tangle is stationary and has

mean L_0.

Using Assumptions 4.1, 4.2, and 4.3 we follow the heuristics described in [3, Section 3]. A first observation is that at any given time t there are on average λh *hidden tips*, those blocks that have been issued after $t - h$ but are not yet visible to the network. As in [3] we assume that typically there are r *revealed tips*, those that have been attached before $t - h$ but are still tips. Hence, we can write the total (average) number of tips as $L_0 = r + \lambda h$. By Assumption 4.3 we consider that the number of tips $L(t)$ is roughly stationary. This implies that since λh tips join the tip pool, during the same time, roughly λh blocks that have been tips at time $t - h$ became referenced and are no longer tips. Hence, the tip pool of size L_0 can be divided into r revealed tips and λh blocks that are no longer tips. This division leads to the crucial observation that a new block (with k parents) approves on average $kr/(r + \lambda h)$ (revealed) tips. Moreover, in the stationary situation where the tip pool size L_0 stays approximately constant, the mean number of chosen tips should be equal to 1; otherwise, the number of tips would change. Solving $kr/(r + \lambda h) = 1$ leads to

$$L_0 = L_0^{(k)} = \frac{k\lambda h}{k - 1}. \tag{9}$$

This result, predicted in [3], has been confirmed through simulation studies in [52], [53] and theoretical results in [58].

A first consequence of (9) is that, if L_0 is large, the expected time for a block to be approved for the first time is approximately

$$h + L_0/(k\lambda) = h + \frac{h}{(k - 1)}. \tag{10}$$

The size of the tip pool is naturally linked to the growth of the WW of a given block; the larger the tip pool the slower the growth of the WW.
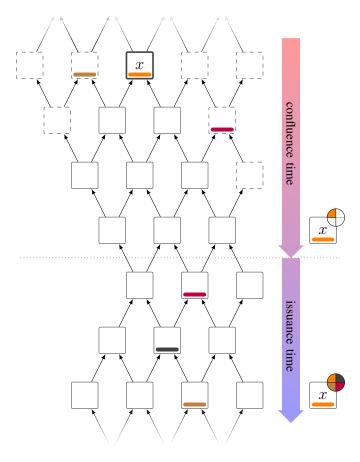
Figure 6: Illustration of the Tangle to display the confluence time and issuance time. The colours in the bottom of the blocks represents the issuing nodes with significant weight. We demonstrate the colours of the "heavy" supporters of block x on the right after each time period. The dashed blocks correspond to blocks that are not in the future cone of x.

Remark 4.4. *For any given λ and h we can choose k sufficiently large such that $k > L_0^{(k)}$. In this case, blocks are referenced essentially immediately after they become* visible, *however, at the cost of a larger block size.*

We can proceed similar to [3] to obtain that

$$\tau_c \approx \frac{h}{W\left(\frac{(k-1)^2}{k}\right)} \left(\log L_0 + \log \varepsilon\right), \qquad (11)$$

where $\log$ denotes the natural logarithm function and W is the principal branch of the Lambert W-function, which is defined as the inverse function to $z = we^w$, i.e. $w = W(z)$. For large k, we can use the approximation

$$W\left(\frac{(k-1)^2}{k}\right) \approx 2\log(k-1) - \log k \approx \log k$$

and, hence, obtain

$$\tau_c \approx \frac{h}{\log k}\log(L_0) \approx \frac{1}{\log k} h\log(\lambda h). \qquad (12)$$

In Section A, we will give more details on the derivation of the confluence time.

Example 4.5. The behaviour of the issuing time τ_{iss} heavily depends on the actual weight distribution, e.g. see Figure 5. However, the extreme case of all nodes having the same weight can be treated more analytically. Extreme is meant here in the sense that the growth of the WW is to some extent the smallest. Hence, let $\mathbf{w}(i) = 1/N$ for all nodes $i \in \mathcal{N}$ and assume that we want to get a bound on the confirmation time, i.e. the first time a given block x reaches $\mathbf{WW}_t(x) \geq \theta$. Denote by X_i the first time a block was sent from node $i \in \mathcal{N}$. The vector of these times $(X_1, \ldots, X_N)$ can be ordered in increasing order and we obtain the so-called order statistics $X_{(1)}, \ldots, X_{(N)}$. In the case where all X_i follow the same exponential distribution $\text{Exp}(\gamma)$ the distribution of the ith order statistic is given by

$$X_{(i)} \sim \frac{1}{\gamma}\sum_{j=1}^{i}\frac{Z_j}{N-j+1}, \qquad (13)$$

where the Z_j are i.i.d. exponential random variables with parameter 1. Eventually, the time it takes that $\lceil \theta N \rceil$ nodes issued a block is distributed as $X_{(\lceil \theta \cdot N \rceil)}$. The expectation is given by

$$\mathbb{E}[X_{(i)}] = \frac{1}{\gamma}\sum_{j=1}^{i}\frac{1}{N-j+1},$$

with $i = \lceil \theta \cdot N \rceil$. Using a standard integral approximation for the above sum, we obtain for large N that

$$\mathbb{E}[X_{(i)}] \approx \frac{N}{\lambda}\left(\log(N) - \log(N-i)\right).$$

Hence, for $i = \theta N$,

$$\tau_{iss} \approx \mathbb{E}[X_{(i)}] \approx \frac{N}{\lambda}\left(-\log(1-\theta)\right).$$

Combing this result with the bound (12) on the confluence time in (8) we obtain the following asymptotic upper bound on the TTC for large k (and the other parameters fixed):

$$\tau_f \lesssim \frac{1}{\log k}h\log(\lambda h) + \frac{N}{\lambda}\left(-\log(1-\theta)\right).$$

5. The Ledger

This section introduces several novel concepts to represent transactions and their interrelationships. Recall that in the standard UTXO conflict-free model, transactions specify the outputs of previous transactions as inputs and create new outputs by spending (or consuming) the inputs. No two transactions are consuming the same input. Such a conflict-free data structure can be implemented in a network where a consensus mechanism filters transactions. The latter is

typically done by choosing a "leader" among the participants, and the leader adds a block of transactions to the conflict-free ledger. To bypass this "centralised" bottleneck, we propose the concept of the Reality-based UTXO Ledger, an augmented version of the standard conflict-free UTXO Ledger that allows more than one output spend. We refer the reader to the parallel work [15], where we discuss all concepts in detail.

In Section 5.1, we recall the definition of a transaction in the UTXO model and the ledger, which is a set of all transactions. In Section 5.2, we introduce definitions of conflicting transactions, conflicts and branches, which represent proper subsets of "non-conflicting conflicts". A reality is a maximal possible branch, and restricting a ledger to a reality results in the conflict-free UTXO Ledger. Finally, in Section 5.3 we discuss how nodes could choose a reality given an abstract weight function defined on the set of conflicts. The selected reality allows a node to express its opinion when issuing new blocks and validating transactions.

5.1. UTXO Model and Transactions

In the Unspent Transaction Output (UTXO) model transactions specify the outputs of previous transactions as inputs and spend them by creating new outputs.

Thus, a transaction consists of a list of inputs and a list of outputs, see Figure 2. Note that outputs must be unique. The uniqueness is typically achieved by creating the output ID with the involvement of a hash function. For example, the output ID could be the concatenation of the index of an output and the hash of a transaction's content. Every output represents a specific amount of the underlying cryptocurrency. The value of all inputs, i.e. spent outputs, must equal the value of all outputs of a transaction. With each output comes a declaration by whom and under which conditions it can be spent. Under unlock conditions, e.g. a signature proving ownership of a given input's address, the transaction issuer is allowed to spend the inputs. We refer to Figure 2 for a general transaction layout.

As said in Section 4.1, blocks contain transactions in their payload. Hereafter, we write $\hat{x}$ to denote the transaction contained in the payload of a block x.

Let us define the transactions and ledger model more formally. We follow the approach of [59].

Definition 5.1 (Output and input). An *output* is a pair of a value $v \in \mathbb{R}^+$ and an unlock condition cond. We write $o = (v, \text{cond})$ to denote the output. An *input* i is a reference to an output. We say the input *consumes* the output.

Definition 5.2 (Transaction). A transaction $\hat{x}$ is a collection of inputs $\text{in}(\hat{x})$, outputs $\text{out}(\hat{x})$, and unlock proofs $\text{unlock}(i)$, $i \in \text{in}(\hat{x})$, where

1) $\text{in}(\hat{x}) = (i_1, \ldots, i_n)$ is a list of inputs, i.e. references to unconsumed outputs. We say that those outputs are spent or consumed by transaction x;
2) $\text{out}(\hat{x}) = (o_1, \ldots, o_m)$ is a list of new outputs produced by transaction $\hat{x}$;

3) $\text{unlock}(i)$ is a proof which performs verification of the unlock conditions of each input i of transaction $\hat{x}$. This is usually done by cryptographic proof of authorization that ensures that the issuer of the transaction satisfies the condition cond of the consumed outputs.

Definition 5.3 (Ledger). The *ledger* is a set of transactions and denoted as $\mathcal{L}$.

The UTXO ledger starts at the so-called *genesis* which contains outputs and no inputs. We emphasize that we use the same term for the ultimate predecessor of all blocks and all transactions. Recall that the genesis-block is written as ρ, whereas the genesis-transaction will be denoted as $\hat{\rho}$.

Typically every output can be consumed by at most one transaction and, hence, the value of all unspent outputs is conserved overall. Specifically, in the standard conflict-free UTXO model, the ledger can not contain a so-called *double spend*, i.e. two transactions that consume the same output of a transaction.

In the following section, we alleviate this conflict-free restriction and allow the Ledger to contain conflicting transactions.

5.2. Reality-based Ledger

In this section, we propose an augmented version of the standard conflict-free UTXO ledger model that allows containing double spends. We suggest different structures that can be used for tracking conflicting transactions without the need for consensus.

First, we explain how the transactions and their in- and outputs result in a DAG structure. The information contained in the Ledger DAG is split into the Conflict Graph, which keeps track of the conflicting transactions only. Then we introduce the concept of branches. A branch forms a possible non-conflicting state of the ledger. We will then derive a concept, called a *reality*, which allows us to reduce $\mathcal{L}$ to a maximal subset of transactions that yield a conflict-free (*Reality-based*) ledger.

Definition 5.4 (Ledger DAG). We define the Ledger DAG $D_{\mathcal{L}}$ to be a DAG whose vertex set is the ledger $\mathcal{L}$. There is a directed edge $(\hat{x}, \hat{y})$ in the edge set of $D_{\mathcal{L}}$ if and only if an input of $\hat{x}$ references an output of $\hat{y}$.

We refer to Appendix B, where we demonstrate this graph together with many other core concepts. Using the notation from Section 2, we write $\leq_{\mathcal{L}}$ to denote the partial order on the set of transactions induced by $D_{\mathcal{L}}$. The past cone of a transaction $\hat{x}$ is denoted by $\text{cone}_{\mathcal{L}}^{(p)}(\hat{x})$, i.e. transaction $\hat{x}$ spends value directly or indirectly from transactions in $\text{cone}_{\mathcal{L}}^{(p)}(\hat{x}) \setminus \{\hat{x}\}$.

Typically, the addition of transactions to this type of data structure is such that only transactions, which create no conflict with any previously recorded transactions are allowed to be added, i.e. the Ledger DAG is conflict-free. However, this requires a consensus mechanism that pre-selects transactions.

Now we introduce a new design for a ledger, where this constraint is replaced by a relaxed one – namely, a new transaction $\hat{x}$ can be added to the ledger if $\text{in}(x)$ are references to outputs which are not already consumed in $\text{cone}_{\mathcal{L}}^{(p)}(\hat{x}) \setminus \{\hat{x}\}$. In the following, we provide an overview of some of the most important concepts of the proposed solution that allows conflicting transactions to co-exist. Thereby, we start with a formal definition of conflicts and conflicting transactions.

Definition 5.5 (Conflicts). Two distinct transactions $\hat{x}, \hat{y} \in \mathcal{L}$ are *directly conflicting* if they have at least one input in common. A transaction $\hat{x} \in \mathcal{L}$ is called a *conflict* if and only if there exists a transaction $\hat{y} \in \mathcal{L} \setminus \{\hat{x}\}$ such that $\hat{x}$ and $\hat{y}$ are directly conflicting.

Definition 5.6 (Conflicting transactions). Two distinct transactions $\hat{x}_1, \hat{y}_1 \in \mathcal{L}$ are said to be *conflicting* if there exist distinct $\hat{x}_2, \hat{y}_2 \in \mathcal{L}$ with $\hat{x}_1 \leq_{\mathcal{L}} \hat{x}_2$ and $\hat{y}_1 \leq_{\mathcal{L}} \hat{y}_2$ such that $\hat{x}_2$ and $\hat{y}_2$ are directly conflicting.

The interrelations between conflicts can be encoded with the help of the Conflict DAG and the Conflict Graph.

Definition 5.7 (Conflict DAG and Conflict Graph). The set of all conflicts is denoted by $\mathcal{C}$ and dubbed the *set of conflicts* of the ledger $\mathcal{L}$. We define the Conflict DAG $D_{\mathcal{C}}$ to be the minimal subDAG of the Ledger DAG induced by $\mathcal{C} \cup \hat{\rho}$ (cf. Definition 2.5). We define the Conflict Graph $G_{\mathcal{C}}$ to be the graph whose vertex set is $\mathcal{C}$ and two conflicts are connected by an edge if and only if these conflicts are conflicting (as transactions).

We can group transactions based on whether they conflict with each other or not.

Definition 5.8 (Conflict-free set and conflicting sets). A subset of transactions $S \subseteq \mathcal{L}$ is called *conflict-free* if it does not contain any two conflicting transactions. We also say that $S_1 \subseteq \mathcal{L}$ is *conflict-free with respect to* $S_2 \subseteq \mathcal{L}$ if there is no $\hat{x}_1 \in S_1$ and $\hat{x}_2 \in S_2$ such that $\hat{x}_1$ and $\hat{x}_2$ are conflicting. Alternatively, S_1 is *conflicting* with S_2 if S_1 is not conflict-free with respect to S_2.

We further specialise conflict-free sets and introduce the notion of branches.

Definition 5.9 (Branch and set of branches). A set of conflicts $B \subseteq \mathcal{C}$ is called a *branch* if and only if the two properties hold:

1) B is conflict-free (cf. Definition 5.8);
2) B is $D_{\mathcal{C}}$-past-closed (cf. Definition 2.8).

Define $\mathcal{B}$ to be the set of all branches. A branch that represents the empty set is called the *main branch*.

We now introduce the concept of a reality which can be defined as a maximal possible branch or, equivalently, a maximal independent set in the Conflict Graph. In other words, a reality aggregates the maximal number of conflicts while preserving non-conflicting nature.

Definition 5.10 (Maximal branch and reality). A branch $B \in \mathcal{B}$ is *maximal* if there exists no other branch $A \in \mathcal{B}$ such that $B \subset A$. A maximal branch is called a *reality*.

Next, we describe the notion of the maximal contained branch of a given transaction which consists of the set of conflicting transactions in the past cone of the given transaction.

Definition 5.11 (Maximal contained branch). Let $\mathcal{B}$ be the set of all branches, and $\text{branch}_{\mathcal{L}}^{(p)} : \mathcal{L} \to \mathcal{B}$ be a function that for a given transaction $\hat{x} \in \mathcal{L}$ returns the maximal branch contained in $\text{cone}_{\mathcal{L}}^{(p)}(\hat{x})$.

We note that there could not be two maximal branches in the ledger past cone of a transaction. Indeed, the past cone of any transaction is conflict-free and, thus, if there would be two maximal branches, we could consider the union of two branches, which has to be also a branch.

Definition 5.12 (Ledger of a reality). Let $R \in \mathcal{B}$ be a reality. Define the *R-ledger*, written as $\mathcal{L}(R)$, to be the set of all transactions $\hat{x} \in \mathcal{L}$ such that $\text{branch}_{\mathcal{L}}^{(p)}(\hat{x}) \subseteq R$.

Recall that a maximal contained branch of a transaction from the R-ledger is a subset of R. Thus, the past cones of any two transactions are conflict-free and so is the R-ledger.

Remark 5.1 (Local Ledger). *As discussed in Section 4.3, there could be subjective versions of the Tangle DAG. Similarly, every node has its own perception of the Ledger. Thereby, we will use subscripts i, t in $\mathcal{L}$, $D_{\mathcal{L}}$ and other related notions if we talk about the point of view of node $i \in \mathcal{N}$ at moment t.*

5.3. Reality selection algorithm

To issue new blocks and validate transactions, each node in the network has to choose a conflict-free part of the ledger that it prefers. For this purpose, it suffices for a node to choose a preferred reality. Once a reality R is chosen, the node can make different operations on the R-ledger.

Definition 5.13 (Preferred reality). Node $i \in \mathcal{N}$ at time t chooses a specific reality $R = R_{i,t} \in \mathcal{B}$ which is called the *preferred reality* for node i.

There could be different ways to choose the preferred reality. We provide a natural reality selection algorithm that takes as an input the Conflict Graph and an abstract weight function $\mathbf{w} : \mathcal{C} \to [0, 1]$ satisfying two properties:

1) monotonicity: for any two conflicts $x, y \in \mathcal{C}$ such that $x \leq_{\mathcal{C}} y$, it holds that

$$\mathbf{w}(x) \leq \mathbf{w}(y);$$

2) consistency: let $x_1, \ldots, x_s$ be pairwise conflicting conflicts.[7] Then it holds that

$$\sum_{i=1}^{s} \mathbf{w}(x_i) \leq 1.$$

7. We say that transactions $S \subseteq \mathcal{L}$ are pairwise conflicting if any pair of transactions $x, y \in S$ are conflicting.

Algorithm 1: Reality selection in Conflict Graph

Data: Conflict Graph $G_{\mathcal{C}} = (\mathcal{C}, E)$
Result: reality $R \in \mathcal{B}$

1   $R \leftarrow \emptyset$
2   $U \leftarrow \mathcal{C}$
3   **while** $|U| \neq 0$ **do**
4      $c^* \leftarrow \arg\max\{\mathbf{w}(c) : c \in \max_{\mathcal{C}}(U)\}$;   /* use $\min \text{hash}(c)$ for breaking ties */
5      $R \leftarrow R \cup \{c^*\}$
6      $U \leftarrow U \setminus \{N_{\mathcal{C}}(c^*) \cup \{c^*\}\}$
7   **end**

Remark 5.2. *In Section 6.4, we introduce the Approval Weight function defined on the set of all transactions, i.e.* $\mathbf{AW} : \mathcal{L} \to [0, 1]$. *Then the required weight function can be obtained as the restriction of the Approval Weight to the set of conflicts, i.e.* $\mathbf{w} = \mathbf{AW}|_{\mathcal{C}}$.

In Algorithm 1 we describe the proposed procedure. In this algorithm, we initialize R as the genesis and U as the set of conflicts. Then we iteratively construct a subset R of conflicts and prune transactions conflicting with R from U. Specifically, we add a conflict to R if this conflict is not conflicting with this set and attains the highest value of the weight function among all $D_{\mathcal{C}}$-maximal elements that remain in U. By construction, Algorithm 1 leads to a maximal independent set in the Conflict Graph or a reality. The number of iterations in the while-loop is bounded by $|\mathcal{C}|$ and the number of $G_{\mathcal{C}}$-neighbours is also bounded by $|\mathcal{C}|$. Thus, it is possible to implement this algorithm with complexity $O(|\mathcal{C}|^2)$.

We refer to Appendix B, where we apply the algorithm as part of an illustrated example.

6. On Tangle Voting

In this section, we present a voting mechanism based on the Tangle and the Ledger DAG. This mechanism allows for selecting realities in the Reality-based Ledger.

In Section 6.2 we give an overview of two suitable DAG structures, which can be utilised to enable voting on the realities. Section 6.3 combines these two structures into a Voting DAG and introduces basic concepts that follow from it. We also address how voting on two DAGs increases the liveness of the protocol. Section 6.4 defines a metric called Approval Weight which is utilised in Section 6.5 to identify a preferred reality and vote for it using a suitable tip selection algorithm.

6.1. Extension of Witness Weight and Liveness Problems

In Section 4 we introduced the Witness Weight, which is a metric used for the confirmation of blocks. In this section, we seek a similar tool for the confirmation of transactions.

The Witness Weight has the property that it is monotonically increasing since it expresses the percentage of the weight that has witnessed a block's existence. The situation is different for transactions where we want to leverage the node's weight to decide between conflicting transactions. To ensure liveness, nodes must have the possibility to change their votes and withdraw their weights from the approval weight of a given transaction.[8] However, changing the opinions might imply that blocks that reference (and vote for) blocks with rejected transactions might never be confirmed.

This situation creates a negative incentive to reference new tips. More precisely, nodes may be incentivized to either reference only blocks from trusted entities, tips of a certain age, or in the worst case, ancient and already confirmed blocks. The last behaviour may eventually lead to no new blocks being confirmed anymore.

The problems above were until now a significant concern of DAG-based consensus protocols, e.g. [3]. We propose to solve these by using the Reality-based Ledger and extending the reference scheme.

6.2. Immutable DAGs

Blocks are the primary information carriers of the network, i.e. they contain transactions and express the opinion of the issuing nodes. The references in the blocks, together with the signature of the nodes and the unlock proofs for the inputs, form two immutable data structures, similar to a blockchain.

First, the Tangle $D_{\mathcal{T}}$ is constructed on the set of blocks $\mathcal{T}$. The interrelations are defined by the references contained in the blocks, which are selected and signed by the issuing nodes (for more details, see Section 4).

Second, the Ledger DAG $D_{\mathcal{L}}$ is constructed on the set of transactions $\mathcal{L}$. Their interrelations are defined by the consumption of inputs, which are the outputs of previous transactions. The consumption and creation of outputs are cryptographically verified by the signature of the fund owner (for more details, see Section 5).

For nodes to objectively agree on a partial order of events, we require the following assumption.

Assumption 6.1 (Past cone completeness)**.** For a transaction $\hat{x}$ that spends an output created in a transaction $\hat{y}$, it holds that the block x is contained in the Tangle future cone of y, i.e. $x \in \text{cone}_{\mathcal{T}}^{(f)}(y)$.

In other words, we have the natural assumption that the spending of the output should happen in the future cones the blocks "creating" these outputs.

Lemma 6.1. *Under Assumption 6.1, the partial order $\leq_{\mathcal{L}}$ induced by $D_{\mathcal{L}}$ is consistent with the partial order $\leq_{\mathcal{T}}$ induced by $D_{\mathcal{T}}$. More specifically, if for some blocks $x, y \in \mathcal{T}$, we have that the corresponding transactions satisfy $\hat{x} \leq_{\mathcal{L}} \hat{y}$, then it holds that $x \leq_{\mathcal{T}} y$.*

Proof. The statement can be shown trivially by induction on the length of the shortest path between $\hat{x}$ and $\hat{y}$ in $D_{\mathcal{L}}$.

---

8. In contrast, if weights are added but not withdrawn, it is possible that two conflicting transactions gain precisely the same weight which would result in an impasse.

The base case, when the length of the path is one, is implied by Assumption 6.1. □

6.3. Voting and Voting DAG

As a consequence of Lemma 6.1 both, the Tangle and the Ledger DAG, are suitable for nodes to express their opinions about which transactions they prefer among any conflicting transactions. More specifically by creating and attaching new blocks, nodes have an implicit way of voting for the "preferred" branches and conflicts. Let us define this more precisely.

We utilise the references contained in a block, which constitute the edges of the Tangle, see Section 4, to express a node's opinion. As by Definition 4.1 a reference contains two fields: r_x, which is a reference to block x and v, which is the value of a label. We call the label v the *vote type* that can take values in $\{v_{\mathcal{T}}, v_{\mathcal{L}}\}$. This label gives additional meaning to the reference to x in the Tangle and defines the following two specialised references.

Definition 6.1 (Block reference). We say a reference $\mathrm{ref}(y) = (r_x, v)$ from a block y to a block x is a *block reference* if y references x. In this case, we set the label $v = v_{\mathcal{T}}$.

To overcome the liveness issues described in Section 6.1 we additionally add a reference that bypasses the block and directly addresses the contained transaction.

Definition 6.2 (Transaction reference). We say a reference $\mathrm{ref}(y) = (r_x, v)$ from a block y to a block x is a *transaction reference* if y references $\hat{x}$. In this case, we set the label $v = v_{\mathcal{L}}$.

Remark 6.1. *Naturally, a block references the transaction that is the content of the block. As such, an honest node would not issue a block with a transaction that is not in its preferred reality (see Section 6.5).*

Example 6.1. Consider Figure 7. Blocks y and y' contain the same transaction $\hat{y}$, but y refers to the transaction $\hat{x}$ in block x and, thus, issues a transaction reference, while block y' refers to the block x and, thus, issues a block reference, instead.

Remark 6.2. *The distinction into the sub-categories (transaction reference and block reference) is only relevant for the purpose of voting; the definition of the Witness Weight, see Section 4.4, remains unaffected.*

We define a data structure that combines the two immutable data structures in Section 6.2 into one single DAG used for propagating the votes.

Definition 6.3 (Voting DAG). The Voting DAG $D_{\mathcal{V}}$ is a DAG whose vertex set $\mathcal{V}$ is the union of the set of blocks $\mathcal{T}$ and the set of transactions $\mathcal{L}$, i.e. $\mathcal{V} = \mathcal{T} \cup \mathcal{L}$. Let v and u be two vertices in $\mathcal{V}$. There exists a directed edge from u to v in $D_{\mathcal{V}}$ if and only if one of the following properties holds:
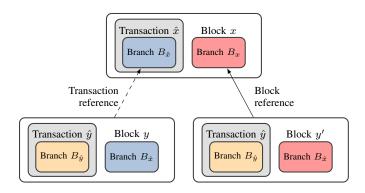1) $u, v \in \mathcal{T}$ and u contains a block reference to v;

Figure 7: Inheritance of branches: we consider two potential blocks y and y' that contain the same transaction $\hat{y}$, but have either a transaction, or a message reference to block x. Thus, a node can vote in two ways. Specifically, block can approve a previous block via a transaction reference or a block reference, and inherit the branch of the referenced transaction or the referenced block, respectively.

2) $u \in \mathcal{T}, v \in \mathcal{L}$ and u contains a transaction reference to transaction v;
3) $u \in \mathcal{T}$ and $v = \hat{u} \in \mathcal{L}$, i.e. v is a transaction in block u;
4) $u, v \in \mathcal{L}$ and transaction u spends the output from transaction v, i.e. $v \in \mathrm{par}_{\mathcal{L}}(u)$.

So far we described how references between blocks are given additional meaning to construct the voting DAG. This DAG allows nodes to express their opinions, recursively. Following Definition 2.7 we define $\mathrm{cone}_{\mathcal{V}}^{(p)}(x)$ as the *voting past cone* of block or transaction x in the Voting DAG.

Definition 6.4 (Voting). A node i expresses a *direct vote* for a vertex $x \in \mathcal{V}$ in the voting DAG $D_{\mathcal{V}}$ by referencing x in a block $y \in \mathcal{T}$, where $\mathbf{issue}(y) = i$. We say node i *indirectly votes* for any vertex in $\mathrm{cone}_{\mathcal{V}}^{(p)}(x)$.

Example 6.2. We illustrate the concept of a Voting DAG in Figure 8. The Voting DAG assembles information from the Tangle and the Ledger DAG. We assume a situation where the node that issues block x does not approve transaction $\hat{y}$ and, thus, can vote neither for blocks y nor z. However, it can vote for transaction $\hat{z}$ by using a transaction vote. More precisely, by creating a transaction reference to block z, the vote of block x avoids vertices $z, y, \hat{y}$ shown in grey.

We can also describe the voting past cone in terms of a recursive equation.

Proposition 6.1. *Suppose a given block $x \in \mathcal{T}$ has block references to block $y_1, \ldots, y_s$ and transaction references to blocks $z_1, \ldots, z_r$ with $s + r = k$. Then the voting past cone of x can be written in a recursive way*

$$\mathrm{cone}_{\mathcal{V}}^{(p)}(x) = x \cup \mathrm{cone}_{\mathcal{L}}^{(p)}(\hat{x}) \cup C_{\mathcal{L}}(x) \cup C_{\mathcal{V}}(x),$$
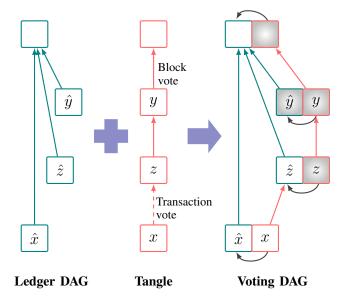
Ledger DAG   Tangle   Voting DAG

Figure 8: Illustration of how the Voting DAG is assembled from the Tangle and the Ledger DAG. By creating a transaction reference to block z, the vote of block x avoids vertices $z, y, \hat{y}$ shown in grey.

where

$$C_{\mathcal{L}}(x) := \operatorname{cone}_{\mathcal{L}}^{(p)}(\hat{z}_1) \cup \ldots \cup \operatorname{cone}_{\mathcal{L}}^{(p)}(\hat{z}_r),$$
$$C_{\mathcal{V}}(x) := \operatorname{cone}_{\mathcal{V}}^{(p)}(y_1) \cup \ldots \cup \operatorname{cone}_{\mathcal{V}}^{(p)}(y_s).$$

The Reality-based Ledger introduces the concept of branches, see Section 5. The consumption of more than one output from different branches creates a new branch, which is the union of the branches of the consumed outputs. Now we extend this concept to blocks, which can combine branches by voting for previous blocks or transactions. More precisely we can relate a given reference in a block with a branch. The branch of the block is then defined as follows.

Definition 6.5 (Voting Branch)**.** Given a block $x \in \mathcal{T}$, we define the *voting branch* of x to be

$$\operatorname{branch}_{\mathcal{V}}^{(p)}(x) := \operatorname{cone}_{\mathcal{V}}^{(p)}(x) \cap \mathcal{C},$$

where $\mathcal{C}$ is the set of conflicts.

Remark 6.3. *We highlight that for the correctness of the protocol, a node has to create references for a new block x in such a way that $\operatorname{branch}_{\mathcal{V}}^{(p)}(x)$ is indeed a branch as defined in Definition 5.9. The property that $\operatorname{branch}_{\mathcal{V}}^{(p)}(x)$ is $D_{\mathcal{C}}$-past-closed trivially follows from the fact that $\operatorname{cone}_{\mathcal{V}}^{(p)}(x) \cap \mathcal{L}$ is $D_{\mathcal{L}}$-past-closed. However, the conflict-free property of $\operatorname{branch}_{\mathcal{V}}^{(p)}(x)$ is not necessarily true in general and has to be checked. We address this issue when we discuss tip selection algorithms in Section 6.5.*

Recall Definition 5.11 that introduces the maximal contained branch of a transaction $\hat{x}$, written as $\operatorname{branch}_{\mathcal{L}}^{(p)}(\hat{x})$.

Using Proposition 6.1, we relate the voting branch of block x and the maximal contained branch of transaction $\hat{x}$ in the following statement.

Proposition 6.2 (Inheritance of branches)**.** *Suppose a given block $x \in \mathcal{T}$ has block references to block $y_1, \ldots, y_s$ and transaction references to blocks $z_1, \ldots, z_r$ with $s + r = k$. A block x inherits the union of the branches that are associated with these votes, i.e. the voting branch can be decomposed as follows*

$$\operatorname{branch}_{\mathcal{V}}^{(p)}(x) = \operatorname{branch}_{\mathcal{L}}^{(p)}(x) \cup B_{\mathcal{L}}(x) \cup B_{\mathcal{V}}(x),$$

where

$$B_{\mathcal{L}}(x) := \operatorname{branch}_{\mathcal{L}}^{(p)}(\hat{z}_1) \cup \ldots \cup \operatorname{branch}_{\mathcal{L}}^{(p)}(\hat{z}_r),$$
$$B_{\mathcal{V}}(x) := \operatorname{branch}_{\mathcal{V}}^{(p)}(y_1) \cup \ldots \cup \operatorname{branch}_{\mathcal{V}}^{(p)}(y_s),$$

Example 6.3. We follow the same example as shown in Figure 7. We assume the maximal contained branch of the transaction in block y is the main branch, i.e. $B_{\hat{y}} = \emptyset$. Block y votes for the transaction contained in block x and, thus, inherits the branch $B_{\hat{x}}$. Since $B_{\hat{y}} = \emptyset$ the voting branch of block y is $B_{\hat{x}}$. Similarly, block y' votes for the block itself and inherits the voting branch B_x. We highlight that the branch of the transaction contained in block y (and y') is not affected by the choice of the vote.

We can associate a given block x with a branch $B_{\mathcal{T}} = \operatorname{branch}_{\mathcal{V}}^{(p)}(x)$. Furthermore, the content of x, which is a transaction $\hat{x}$, also can be associated with a branch $B_{\mathcal{L}} = \operatorname{branch}_{\mathcal{V}}^{(p)}(\hat{x})$. Due to Lemma 6.1 we have that $B_{\mathcal{L}} \subseteq B_{\mathcal{T}}$. Since a node may change its opinion about a conflict and vote for a conflicting transaction to $\hat{x}$, the vote is only valid from the point-of-view of the referencing block y. A later change of the node's vote is possible by issuing another block that votes for a conflicting transaction to $\hat{x}$.

Definition 6.6 (Change of vote and current vote)**.** Let $\hat{x}$ be a transaction for which node $i \in \mathcal{N}$ voted for. Let transaction $\hat{y}$ be conflicting with $\hat{x}$. If node i votes for $\hat{y}$ after it voted for $\hat{x}$, node i is no longer approving $\hat{x}$. We say i *revokes* its vote from $\hat{x}$. If i's most recent vote is approving $\hat{x}$, i.e. the vote is also not revoked, we say i's *current vote* is approving $\hat{x}$.

Remark 6.4. *The notion of "time" and its implications on the meaning of "after" in Definition 6.6 are crucial. Natural choices are the timestamp inside a transaction or the solidification time of a block that contains a given transaction.*

Example 6.4. The principle of Definition 6.6 is demonstrated in Figure 1. Specifically, transactions $\hat{x}$ and $\hat{y}$ are conflicting; block references are depicted with solid edges, whereas transaction references are depicted with dashed edges. Initially, brown and purple nodes voted for $\hat{x}$. However, after a while, green nodes have revoked their votes from $\hat{x}$ and their latest votes are approving $\hat{y}$. The supporters of $\hat{x}$ and $\hat{y}$ are shown in the top-right corners of blocks for each of the two periods.

6.4. Approval Weight and Confirmation Rule for Transactions

Nodes must be able to track the progress of the acceptance of a transaction. We extend the concepts of Witness Weight, introduced in Section 4.4, to the *Approval Weight* (AW) of transactions. The objective is then to define a parameterisable confirmation condition for transactions similar to the one discussed for blocks in Section 4.5.

Definition 6.7 (Transaction Supporters and Approval Weight)**.** Let $\hat{x} \in \mathcal{L}$ be a transaction. Denote by $\mathrm{sprt}_{\mathcal{L}}(\hat{x})$ the set of nodes that has a current vote for $\hat{x}$. These nodes are called *supporters* of $\hat{x}$. We define the function $\mathbf{AW} : \mathcal{L} \to [0,1]$ which is called the *Approval Weight* (AW)

$$\mathbf{AW}(\hat{x}) := \sum_{j \in \mathrm{sprt}_{\mathcal{L}}(\hat{x})} \mathbf{w}(j) \qquad (14)$$

Clearly, the AW describes the percentage of the network approving a given transaction.

Remark 6.5. *The WW of a block x and the AW of its contained transaction $\hat{x}$ are related, but there is no "monotonicity property". If $\overline{x}$ is only contained in the block x we have that $\mathbf{AW}(\hat{x}) \leq \mathbf{WW}(x)$. If $\overline{x}$ is contained in more than one block* [9]*, we can have that the AW of the transaction is even larger than the WW of each enveloping block.*

The supporter of transactions can be updated using propagation of the supporter information through the voting DAG. More precisely on arrival of a block x we traverse $\mathrm{cone}_{\mathcal{V}}^{(p)}(x)$. We propose Algorithm 2 to update transactions supporters when a new block is processed. The AWs are then updated using Equation (14).

Similar to Definition 4.5 we define the confirmation of a transaction. We will use subscripts i and t such as $\mathbf{AW}_{i,t}$ if we talk about the perception of the ith node at moment t.

Definition 6.8 (Confirmed transaction)**.** Let $\theta \in (0.5, 1]$ be a fixed threshold. We say that a transaction $\hat{x} \in \mathcal{L}$ is *confirmed* for a node $i \in \mathcal{N}$ at time t if $\mathbf{AW}_{i,s}(\hat{x}) \geq \theta$, for some $s \leq t$.

We also define the AW of a branch, which will form the base for the algorithm in the next section. The supporters of a branch are equal to the intersection of the supporters of the conflicts in the branch. More formally we have the following.

Definition 6.9 (Branch Supporters and Approval Weight)**.** Let $B \in \mathcal{B}$ be a branch. We define $\mathrm{sprt}_{i,t}^{\mathcal{L}}(B)$ to be the set of nodes that issued blocks that approve all conflicts in B. Similarly we define the AW for B as

$$\mathbf{AW}(B) := \sum_{j \in \mathrm{sprt}^{\mathcal{L}}(B)} \mathbf{w}(j). \qquad (15)$$

We define the AW of the main branch, i.e. the empty set $\emptyset$, to be 1.

---

9. A fund owner may request several nodes to broadcast a transaction $\hat{x}$, or a node may issue several different blocks with $\hat{x}$ contained.

Algorithm 2: Updating transaction supporters when new block arrives

Data: Tangle DAG $D_{\mathcal{T}}$, Ledger DAG $D_{\mathcal{L}}$, new block x issued by node j, $\{\mathrm{sprt}_{\mathcal{L}}(y)\}_{y \in \mathcal{L}}$
Result: updated $\{\mathrm{sprt}_{\mathcal{L}}(y)\}_{y \in \mathcal{L}}$

1 **for** $\forall z \in \mathrm{cone}_{\mathcal{V}}^{(p)}(x) \cap \mathcal{L}$ **do**
2 $\quad$ $\mathrm{sprt}_{\mathcal{L}}(z) \leftarrow \mathrm{sprt}_{\mathcal{L}}(z) \cup \{j\}$
3 **end**
4 **for** $\forall z \in \mathcal{L}$ *that are conflicting with* $\mathrm{cone}_{\mathcal{V}}^{(p)}(x)$ **do**
5 $\quad$ $\mathrm{sprt}_{\mathcal{L}}(z) \leftarrow \mathrm{sprt}_{\mathcal{L}}(z) \setminus \{j\}$
6 **end**

6.5. Tip Selection Algorithm

The consensus protocol relies substantially on an implicit voting mechanism. Nodes express their opinions and votes by choosing the references in their newly issued blocks. The process that determines the references is called the *Tip Selection Algorithm* (TSA) and is discussed in this section.

With every block, a node can vote on which parts of the Tangle and the Ledger DAG it prefers by using block or transaction references. The preferred parts of the Tangle and the Ledger DAG are defined by the preferred reality. Following the algorithm described in Section 5.3, a node i at moment t keeps its preferred reality $R = R_{i,t}$ up to date.

We now describe a tip selection mechanism that considers both block and transaction votes. Note that due to Lemma 6.1 the Ledger DAG induces a partial order consistent with the one induced by the Tangle and, thus, voting on the Ledger DAG allows expressing a more selective, albeit less efficient vote than on the Tangle.

Let us define some reality-dependent tip sets on the Tangle DAG and the Ledger DAG.

Denote by $\mathbf{T}_{\mathcal{T}}(R) \subset \mathcal{T}$ the tips in the Tangle DAG whose Tangle past cones contain only transactions in reality R. More precisely, for any $x \in \mathbf{T}_{\mathcal{T}}(R)$, there is no $y \in \mathrm{cone}_{\mathcal{T}}^{(p)}(x)$ such that $\hat{y} \in \mathcal{C} \setminus R$.

Denote by $\mathbf{T}_{\mathcal{L}}(R) \subset \mathcal{L}$ the tips in the Ledger DAG whose past cones contain conflicts from R only. In other words, it holds that for any $\hat{x} \in \mathbf{T}_{\mathcal{L}}(R)$, there is no $\hat{y} \in \mathrm{cone}_{\mathcal{L}}^{(p)}(\hat{x})$ such that $\hat{y} \in \mathcal{C} \setminus R$. A node should apply the following tip selection and reference setting.

Definition 6.10 (Uniform Random Tip Selection on a reality)**.** To issue a new block, node i chooses tips to approve uniformly at random from all tips in the Tangle DAG until k references are created. For a randomly chosen tip, the node proceeds with the following steps:

1) if the selected block is in the set $\mathbf{T}_{\mathcal{T}}(R)$, a block reference is created;
2) otherwise, if the selected tip contains a transaction that is in the set $\mathbf{T}_{\mathcal{L}}(R)$, a transaction reference is created;
3) if neither of the above apply, the block is discarded instead.

Algorithm 3: Uniform Random Tip Selection restricted on reality R

Data: Tangle DAG $D_{\mathcal{T}}$, Ledger DAG $D_{\mathcal{L}}$, preferred reality $R \in \mathcal{B}$, number of references k

Result: tips $L_{\mathcal{T}} \cup L_{\mathcal{L}}$

1 $L_{\mathcal{T}} \leftarrow \emptyset$
2 $L_{\mathcal{L}} \leftarrow \emptyset$
3 $cnt \leftarrow 0$
4 **while** $cnt < k$ **do**
5 | Choose tip x uniformly at random in $D_{\mathcal{T}}$
6 | Set $Q_{\mathcal{V}}$ to be conflicts contained in $\mathrm{cone}_{\mathcal{V}}^{(p)}(x)$
7 | Set $Q_{\mathcal{L}}$ to be conflicts in $\mathrm{cone}_{\mathcal{L}}^{(p)}(\hat{x})$
8 | **if** $Q_{\mathcal{V}} \subseteq R$ **then**
9 | $cnt \leftarrow cnt + 1$
10 | $L_{\mathcal{T}} \leftarrow L_{\mathcal{T}} \cup \{x\}$
11 | **else**
12 | **if** $Q_{\mathcal{L}} \subseteq R$ **then**
13 | $cnt \leftarrow cnt + 1$
14 | $L_{\mathcal{L}} \leftarrow L_{\mathcal{L}} \cup \{x\}$
15 | **end**
16 | **end**
17 **end**

We call this algorithm the *Uniform Random Tip Selection Algorithm restricted on the reality R* (or R-URTS for short) and refer to Algorithm 3 for a pseudo code.

We refer to Appendix B, where we demonstrate this algorithm as part of an illustrated example.

A node may have voted previously for a branch that is no longer its preferred branch. It has therefore to change its vote. With the above tip selection nodes are allowed to vote for branches they previously did not "prefer" (by voting for a conflicting transaction) and vote "against" branches they previously voted for. Every node must therefore keep the supporters for each branch and their AW up to date. An important consequence is that the AW of certain branches may increase in time while for others it may decrease in time.

The addition of the transaction vote demonstrates that solutions for the Tip Selection Algorithm can be found that mitigate or reduce liveness issues and that transactions eventually will be considered for tip selection. Thus, in the following, we work under the following assumption.

Assumption 6.2 (Block inclusion)**.** Let R be the preferred reality. The tip selection satisfies that for every transaction $\hat{x} \in \mathcal{L}(R)$ (see Definition 5.12) we have that there is at least one element in $\mathrm{cone}_{\mathcal{V}}^{(f)}(x)$ that the tip selection algorithm can pick up. In particular, there exists at least one available tip in the union $\mathbf{T}_{\mathcal{T}}(R) \cup \mathbf{T}_{\mathcal{L}}(R)$.

We also refer to Section 12 for a more detailed discussion.

7. Communication and Adversary Models

Before stating the security requirements of the protocol, we have to make assumptions about the underlying communication model. It is common to describe the uncertainty related to the communication by an attacker that controls the delays of the blocks. The communication model defines the limits the adversary can delay the communication between the nodes. As a model, it is only a simplification, but it allows a systematic study of the most critical components.

For simplicity, we also analyse the voting mechanism without details such as the TSA. We want to emphasise that our modelling can also be applied to other consensus protocols, thus, providing a framework for comparing different DLTs.

7.1. Communication Model

The participating nodes communicate over a peer-to-peer (P2P) protocol or network. In this P2P protocol, nodes send their signed blocks to their neighbouring peers. Neighbours forward blocks from other nodes in the overlay network only if they have verified its validity; if a transaction is invalid, the propagation stops. The transmission of a block between two nodes is done by sending a *package* containing the block.

There are three basic (or classic) models for the P2P communication between the nodes: the synchronous model, the asynchronous model, and the partial synchronous model, e.g. see [60] and [61].

In the *synchronous model*, there exists some known finite time bound Δ by which an adversary can delay the delivery of a package. In the asynchronous model, an adversary can delay the delivery of a package by an unknown finite amount of time. There is no bound on the time to deliver a block but each package must eventually be delivered. In the partial synchronous model, we assume that there is some finite unknown upper bound Δ on block delivery. This bound is not known in advance and can be chosen by the adversary.

A partially synchronous system can be seen as initially asynchronous that becomes eventually synchronous. The time at which the system becomes synchronous is called the Global Stabilisation Time (GST).

We also consider a probabilistic synchronous model, see [49]. In this model we assume that for every $\varepsilon > 0$ and $\delta \in [0, 1]$, a proportion δ of the blocks is delivered within a bounded (and known) time $\Delta = \Delta(\varepsilon, \delta)$, that depends on ε and δ, with probability of at least $1 - \varepsilon$. The probabilistic synchronous model is similar to the asynchronous model with crash failure faults, see [61].

The specific implementations for a consensus mechanism depend heavily on the underlying synchronicity assumption. It also seems appropriate to distinguish between consensus protocols that find consensus on one data set and consensus protocols that find consensus on a growing number of decisions. The latter allows to "strengthen the synchronicity" between the nodes if the data are related by references.

7.2. The Tangle, Solidification, and Synchronicity

The references that form the Tangle are essential for the consistency of information every node has. Consider that a package propagates to only part of the network, e.g. lost during some of the propagation processes on the communication layer. However, nodes that have received the block start building on it and gossip their blocks to the network. These new blocks contain references to the partially missing block. Since nodes must know the past cone of any block to have a complete Tangle history from that blocks' point of view, we use a mechanism called the *solidification process*. In this mechanism, nodes that receive a given block only process it if its past cone is complete or, otherwise, ask their peers for the missing referenced block (for more details, see Section 4.3). In other words, the solidification process is a mechanism to recover lost blocks and, hence, strengthens the "synchronicity" of the communication model. We think that this, to some extent, supports the assumption that all blocks are delivered within a bound time Δ with high probability.

7.3. Adversary Model

We distinguish between three types of nodes: *honest*, *faulty*, and *malicious*. Honest nodes follow the protocol, faulty nodes are not working properly (e.g. not sending any transactions), and malicious nodes are trying to disturb the protocol by not following the rules actively. In most scenarios, we assume that the malicious nodes are controlled by an abstract entity that we call the *attacker*. We assume that the attackers are computationally limited and cannot break the signature schemes or the cryptographic hash functions involved. However, we assume that the attacker is omniscient and "knows immediately" about all state changes of the honest nodes.

In classic consensus protocols, the communication model already covers the adversary behaviours, as delaying blocks is essentially the only way an attacker can influence the system. This is no longer true for our consensus protocol. Here, adversarial strategies can be divided into two main categories: attacks on the protocol level and attacks on the voting layer.

7.4. Configuration Graph and Schedule

How events in distributed systems are triggered depends on some external causes that are often referred to as the environment. We follow [62] and model this environment using the abstraction of a scheduler.

To this end, we consider a communication network on which all communications between the nodes are carried out. These networks are often referred to as P2P networks. We model them using a directed graph whose vertex set is the set of participating nodes. There is a directed edge from i to j, if node i can send packages directly to node j.
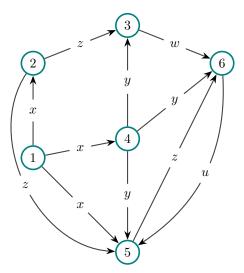

Figure 9: Illustration of a network of 6 nodes. Packages u, w, x, y, z are send along directed edges representing communication channels.

We assume this graph to be connected. Along the directed edges of this graph *packages* are exchanged by the nodes. In our case these packages contain blocks.

Definition 7.1 (Packages and communication graph). For each block x sent from node i to node j we add a directed edge, called *package*, from i to j that is labelled by the vector

$$e(x, i, j) := (x, i, j, t(x), \delta_{i,j}(x)). \tag{16}$$

This label indicates that package x was sent at time $t(x)$ from node i and arrives at node j at time $t(x) + \delta_{i,j}(x)$. The state space of all packages is denoted by $\mathcal{M}$. We dub the resulting graph the *communication graph G* of the protocol.

Essentially a node i does the following once it receives a package e from node j. It checks if the block x that is contained in the package e was already treated. If this is the case, the node's status remains unchanged, and no new package is issued. If the block is new, the node checks its validity and adds the block to its local Tangle. If applicable, it updates the supporters of branches and conflicts and, if the transaction contained in the block is conflicting, adds a new conflict to the set of conflict. After this step, the node forwards the block in new packages to all its neighbours from which the node has not received the block.

A node may also create blocks. Once it creates a block x, it attaches x to its local Tangle. Then, it creates a package e containing block x and sends copies of this package to all of its neighbours.[10]

Example 7.1. We illustrate the concept of networks and packages in Figure 9. In this figure, the network consists of six nodes. Directed edges exist between some of them and show the communication channels. We point out that these

---

10. In this example, we use a flooding protocol for the dissemination of blocks.

communication does not necessarily have to be symmetric. Packages containing blocks can be sent along the edges.

Every node keeps a local version of the Tangle $D_{\mathcal{T}_i}$ that we consider as the (local) configuration ω_i of node i. For ease of presentation, we consider the following simplified version of the OTV that does not keep track of where the actual blocks are attached in the Tangle but only keeps track of the supporters of the branches or conflicts. The (local) state space is therefore given by $\mathcal{Q} = \underbrace{2^{\mathcal{N}} \times \ldots \times 2^{\mathcal{N}}}_{|\mathcal{C}|}$, where $\mathcal{C}$ is a fixed set of conflicts and $2^{\mathcal{N}}$ is the set of all possible subsets of $\mathcal{N} = \{1, \ldots, N\}$.

Remark 7.1. *The simplified version described above allows a more accessible analysis of the voting on conflicting transactions. This comes with the cost of not describing the confirmation of non-conflicting transactions. We give more details on the "liveness" of these transactions in Section 8.1.*

We interpret the packages a node i receives as input assignments with values in the space of all packages $\mathcal{M}$. Each input assignment e yields an update of the current configuration, and each configuration ω_i leads to an output assignment. We therefore consider two functions

$$I(e, \omega_i) : \mathcal{M} \times \mathcal{Q} \mapsto \mathcal{Q},$$

and

$$O(\omega_i) : \mathcal{Q} \mapsto \mathcal{M}^{|\mathcal{N}_i|-1},$$

where $\mathcal{N}_i$ are the neighbours of node i in the communication graph G. A node i runs an algorithm $\mathbf{A} = (I, O)$ that reacts to incoming packages by updating its internal state ω_i and eventual sending outgoing packages indicating its state update. We also consider the configuration of the whole system that takes values $\overline{\omega} = \{\omega_1, .., \omega_N\} \in \mathcal{Q}^N$. The corresponding algorithm is denoted by $\overline{\mathbf{A}}$.

The creation of blocks uses randomness (by design) through the TSA. Moreover, issuing times of blocks may depend on the interactions of the node with the environment of our system. For this reason, we model the time between two successive blocks of one given node by random variables. oreover, the latency between packages of two given nodes is described by random variables. This randomness turns our protocol into a *random* protocol, and the randomness is described by the probability measure $\mathbb{P}$. As we consider a simplified model and are only interested in the supporters of given branches, the randomness enters only in the "time components" of the packages or edges. Consequently, edges become random variables.

Definition 7.2 (Configuration graph). Let e be a package sent from i to j in G and $\overline{\omega}, \overline{\omega}' \in \mathcal{Q}^N$ be two (global) configurations. We write $\overline{\omega} \overset{e}{\to} \overline{\omega}'$ if and only if

$$\mathbb{P}(I(e, \omega_i) = \omega_i') > 0$$

for some i and, $\mathbb{P}(\omega_l' = \omega_l) = 1$ for all $l \neq i$. We say that $\overline{\omega}'$ is *accessible* from $\overline{\omega}$ by e. The notation of accessibility defines a directed graph on the set of (global) configurations that we dub the *configuration graph* of the algorithm.

Definition 7.3 (Valid packages). A package (or edge) e from node i to node j is called *valid* given a global configuration $\overline{\omega}$ if and only if

$$\mathbb{P}(O(\omega_i) \ni e) > 0.$$

In other words, any valid edge must be the outcome of the algorithm $\overline{\mathbf{A}}$. A sequence of edges $e_1, e_2, \ldots$ is called *valid* given an (initial) configuration $\overline{\omega}(0)$ if and only if e_1 is valid given $\overline{\omega}(0)$ and inductively e_ℓ is valid given $\overline{\omega}(\ell-1)$, where $\overline{\omega}(\ell-1)$ is such that $\overline{\omega}(\ell-2) \overset{e_{\ell-1}}{\to} \overline{\omega}(\ell-1)$.

In the following, we assume that honest nodes only issue valid packages.

Definition 7.4 (Communication of configurations). We say that the (global) configuration $\overline{\omega}'$ is *accessible* from a configuration $\overline{\omega}$ if and only if there exists a finite valid path from $\overline{\omega}$ to $\overline{\omega}'$ in the configuration graph. In this case, we write $\overline{\omega} \to \overline{\omega}'$ We define that a configuration is accessible from itself. Two configurations $\overline{\omega}$ and $\overline{\omega}'$ are said to *communicate* if and only if they are accessible from each other. In this case, we write $\overline{\omega} \leftrightarrow \overline{\omega}'$.

The relation $\leftrightarrow$ defines an equivalence relation on the set of configurations.

Definition 7.5 (Communication classes). The equivalence classes of the equivalent relation $\leftrightarrow$ are called the *communication classes* of the set of configurations. A (communication) class is called *closed* if and only if it has no outgoing edges, and *open* otherwise.

The closed communication classes play a vital role as they describe the outcome of the protocol. Let $R \in \mathcal{B}$ be a reality. Then the configuration with $\mathrm{sprt}_{\mathcal{L}_i}(R) = \mathcal{N}$ for all $i \in \mathcal{N}$ is a closed class. Let us note here that we are still assuming all nodes to be honest and behave according to the protocol.

Definition 7.6 (Consensus state). A state ω is called a *consensus state* if and only if

$$\mathrm{sprt}_{\mathcal{L}_i}(R) = \mathcal{N}, \quad \forall i \in \mathcal{N}, \tag{17}$$

for some reality R.

Remark 7.2. *Let us stress that the definition of "consensus state" is only about agreeing on the preferred reality. It does not take into account the meaning of confirmation; see Definition 4.6. Liveness and safety with respect to confirmation are discussed in the following sections.*

We make a crucial assumption about the communication layer.

Assumption 7.1 (Random block issuance and package delay). Block issuances and package delays are random and satisfy:
1) Nodes issue new blocks independently and distributed according to some probability distribution μ_{iss}.
2) The delays of packages between two nodes are independent and distributed according to some probability distribution μ_{pack}.

3) Block issuances and package delays are independent.
4) With a positive probability packages are delivered faster than new blocks are issued. More precisely, if $X \sim \mu_{\text{iss}}$ and $Y \sim \mu_{\text{pack}}$, then $\mathbb{P}(Y < X) > 0$.

Lemma 7.1. *Under Assumption 7.1, for every given configuration $\overline{\omega}$ there exists a consensus state $\overline{\omega_c}$ such that $\overline{\omega} \xrightarrow{e} \overline{\omega_c}$.*

Proof. Let $\overline{\omega}$ be a configuration. We wait until all existing packages and corresponding changes of votes are sent to all other nodes. During this time, no new block is issued with a positive (non-zero) probability. After every node has seen all current blocks every node has the same perception of the supporters of the different realities. In other words, nodes agree on the AWs of the different branches. Now, every node changes its opinion to its preferred reality, issues transactions indicating their change of vote, and gossips them using packages on the communication graph. Once all these packages are seen by all nodes a consensus state is reached. □

There are two immediate consequences of Lemma 7.1.

Corollary 7.1. *Under Assumption 7.1, a communication class is closed if and only if it consists of one consensus state.*

Corollary 7.2. *Under Assumption 7.1 (and in absence of an adversary), the protocol converges ($\mathbb{P}$-almost surely) to a consensus state.*

Definition 7.7 (Schedule)**.** A *schedule* on the communication graph G is a sequence of (finite or infinite) valid edges $e_1, e_2, \ldots$. A (finite or infinite) *execution* of a sequence of edges $e_1, e_2, \ldots$ by $\overline{\mathbf{A}}$ on G is a sequence of configurations $\overline{\omega}(0) \xrightarrow{e_1} \overline{\omega}(1) \xrightarrow{e_2} \cdots$, where $\overline{\omega}(0)$ is the initial (global) configuration.

The above definitions can naturally extend to models that distinguish between honest and adversary nodes. We assume that adversary nodes do not have to follow the algorithm $\mathbf{A}$ but can produce messaging voting for non-preferred realities. On the communication level, adversary nodes may be more potent than honest nodes, i.e. issuing blocks more frequently, and may delay the relaying of honest packages. Nevertheless, we assume that Assumption 7.1 holds for all honest and malicious nodes. We say that the protocol reaches a consensus state if all honest nodes eventually prefer the same reality. Let us denote by $\mathcal{N}_h$ and $\mathcal{N}_a$ the set of honest and malicious nodes. In analogy to the above, we obtain the following result.

Theorem 7.1 (Eventual consistency - random blocks)**.** *Assume Assumption 7.1 to hold for the blocks and packages of honest and malicious nodes and let q be the weight of the adversary. Then, all honest nodes will ($\mathbb{P}$-almost surely) eventually prefer the same reality if $q < 1/2$.*

Proof. Since $q < 1/2$ a consensus state is reached if all honest nodes have the same preferred reality and all nodes know about it, i.e.

$$\text{sprt}_{\mathcal{L}_i}(R) \supset \mathcal{N}_h, \quad \forall i \in \mathcal{N}_h, \tag{18}$$

for some reality R. We have to prove that for every given configuration ω there exists an available consensus state. This is proven similar to Lemma 7.1 together with the situation where an adversary is neither issuing a block nor can delay the honest packages, which occurs with a positive probability under Assumption 7.1. □

8. Liveness and Safety

In the previous section, we were interested in the eventual convergence and proved an optimal result in Theorem 7.1 under the assumption of random blocks issuance and random package delay. This section adds the confirmation status of transactions into our considerations. We divide security into liveness and safety to allow a more detailed and quantitative analysis.

From a general point of view, liveness means that eventually, good things will happen, and safety means that nothing wrong will ever happen. In our situation, this translates into the following. The safety condition is that any two honest nodes should always reach an agreement and that this decision satisfies the specified validity conditions. Furthermore, no two nodes should ever confirm conflicting transactions. The liveness property is that each honest node should eventually make a decision on the confirmation status of a transaction, i.e. in our case all nodes reach the confirmation threshold θ, see Definitions 4.5 and 6.8, eventually.

Remark 8.1. *In general, one requires in addition that the consensus protocol satisfies integrity. Integrity requires that the eventual outcome of the consensus protocol was initially proposed by at least one node. Since in OTV honest nodes always pick a maximal branch, the integrity property is satisfied once the protocol terminates.*

8.1. Non-Conflicting Transactions

Liveness of a non-conflicting transaction is the property that it will eventually be included in the ledger state. In the strongest form, it means that every non-conflicting transaction will be confirmed, see Definitions 4.5 and 6.8. Therefore, the security threshold for liveness is at most a proportion $(1 - \theta)$ of the weight, as an attacker or faulty nodes holding a proportion $(1 - \theta)$ can stop the confirmation by not issuing any blocks anymore.

Liveness is inherently linked with the TSA and the orphanage problem. We assume the following Assumption on the TSA[11] and we refer to Section 6.1 for a discussion.

Proposition 8.1 (Liveness and safety of non-conflicting transactions)**.** *We assume in the asynchronous model that the tip pool size is stationary, and that Assumption 6.2 is satisfied. The weight of the malicious nodes is q. Then, eventually every non-conflicting transaction is confirmed for all honest nodes if $q < 1 - \theta$.*

Proof. Let x be a block containing a non-conflicting transaction $\hat{x}$ and consider an arbitrary honest node i. Each time

---

11. Formally it is more a requirement on the definition of "tips".

this node issues a new block, the probability that it refers to (and votes for) x is positive, due to Assumption 6.2. At this point, it is important to have the second type of reference that allows to only vote for the transaction $\hat{x}$ and not the whole Tangle past cone of x. Let us denote by p_j this last probability for the jth issued block. Then, due to the assumption on the stationarity of the tip pool size, there exists some $\varepsilon > 0$ such that $p_j \geq \varepsilon$ for infinitely many indices j. Assumption 6.2 guarantees the independence of these events, and the Lemma of Borel-Cantelli implies that node i eventually votes for block x. Then, since the number of nodes is finite, all nodes eventually vote for x. $\qquad\square$

Some discussion on the validity of the stationary tip pool size assumption is appropriate. This kind of assumption was also made throughout Section 4.7 as Assumption 4.3. Let us review this assumption in the light of the communication and the adversary model.

An attacker can delay blocks with honest transactions such that the network delay h increases. This, in turn, will inflate the tip pool size and the time to confirmation [63]. In the asynchronous model, this could lead to memory overflow of the nodes or halt confirmation of certain transactions. While this attack is theoretically possible in this model, it is more of a theoretical interest than a practical issue. We also want to note here that nodes do have an efficient way to "synchronize" their perceptions of the Tangle due to the solidification process; see Section 7.2.

On the Tangle layer, the "worst-case scenario" seems to be the following. The adversary issues blocks, referencing already referenced blocks, not removing any tips from the tip pool. Under the assumption that nodes can issue blocks proportionally to their weight, we obtain that $q/(1-q)$ malicious blocks are issued for each honest block. Honest nodes can increase the number of references to keep the Tangle width stationary. More precisely, it is sufficient that the honest nodes' blocks, on average, remove $K := (q/(1-q)) + 1$ tips. In other words, we can choose the number of references $k > K$ to guarantee robustness against this attack. For instance, $q = 1/2$ leads to $K = 2$.

8.2. Conflicting Transactions

Theoretical results on the liveness and safety of conflicting transactions rely heavily on the assumptions of the underlying communication and adversary model. Moreover, the analysis of the OTV protocol is complex: it requires modeling of the networking part, modeling of the weight distribution, and various (even an infinite number of) adversarial strategies. The following section shows that an adversary can hinder consensus finding in specific situations or edge cases. However, we want to emphasize that this interference only influences the liveness of conflicting transactions and that an appropriate TSA guarantees liveness of non-conflicting transactions; see Proposition 8.1. In Section 10, we add a feature to the protocol that allows us to obtain theoretical results on the liveness of conflicting transactions.

9. Impossibility Results and Metastability

Impossibility results play an essential role in the theory of consensus protocols, as they emphasize the limitations and critical edge cases. The most famous impossibility result is the FLP-result, [19], which states that achieving consensus in the asynchronous communication model is in general impossible for deterministic protocols. From a general point of view, this impossibility is due to the possible delay of packages in the P2P communication and the resulting "symmetric" situation that hinders consensus finding.

We will consider the situation of two or more directly conflicting transactions. It is the role of the consensus mechanism to reach an agreement on which transaction should eventually be accepted. One may consider that keeping conflicting transactions in an undecided state, i.e. violating the liveness, is acceptable. However, this is problematic for several reasons. For example, if nodes keep transactions indefinitely undecided, this could drastically inflate the communication required on the voting layer and prevent the pruning capability of the ledger. Transactions that are undecided for a long time can also harm safety. There is always a chance that some node confirms an "undecided" transaction. While the probability of this event might be small, it is still positive, and hence this unlikely event will happen at some point in time. We also note that simply rejecting malicious transactions does not provide a solution since this would allow delayed cancellation of transactions, thus, violating the system's safety.

In this section, we give examples where the liveness and safety of conflicting transactions are not satisfied; more complicated examples can be constructed following the same principles. They constitute an impossibility result in the sense that the proposed protocol does not guarantee liveness or safety under the asynchronous communication model. These situations rely on strong assumptions about the attackers. We distinguish between attacks on the communication level and those on the voting level. By requiring both levels we give a theoretical result when safety cannot be guaranteed, Lemma 9.1.

9.1. Communication Level

We start with an example where an attacker does not take part directly in the voting but only controls the schedule of the honest nodes' blocks. Let us point out that the attacker does not need to control any weight in this scenario.

The first adversary attack is dubbed a *metastability* attack since it tries to keep the honest nodes in an undecided situation. We refer to [50] for more details and analysis of these kinds of attacks. On a conceptual level, these kinds of attacks exploit a situation where the system is kept in a roughly symmetric condition between two incompatible options. Once the symmetric scenario is broken, nodes likely converge quickly on one of the options.

Example 9.1 (Metastability Attack I). We consider $N = 4$ participating nodes $\{1, 2, 3, 4\}$ that communicate directly;
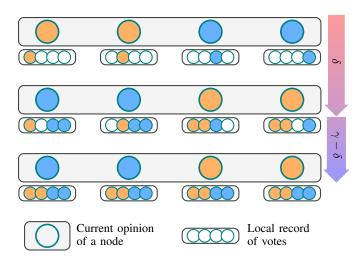
Figure 10: Illustration of Example 9.1. Nodes are voting for transaction $\hat{x}$ (blue) or $\hat{y}$ (orange). Each node ends up with the opposite opinion it started with, thus, creating a deadlock.

the communication graph is the complete graph with four vertices. We assume that every node has the same weight, i.e. $m_i = 1/4$ for all $i \in \mathcal{N}$. We consider the scenario of a simple double spend. The set of conflicts is, therefore, $\{\hat{x}, \hat{y}\}$. We assume for the sake of simplicity that a node prefers its own opinion if both conflicts have 50% of AW. Nodes 1 and 2 starts with an initial like of conflict $\hat{x}$ and nodes 3 and 4 prefer $\hat{y}$. At the time t_0, every node i communicates its vote to each of its neighbors by attaching a block x_i. The attacker delays these blocks (more precisely, the corresponding packages) either by some $\delta > 0$ or $\gamma > \delta$. More precisely, we have the following edges, as defined in Equation (16), in our communication graph:

$$(x_i, i, 3, t_0, \delta), (x_1, i, 4, t_0, \delta), \quad i \in \{1, 2\},$$

$$(x_j, j, 1, t_0, \delta), (x_j, j, 2, t_0, \delta), \quad j \in \{3, 4\},$$

and

$$(x_1, 1, 2, t_0, \gamma), (x_2, 2, 1, t_0, \gamma),$$
$$(x_3, 3, 4, t_0, \gamma), (x_4, 4, 3, t_0, \gamma).$$

At time γ this schedule leads to an inversion of the preferred conflicts, see Figure 10. An attacker that controls the communication level could therefore delay consensus finding arbitrarily. To make the description of the former attacker more formal, we must specify the assumption on the issuance of block and the communication model. For instance, in the synchronous model, with a known upper bound Δ on the network delay, such an attack is successful if the $\delta, \gamma < \Delta$ and the honest nodes issue blocks periodically. In the asynchronous setting, an attacker can adjust the delays δ and γ even if the honest nodes do not continuously issue their transactions simultaneously.

Remark 9.1. *The situation described above is undoubtedly a special case and mainly of theoretical interest. However, it raises the question under which conditions such schedules exist and how realistic they appear in real applications.*

9.2. Voting Level

In this section, we describe situations, where an attacker can successfully interfere in the consensus finding by using the voting layer. We do not need conditions to control communication between honest nodes but relatively strong assumptions about the adversary's ability to issue new blocks and reliably forward them to the honest nodes.

Example 9.2 (Metastability Attack II)**.** We again consider the situation of one double spend, i.e. a set of conflicts $\{\hat{x}, \hat{y}\}$. In this attack, the adversary votes for the minority, i.e. the conflict that has less AW. The attack is supposed not to influence the communication layer, and we work under the assumption of the synchronous communication model. We assume that the propagation of blocks happens fast, i.e. each block causes a state update in all other nodes. Furthermore, we assume that the adversary can issue at a high rate, such that for every other honest node's block, the adversary can issue a block.

We consider an even number N_h of honest nodes and three malicious nodes, and where each node holds the same weight. We say if a node votes for $\hat{x}$ or $\hat{y}$ it is in set X and Y, respectively. The protocol starts with $1/2 N_h$ honest nodes initially voting for $\hat{x}$ and $1/2 N_h$ honest nodes voting for $\hat{y}$. We refer to Figure 11 for an illustration. Next, the adversary votes for $\hat{x}$ (with all three nodes), resulting in a vote of $|X|/|Y| = (1/2 N_h + 3)/(1/2 N_h)$. Nodes in X will continue to vote in favour of $\hat{x}$. On the other hand, an honest node in Y will eventually change its vote and issue a transaction in favor of $\hat{x}$, thus, changing from set Y to X. Now, before any other honest nodes can express their vote, the attacker switches its vote to $\hat{y}$. Hence, in total we have $|X|/|Y| = (1/2 N_h + 1)/(1/2 N_h + 2)$. Honest nodes will now vote for $\hat{y}$. However, as soon as a node from X changes its vote, the resulting situation is symmetric to the initial condition. Thus, the adversary can repeat this ad infinitum.

Remark 9.2. *We want to note that in Example 9.2 the attacker heavily relies on the capability of an adversary to immediately adapt its opinion before more than 2 honest nodes changed their vote to the majority.*

The next example, the *Bait-and-Switch Attack*, depends less on the adversaries issuance rate but requires a higher amount of weight.

Example 9.3 (Bait-and-Switch Attack)**.** We consider a situation where the adversary possesses the node with the highest weight. The strategy is to switch frequently the opinions such that the honest nodes are constantly " chasing the ever-changing heaviest branch". For example, consider N_h honest nodes with total weight w_h and individual weight
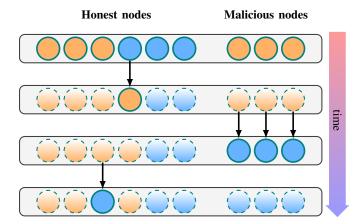
Figure 11: Illustration of Example 9.2. Nodes are voting for transaction $\hat{x}$ (blue) or $\hat{y}$ (orange).

w_h/N_h and one adversary node with weight w_a. Let n_{cr} be the largest natural number such that

$$n_{cr} \cdot \frac{w_h}{N_h} < w_a.$$

In the beginning, the malicious node spends an output in a transaction $\hat{x}_1$. Then, before n_{cr} nodes with a total weight of less than w_A express their vote, the adversary spends the same output in transaction $\hat{x}_2$, i.e. creates a conflicting transaction with $\hat{x}_1$, and (implicitly) votes for the new transaction $\hat{x}_2$. Since $\hat{x}_2$ becomes the heaviest branch, all honest nodes will vote for this transaction. The adversary repeats this procedure by creating additional double spends repetitively.

9.3. Communication and Voting Level

In the previous sections, we presented examples of how an adversary can harm the liveness of conflicting transactions. The attacker strategies required either substantial control of the communication layer or a high issuance rate combined with considerable weight. In this section, we prove an impossibility result for safety that involves an attack strategy that uses both levels.

Definition 9.1 (Broken safety). We say that *safety is broken* if and only if there exist two honest nodes i, j and conflicting transactions $\hat{x}$ and $\hat{y}$ such that for some times s, t we have

$$\mathbf{AW}_{i,t}(\hat{x}) > \theta \text{ and } \mathbf{AW}_{j,s}(\hat{y}) > \theta.$$

We have the following "negative" result.

Lemma 9.1. *Let $q > \theta - 0.5$ be the weight of the adversary. Assume that the weight of the honest nodes is equally distributed on sufficiently many honest nodes. Then, there exists an adversary strategy that breaks safety.*

Proof. Let us choose a number of honest nodes N_h sufficiently large such that there exists some $N_h^* < N_h$ such that

$$\frac{\theta - q}{1 - q} < \frac{N_h^*}{N_h} < \frac{0.5}{1 - q}.$$

An attacker starts issuing two conflicting transactions $\hat{x}$ and $\hat{y}$. The attacker decomposes the honest nodes into two groups X and Y such that each of these groups forms a connected subgraph of the underlying communication layer, while the attacker is connected to both groups. Group X consists of N_h^* nodes and group Y of $N_h - N_h^*$ nodes. The attacker interferes with the schedule such that nodes in each group only receive blocks from their group. The attacker changes the schedule such that the nodes in X receive transaction $\hat{x}$ before $\hat{y}$ and the nodes in Y receive $\hat{y}$ before $\hat{x}$. All honest nodes prepare their initial statement of their preferred transaction ($\hat{x}$ for group X and $\hat{y}$ for group Y) and send them to their neighbours.

The attacker sends to X blocks that state that it prefers $\hat{x}$. As a consequence, nodes from X confirm transaction $\hat{x}$ since $\mathbf{AW}(\hat{x}) = (1 - q)\frac{N_h^*}{N_h} + q > \theta$.

After this, the attacker sends blocks to Y (and X) that it votes now for transaction $\hat{y}$. Without the vote of the attacker for transaction $\hat{x}$ the AW of $\hat{x}$ in X reduces to $\mathbf{AW}(\hat{x}) = (1 - q)\frac{N_h^*}{N_h} < 0.5$.

Next, the attacker lets X know about the preferences of Y. At this point $\mathbf{AW}(\hat{y}) > \mathbf{AW}(\hat{x})$ and as a consequence nodes from X update their preferred reality and vote for $\hat{y}$. This eventually leads to $\mathbf{AW}(\hat{y}) > \theta$ for all nodes. As by Definition 9.1 safety is broken. $\square$

The above proof indicates that the attacker needs very strong control over the communication layer to conduct such an attack. Nevertheless, it gives a reasonable theoretical security threshold for the protocol's safety. All the more since we can prove safety under the assumption $q < \theta - 0.5$ in Section 10.

9.4. Realistic Conditions

The above examples illustrate that the *two dimensions*, namely the communication and voting level, may interact either in favor of the attacker or in favor of the robustness of the protocol. In all cases, it seems that the attacker needs excellent control of the communication layer of the protocol. Randomness or uncertainty on the communication layer may interfere with the adversary strategy and finally lead to convergence of the honest nodes' opinions.

We conjecture that these strong assumptions are not met in most reasonable real-world scenarios and that the attacks that rely solely on the communication level are hard to perform in practice.

With a completely random schedule of packages, the system will eventually converge to a consensus state in situations where an attacker controls not more than half of the total weight, see Theorem 7.1. However, this convergence time can be *impracticably long* for real-world applications and it is possible that safety (for the confirmation) can be

broken as shown by Lemma 9.1. The theoretical treatment of the inherent randomness of real-world implementation systems is at best in an early state, and a quantification or even its control seems currently out of reach. We refer to [62] for a theoretical approach to describe the entropy related to the scheduling of the transactions.

The following section proposes a more sophisticated variation that allows a more straightforward theoretical treatment and provides the "optimal" safety thresholds.

10. Synchronized Random Reality Selection

In the previous section, we demonstrated that under several conditions, the protocol presented so far might lead to situations where nodes cannot come to an agreement between several valid options. This section offers a mechanism to overcome this scenario by utilising external randomness. As shown in [50], [64], [65] common randomness can successfully navigate a system away from such an undesired situation.

Pre-consensus classes are those classes from which the network reaches a consensus eventually. The aim of the design of the consensus protocol is, therefore, to construct the protocol so that its global state reaches such a pre-consensus state fast and that from there, the actual consensus state is inevitable.

The OTV is an asynchronous protocol and comes with advantages and disadvantages. One disadvantage is the lack of synchronization possibilities between nodes that could be used against adversarial attacks on the communication level. The arguments and examples in the previous section showed that it is theoretically possible for an attacker to keep the honest nodes in an undecided situation for a long time. To exclude these cases and obtain theoretical results, we use a distributed random number generation (dRNG) process to synchronize the nodes and interfere with a possible adversary.

We choose a parameter $\mathcal{D}$ describing the length of epochs between synchronizations times. In other words, once in every $\mathcal{D}$ time units, we synchronize the nodes with the help of a given dRNG process. This procedure is inspired by the paper [64], where a dRNG is used to construct a voting-based consensus protocol in a Byzantine environment. The dRNG allows the consensus protocol to reach a pre-consensus state with a positive (non-zero) probability. This probability is uniform in the opinions and votes of the nodes, and hence, the protocol enters a pre-consensus class in a geometrically distributed number of periods of length $\mathcal{D}$. In the last step, we then prove that consensus is reached from the pre-consensus state.

We consider a system of $N = N_h + N_a$ nodes with N_h honest nodes and N_a adversarial nodes. The honest nodes are identified with the set $\mathcal{N}_h = \{1, \ldots, N_h\}$ and the adversarial nodes with $\mathcal{N}_a = \{N_h + 1, \ldots, N_h + N_a\}$.

We start with stating our model assumptions.

Assumption 10.1. We make the following assumptions:

10.1.1 Every block from an honest node is received by another honest node during time $\mathbf{d} = \mathbf{d}(\varepsilon)$ with probability of

at least $1 - \varepsilon$. The constant $\varepsilon > 0$ can be chosen arbitrarily small. The events for each block are independent of each other.

10.1.2 The adversary controls a proportion q of the weight. The adversary might have an influence on the schedule of the blocks to the extent of 10.1.1.

10.1.3 The set of conflicts $\mathcal{C}$ is fixed and does not vary in time. All nodes perceive the same $\mathcal{C}$.

10.1.4 There exists a dRNG that publishes a random variable every $\mathcal{D}$ unit of times. The random variable is uniformly distributed on the interval $[0.5, \theta]$, where θ is the confirmation threshold; see Section 4.5. This value is received (independently) by every given node before time $\mathbf{d}$ (in every epoch) with a probability of at least $1 - \varepsilon$.

10.1.5 Honest nodes of cumulative weight of at least θ issue blocks expressing support for their preferred reality[12] at least every $\mathcal{D}/2$ time units with a probability of at least $1 - \varepsilon$.

Let us comment on the validity of the above assumptions. Assumption 10.1.1 is essentially a probabilistic synchronicity assumption. The fact that the probability ε can be chosen arbitrarily small is supported by the fact that votes are blocks in the Tangle that can be re-broadcast or obtained by solidification requests; see Section 7.2. The independence assumption is essential and the study of correlated errors is out of the scope of this paper. Assumption 10.1.2 is natural in a probabilistic synchronous model. Assumption 10.1.3 is essentially for ease of presentation. As nodes will consider only conflicts of a certain age, older than $\mathcal{D}$, Assumption 10.1.1 ensures that nodes already have the same perception of the sets of conflicts with a very high probability. Assumption 10.1.4 was used in previous work, [50], [64], [65]. A sequence of such common random numbers can be either provided by an external source or generated by the nodes of the system themselves; see e.g. [66]–[71]. Let us stress that it is necessary that the randomness of the dRNG is not predictable and obtained in each epoch by the majority of the weight with a positive probability. However, we do not require that all honest nodes agree on this random number.[13] The last Assumption 10.1.5 is an (almost) necessary condition to ensure that transaction have a chance to be confirmed.

In the beginning, before time $\mathcal{D}$, the AWs for each conflict $c \in \mathcal{C}$ grow through votes according to the mechanism described in Section 6.5. At the end of this initial interval, every node has its own perception of the AW of a conflict c, written as $\mathbf{AW}_{i,\mathcal{D}}(c)$.

After the arrival of the first dRNG randomness X (between $\mathcal{D}$ and $\mathcal{D}+\mathbf{d}$), every honest node chooses its preferred reality and adheres with it during the next interval of length $\mathcal{D}$.

---

12. In other words, for every conflict c in the preferred reality the node issues at least one block stating that the node votes for this conflict c and it doesn't issue any block stating that the node votes for a transaction conflicting with c.

13. The idea is that a "weak consensus" on the randomness of the dRNG leads to an eventual "strong consensus" on the ledger state.
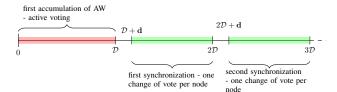
Figure 12: The different epochs in the synchronisation.

Algorithm 4: Reality selection algorithm with common coin

Data: Conflict Graph $G_{\mathcal{C}} = (\mathcal{C}, E)$, common randomness X distributed uniformly in $[0.5, \theta]$

Result: preferred reality $R \in \mathcal{B}$

1 $R \leftarrow \emptyset$
2 $U \leftarrow \mathcal{C}$
3 **while** $|U| \neq 0$ **do**
4    $c^* \leftarrow \arg\max\{\mathbf{AW}(c) : c \in \max_{\mathcal{C}}(U)\}$; /* use max hash(c) for breaking ties */
5   **if** $\mathbf{AW}(c^*) > X$ **then**
6       $R \leftarrow R \cup \{c^*\}$
7       $U \leftarrow U \setminus \{N_{\mathcal{C}}(c^*) \cup \{c^*\}\}$
8   **else**
9   break the while-loop
10   **end**
11 **end**
12 **while** $|U| \neq 0$ **do**
13    $c^* \leftarrow \arg\max\{\text{hash}(c||X) : c \in \max_{\mathcal{C}}(U)\}$
14    $R \leftarrow R \cup \{c^*\}$
15    $U \leftarrow U \setminus \{N_{\mathcal{C}}(c^*) \cup \{c^*\}\}$
16 **end**

In Algorithm 4, we describe an iterative procedure, inspired by [72], for choosing a preferred reality by a node. First, it initialises set R to be the empty set and U to be the set of conflicts $\mathcal{C}$. At every step of the first while-loop, the node finds a conflict c^* in U with the highest AW. If $\mathbf{AW}(c^*) > X$, then we add c^* to R, remove all transactions from U conflicting with R and repeat this step. We additionally require c^* to be from $\max_{\mathcal{C}}(U)$ (see Definition 2.6) to guarantee that after adding c^* to R, the updated set R is a branch. If $\mathbf{AW}(c^*) \leq X$, then we run the next iterative procedure (while-loop) which updates R by c^*, where c^* is the conflict c in U attaining the largest hash of the concatenation $c||X^{14}$ and proceed similarly until U becomes empty. By construction, the resulting set R is a maximal branch or a reality. We summarize these results in the following proposition.

Proposition 10.1. *The resulting set R in Algorithm 4 is a reality.*

Denote by $\text{sprt}_{\mathcal{L}_{i,t}}^{(h)}(\hat{x})$ the set of honest nodes seen from node i at time t that issued a block that votes for

---

14. We assume that $c||X$ can be treated as a binary string for a proper usage of the hash function as noted in Remark 4.1

---

a transaction $\hat{x}$ (for a similar definition of supporters, see Definition 6.7). The *honest AW* of $\hat{x}$ seen from node i at time t is defined as

$$\mathbf{AW}_{i,t}^{(h)}(\hat{x}) := \sum_{j \in \text{sprt}_{\mathcal{L}_{i,t}}^{(h)}(\hat{x})} \mathbf{w}(j)$$

Due to Assumption 10.1.5 and since the honest nodes change their vote at most once, every other honest node sees this vote with a very high probability. In other words, every honest node has the same perception of the votes of all other honest nodes (with high probability). In this case, we can speak of the *honest AW seen by the honest nodes* of a transaction $\hat{x}$:
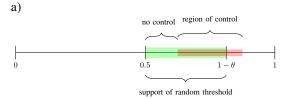
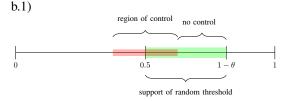$$\mathbf{AW}_t^{(h)}(\hat{x}) := \mathbf{AW}_{1,t}^{(h)}(\hat{x}) \qquad (19)$$

if it holds that $\mathbf{AW}_{i,t}^{(h)}(\hat{x}) = \mathbf{AW}_{j,t}^{(h)}(\hat{x})$ for all $1 \leq i, j \leq N_h$.

Adversarial nodes may change their opinions. In particular, they can do this close to the threshold time $\mathcal{D}$ such that honest nodes may have different perceptions of the adversarial votes. However, this difference in perception is bounded by the weight of the adversary. For every $c \in \mathcal{C}$ we define, similar to [72], the *regions (or intervals) of adversarial control* as

$$I_t(c) = [\mathbf{AW}_t^{(h)}(c), \mathbf{AW}_t^{(h)}(c) + q]; \qquad (20)$$

see Fig. 13. The lower (resp. upper boundary) of this interval is precisely the overall AW of the conflict when all malicious nodes vote against (resp. for) it.
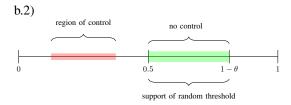
a)

b.1)

b.2)

Figure 13: Region of adversarial control.

We summarize the above considerations in the following statement.

Algorithm 5: Voting protocol for a node i

---
1 $e \leftarrow 0$;   /* epochs index */
2 $X_e \leftarrow 0$;   /* common random variable */
3 **while** *the node did not confirm a reality* **do**
4 | Obtain reality R_i by Algorithm 4 with X_e
5 | Before time $(e+1)\mathcal{D}$ issue new blocks with
   | references selected by Algorithm 3 for R_i
6 | Before time $(e+1)\mathcal{D}$ receive new blocks
7 | $e \leftarrow e+1$
8 | Wait time $\mathbf{d}$ to get common r.v. X_e
9 **end**

Lemma 10.1. *Assume that the honest nodes have the same perceptions on the honest AWs. Then, for all i, $1 \le i \le N_h$, it holds that*

$$\mathbf{AW}_{i,t}(c) \in I_t(c). \tag{21}$$

The above holds for every adversary strategy that satisfies Assumption 10.1.2. The idea is now to choose the support of the dRNG in such a way that independent of the honest AWs and the adversarial strategy all honest nodes will decide on the same reality with a positive probability. Every $\mathcal{D}$ time units we have therefore also a positive probability that all nodes decide on the same reality. It takes, hence, a geometrically distributed number of such intervals until all honest nodes agree on the same reality.

Definition 10.1. [Convergence to a consensus state] We say that the protocol *converges to a consensus state* if and only if there exist some reality R and some (random) time T such that

$$\mathbf{AW}_{i,t}(R) > \theta, \forall i \in \{1, \dots, N_h\}, \forall t > T. \tag{22}$$

Remark 10.1. *Definition 10.1 is similar to the definition of a consensus state; see Definition 7.6. While it describes the asymptotic behaviour of the protocol, it delivers not a practicable criterion for confirmation.[15] A "confirmation rule", as in Definition 4.6, however, is always susceptible to possible "re-orgs"[16] of the ledger state; see also Lemma 9.1. Quantifying the probabilities that such re-orgs happen depends on the precise communication and adversarial models and is out of this paper's scope.*

This discussion can be turned into a formal protocol description written in Algorithm 5 and we obtain the following theorem.

Theorem 10.1 (Liveness and safety - Synchronisation). *Let*

$$q < \min\left\{1 - \theta, \theta - \tfrac{1}{2}\right\}$$

be the weight of the adversary. Then, under Assumption 10.1, the protocol (described by Algorithm 5) converges to a consensus state.

---

15. The "probabilistic" reason for this is that T is not a stopping time.
16. A re-org is the procedure that a transaction that was confirmed is no longer in the preferred reality.

Proof. We start the protocol at time $t_0 = 0$ with a fixed set of conflicts $\mathcal{C}$ of size $|\mathcal{C}|$ and let the nodes exchange their votes until time $\mathcal{D}$. We let $\varepsilon > 0$ be arbitrary but fixed and determine its value at the end of the proof. Every node waits until time $\mathcal{D} + \mathbf{d}$. If the node received the first random number X_1 it will perform Algorithm 4 with X_1 as a random number. If a node did not receive the random number on time it will use Algorithm 4 with the threshold of θ (instead of random X_1). Between $\mathcal{D} + \mathbf{d}$ and $2\mathcal{D}$ every honest node will not change its preferred reality. Let A_1 be the event that all honest nodes voted for their preferred reality and that these votes are seen by all other honest nodes. Let B_1 be the event that all honest nodes expressed their preferred reality on time, see Assumption 10.1.5, and C_1 that all these blocks arrived at every other honest node before time $2\mathcal{D}$. Since $A_1 = B_1 \cap C_1$ we have that

$$\begin{aligned}
\mathbb{P}(A_1) &= \mathbb{P}(C_1|B_1)\mathbb{P}(B_1) \\
&\ge (1-\varepsilon)^{|\mathcal{C}|N_h}(1-\varepsilon)^{N_h} \\
&= (1-\varepsilon)^{N_h(|\mathcal{C}|+1)}.
\end{aligned}$$

At time $2\mathcal{D} + \mathbf{d}$ with probability of at least $(1-\varepsilon)^{N_h}$ the new random number X_2 is received by all honest nodes. Hence, with probability

$$p(\varepsilon) := (1-\varepsilon)^{N_h(|\mathcal{C}|+1)}$$

all honest nodes agree on the honest AWs, defined in Equation (19) and the threshold X_2. We write $\mathbf{AW}^{(h)}(c) := \mathbf{AW}_{2\mathcal{D}}^{(h)}(c)$. Let us note here that no honest node can perceive the honest AW. However, for the analysis, we assume a *perfect view* or total information on the status of the system.

We start a recursive argument on the Conflict Graph by initialising $R = \emptyset$ and $U = \mathcal{C}$. Define the conflict chosen by Algorithm 4 inside the first while-loop at every iteration $c^* := \arg\max\{\mathbf{AW}^{(h)}(c), \quad c \in \max_{\mathcal{C}}(U)\}$. We distinguish two cases.

Case A: $\mathbf{AW}^{(h)}(c^*) > 0.5$. The support of the random threshold does lie above 0.5; see also Figure 13. More, precisely, the probability ξ_A that every node will include this conflict in its preferred reality (using Algorithm 4) satisfies $\xi_A > \mathbf{AW}^{(h)}(c^*) - 0.5 > 0$. All conflicts that conflict with c^*, i.e. the neighbours in the Conflict Graph $N_{\mathcal{C}}(c^*)$, are not preferred. Note here, that since every honest node might have a different perception of the actual AWs, it may run Algorithm 4 in a different "order". However, as no two neighbours in the Conflict Graph can have more than 0.5 of the honest AW, the algorithm treats all "A cases" before the following case.

Case B: $\mathbf{AW}^{(h)}(c^*) \le 0.5$. In this case, all conflicts in $c^* \cup N_{\mathcal{C}}(c^*)$ have an honest AW of less than 0.5. (This is because, in Algorithm 4, nodes treat conflicts in the order of "decreasing AW".) Since $q < \theta - 0.5$, with a positive probability ξ_B none of these conflicts will have AWs above the threshold X_2 and none of them will be added to the preferred reality in the first while-loop of Algorithm 4.

We now remove the conflicts $c^* \cup N_{\mathcal{C}}(c^*)$ from the set U and continue this procedure until the set U is the empty set.

We set $\xi = \min\{\xi_A, \xi_B\}$. Let K be the size of the largest maximal independent set in the Conflict Graph. Eventually, with a positive probability of at least ξ^K the nodes agree on the preferred conflicts originating from case A. The nodes have to fill up the maximal branch with the second while-loop in Algorithm 4. Since they agree on the value of X_2 they also agree on the preferred reality.

Altogether, with a positive probability of at least $p(\varepsilon) \cdot \xi^K$ all honest nodes vote for the same reality during the next epoch of length $\mathcal{D}$. If this happens, an AW of more than θ is obtained in the next epoch. Otherwise, we repeat this procedure until it is satisfied. The number of epochs necessary follows a geometric random variable. □

Remark 10.2. *The above proof offers a possibility to estimate the "consensus time" T. In fact, its expectation is bounded above by $\mathcal{D} \cdot (1 + (p(\varepsilon) \cdot \xi^K)^{-1})$. This quantitative analysis is one main difference to Theorem 7.1, where no bounds on the "consensus time" are obtained. Another crucial difference is that Theorem 10.1 does not require assumptions on the randomness of the packages and issuance as in Assumption 7.1.*

Remark 10.3. *The assumption that the set of conflicts is fixed reduces to the assumption that the set of conflicts is bounded during the run-time of the protocol. The results, therefore, also apply to sets of conflicts that may evolve over time. However, the quantitative bounds in the proof get worse for larger sets of conflicts.*

11. Performance studies

We summarize some of the performance analysis obtained in [73] via agent-based simulations to validate the performance of the presented concepts. The used simulator [74] is written in Go and is open source. In this simulator, the necessary components of the consensus protocol are implemented, however, some of them are simplified. In the following we give a short description but refer to [73] for more details and further simulation results.

The simulated environment reflects a situation in which network participants are connected in a peer-to-peer network, where each node has the same number of neighbors. Nodes can gossip, receive blocks, request for missing blocks, and state their opinions whenever conflicts occur. The underlying network topology is modeled by a Watts-Strogatz network. In order to mimic a real world behaviour the simulator allows to specify the network delay and packet loss for each node's connection.

Nodes are modeled as different independent agents that concurrently issue new blocks. This means that different nodes can have different perceptions of the Tangle and Approval Weights, at any given moment of time. The number of nodes does not change during the simulation period, and all the honest actors are actively participating in the consensus mechanism. While the simulator allows to model different weight distributions, we focus here on the case of a Zipf distribution with $s = 0$, i.e. every node has the same weight.
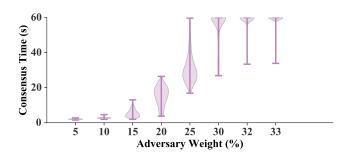

Figure 14: Consensus time distributions under Bait-and-Switch attack, without SRRS ($N = 100$), taken from [73].
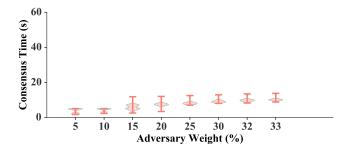

Figure 15: Consensus time distributions under Bait-and-Switch attack, with SRRS ($N = 100$), taken from [73].

Here, we focus on the robustness of the consensus protocol against the Bait-and-Switch attack, 9.3, and illustrate the influence of the Synchronized Random Reality Selection (SRRS) introduced in Section 5.3.

We present simulation studies with the following specific setup. We consider $N = 100$ honest nodes with equal weight and one adversary node with weight q (out of a total weight of 1). The block issuance time interval of nodes follows a Poisson distribution with issuance rates proportional to the nodes' weight. The total throughput is approximately constant at about 100 blocks per second. The parents count (or number of references) is set to $k = 8$. The default confirmation threshold is set to $\theta = 2/3$. The peer-to-peer network is a realization of a Watts-Strogatz network with rewiring probability 1 and 8 neighbors for each node. The latency between two nodes in the peer-to-peer network is set to be 0.1 seconds and we assume the adversary to have no influence on the communication layer. The maximal simulation time is set to 60 seconds.

The access to all Tangles of all nodes in the simulator allows to "objectively" measure the confirmation time as proposed in [73] for each node. These can be combined to extract the consensus time, which is defined as the time between the creation of a conflict and the time when all honest nodes confirm the same spending or branch. As such, for any given conflict, it is strictly larger than the confirmation time at any node. By measuring the consensus time, the safety and liveness of the protocol can be analyzed.

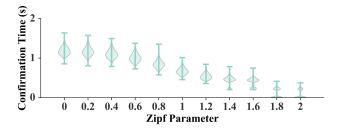Figure 14 shows the consensus time for the Bait-and-

Figure 16: Confirmation time distributions of blocks with the Zipf parameter s, taken from [73].
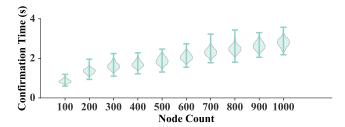

Figure 17: Confirmation time distributions of blocks with the number of nodes, for $s = 0.9$, taken from [73].

Switch strategy as a function of the adversarial weight if SRRS is disabled. It is interesting to note that there this some "inherent randomness" in the protocol as blocks are issued randomly. This seems sufficient to guarantee the security against an attacker with at most 20% of total weight. In Figure 15 we see the effectiveness of the SRRS, that makes the protocol robust against the Bait-and-Switch attack up to the theoretical limit of $q = 1/3$.

We conclude this section with a brief analysis of the performance with the degree of decentralization and the size of the network. This also allows to support the values for the growth of the Witness Weight in Figure 5. Figure 16 shows the confirmation time distributions for several Zipf parameters s with $N = 100$. The confirmation time increases with the "decentralization" of the network, as also discussed in Section 6.4. Nevertheless, Figure 5 shows, that in the extreme case where all nodes have equal weight, i.e. $s = 0$, transaction are still confirmed within 2 seconds. In Figure 17 we show the dependence of the confirmation times with respect to the size of the network, for $s = 0.9$. As described in Section 5, the Witness Weight increases slower with a larger number of nodes. However, as Figure 16 shows the increase is sublinear, resulting in low confirmation times of ~ 3 seconds, even for 1000 nodes.

12. Outlook - Future Research

The proposed consensus mechanism in combination with the Reality-based Ledger supports the parallelisation of many processes, such as processing, booking and voting. This can lead to a significant performance boost since it can enable multi-threaded concurrency. The potential for multi-threadedness of our solution, the capability to work in an asynchronous setting and the leaderless approach can offer a highly performant consensus and ledger solution. Detailed and sound performance analysis will be necessary to validate theoretically predicted properties.

Since the ledger can be progressed without having global knowledge of new transaction additions to the ledger, it is possible that nodes can reach consensus with our mechanism even without learning about all blocks. As a consequence, the approach may enable certain sharding solutions directly on the Tangle layer, in which nodes only observe a proportion of the total ledger. However, this approach may lower performance and potentially lower security and/or liveness. To address the viability of our solution for a sharded scenario key questions such as necessary assumptions and a full security analysis are vital.

The weight system from which the Approval Weight is derived can be constructed from multiple sources and in various settings. For example, the weight may be derived from the token value and the system can be operated permissioned or permissionless. A different approach is to obtain the weights through reputation systems, which has so far received little attention.

By introducing the transaction reference in addition to the block reference in Section 6, the orphanage of transactions can be reduced through Algorithm 3. However, it does not solve the problem entirely. For instance, an honest transaction can be referenced (directly) only by eventually rejected transactions and may never reach sufficient AW to be considered confirmed. This can be improved in several ways. First, nodes may keep their "own" transactions as tips until they are confirmed. This resembles an automated way of reattaching blocks. Second, nodes may also retain transactions that are in their preferred reality but for which they have not yet voted for in the tip pool. The transactions may then be supported via a transaction reference. Third, one could allow block and transaction references to be conflicting for a given block. The transaction can then be prioritised over block references in a transaction. This enables an efficient way to remove parts of branches from the referenced aggregated branch. Another possible solution for a more accurate voting is to introduce more reference types which would eventually allow nodes to remove more explicitly certain branches from the supported branches of referenced blocks. The above examples demonstrate that solutions for the Tip Selection Algorithm can be found that mitigate or reduce orphanage, however, they require thorough analysis to cover edge cases.

13. Conclusion

We have introduced a new leaderless consensus protocol that can be seen as a generalisation of the Nakamoto consensus. Our protocol is based on the Tangle, which not only forms a partially ordered communication record between participants in a peer-to-peer network, but also serves as an efficient way to implicitly vote on the history of the underlying ledger. These nodes are associated with reputation-based

weights which are used to reach consensus on the acceptance of transactions to the ledger. The leaderless nature of the protocol allows asynchronous and concurrent writing access to the ledger. It also eliminates the need for shared "memory pools" for pending transactions and the special roles of miners or validators.

We provide formal definitions and proofs for the functionalities of the protocol, as well as pseudo-code for the various core algorithms. Furthermore, liveness and security of the protocol are analysed and several attack scenarios discussed in detail. We proved an impossibility result for safety in the asynchronous communication model. However, by introducing a synchronisation mechanism that utilises a common random coin, we proved theoretical results on the safety of the protocol. Finally, we presented initial simulation studies that confirm the performance of the protocol with confirmation times in the order of second, and robustness up to a theoretical upper bound of the adversary weight of $1/3$.

Appendix A.
Estimates on Confluence Time

This section gives an upper bound on the confluence time τ_c.

In the case where the network is in a low load regime, we can assume that the tip pool size is small. Then after several approvals, all new transactions will indirectly reference this transaction. In the high load regime, the tip pool size $L_0 \gg k$ and the confluence time can be larger. Denote $K(t)$ the number of tips that approve the given transaction x at time t. A new transaction at time t chooses k tips based on the state of the Tangle at time $t - h$. Hence, the probability of a new transaction approving at least one of the $K(t - h)$ tips that are approving x is given by

$$1 - \left(1 - \frac{K(t - h)}{L_0}\right)^k. \tag{23}$$

As mentioned above, during a time interval h we have that λh new tips arrive and λh tips are approved. Hence, the probability that a transaction that was a tip at time $t - h$ is no longer a tip at time t is

$$\frac{\lambda h}{L_0} = \frac{k - 1}{k}. \tag{24}$$

Therefore, at time t we have that $(1)/kK(t - h)$ previous tips are still tips and $(k-1)/kK(t-h)$ have been referenced and are no longer tips. We denote by A the set of the tips referencing x that are still tips and by B the tips referencing x that got approved in $[t - h, h]$. We write

$$p_A = \frac{K(t - h)}{kL_0}, \text{ resp. } p_B = \frac{(k - 1)K(t - h)}{kL_0} \tag{25}$$

for the probabilities to choose a given parent from the set A, resp. the set B. Let p_1 be the probability to approve at least one transaction from B but not from A and let p_2 be the probability that at least two parents are chosen from the

set A. Let Y_A be the number of tips approved from set A. Then, note that in the first event, the number of tips that reference the given transaction increases by a factor 1 and in the second event the number of tips decreases by a factor $Y_A - 1$.

The probability of the first event can be described by a binomial distribution. In fact,

$$p_1 = \sum_{i=1}^{k} \binom{k}{i} p_B^i (1 - p_A - p_B)^{k-i}. \tag{26}$$

Since p_B is assumed to be small the two leading terms are for $i \in \{1, 2\}$ and we obtain

$$p_1 \approx kp_B + \frac{1}{2}k(k - 1)p_B^2. \tag{27}$$

The random variable Y_A follows a Binomial distribution $Bin(k, p_A)$, hence,

$$\mathbb{P}[Y_1 \geq 2] = \sum_{i=2}^{k} \binom{k}{i} p_A^i (1 - p_A)^{k-i}. \tag{28}$$

For $K(t-h)$ small, and, thus, p_A small, the leading term in the above expression is for $i = 2$. Hence, the second event happens with probability approximately equal to

$$p_2 = \frac{1}{2}k(k - 1)p_A^2, \tag{29}$$

and the tip pool size is reduced essentially by 1. Similarly to [3] we can write now a differential equation for $K(t)$. We consider only the first order terms of p_1 and p_2 since we assume $K(t)$ to be small:

$$\frac{dK(t)}{dt} = (p_1 - p_2)\lambda \approx \lambda \frac{(k - 1)K(t - h)}{L_0} \tag{30}$$

Using Equation (9) we can write

$$\frac{dK(t)}{dt} \approx \frac{(k - 1)^2 K(t - h)}{kh}, \tag{31}$$

with boundary condition $K(0) = 1$. Following the lines of [3] we obtain a solution of the form

$$K(t) = \exp\left(W\left(\frac{(k - 1)^2}{k}\right)\frac{t}{h}\right), \tag{32}$$

where $W(\cdot)$ is the so-called Lambert W-function. Taking the logarithm at both sides we find that the time when $K(t)$ reaches εL_0 is roughly

$$\tau_c \approx \frac{h}{W\left(\frac{(k-1)^2}{k}\right)} (\log L_0 + \log \varepsilon). \tag{33}$$

For large k we can approximate $W\left(\frac{(k-1)^2}{k}\right) \approx 2\log(k - 1) - \log k \approx \log k$ and obtain

$$\tau_c \approx \frac{h}{\log k} \log(L_0) \approx \frac{1}{\log k} h \log(\lambda h). \tag{34}$$
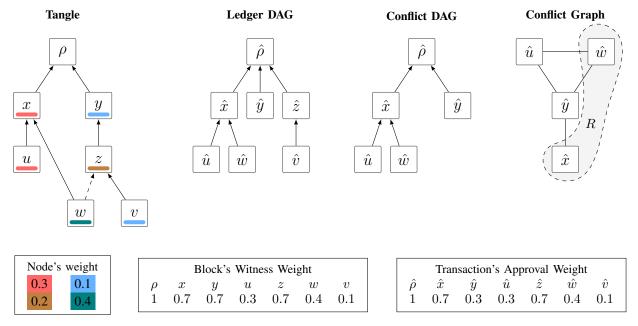
Figure 18: The Tangle, the Ledger DAG, the Conflict DAG and the Conflict Graph are shown. The Tangle starts with the genesis ρ and includes six other blocks x, y, z, u, v, w. Blocks x and y contain directly conflicting transactions $\hat{x}$ and $\hat{y}$. Similarly, blocks u and w contain directly conflicting transactions $\hat{u}$ and $\hat{w}$. Weights of four issuing nodes, which are identified with unique colors, are depicted. The WW of blocks and AW of transactions are computed. In addition, the preferred reality R is highlighted on the Conflict Graph.

Appendix B.
Illustrative example

In this section, we demonstrate in Figure 18 the most important concepts introduced in the paper using a toy example. In this example, blocks have two references which are identical in some cases.

The Tangle starts with the genesis ρ and six blocks are issued in the order x, y, z, v, w, u by four distinct nodes which are identified with unique colors (red, blue, brown, green) and have weights $0.3, 0.1, 0.2, 0.4$. In Figure 18 we demonstrate the Tangle, the Ledger DAG, the Conflict DAG and the Conflict Graph. Transactions $\hat{x}$ and $\hat{y}$ consume the same output of $\hat{\rho}$, thereby they are directly conflicting transactions. Similarly, we say that $\hat{u}$ and $\hat{w}$ are directly conflicting as their input is the same output of $\hat{x}$. Thus, the Conflict DAG consists of the genesis $\hat{\rho}$ and $\hat{x}, \hat{y}, \hat{u}, \hat{w}$ and can be seen as the subDAG of the Ledger DAG induced by its vertices (see Section 5.2). The Conflict Graph shows the conflicting dependencies between $\hat{x}, \hat{y}, \hat{u}, \hat{w}$, e.g. $\hat{y}$ is connected with $\hat{x}$ as they are directly conflicting and $\hat{y}$ is connected with all conflict-successors of $\hat{x}$, i.e. $\hat{u}$ and $\hat{w}$.

To demonstrate the steps of our protocol we discuss the actions from the point of view of the "green" node for issuing block w. Before block w was issued (i.e. at time when blocks x, y, z, v were issued only), the preferred reality (see Algorithm 1) for the node was $R = \{\hat{x}\}$ as $\mathbf{AW}(\hat{x}) > \mathbf{AW}(\hat{y})$. Suppose that the node decided to issue a block w and selected the two tips x and z by Algorithm 3.

Since the voting branch of x is $\mathrm{branch}_{\mathcal{V}}^{(p)}(x) = \{\hat{x}\} \subseteq R$ and the voting branch of z is $\mathrm{branch}_{\mathcal{V}}^{(p)}(z) = \{\hat{y}\} \not\subseteq R$ (see Definition 6.5), the node set a block reference from w to x only. After checking that the maximal contained branch of transaction $\hat{z}$ is the main branch (or the empty set), the node put a transaction reference from w to z shown in Figure 18 by the dashed arrow.

We observe that the Approval Weight of transactions is often equal to the Witness Weight of the corresponding blocks. However, this is not always the case. For instance, the Approval Weight of transaction $\hat{y}$ is the sum of weights of nodes supporting it. In this case, the "brown" and "blue" nodes are the supporters of $\hat{y}$, but not the "green" node because of the transaction reference from w to z. Therefore, $\mathbf{AW}(\hat{y}) = 0.2 + 0.1 = 0.3$. On the other hand, $\mathbf{WW}(y) = 0.2 + 0.1 + 0.4 = 0.7$ since the "green" node witnesses the block y.

To find the preferred reality, a node must follow Algorithm 1. Specifically, the reality R is constructed step-by-step by looking at the Conflict Graph (see Figure 18). At the first step, one includes $\hat{x}$ in R as it attains the highest Approval Weight and it is the closest vertex to the genesis. Then we remove $\hat{x}$ and all conflicts which are conflicting with $\hat{x}$, i.e. $\hat{y}$ is removed. At the second step, we choose $\hat{w}$ as its Approval Weight is higher that the one of $\hat{u}$. After this step, we remove both $\hat{w}$ and $\hat{u}$. Since the empty set remains, we finish with constructing reality $R = \{\hat{x}, \hat{w}\}$.

We also highlight that if at the next moment the "brown" node, which is supposed to be honest, decides to issue a

new block and attach it to block w (with a block reference), then it would change its vote on conflicting transaction $\hat{y}$ (see Definition 6.6). Specifically, the Approval Weight of $\hat{y}$ would be dropped by the weight of the "brown" node and become 0.1. In contrast, the Approval Weight of $\hat{x}$ would gain and become 0.9.

Appendix C.
Glossary

Approval Weight A function that computes the "relative" part of the network that approves a given transaction

Conflict A transaction that consumes the same output as a distinct transaction

Conflicting transactions Two transactions that contain two transactions in their past cones which consume the same output of some transaction

Cone A set of vertices in a DAG that are reachable from a given vertex by following the directions (past cone) and the opposite directions (future cone) of edges in the DAG.

Branch A set of conflicts which does not contain conflicting transactions and is past-closed

Branch DAG A DAG that represents the relations between branches

Ledger DAG A data structure that stores all transactions in the form of a DAG

Tangle DAG A data structure that stores all blocks in the form of a DAG

Voting DAG An augmented DAG that represents a combination of the Tangle DAG and the Ledger DAG and is used for determining voting cones

Genesis The transaction that is the ultimate predecessor of any transaction of the UTXO ledger.

Block An element of the Tangle DAG, constituted of identified data that refer to at least two blocks

Node A machine that is a part of the network. Its role is to issue new blocks and validate pre-existing ones

Reality A maximal branch

Solidification The process of retrieving missing blocks in the past cone of a given block which can be requested by a node

Witness Weight A function that computes the "relative" part of the network that approves a given block

Acknowledgment

References

[1] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008.

[2] Y. Sompolinsky and A. Zohar, "Secure high-rate transaction processing in bitcoin," in *Financial Cryptography and Data Security*, R. Böhme and T. Okamoto, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 507–527.

[3] S. Popov, "The Tangle," 2015.

[4] A. de Vries and C. Stoll, "Bitcoin's growing e-waste problem," *Resources, Conservation and Recycling*, vol. 175, p. 105901, 2021.

[5] I. Makarov and A. Schoar, "Blockchain analysis of the bitcoin market," National Bureau of Economic Research, Working Paper 29396, October 2021.

[6] P. Daian, S. Goldfeder, T. Kell, Y. Li, X. Zhao, I. Bentov, L. Breidenbach, and A. Juels, "Flash boys 2.0: Frontrunning in decentralized exchanges, miner extractable value, and consensus instability," in *2020 IEEE Symposium on Security and Privacy (SP)*, 2020, pp. 910–927.

[7] D. Rosenthal, "EE380 Talk," 2022. [Online]. Available: https://blog.dshr.org/2022/02/ee380-talk.html

[8] M. Dotan, Y.-A. Pignolet, S. Schmid, S. Tochner, and A. Zohar, "SOK: Cryptocurrency Networking Context, State-of-the-Art, Challenges," in *Proceedings of the 15th International Conference on Availability, Reliability and Security*, ser. ARES '20. New York, NY, USA: Association for Computing Machinery, 2020.

[9] M. Belotti, N. Božić, G. Pujolle, and S. Secci, "A Vademecum on Blockchain Technologies: When, Which, and How," *IEEE Communications Surveys Tutorials*, vol. 21, no. 4, pp. 3796–3838, 2019.

[10] Q. Wang, J. Yu, S. Chen, and Y. Xiang, "Sok: Diving into dag-based blockchain systems," *ArXiv preprint arXiv:2012.06128*, 2020.

[11] V. Buterin, "Ethereum: A Next-Generation Smart Contract and Decentralized Application Platform," 2013.

[12] M. M. T. Chakravarty, J. Chapman, K. MacKenzie, O. Melkonian, M. Peyton Jones, and P. Wadler, "The Extended UTXO Model," in *Financial Cryptography and Data Security*, M. Bernhard, A. Bracciali, L. J. Camp, S. Matsuo, A. Maurushat, P. B. Rønne, and M. Sala, Eds. Cham: Springer International Publishing, 2020, pp. 525–539.

[13] T. Rocket, M. Yin, K. Sekniqi, R. van Renesse, and E. G. Sirer, "Scalable and probabilistic leaderless BFT consensus through metastability," *arXiv preprint arXiv:1906.08936*, 2019.

[14] S. Popov, H. Moog, D. Camargo, A. Capossele, V. Dimitrov, A. Gal, A. Greve, B. Kusmierz, S. Mueller, A. Penzkofer, O. Saa, W. Sanders, L. Vigneri, W. Welz, and V. Attias, "The Coordicide," 2019.

[15] S. Müller, A. Penzkofer, N. Polyanskii, J. Theis, W. Sanders, and H. Moog, "Reality-based UTXO Ledger," *arXiv preprint arXiv:2205.01345*, 2022.

[16] T. D. Chandra and S. Toueg, "Unreliable failure detectors for reliable distributed systems," *J. ACM*, vol. 43, no. 2, p. 225–267, mar 1996.

[17] A. Penzkofer, B. Kusmierz, A. Capossele, W. Sanders, and O. Saa, "Parasite Chain Detection in the IOTA Protocol," in *2nd International Conference on Blockchain Economics, Security and Protocols (Tokenomics 2020)*, 2020, pp. 8:1–8:18.

[18] L. Lamport, R. Shostak, and M. Pease, "The byzantine generals problem," *ACM Trans. Program. Lang. Syst.*, vol. 4, no. 3, p. 382–401, jul 1982.

[19] M. J. Fischer, N. A. Lynch, and M. S. Paterson, "Impossibility of distributed consensus with one faulty process," *J. ACM*, vol. 32, no. 2, p. 374–382, apr 1985.

[20] C. Dwork, N. Lynch, and L. Stockmeyer, "Consensus in the Presence of Partial Synchrony," *J. ACM*, vol. 35, no. 2, p. 288–323, apr 1988.

[21] F. Cristian and C. Fetzer, "The timed asynchronous distributed system model," *IEEE Transactions on Parallel and Distributed Systems*, vol. 10, no. 6, pp. 642–657, 1999.

[22] A. Dembo, S. Kannan, E. N. Tas, D. Tse, P. Viswanath, X. Wang, and O. Zeitouni, *Everything is a Race and Nakamoto Always Wins*. New York, NY, USA: Association for Computing Machinery, 2020, p. 859–878.

[23] P. Gaži, A. Kiayias, and A. Russell, *Tight Consistency Bounds for Bitcoin*. New York, NY, USA: Association for Computing Machinery, 2020, p. 819–838.

[24] R. Pass, L. Seeman, and A. Shelat, "Analysis of the blockchain protocol in asynchronous networks," in *Advances in Cryptology – EUROCRYPT 2017*, J.-S. Coron and J. B. Nielsen, Eds. Cham: Springer International Publishing, 2017, pp. 643–673.

[25] C. Natoli and V. Gramoli, "The balance attack against proof-of-work blockchains: The R3 testbed as an example," *CoRR*, vol. abs/1612.09426, 2016.

[26] R. Guerraoui and A. Schiper, "Consensus: the big misunderstanding [distributed fault tolerant systems]," in *Proceedings of the Sixth IEEE Computer Society Workshop on Future Trends of Distributed Computing Systems*, 1997, pp. 183–188.

[27] M. K. Aguilera, *Stumbling over Consensus Research: Misunderstandings and Issues*. Berlin, Heidelberg: Springer-Verlag, 2010, p. 59–72.

[28] M. Ben-Or, "Another advantage of free choice (extended abstract): Completely asynchronous agreement protocols," in *Proceedings of the Second Annual ACM Symposium on Principles of Distributed Computing*, ser. PODC '83. New York, NY, USA: Association for Computing Machinery, 1983, p. 27–30.

[29] M. O. Rabin, "Randomized byzantine generals," in *24th Annual Symposium on Foundations of Computer Science (sfcs 1983)*, 1983, pp. 403–409.

[30] V. B. Mišić, J. Mišić, and X. Chang, "On Forks and Fork Characteristics in a Bitcoin-Like Distribution Network," in *2019 IEEE International Conference on Blockchain (Blockchain)*, 2019, pp. 212–219.

[31] M. Iqbal and R. Matulevičius, "Exploring Sybil and Double-Spending Risks in Blockchain Systems," *IEEE Access*, vol. 9, pp. 76 153–76 177, 2021.

[32] M. Neuder, D. J. Moroz, R. Rao, and D. C. Parkes, "Low-cost attacks on Ethereum 2.0 by sub-1/3 stakeholders," 2021.

[33] J. P. T. Lovejoy, "An empirical analysis of chain reorganizations and double-spend attacks on proof-of-work cryptocurrencies," *Thesis, Massachusetts Institute of Technology. Department of Electrical Engineering and Computer Science*, 2020.

[34] V. Bagaria, S. Kannan, D. Tse, G. Fanti, and P. Viswanath, "Prism: Deconstructing the blockchain to approach physical limits," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 585–602.

[35] Y. Sompolinsky and A. Zohar, "Accelerating bitcoin's transaction processing. fast money grows on trees, not chains," *IACR Cryptol. ePrint Arch.*, vol. 2013, p. 881, 2013.

[36] Y. Lewenberg, Y. Sompolinsky, and A. Zohar, "Inclusive block chain protocols," in *International Conference on Financial Cryptography and Data Security*. Springer, 2015, pp. 528–547.

[37] Y. Sompolinsky, Y. Lewenberg, and A. Zohar, "Spectre: A fast and scalable cryptocurrency protocol," Cryptology ePrint Archive, Report 2016/1159, 2016.

[38] Y. Sompolinsky, S. Wyborski, and A. Zohar, *PHANTOM GHOSTDAG: A Scalable Generalization of Nakamoto Consensus: September 2, 2021*. New York, NY, USA: Association for Computing Machinery, 2021, p. 57–70.

[39] A. Miller, Y. Xia, K. Croman, E. Shi, and D. Song, "The Honey Badger of BFT Protocols," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 31–42.

[40] S. Duan, M. K. Reiter, and H. Zhang, "BEAT: Asynchronous BFT Made Practical," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 2028–2041.

[41] L. Baird and A. Luykx, "The Hashgraph Protocol: Efficient Asynchronous BFT for High-Throughput Distributed Ledgers," in *2020 International Conference on Omni-layer Intelligent Systems (COINS)*, 2020, pp. 1–7.

[42] A. Gagol, D. Lesniak, D. Straszak, and M. Swietek, "Aleph: Efficient atomic broadcast in asynchronous networks with byzantine nodes," *CoRR*, vol. abs/1908.05156, 2019.

[43] G. Danezis, L. Kokoris-Kogias, A. Sonnino, and A. Spiegelman, "Narwhal and Tusk: A DAG-Based Mempool and Efficient BFT Consensus," in *Proceedings of the Seventeenth European Conference on Computer Systems*, ser. EuroSys '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 34–50.

[44] I. Keidar, E. Kokoris-Kogias, O. Naor, and A. Spiegelman, "All You Need is DAG," in *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing*, ser. PODC'21. New York, NY, USA: Association for Computing Machinery, 2021, p. 165–175.

[45] N. Giridharan, L. Kokoris-Kogias, A. Sonnino, and A. Spiegelman, "Bullshark: DAG BFT protocols made practical," *CoRR*, vol. abs/2201.05677, 2022. [Online]. Available: https://arxiv.org/abs/2201.05677

[46] M. A. Schett and G. Danezis, "Embedding a Deterministic BFT Protocol in a Block DAG," in *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing*, ser. PODC'21. New York, NY, USA: Association for Computing Machinery, 2021, p. 177–186.

[47] L. Yang, V. Bagaria, G. Wang, M. Alizadeh, D. Tse, G. Fanti, and P. Viswanath, "Prism: Scaling bitcoin by 10,000x," *arXiv preprint arXiv:1909.11261*, 2019.

[48] G. Wang, S. Wang, V. Bagaria, D. Tse, and P. Viswanath, "Prism removes consensus bottleneck for smart contracts," in *2020 Crypto Valley Conference on Blockchain Technology (CVCBT)*, 2020, pp. 68–77.

[49] S. Müller, A. Penzkofer, B. Kuśmierz, D. Camargo, and W. J. Buchanan, "Fast Probabilistic Consensus with Weighted Votes," in *Proceedings of the Future Technologies Conference (FTC) 2020, Volume 2*, 2021, pp. 360–378.

[50] S. Popov and S. Müller, "Voting-based probabilistic consensuses and their applications in distributed ledgers," *Annals of Telecommunications*, vol. 77, no. 1, pp. 77–99, 2022.

[51] A. Cullen, P. Ferraro, W. Sanders, L. Vigneri, and R. Shorten, "Access Control for Distributed Ledgers in the Internet of Things: A Networking Approach," *IEEE Internet of Things Journal*, pp. 1–1, 2021.

[52] B. Kusmierz, W. Sanders, A. Penzkofer, A. Capossele, and A. Gal, "Properties of the Tangle for Uniform Random and Random Walk Tip Selection," in *2019 IEEE International Conference on Blockchain (Blockchain)*, 2019, pp. 228–236.

[53] P. Ferraro, C. King, and R. Shorten, "Distributed ledger technology for smart cities, the sharing economy, and social compliance," *IEEE Access*, vol. 6, pp. 62 728–62 746, 2018.

[54] W. Li, "Zipf's Law everywhere," *Glottometrics*, vol. 5, pp. 14–21, 2002.

[55] L. A. Adamic and B. Huberman, "Zipf's law and the Internet," *Glottometrics*, vol. 3, pp. 143–150, 2002.

[56] D. Kondor, M. Pósfai, I. Csabai, and G. Vattay, "Do the Rich Get Richer? An Empirical Analysis of the Bitcoin Transaction Network," *PLOS ONE*, vol. 9, no. 2, pp. 1–10, 02 2014.

[57] B. Kuśmierz, S. Müller, and A. Capossele, "Committee Selection in DAG Distributed Ledgers and Applications," in *Intelligent Computing*, K. Arai, Ed. Cham: Springer International Publishing, 2021, pp. 840–857.

[58] C. King, "The fluid limit of a random graph model for a shared ledger," *Advances in Applied Probability*, vol. 53, no. 1, p. 81–106, 2021.

[59] E. Drąsutis, "IOTA Smart Contracts," accessed January 2022.

[60] C. Dwork, N. Lynch, and L. Stockmeyer, "Consensus in the Presence of Partial Synchrony," *J. ACM*, vol. 35, no. 2, p. 288–323, Apr. 1988.

[61] T. D. Chandra and S. Toueg, "Unreliable Failure Detectors for Reliable Distributed Systems," *J. ACM*, vol. 43, no. 2, p. 225–267, Mar. 1996.

[62] J. Beauquier, P. Blanchard, J. Burman, and R. Guerraoui, "The Benefits of Entropy in Population Protocols," in *19th International Conference on Principles of Distributed Systems (OPODIS 2015)*, ser. Leibniz International Proceedings in Informatics (LIPIcs), E. Anceaume, C. Cachin, and M. Potop-Butucaru, Eds., vol. 46. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2016, pp. 1–15.

[63] A. Penzkofer, O. Saa, and D. Dziubałtowska, "Impact of delay classes on the data structure in IOTA," in *CBT2021 - 5th Cryptocurrencies and Blockchain Technology workshop*, 2021.

[64] S. Popov and W. J. Buchanan, "FPC-BI: Fast probabilistic consensus within byzantine infrastructures," *Journal of Parallel and Distributed Computing*, vol. 147, pp. 77–86, 2021.

[65] A. Capossele, S. Müller, and A. Penzkofer, "Robustness and efficiency of voting consensus protocols within byzantine infrastructures," *Blockchain: Research and Applications*, vol. 2, no. 1, p. 100007, 2021.

[66] I. Cascudo and B. David, "SCRAPE: Scalable randomness attested by public entities," in *International Conference on Applied Cryptography and Network Security*. Springer, 2017, pp. 537–556.

[67] A. K. Lenstra and B. Wesolowski, "Trustworthy public randomness with sloth, unicorn, and trx," *International Journal of Applied Cryptography*, vol. 3, no. 4, pp. 330–343, 2017.

[68] S. Popov, "On a decentralized trustless pseudo-random number generation algorithm," *Journal of Mathematical Cryptology*, vol. 11, no. 1, pp. 37–43, 2017.

[69] P. Schindler, A. Judmayer, N. Stifter, and E. Weippl, "HydRand: Practical Continuous Distributed Randomness," Cryptology ePrint Archive, Report 2018/319, 2018.

[70] E. Syta, P. Jovanovic, E. K. Kogias, N. Gailly, L. Gasser, I. Khoffi, M. J. Fischer, and B. Ford, "Scalable bias-resistant distributed randomness," in *2017 IEEE Symposium on Security and Privacy (SP)*, 2017, pp. 444–460.

[71] B. Wesolowski, "Efficient verifiable delay functions," *J. Cryptol.*, vol. 33, pp. 2113–2147, 2020.

[72] S. Müller, R. C. Nitchai, and S. Popov, "FPC on a set," 2021.

[73] B.-Y. Lin, D. Dziubałtowska, P. Macek, A. Penzkofer, and S. Müller, "Robustness of the Tangle 2.0 Consensus," *arXiv preprint arXiv:2208.08254*, 2022.

[74] IOTA Foundation. (2021) Tangle 2.0 simulator. [Online]. Available: https://github.com/iotaledger/multiverse-simulation