The Coordicide

Serguei Popov, Hans Moog, Darcy Camargo, Angelo Capossele, Vassil Dimitrov, Alon Gal, Andrew Greve, Bartosz Kusmierz, Sebastian Mueller, Andreas Penzkofer, Olivia Saa, William Sanders, Luigi Vigneri, Wolfgang Welz, Vidal Attias
(IOTA Foundation)

January 2020

Contents

Glossary

Byzantine node A participant in a distributed system which tries to damage its operation intentionally, e.g., by not forwarding messages to other participants and/or sending conflicting messages to different participants.

Consensus The problem of agreeing on a specific data or value in distributed multi-agent systems in presence of faulty processes.

Coordinator A trusted entity issuing milestones in order to guarantee finality and protect the Tangle against attacks.

Dictionary attack A form of brute force attack technique for defeating an authentication mechanism by trying to determine its passphrase by trying millions of likely possibilities, such as words in a dictionary.

Eclipse attack A cyber-attack that aims to isolate and attack a specific user, rather than the whole network.

Genesis The first transaction ever generated in the Tangle.

Heartbeat A periodic signal generated by hardware or software to indicate normal operation or to synchronize other parts of a computer system.

History The list of transactions directly or indirectly approved by a given transaction.

Milestone A special transaction issued by the Coordinator. One of its features is to determine finality of the transactions it approves.

Neighboring nodes Nodes sharing the same link in a network.

Node A machine which is part of the IOTA network. Its role is to issue new transactions and to validate already existing ones.

Peering The procedure of discovering and connecting to other network nodes.

Proof-of-Work A piece of data which is difficult (costly, time-consuming) to produce but easy for others to verify and which satisfies certain requirements.

Rainbow table attack A precomputed table for reversing cryptographic hash functions.

Random walk A mathematical object that describes a path that consists of a succession of random steps on some mathematical space.

Salt A random number used as an additional input to a one-way function that hashes data.

Social engineering The use of deception to manipulate individuals into divulging confidential or personal information that may be used for fraudulent purposes.

Sybil attack A Sybil attack is an attempt to gain control over a peer-to-peer network by forging multiple fake identities.

Tip A transaction that has not been approved yet.

Transaction A message that transfers funds or information between two nodes. A transaction is *solid* if its entire history is known.

Acronyms

ASIC	Application-Specific Integrated Circuit.
CA	Cellular Automata.
DAG	Directed Acyclic Graph.
DLT	Distributed Ledger Technology.
IF	IOTA Foundation.
IRI	IOTA Reference Implementation.
PoW	Proof-of-Work.
TSA	Tip Selection Algorithm.
VDF	Verifiable Delay Function.

1 Introduction

IOTA's vision aims to establish a real-time economy for Internet-of-Things and the future Internet through a secure zero fee payment and data transmission system. Realizing this vision is subject to a combination of features that cannot be found in current distributed ledger technologies (DLTs): First, a significantly higher throughput is required than in blockchains which have an intrinsic bottleneck, forcing transactions to be aggregated under a chain-type data structure; second, fees can be considered a barrier for micro transactions, but they are necessary in PoW-based DLTs where the network distinguishes between miners and users. Conversely, IOTA utilizes a directed acyclic graph (DAG) structure as explained in the original IOTA white paper [66] which permits a theoretical infinite throughput[1]. Furthermore, enabling each network participant to both issue and approve transactions allows IOTA to eliminate the fees found in blockchain architecture, thus facilitating a micropayment-ready network (see also [67]).

One common problem for early stage DLTs is that the networks are not robust enough for proposed security mechanisms to function as intended, since such security mechanisms presuppose a mature network. Therefore, it is typical that DLTs employ various "bootstrapping" security measures at the outset, ensuring network growth to the mature stage can take place[2]. Thus, in its current implementation, IOTA relies on a centralized Coordinator to provide security given the risk of dishonest actors seeking to undermine the nascent network. IOTA's definition of consensus requires a confirmed transaction to be referenced (either directly or indirectly) by a signed transaction issued by the Coordinator. In other words, the Coordinator can be thought of as a "finality device".

We believe that the vision of cryptocurrency networks based on Nakamoto consensus can be improved upon by changing the key underlying assumption about those controlling the majority of the network's hashing power being considered "honest" by definition (the "longest chain wins" rule). In IOTA, the requirement for honest actors to control a majority of the network's hashing power is currently replaced by the use of the Coordinator. The Coordinator is a temporary measure as the IOTA network develops beyond Nakamoto's vision for network consensus. The Coordicide project is focused on the removal of the Coordinator through the implementation of several network components, as discussed in this working paper. Despite these additional components, all existing fundamental design features of the Tangle remain in-place.

In line with the IOTA Foundation's charter as a non-profit organization, our goals as a research department include transparency, collaboration, and community engagement. We aim to open our research work in order to obtain feedback from academia as well as the broad community of enthusiasts. Since we have released the IOTA white paper, we have indeed seen a flourishing

[1] The actual throughput is bounded by hardware limitations and by laws of physics.

[2] See e.g. https://en.bitcoin.it/wiki/Checkpoint_Lockin for an example of a security mechanism of this sort

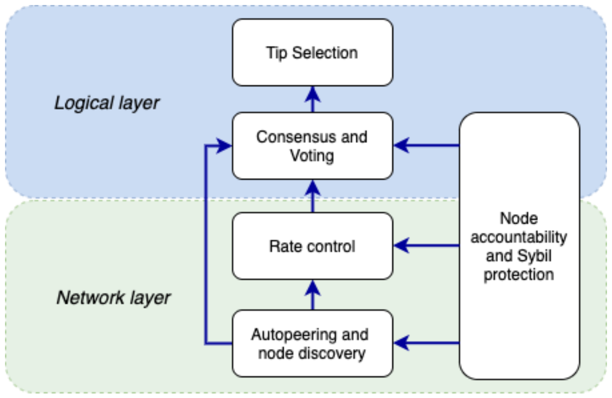


Fig. 1: Interconnections between the Coordicide building blocks.

literature, would it be people analysing and proposing improvements on the protocol itself [3, 14, 30, 47, 51, 74], building systems on top of the IOTA protocol [15, 20, 59, 80] or studying the market viability [6, 28, 38, 72]. One note of caution, however, is in order: Since our research is highly dynamic in nature, proposed ideas need to be simulated and tested in order to develop specific network components which we feel confident to deploy on the main network. We stress that some of the ideas presented here are works in progress and as such are not fully fleshed out. They are therefore likely to be modified as we make progress and perform simulations.

At a high level, our vision for the Coordicide can be explained in the following way. We are looking for a *probabilistic consensus* — with probability very close to 1, all honest participants of the network would agree on which transactions should be considered valid. It is important to remark that one should not be afraid of the probabilistic nature of it — if something occurs with strictly positive probability, this doesn't yet mean it would ever occur *in practice*[3]. Another important idea is that, while we do need *total* consensus on what is really important (transactions' validity), we may not need total consensus on everything. Therefore, we may use an *approximate* consensus[4] to achieve the total one with high probability. It is also a feature of our approach that the consensus (on transactions' validity) is an *attracting state* in the following sense: small deviations from the absolute consensus on secondary things (e.g., local clocks, random number sequence that the nodes see, mana vectors) with high probability do not lead to deviations from consensus on transactions' validity. Still, we keep the basic protocol simple: the only "hard" rule remains to be that

[3]As a quick example, try guessing at least one private key from Bitcoin addresses which belong to Satoshi; yet, the probability that a random 256-bit number is one of those private keys is strictly positive.

[4]For example, on *time* — it is something on which the approximate consensus already exists; another example when the total consensus is not necessary is the common sequence of random numbers: as explained below, it is already enough if most of the participants agree on the same number frequently enough.

a new transaction should approve two other transactions; what eventually stays in the main tangle (not orphaned), is valid[5]. The above is in line with the idea that "IOTA is free" [67]. This is because the actors will have flexibility to adapt the system to different environments by only adjusting the node's behaviors.

In order to get rid of the Coordinator, a number of challenges must be solved. This working paper covers those challenges, which are summed up by the building blocks in Fig. 1. In the following, we give a concise overview about the current state of the Coordicide as well as future research directions:

- *Node accountability.* In Section 3 we propose the concept of *global node IDs* and we describe a novel Sybil protection mechanism that does not require node owners to risk or disclose their funds. Identifying the issuing node of a message is fundamental to enforce a specific network topology (through autopeering) or to penalize bad behaviours (through rate control).

- *Autopeering and node discovery.* An automated process to discover and reliably connect to neighbors is needed in every distributed system. In Section 4 we discuss an autopeering proposal for the IOTA network.

- *Rate control.* To ensure the network does not exceed its capacity, in Section 5 we introduce a mechanism to control the rate of transactions that are propagated through the network. This method selectively filters some transactions out according to the statistics of the issuing node.

- *Consensus and Voting.* The previous building blocks lead to an extended consensus framework described in Section 6. The new protocol does not use the tip selection algorithm as a tool for consensus, although it is still has its importance as it can be seen in Section 7. Instead, it proactively resolves conflicts through voting. We describe two voting mechanisms where nodes query other nodes to find out their current opinion on the network status (Section 6).

- *Tip Selection* With the decoupling from the consensus mechanism, the tip selection algorithm has more freedom to achieve a better performance in its other tasks, as keeping the good structure of the network, demotivating lazy behavior and quickly approve new honest transactions. In Section 7 we talk about the impact and motivation of the tip selection, as well as present more details on our main proposal for a new tip selection algorithm.

[5]One of the reasons why we prefer using this approach, is the following: if we allow conflicting transactions on the Tangle, this has to go coupled with a precise conflict-resolution rule, which can be never changed and likely has "long-range" dependencies.

2 Status Quo

In the introduction, we mentioned that the current IOTA main network uses the Coordinator to reach consensus and, more generally, to guarantee the security of the network. However, this centralized component should only be considered as a necessary bootstrapping mechanism, rather than a long-term solution. In this section, we first discuss the current status of the IOTA main network implemented through the IOTA reference implementation (IRI) software[6], and then we describe the challenges we are facing when building a *Coo-less* network (i.e., a network without the Coordinator). Since the Coordinator and its milestones are currently deeply embedded in IRI, removing those dependencies not only implies comprehensive changes to the software, but also leads to new research questions.

2.1 Current IOTA implementation

The current IOTA main network is implemented according to the IRI software, in which the Coordinator plays an important role. In the following we describe the main tasks implemented in the current main network, not all of them strictly related to consensus:

- *Manual peering.* In order to join the Tangle, a node is required to connect to some existing nodes (*peering*). The current IRI software only permits *manual* peering, i.e., a node operator has to manually look for the addresses of other Tangle's nodes. Peering is fundamental to propagate transactions and to synchronize to the current status of the ledger. As for the latter, milestones are useful anchors to determine whether two nodes have fallen out of synchronization: If a node's latest solid milestone is much older than its peers', it is probably lagging behind.

- *Rate control mechanism.* In order to issue a transaction, a node must solve a cryptographic puzzle (Proof-of-Work). This is necessary to guarantee that nodes do not arbitrarily spam the network, or to avoid that they inject more transactions than the network can handle.

- *Tip selection strategy.* Approving transactions is a fundamental procedure which leads to the DAG structure of the Tangle. To approve a transaction, a node must verify that no inconsistencies with respect to the ledger state are introduced. Although it is not possible to enforce which transaction to validate, the original IOTA white paper suggests a tip selection algorithm based on a random walk which: (i) Discourages lazy behavior and encourages approving fresh tips; (ii) continuously merges small branches into a single large branch, thus increasing confirmation rate; (iii) in case of conflicts, kills off all but one of the conflicting branches.

[6]https://github.com/iotaledger/iri

- *Consensus.* The main role of milestones is to determine the consensus. The Tangle applies a simple rule: A transaction is confirmed if and only if it is referenced by a milestone. In IRI, this is reflected in the `getBalances` and `getInclusionStates` API calls, which indicate how many tokens an account has and whether a transaction is confirmed, respectively.

- *IRI code optimization.* Instead of computing the full ledger state starting from the genesis, an intermediate state is saved for each milestone; similarly, milestones are used in *local snapshots*, i.e., the IRI pruning mechanism, which allows nodes to avoid storing older parts of the Tangle.

2.2 New research challenges

The most significant modelling assumption in the IOTA white paper is the *honest transaction majority* condition [14]: Specifically, to be considered valid, the white paper consensus algorithm requires that the majority of transactions *always* come from honest network participants. The implication is that honest nodes may need to continuously send transactions, regardless of whether they are actually using the network or not. Furthermore, achieving this hashing majority must be expensive, otherwise it would be easy for malicious agents to buy enough hashing power and overtake the network. In addition to this incentivization problem, issuing transactions is subject to Proof-of-Work (PoW). Due to its complexity, slow nodes would be excluded from participating in the network.

The above concerns directly lead to new challenges and research questions which will be investigated throughout the paper:

- *Peering.* We require an automatic way to connect to existing nodes. Such an autopeering mechanism generates a network topology which becomes fundamental to deal with eclipse attacks and acts as a main component in certain voting protocols (see later Cellular consensus).

- *Sybil protection.* When node identities are introduced, it is necessary to have a mechanism that prevents the creation of counterfeit identities to gain disproportionate throughput or voting weight.

- *Rate control.* A more efficient rate control algorithm is needed to solve the following tradeoff: If the PoW difficulty is too high, then small devices (e.g., phones or sensors) would take an unreasonably long amount of time to compute it, and will therefore be unable to send transactions; on the other hand, low difficulty can favor network congestion and/or spam attacks.

- *Consensus.* We need a consensus mechanism which is solid under the honest node majority assumption (ensured by an appropriate Sybil protection mechanism) without the support of the Coordinator.

- *Tip Selection.* In the current proposal, the security of the system does not rely solely on tip selection anymore. Therefore, there is now more freedom in choosing the (recommended) tip selection algorithm. We require it to be low in computational power, to make honest transactions approved quickly and to demotivate lazy behavior, this all while keeping a good structure for the Tangle.

In next section 3, we will introduce the notion of node identity which is a prerequirement to the solution of the above research topics.

2.2.1 GoShimmer

These new concepts and the research results tied to this should be tested in an experimental manner, in order to proceed to the next level of implementation in a protocol. An important step, therefore, is to introduce a code-base on which experiments can be be performed and the hypotheses thoroughly tested. This is achieved by implementing the concepts of the Coordicide blueprint into a prototype, called *GoShimmer*[7].

GoShimmer is designed in a modular fashion, where each module represents one of the essential Coordicide's components as well as core components necessary to work as a full-node (e.g., gossip layer, ledger state, API). This approach enables to convert the concepts piece-by-piece and more importantly, simultaneous but independent of each other, into a prototype.

[7]https://github.com/iotaledger/goshimmer

3 Node accountability

In a network without the Coordinator, several applications require to reliably associate transactions or other messages with the node which issued them. These applications include:

- *Rate control.* In an overload scenario, where the nodes are trying to issue more transactions than the overall network can handle (due to its physical limits), particular transactions originating from the most heavily contributing nodes should be blocked or penalized.
- *Voting-based consensus mechanisms.* To prevent double voting and to associate votes with node weights, the actual votes must be linked to node IDs.

In Section 3.1, we suggest a way to associate global identities to nodes. Since this may expose the network to potential Sybil attacks, in Section 3.2 we introduce *mana*, a novel anti-Sybil mechanism.

3.1 Global node identities

In order to identify nodes, it is necessary to introduce global node identities. To this end, we envision using common public key cryptography to sign certain data and to link it to its issuing node in a tamper proof way. Additionally, we require that the issuing node adds its public key to every signed message. This way, every node can verify the authenticity of the issuing node without the need for some form of global database of IDs and keys. It is important to note that these mechanisms only need to be implemented to protect the communication layer and that keys, IDs and signatures do not need to be stored in the Tangle once processed by the node. This allows for greater flexibility as the actual signing scheme can be exchanged without any impact on stored data. In contrast to any data stored in the Tangle, the communication layer, therefore, does not necessarily require the use of post-quantum cryptography right now, but it can be swapped when quantum attacks become more imminent in the future.

When node identities are relevant, a distributed system becomes vulnerable to Sybil attacks [32], where a malicious entity masquerades as multiple counterfeit identities. This would overcome any mechanism that relies on a limited number of such identities and would open the network to coordinated attacks. A possible way to deal with this problem is described in the following section.

3.2 Sybil protection principles

One very common way to make such a Sybil attack harder is the so-called resource testing, where each identity has to prove the ownership of certain difficult-to-obtain resources. Since in the cryptocurrency world users own a certain amount of tokens, we propose a Sybil protection mechanism based on the ownership of such tokens. Specifically, we allow a user to assign a certain reputation value to a given node alongside its issued transaction. Such a reputation value, called *mana*, is equivalent to the total funds transferred within the transaction.

As anticipated, *mana* is a crucial aspect in rate control, autopeering and voting. In the future, we expect mana to be only a specific aspect of a more comprehensive reputation system which includes other criteria, such as penalties for misbehavior or incentives for helping the network. Also, criteria which are *external* to the system (informally, any sort of “real-world importance”) may be used.

Specifically, mana is a function whose input is a transaction, and whose output is the mana state, a vector which gives the amount of *mana* staked to each node. There are several ways of defining a mana system, and through the last months of research we decided on one of such systems that have all the desirable properties the network wants.

We stress here that this process does not influence the actual balances in any way, but it is only used to give higher weight to “trusted” nodes.

In the next subsection we present a base for our system of choice, and following we properly define the model the network will use, as well as describe why this alteration of the base model is more desired.

3.3 The mana system

The principles of a mana system is that one should get or have more mana the more one contributes to the network. Contribution is naturally associated to how much stake one holds, but although having tokens helps the network, one should not be able to “mine” an unrestrained amount of mana by simply holding some quantity of tokens for a large amount of time, or by frequently sending tokens around.

Now we will define a mana system that has the described properties, and conclude the subsection with a possible improvement that would keep abrupt variations in mana from happening.

To achieve this we introduce three new concepts:

- *Pending mana.* Addresses generate pending mana at a rate proportional to the stake they hold.
- *Mana.* When funds (i.e., IOTA tokens) are spent from an address, the pending mana that has been generated by this address, is converted to mana and pledged to a node[8]. Pending mana is now generated by the funds on the receiver’s address.
- *Decay.* Both mana and pending mana decay at a rate proportional to its value, hence keeping mana from growing unrestrained over time.

[8]In practice, it may be also be a good idea to only credit mana to the node after a (small) amount of time ε_0, as a further protective measure against actors who might try moving funds very fast in order to possibly mess with the system by using network delays.

Let us now formalize these ideas and define the following quantities:

- The amount of tokens an address x holds at a given time, S;
- The generation rate for pending mana, α;
- The decay rate coefficient for mana and pending mana, γ;
- The pending mana an address x holds at time t, $m_x(t)$;
- The mana a node Z has at time t, $M_Z(t)$.

Consistently with the above informal explanation, the evolution of the pending mana on an address (while the tokens are not moved) satisfies the differential equation

$$\frac{d}{dt}m_x(t) = \alpha S - \gamma m_x(t). \tag{1}$$

Hence, if an address has S tokens at time 0, its pending mana can be calculated by solving (1) to obtain that

$$m_x(t) = m_x(0)e^{-\gamma t} + \frac{\alpha S}{\gamma}(1 - e^{-\gamma t}). \tag{2}$$

Note that the number of tokens on an address is a piecewise constant function; so, the above equation can be used to calculate the pending mana in the general case, by considering the intervals where the address' balance remains constant. In particular, in the case where the address had no tokens before the initial time (it received S tokens at time 0), (2) reduces to

$$m_x(t) = \frac{\alpha S}{\gamma}(1 - e^{-\gamma t}). \tag{3}$$

Observe that even though we stated that pending mana decays over time, we can see that (3) is strictly increasing. This happens due to the decay being proportional to its value, while the generation being a constant rate (equal to the number of tokens). This means that the decay rate is always lower than the generation rate and thus the pending mana is always increasing, but slower and slower as time increases, up to a point where we cannot see its growth anymore.

After being pledged to a node, mana decays over time. Hence the evolution of mana (without further pledging from addresses) for a node satisfies

$$\frac{d}{dt}M_Z(t) = -\gamma M_Z(t),$$

and therefore

$$M_Z(t) = M_Z(0)e^{-\gamma t}. \tag{4}$$

Observe that from (2), (3) and (4) the node can always calculate the mana and pending mana of all nodes in the system, given that it knows the values of α and γ (which are public).

This model has two desired properties:

1. **Mana cannot be gamed**, in the sense that the amount of mana received will not change depending on how often the user converts pending mana into mana or in how many addresses the tokens are split. To illustrate this let us give two examples.

   First consider that an address has S tokens and will pledge the mana generated to node Z at two points in time: at time $t/2$ and at time t (of course considering the user doing this also has control over the receiving address). We will show that pledging more often will give the same mana. Let us denote the mana received at the first pledging by m_1 and at the second pledging by m_2. The values of m_1 and m_2 are the same (as we have the same number of tokens generating pending mana over the same amount of time) and can be calculated by (3):

   $$m_1 = m_2 = \frac{\alpha S}{\gamma}(1 - e^{-\gamma t/2}).$$

   However, at time t, when Z receives the mana m_2, the mana m_1 will have decayed for time $t/2$, hence using (4) we have that the mana that Z has at time t is

   $$\begin{aligned} M_Z(t) &= m_1 e^{-\gamma t/2} + m_2 \\ &= \frac{\alpha S}{\gamma}(1 - e^{-\gamma t/2})(1 + e^{-\gamma t/2}) \\ &= \frac{\alpha S}{\gamma}(1 - e^{-\gamma t}), \end{aligned}$$

   which is the same it would have received by pledging only once.

   The second example is much simpler. If instead of having an address with S tokens we have n addresses with S/n tokens, all pledging to Z at time t, then by (3) the total mana received is

   $$\begin{aligned} M_Z(t) &= n \times \frac{\alpha(S/n)}{\gamma}(1 - e^{-\gamma t}) \\ &= \frac{\alpha S}{\gamma}(1 - e^{-\gamma t}), \end{aligned}$$

   which again is the same value.

2. **Mana is bounded**, both for a node with addresses summing S tokens and, therefore, also for the entire network. To check this, observe that by (4), as the pending mana becomes mana, its value at the node only decays. Hence the maximal value of mana is the same as the maximal value that (3) achieves, which is the asymptotic value as $t \to \infty$:

   $$\begin{aligned} \lim_{t\to\infty} m_x(t) &= \lim_{t\to\infty} \frac{\alpha S}{\gamma}(1 - e^{-\gamma t/2}) \\ &= \frac{\alpha S}{\gamma}. \end{aligned}$$

Since this bound is a constant times the number of tokens, if we sum the maximal mana on all the possible addresses on the network, we get that at any time the maximal mana in the network can be bounded above in the following way

$$\text{Total amount of Mana} \leq \frac{\alpha}{\gamma} \times \text{Total amount of IOTA tokens.} \tag{5}$$

This approach has some interesting features:

- Once an address gains control over the tokens, they must wait for their pending mana to gain their full potential.
- Mana is easy to store and is snapshotable: its decay and growth can be deduced from changes in time and balance: if a nodes' pending mana and balance at t_1 is known, then its potential mana at time t_2 can be deduced for any $t_2 > t_1$.

This proposal of mana satisfies most of our needs but it still could be criticized for its abrupt variations over time: since we could have a large quantity of pending mana becoming mana for a node at once, many applications that depend on how much mana a node currently has could abruptly change their vision. To prevent this from happening, we will use an averaged version of mana $\bar{M}_Z(t)$ that is defined for a certain parameter Δ as

$$\bar{M}_Z(t) = \frac{1}{\Delta} \int_{t-\Delta}^{t} M_Z(s)ds. \tag{6}$$

In mathematical terms, the mana system $\bar{M}_Z$ is a **smoothing** of the mana system M_Z, obtained by applying **moving averages** to their value over time.

There are two things to notice about $\bar{M}_Z$:

- The system $\bar{M}_Z$ is as simple as M_Z, in the sense that if we have the values of M_Z, the calculation of $\bar{M}_Z$ can be done without significant computational cost.
- Differently from $M_Z(t)$, the function $\bar{M}_Z(t)$ is continuous. This means that after the pledge of mana from an address, the value of $\bar{M}_Z(t)$ will slowly increase until it achieves its maximum value. The speed of this increase will depend on the parameter Δ.

3.4 Comparison with existing schemes

The amount of *mana* people can delegate is determined by how many tokens they own, which means that people who own more tokens will have a larger influence in this process. In particular, nodes could accumulate large amounts of *mana* without having much stake in the network of their own. In a traditional *proof-of-ownership* Sybil protection mechanism, each node has to prove that it owns a certain amount of collateral. Conversely, delegating *mana* brings several key advantages. First, since *mana* is credited as part of regular transactions, nodes do not have to constantly use their address's private keys to sign, which would pose a severe security risk. Furthermore, this approach does not require all node operators to own or declare high amounts of tokens; finally, users can issue additional *mana* to nodes providing good service to the community.

Since we have now established reliable node identities, we can use these identities to discover and connect to other nodes in the network while taking their mana into account.

4 Autopeering

In the IOTA protocol, a node (or peer) is a machine storing the information about the Tangle, IOTA's underlying data structure. In order for the network to work efficiently and for the nodes to be kept up-to-date about the ledger state, nodes exchange information, such as new transactions, with each other. Each node establishes a communication channel with a small subset of nodes (i.e., neighbors) via a process called *peering*. The peering process is potentially vulnerable to various attacks. For instance, if all of a node's neighbors are controlled by an attacker, then the attacker has complete control over the node's view of the Tangle, leaving the victim node vulnerable to a host of scams. This type of attack is known as an Eclipse attack [43, 46, 49].

Currently, in the *mainnet* IOTA network, nodes use a manual peering process to mutually register as neighbors. However, manual peering might be susceptible to these attacks (including social engineering ones) and can be also generally inconvenient. To this end, and to simplify the setup process of new nodes, we introduce a mechanism that allows nodes to choose their neighbors *automatically*. The process of nodes choosing their neighbors without manual intervention by the node operator is called *autopeering*.

Specifically, in this section we propose an autopeering mechanism which achieves two important goals: First, it creates an infrastructure where new nodes can easily join the network; second, we make sure that an attacker cannot target specific nodes during the peering process, i.e., we ensure the network to be secure against Eclipse attacks.

For more technical details, we refer the interested reader to the source code of our autopeering simulator[9] as well as its integration on *GoShimmer*[10].

4.1 Peer discovery

In order to establish connections, a node needs to discover and maintain a list of the reachable IP addresses of other peers (*peer discovery*). Then, the node has to select to which peers it establishes connections as well as how to handle incoming connections (*neighbor selection*).

To bootstrap the *peer discovery*, a node must be able to reach one or more *entry nodes*. Thus, the protocol provides a hard coded list of trusted "entry nodes" run by the IF or by trusted community members that answer to peer-discovery requests from new nodes. This approach is a common practice of many distributed networks [61]. Moreover, Public Key-based Cryptography (PKC) should be used for mutual authentication in order to avoid "malicious nodes" from hijacking the bootstrap phase. One way to implement such authentication is to use a *ping-pong* protocol: a peer X sends a *ping* message, containing its public key, signed with its private key to a peer Y, which in turn, replies with a signed *pong* containing Y's public key. Both peers authenticate to each other by verifying that the respective signatures are valid and add the other peer to

[9] https://github.com/iotaledger/autopeering-sim
[10] https://github.com/iotaledger/goshimmer

the *verified peer-list* (i.e., the list of authenticated peers). In the case of *entry-nodes*, a peer must know in advance their public keys, to verify the signed pong messages.

In a distributed environment, the list of reachable peers changes over time since nodes can continuously join or leave the network. To keep this list up-to-date, we assume that nodes periodically communicate a subset of their known peers to others. Each node, upon reception of a list of known peers from other nodes, adds this information to its *unverified peer-list* (i.e., the list of known peers to be authenticated). Once the *unverified peer-list* of a peer is non-empty, a peer X sends a signed *discovery-request* message to some peer Y, which belongs to its *unverified peer-list.* Peer Y replies with a signed *discovery-response* message containing a subset of its known peers. Upon reception and verification of the *discovery-response* message, peer X adds the received peers to its *unverified peer-list* and starts the authentication *ping-pong* protocol with them. This mechanism is simple and effective as it allows every node to learn about other network participants.

It is important to note that this mechanism only requires to have access to a large enough fraction of the network such that the *verified peer-list* (i.e., potential neighbors) contains "enough" nodes[11].

4.2 Neighbor selection

The goal of the *neighbor selection* is to prevent attackers from "tricking" other nodes into becoming neighbors. Neighbors are established when one node sends a *peering-request* message to another node, which in turn accepts or rejects the request with a *peering-response* message. To prevent attacks, we attempt to make the *peering-request* verifiably random. If the requests are random, attackers cannot create nodes to which the target node will send requests. Furthermore, a node must check that the incoming requests are indeed random, in order for it to screen out attacking requests.

Nodes choose half of their neighbors themselves and let the other half be comprised of neighbors that choose them. The two distinct groups of neighbors are consequently called:

- *Chosen neighbors.* The peers that the node proactively chooses from its list of neighbors.
- *Accepted neighbors.* The peers that choose the node as their neighbor.

In order to select *chosen neighbors* from the list of potential peering partners, we measure the distance between two nodes through the distance function d, defined by

$$d(nodeId_1, nodeId_2, \zeta) = hash(nodeId_1) \oplus hash(nodeId_2 + \zeta),$$

[11] The required number of potential peers needed in the list depends on the gossip protocol as well as global system parameters such as the number of neighbors.

where ζ is a public *salt*[12] (we discuss how salts are chosen in Subsection 4.4).

In order to connect to new neighbors, each node with ID *ownId* and public salt ζ keeps a list of potential peers that is sorted by their distance $d(ownId, \cdot, \zeta)$. Then, the node sends them peering requests in ascending order, containing its own node ID, its current public salt and its address (i.e., IP + port). After that, the requested node can decide to either accept or reject the connection as explained below. The connecting node repeats this process until it has established connections to enough neighbors or it finds closer peers. Those neighbors make up its list of *chosen neighbors*. This entire process is also illustrated in Algorithm 1.

```
Algorithm 1: Select chosen neighbors
  Input: desired amount of neighbors $k$, current list of chosen neighbors $\mathcal{C}$,
         list of potential peers $\mathcal{P}$
  $\mathcal{P}_{sorted} \leftarrow$ sortByDistanceAsc($\mathcal{P}$, $ownId$, $\zeta$)
  foreach $p \in \mathcal{P}_{sorted}$ do
      $peerRequest \leftarrow$ sendPeerRequest($p$)
      if $peerRequest.accepted$ then
          append($\mathcal{C}$, $p$)
          if $|\mathcal{C}| \geq k/2$ then
              return
```

Similarly to the previous case, in order to accept neighbors, every node with ID *ownId* must generate a *private* salt ζ^*. When it receives a peering request from a node with ID *remoteId*, it measures $d(ownId, remoteId, \zeta^*)$ and only accepts the request if at least one of the following conditions is satisfied:

- The connecting node is closer than an existing *accepted neighbor*.
- The node receiving the peer request does not have enough neighbors.

In addition, the receiving node can (and should) apply a statistical test to the request, and only accept the request if

$$d(remoteID, ownId, \zeta_{remote}) < \theta$$

for a fixed threshold θ. This test determines if the request was indeed random.

4.3 Network reorganization and eclipse protection

Although we have not yet discussed how salts are selected, we assume in this section that public salts are unpredictable and verifiably random - that is, everyone can check that they are random.

[12]Salts defend against dictionary attacks or against their hashed equivalent, the precomputed rainbow table attack.

The public and the private salts help to create an asymmetric perception of the network, which is supposed to discourage an attacker from harming the system. Through this, the only way to target a node in the autopeering process is by brute forcing different node identities and hoping to get closer (in terms of distance d) than an existing neighbor.

The effectiveness of this brute force method is determined by a statistical test and a threshold θ. The smaller the value of θ, the more nodes an attacker requires to have a reasonable chance of being successful. For example if we set $\theta = .01$, an attacker only has 1% of creating a connection with a desired peer. In a similar way, the statistical test allows other nodes to judge the quality of a connection. More specifically, if the distance is small enough, the connection is assumed to be good.

Furthermore, to prevent attacks nodes will change their salts periodically. This frequent reorganization brings a twofold benefit: First, it prevents attackers from affecting the network topology; second, it supports new nodes that want to join the network as their peering requests will be accepted with larger probability.

4.4 Choosing salts

In this subsection we discuss how to choose salts. Contrary to private salts, which can be any randomly generated number, the public salts have to be verifiably random. That is, nodes must be able to verify the public salts used in requests. If an adversary could choose them completely at will, then salts would be mineable, in a manner where the adversary can achieve a minimum distance for any given distance function.

The simplest way to set public salts would be to use a centralized random number beacon (such as the beacon operated by NIST [53]). Indeed, a node could set its public salt to be its ID hashed with every 1000th random number issued by the beacon. However, such a centralized option is unpalatable in a DLT space.

Another similar option is using the same distributed random generator used in the Fast Probabilistic Consensus (FPC) protocol (see Section 6.1). However, there are some issues that would need to be resolved: nodes would receive the random number through gossip, but new nodes would not be able to participate in the gossip without any neighbors.

Another option is to use hash chains [56]. When a new node joins the network, it creates a hash chain, $\zeta = \{\zeta_0, \zeta_1, \zeta_2, \ldots, \zeta_m\}$, where $\zeta_{i+1} = hash(\zeta_i)$. Then, when nodes share their ID in the peer discovery, they also reveal the number ζ_m. The salt would then be some value element of the hash chain ζ_i with i decreasing on the time. Thus, accepting nodes can verify that the salt is correct, without compromising the unpredictability of the salts. These hash chains do leave the attacker some amount of freedom, since a desirable hash chain can be mined. However, since this would require advanced planning, and at most one ζ_i could be mined in the chain, a desired distance to a certain node can only be created for a short period of time.

4.5 Sybil protection

By creating large amounts of nodes for free, an attacker may gain influence on the network topology and might be able to perform eclipse attacks. In order to effectively prevent this kind of attacks, we introduce a Sybil protection. As discussed in Section 3.2, the Sybil protection in the IOTA protocol is given by *mana*.

We outline the main idea how *mana* can be implemented into our autopeering mechanism. The simplest way is to require nodes to have a minimal amount of *mana* to participate in the autopeering. This is problematic for two reasons. First, we want as many nodes to participate as possible, and setting a *mana* requirement could hurt participation. Second, attacking nodes will be encouraged to split up their nodes into several smaller attacking identities. Moreover, any hard threshold criterion[13] may lead to poor connectivity properties of the autopeering network. For these reasons, our preferred option is to weight the distance function by *mana*.[14] This would make low *mana* nodes less favorable but still allow them to participate. Moreover, nodes with similar *mana* are likely to be paired with each other. Thus, to eclipse a high *mana* node, an attacker would also need many high *mana* nodes.

[13]For instance, we could pick a parameter $a > 1$, and a node with M *mana* would only peer with nodes whose *mana* is in the interval $[M/a, Ma]$.

[14]For example, we can modify the distance between nodes X and Y defined at the beginning of this section to

$$d(X,Y,\zeta) = f(|k_X - k_Y|)\,[hash(nodeId_X) \oplus hash(nodeId_Y + \zeta)]\,,$$

where k_i is the rank (i.e. the position in the ordered list of nodes) of the node i according to mana, and $f(s)$ is some increasing function, such as $e^{\alpha s}$ or s^3.

5 Rate control

A basic goal of every communication network is to handle the traffic injected by its nodes by limiting the rate of transactions joining the network. In fact, such a traffic could lead to unpleasant situations such as network congestion, due to resource limitations, or spam, due to malicious actors:

- *Congestion control.* In most networks, there are circumstances where the incoming traffic load is larger than what the network can handle. If nothing is done to restrict the influx of traffic, bottlenecks can slow down the entire network. A similar analysis can be applied to distributed ledgers, where the incoming traffic (i.e., transactions issued by the nodes of the network) exploits limited resources such as bandwidth, computational power, or disk space. Additionally, nodes can lose synchronization with each other, sometimes without being aware of it.
- *Spam detection.* Gossip protocols (which are currently implemented to forward transactions in the IOTA network) are an efficient and reliable way to disseminate information. These protocols have nevertheless a drawback: They are unable to limit the dissemination of spam messages. Indeed, messages are redundantly distributed in the network and it is enough that a small subset of nodes forward spam messages to have them received by a majority of nodes.

Rate limitation strategies for communication networks are well studied in the case of both congestion control [52] and spam detection [33]. The former has been deeply investigated after the Internet collapse in the 80's, and solutions based on *Additive Increase Multiplicative Decrease* have been proposed to properly modulate the influx of packets in the network [25, 21, 50]. We plan to adopt a similar strategy to deal with potential congestion in the Tangle. On the other hand, the seminal work by Dwork and Naor [33] in 1993 paved the way to research in spam prevention. In the context of DLTs, many blockchains use PoW as a built-in rate limitation mechanism. However, PoW leads to undesirable side effects such as mining races: The discrepancy between smaller general purpose devices and optimized hardware with respect to the PoW performance is several orders of magnitude. Hence, any rate control based on PoW would eventually leave smaller devices behind. A new transaction rate control mechanism for the Tangle is therefore required to deal with the global and per-node limitations of the network. In the rest of the section, we will investigate two anti spam mechanisms based on varying difficulty PoW (Section 5.1) and on non-parallelizable functions (Section 5.2).

5.1 Adaptive PoW

In a pure PoW-based architecture, a high difficulty value would prevent low-power nodes from issuing transactions, which is not desirable, especially in the context of Internet-of-Things; on the other hand, low difficulty can quickly lead to network congestion. We propose an adaptive PoW algorithm to allow every node to issue transactions while penalizing spamming actions.

5.1.1 Algorithm

All nodes in the network have knowledge of three global parameters:

- *Base difficulty* d_0. It sets the minimum difficulty of PoW.
- *Adaptation rate* $\gamma_i \in [0, 1]$. It provides the rate at which the PoW difficulty will be adjusted, which depends on the mana m_i owned by node i.
- *Time window* $W > 0$. This parameter defines the granularity of the algorithm, and its role will be clarified below.

Similarly to the current implementation, issuing a new transaction must be preceded by the computation of some PoW. At time t, node i must perform a PoW with difficulty $d_i(t)$ which can be calculated by

$$d_i(t) = d_0 + \lfloor \gamma_i \cdot a_i(t) \rfloor \,, \tag{7}$$

where $a_i(t)$ represents the number of transactions issued by node i in the time interval $[t - W, t]$. Note that when $\gamma_i = 0$, the algorithm becomes equivalent to the current IOTA implementation.

Again, received transactions are processed in FIFO order. Let us assume we receive a transaction with difficulty d_i signed by node i. To decide whether this transaction should be forwarded or not, a node counts how many transactions $r_i(t)$ signed by i has been received in the last W seconds. In accordance to the formula given by Eq. (7), the node forwards the transaction if the following condition is satisfied:

$$d_i \geq d_0 + \lfloor \gamma_i \cdot r_i(t) \rfloor \,.$$

Due to the asynchronous nature of the system, some of the transactions may be received in burst, invalidating the previous formula. We consider a correcting term $c > 0$ which allows to accept transactions with lower difficulty. This parameter is known by all nodes. We modify the previous formula into:

$$d_i \geq \max\{d_0;\; d_0 + \lfloor \gamma_i \cdot r_i(t) - c \rfloor\}.$$

For the sake of simplicity, we assume incoming transactions are checked in the same order as they are issued by the sending node. As the expected time needed to perform the PoW is typically much larger than the network latency h, this is a reasonable assumption.

Assume that a transaction is seen for the first time at time t_0. Every node will store the *id* of the node issuing the transaction, the PoW difficulty computed and the time t_0. The identity *id* of the issuing node as well as its mana m_{id} can be determined using the methods described in Section 3. Based on this information, it can then be checked that the difficulty of the most recent transaction is indeed sufficient. This idea is more formally described in Algorithm 2.

Algorithm 2: Rate control algorithm

```
Input: incoming transaction $tx$, set of known transactions $\mathcal{X}$, time
       window $W$, basic difficulty $d_0$, adaptation rate $\gamma_i$.
Output: forward or ignore $tx$.
$t_0 \leftarrow$ time($tx$);
$id \leftarrow$ nodeId($tx$);
$\mathcal{T} \leftarrow tx' \in \mathcal{X} \mid$ time($tx'$) $\in (t_0 - W, t_0] \wedge$ nodeId($tx'$) $= id$;
if difficulty($tx$) $\geq d_0 + \gamma_i \cdot |\mathcal{T}|$ then
    return forward $tx$;
return discard $tx$;
```

5.1.2 Theoretical analysis

In the IOTA protocol implementation the number of operations triples for each increment of the difficulty. Let b be the *mean* number of operations needed to solve the PoW at difficulty 1. Furthermore, following the definition of *gamma*, $1/\gamma$ is the maximum number of transactions that can be sent at a given difficulty during a time window W (note that we omit the index i as we perform the analysis on a single node). Hence, the number of operations that can be computed by a node during a time window, i.e., $\mu \cdot W$, produces a bound on the node's throughput:

$$\begin{aligned}\mu \cdot W &\geq \frac{b \cdot 3^{d_0}}{\gamma} + \frac{b \cdot 3^{d_0+1}}{\gamma} + \ldots + \frac{b \cdot 3^{d_0+n}}{\gamma} \\ &= \frac{b}{\gamma} \cdot \sum_{i=0}^{n} 3^{d_0+i} \\ &= \frac{b}{\gamma} \cdot \frac{3^{d_0+n} - 1}{3^{d_0} - 1},\end{aligned}$$

which, after elementary computations, gives

$$\begin{aligned}n &\leq \log_3 \left(\frac{\gamma \cdot W \cdot (3^{d_0} - 1)}{b} \cdot \mu + 1 \right) - d_0 \\ &\approx \log_3 \left(\frac{\gamma \cdot W}{b} \cdot \mu \right). \end{aligned} \tag{8}$$

Furthermore, consider that the throughput is equal to the total number of transactions issued by a node, i.e., $\frac{n-d_0}{\gamma}$, over a time window W. Hence, the throughput for a node with computational capability μ is

$$\begin{aligned}\theta(\mu) &= \frac{n - d_0}{\gamma \cdot W} \\ &\leq \frac{\log_3 \left(\frac{\gamma \cdot W}{b} \cdot \mu \right) - d_0}{\gamma \cdot W}.\end{aligned}$$

5.1.3 Simulations

We built a Python simulator with Jupyter notebook[15] in order to verify whether the adaptive PoW algorithm can mitigate the problems described in the beginning of this section. In the simulations, we considered a simple scenario with a single node to evaluate the maximum throughput it can generate.

Suppose a node has computational capability μ measured in number of operations per second. We assume a node can belong to one of the following three categories depending on its computational power:

- IoT, where $\mu = 10^{-1} \times 10^6$;
- Laptop, where $\mu = 10^0 \times 10^6$;
- FPGA, where $\mu = 10^6 \times 10^6$.

Furthermore, we assume that the number of operations needed to solve PoW at difficulty 14 is a random variable uniformly distributed with mean $3^{14} \approx 5 \times 10^6$. Hence, an IoT device will be able to solve it with an average time of 1 minute.

Finally, we set the following global parameters:

- Base difficulty $d_0 = 10$;
- Adaptation rate $\gamma = 0.1$, which means the node increases the PoW difficulty every 10 transactions sent within the time window;
- Time window $W = 1000$ s.

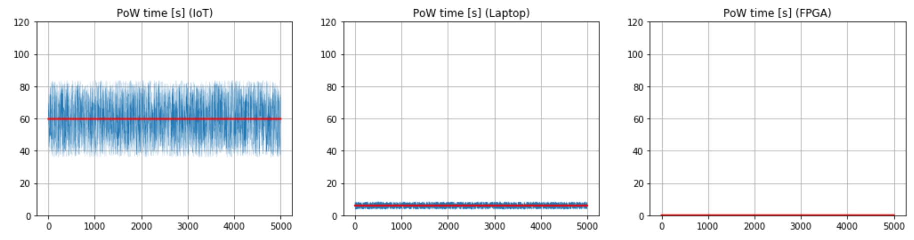
In the following subsection, we present the simulation results when a node wants to issue 5000 transactions in the shortest possible time, i.e., we aim to maximize the total throughput.

In this subsection, we present the simulations results based on the current fixed PoW algorithm, i.e., when $\gamma = 0$. In Fig. 2a, we show the time (in seconds) needed to compute the PoW for IoT, laptop and FPGA. The red line shows the average time needed to compute the PoW. As we expect, IoT devices require on average 1 minute for a PoW difficulty of 14. Laptops improve this time to about 6 seconds, while FPGAs run 6 orders of magnitude faster than laptops (i.e., they solve the PoW in a few microseconds).

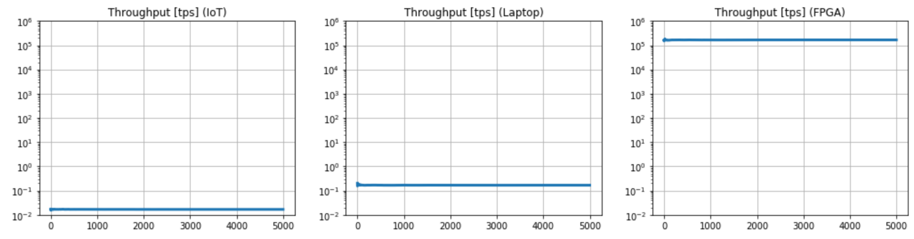
According to these numbers, in Fig. 2b we show the throughput generated by the nodes, computed as the number of transactions per second. In the current IOTA implementation, the usage of FPGAs can speed up the computation by several orders of magnitude, preventing low-power nodes to access the network successfully and enabling spam attacks.

In this section, we analyze the adaptive PoW algorithm. In Fig. 3a we show how the PoW difficulty changes over time. In particular, we see an initial transient phase where the difficulty progressively increases. Then, the difficulty

[15] https://github.com/iotaledger/adaptive-pow-sim



(a) Time (in seconds) needed to compute fixed PoW (difficulty = 14). The red line shows the average PoW time.



(b) Throughput (in transactions per second) with fixed PoW (difficulty = 14).

Fig. 2: Fixed PoW

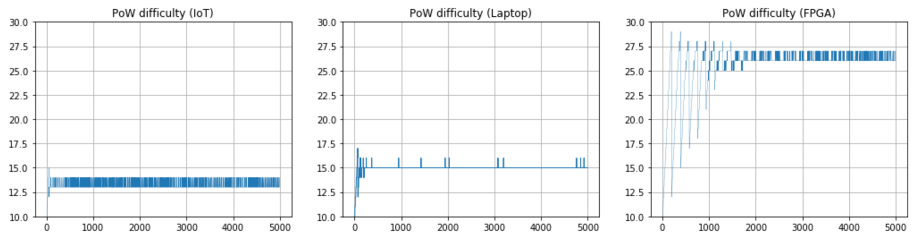
oscillates around a certain value, which depends on the computational capabilities of the device: for instance, IoT devices are able to solve a PoW with difficulty 14, while FPGAs can solve up to a difficulty of 27.

Such a different difficulty mitigates the gap between the solution time for the PoW between the different devices: while IoT can solve the puzzle faster than in the fixed scenario, high power devices see their PoW time raising considerably. This is the key principle behind the adaptive PoW algorithm: make life easy to IoT devices (which is one of the most important IOTA use cases), while bound the power of FPGAs and ASICs (Fig. 3b).

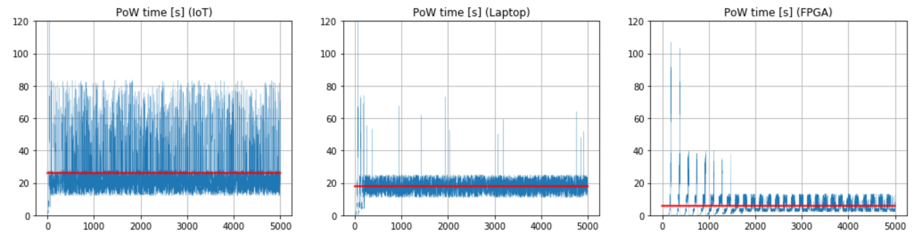
In fact, it is interesting to see that specialized hardware cannot spam the network indefinitely or create congestion, because its allowed throughput is capped in the same order of magnitude of low-power devices. Fig. 3c shows this fundamental result, which proves the validity of the proposed approach.

5.2 Verifiable delay functions

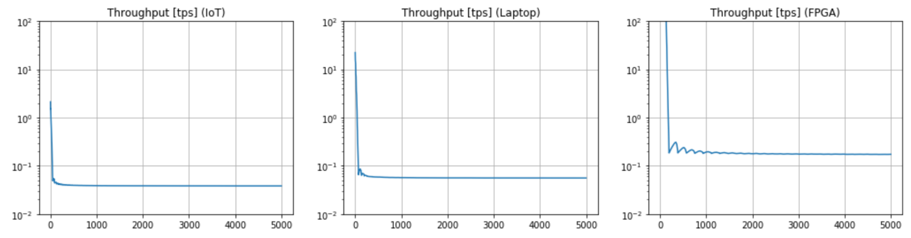
While the adaptive PoW algorithm described in Section 5.1 mitigates some of the drawbacks of PoW, we believe that, in the current era of distributed ledger ecosystems, the need for more efficient algorithms is evident. In the following, we present a more sustainable mechanism that might be used as a complete replacement of the PoW component based on *verifiable delay functions* (VDFs).



(a) PoW difficulty variation over time.



(b) Time (in seconds) needed to compute adaptive PoW. The red line shows the average PoW time.



(c) Throughput (in transactions per second) with adaptive PoW.

Fig. 3: Adaptive PoW

5.2.1 Preliminaries

Informally, the VDFs are special functions that are (i) difficult to evaluate, even under the assumption of using unbounded parallelism (i.e., using an infinite number of CPUs) [12] but (ii) easy to verify. Compared to PoW, these functions bring the following advantages:

- VDFs can be considered more environment friendly since they avoid mining races.
- As they are not parallelizable, they make the usage of dedicated hardware inefficient (e.g., ASIC), inherently solving the problem of unfairness between slow and fast nodes.

The first ideas about verifiable delay functions can be traced back to the seminal paper of Dwork and Naor in the field of spam protection [33], but it is only after the recent paper by Boneh *et al.* [10] that the interest in the development and implementation of VDFs has substantially increased. Various researchers have proposed different VDFs based on specific number-theoretic functions (e.g., modular exponentiation [33, 64], supersingular isogenies over elliptic curves [29], pairings over elliptic curves, injective rational maps between extensions of finite fields [10]). In fact, VDFs are already an essential ingredient in some DLT designs (e.g., Chia Network[16]). Furthermore, [10] has shown a potential application for decentralized randomness.

It is worth reiterating that spam detection is a field where VDFs has already been applied by the Dwork-Naor algorithm, which uses the square root over finite fields puzzle. The main reason why their work has been considered as impractical is the fact that one has to use rather large finite fields to make the algorithm useful. At the time of the suggestion of the algorithm, early 1990s, the existing libraries for handling multiple-precision arithmetic were orders of magnitude slower than the current ones.

5.2.2 Protocol

In this section, we present a novel anti-spam mechanism for IOTA based on Wesolowski VDF [78] which aims to overcome the limitations of PoW (we choose the Wesolowski construction because it guarantees fast verification time and low overhead). The research on VDF is actively ongoing and this algorithm can be considered as a future replacement for adaptive PoW.

As an initial setup for the protocol, the network participants are required to agree on (i) a public parameter N, which will serve as the modulus for the VDF, and on (ii) a cryptographic hashing function H, which will be used in the computation of the proof. As for (i), the parameter $N = p \cdot q$ is an RSA modulus computed as the product of two prime numbers of the same order, i.e., having the same bit-length. The security of the entire mechanism relies on the length of N: the longer the modulus is, the more difficult it is to find its factorization. A

[16] www.chia.net

fundamental question is how to generate the RSA modulus without disclosing its factorization to any of the network participants or relying on any sort of centralization. The literature on distributed RSA keys generation already counts several existing solutions [11, 27, 45, 40]. We describe our proposal in [4] where our algorithm exploits smooth numbers to generate fast prime numbers [31] and, hence, to reduce the overall communication complexity. As for (ii), for a given binary message $m \in \{0,1\}^*$, we define a hashing function H with security level k (i.e., breaking the hashing function's security would take approximately 2^k computations) such that $H(m) : \{0,1\}^* \mapsto [0, 2k]$. Having this in mind, we also define $H_{prime}(m)$ as the smallest prime greater or equal to $H(m)$ for a given message m:

$$H_{prime}(m) = \min\{x \in \mathbb{N} \mid x \geq H(m) \land x \texttt{ prime}\}.$$

Each time a node tries to issue a transaction, it has to compute the VDF solution y, which can be solved only using τ sequential squarings, such that

$$y = x^{2^\tau} \bmod N, \tag{9}$$

where $x = H(m)$. The evaluation takes as inputs the message hash x and a difficulty $\tau \in \mathbb{N}$, computed depending on mana. Furthermore, we enforce that a node must choose m as the VDF output of its previous transaction. In this way, the protocol ensures the sequentiality of the VDF evaluations because a node has to wait for a VDF to complete before starting a new computation. In other terms, VDFs cannot be run in parallel in order to overcome node's allowed throughput.

Algorithm 3: Evaluation and proof of the VDF

```
input  : $m \in \{0,1\}^*$, $\tau \in \mathbb{N}$
output: $\pi \in [0, N-1]$, $l$ prime $\in [0, 2^{2k} - 1]$
$x \leftarrow H(m)$
$y \leftarrow h$
for $k \leftarrow 1$ to $\tau$ do
    $y \leftarrow y^2$
end
$l \leftarrow H_{prime}(x + y)$
$\pi = x^{\lfloor 2^\tau / l \rfloor}$
return $(\pi, l)$
```

Unlike PoW, in VDF there is no trivial way to verify whether a node has indeed performed the work to obtain the output y. For this reason, we propose to attach an additional number, the *proof*, to help other nodes during the verification. In order to generate the proof, the node computes $l = H_{prime}(x + y)$ and $\pi = x^{\lfloor 2^\tau / l \rfloor}$. The summation between x and y ensures that the challenge has been successfully solved. In order to keep the transaction size small, the

Algorithm 4: Verification of the VDF

```
input  : $x, \tau, \pi, l$
output: $\top$ or $\bot$
$x \leftarrow H(m)$
$r \leftarrow 2^{\tau} \bmod l$
$y \leftarrow \pi^{l} \cdot x^{r}$
if $l = H_{prime}(x + y)$ then
    return $\top$
else
    return $\bot$
end
```

evaluator will only attach the resulting proof, i.e., the pair $\{l, \pi\}$, since the solution y can be inferred from it. The Algorithm 3 describes the evaluation and proof processed by a node.

A node receiving a new transaction has to reconstruct the solution y to verify whether the VDF has been properly evaluated. To do it, the node will compute $y = \pi^{l} \cdot x^{r}$, where $r = 2^{\tau} \bmod l$. The challenge y is then correct if $l' = l$, where $l' = H(x + y)$. We describe the verification task in Algorithm 4.

6 Consensus and Voting

Due to the propagation delay of transactions in the network, not all nodes share the same vision of the Tangle at the same time. This might lead to situations where the validation process lets multiple transactions in conflict with each other join the Tangle. It is a fundamental assumption of the IOTA white paper that the Tangle itself can indeed contain conflicting transactions. In case of a conflict, however, the nodes need to decide which transaction(s) should be considered valid, i.e., they need to come to a *consensus* on those conflicting transactions.

In the original IOTA white paper this is solely achieved by consistently applying the tip selection algorithm (TSA), i.e., the mechanism used by (honest) nodes to select the transactions to approve, which currently uses a biased random walk. In case of a conflict, this bias will eventually leave all but one of the conflicting branches behind. However, this approach is not suited for our current vision of the Tangle, as conflict resolution is slow and it leads transactions that chose the "wrong" branch to be orphaned, creating the need for a large number of reattachments.

In this section we discuss a consensus mechanism which we call Shimmer and that is a mechanism to achieve consensus that is robust against potential attacks. The Shimmer voting scheme is named after an extraordinary behavior seen in nature. Bees "synchronize" their movement to defend themselves against predators. They do this without any centralized entity, and only know when to "change their state" by observing the behavior of their peers. Individual autonomous agents that act according to some predefined rules can be found in many systems in nature, such as bees, ants, schools of fish and even in some areas of physics. Very simple rules can create incredibly complex features that, over time, manifest as emergent properties of a system. The Shimmer consensus mechanism works in the same way. Instead of trying to reconstruct the opinion of every other node, we care only about the opinions of a very small subset of nodes and let consensus be formed organically as an emergent property of the network.

More specifically the idea is that nodes query other nodes about their current opinion of the ledger, and adjust their own opinion over the course of several rounds based on the proportion of other opinions they have observed. Whilst the voting models have their limitations, they have been successfully applied in a wide range of engineering and economical applications [5, 62, 70], leading to the emerging science of sociophysics [18]. We describe two voting mechanisms where nodes communicate to each other to decide, in case of a conflict, which transaction(s) should be accepted in the Tangle.

For this section, we only consider algorithms that find consensus on the value of a single bit, i.e., of a single conflict. The result of this consensus process can then be used to mark a transaction as either "liked" or "disliked".

So, the general idea is to let the nodes talk to each other in order to resolve the conflicts *pro-actively*. The conflict resolution is performed starting from an initial opinion on the ledger status described as follows: Consider a transaction

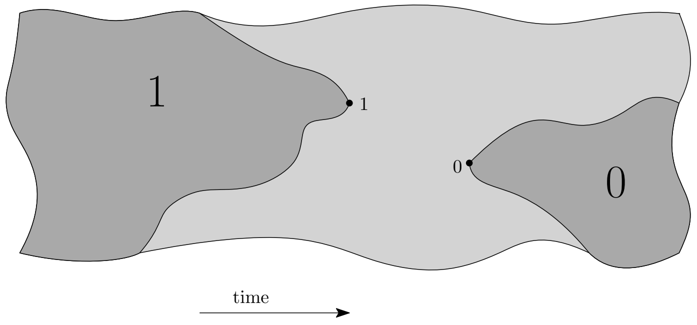


Fig. 4: Votes must be monotonous

v. If in a given interval a node does not see any other transaction spent from the same address, we say that the node *likes* transaction v; otherwise, the node *dislikes* v[17]. The decision whether a transaction is liked or disliked must then be taken into account for the tip selection. The most straightforward way of integrating this is to simply remove all the tips in the future cones of disliked transactions from the tip selection.

After that, we periodically apply a voting scheme to every transaction in the Tangle where each node asks for the opinion of some of its neighbors. After the vote, a transaction is either definitely liked or definitely disliked by a node. We would like to keep monotonicity in the sense that if a node likes u then it likes any transaction u approves, and if the node dislikes v then it dislikes any transaction that approves v, see Fig. 4. To achieve this, we can safely assume that we can only like transaction v when we like all of its past cone, and if we dislike v then we dislike all of its future cone.

In the following two subsections, we will describe two voting mechanisms we are considering. The first one, called *Fast Probabilistic Consensus* [68], is certified by rigorous mathematical proofs; however, this solution requires nodes to accept connections from nodes which are not neighbors, and uses decentralized randomness, that needs to be acquired as part of an additional layer. On the other hand, the cellular automaton approach of Section 6.2 does not have those requirements and seems to be faster from simulation results; however, this scheme lacks rigorous proofs and requires a stricter autopeering solution to avoid Eclipse attacks, and formation of "islands" of adversarial nodes. The two solutions can be considered as different non-mutually exclusive implementations of the voting mechanism, and they can be used in combination to build a robust framework.

For more technical details, we refer the interested reader to the source code of our FPC simulator[18], as well the integration as a prototype version of FPC

[17]It is important to note, that this rule does not include reattachments: If $v_1, \dots, v_k$ are all reattachments of the same transaction, we either like all or none of them.

[18]https://github.com/iotaledger/fpc-sim

and the cellular automaton approach in *GoShimmer*[19].

6.1 Fast probabilistic consensus (FPC)

The paper [68] introduces a protocol of low communicational complexity which allows a set of nodes to come to a consensus on a value of a bit by means of (possibly randomized) voting (see e.g., [7, 23, 24, 34, 35, 57] and references therein for the vast available literature on this subject). The FPC relies on the idea that randomised voting, i.e., random queries, is in some situations sufficient for good performance, and due the small message complexity the protocol becomes scalable. Another advantage of this randomness is robustness in less reliable networks and in situations with inevitable churns (nodes join and leave), e.g., we refer to [77] for an application of this idea in distributed machine learning.

We mention also the classical work on (probabilistic) Byzantine consensus protocols, see e.g., [2, 8, 13, 19, 36, 41, 71], where, typically, the communicational complexity is much larger.

In order to analyze the security of FPC, we must define our assumptions about the behaviors of the participants in the system. An *honest* participant is one that we assume to follow the system's protocol rules as specified, hence representing a participant exhibiting no adversarial behavior. A *Byzantine* (a.k.a., *active* or *malicious*) participant is one we make no assumptions about; such a participant can behave in arbitrary fashion, without any restriction. Since this is the strongest adversarial model we will focus primarily on Byzantine attacks. We further assume that the Byzantine attacker is *omniscient*, i.e., it is aware of the current opinion of every other node and observes all communications[20] and that the adversary is *rushing*, i.e., it may delay sending its own messages in any given round until after the honest parties send their messages in that round. We refer to [39] for more details on threat modeling and different kind of attackers.

We have to make assumptions on the communication model of the FPC. We assume the communication between two nodes to satisfy *authentication*, i.e., senders and receivers are who they claim to be, and *data integrity*, i.e., data is not changed from source to destination. As we consider omniscient adversaries we do not assume *confidentiality*. For the communication of the opinions between nodes we assume a *probabilistic synchronous model*, in which for every $\varepsilon > 0$ and $\gamma > 0.5$, a majority proportion γ of the messages is delivered within a bounded (and known) time, that depends on ε and γ, with probability of at least $1 - \epsilon$. We want to emphasize that, due to its random nature, FPC still shows good performances in situations where not all queries are answered in due time.

The aim is to build systems that could withstand the most participants being Byzantine. In practice we have to make threshold security assumptions, such as that over half or over two-thirds of the participants are honest. The

[19]https://github.com/iotaledger/goshimmer

[20]This is one reason why we do not consider *honest-but-curious* (a.k.a., *passive* or *semi-honest*) adversaries separately.

distinguishing feature of the mechanism described in [68] is that a larger number of adversarial (or Byzantine) nodes is allowed, which may be a (fixed) proportion of the total number of nodes. Those adversarial nodes intend to either delay the consensus, or break it (i.e., make at least a couple of honest nodes come to different conclusions). It is shown that, nevertheless, the protocol works with high probability when its parameters are suitably chosen, and some explicit estimates on the probability that the protocol finalizes in the consensus state in a given time are also provided (see Theorems 2.1 and 2.4 of [68]).

Differently from the classical work in this area, it is not required that the consensus should be achieved, with high probability, on the initial majority value. Rather,

- if, initially, no *significant majority*[21] of nodes prefer 1, then the final consensus will be on the value 0 with high probability;
- if, initially, a *supermajority*[22] of nodes prefer 1, then the final consensus will be 1 with high probability.

To explain why this is relevant in cryptocurrency applications, consider a situation when there are two contradicting transactions; for example, one of them transfers all the balance of address A_1 to address A_2, while the other transfers all the balance of address A_1 to address $A_3 \neq A_2$. In the case when neither of the two transactions is strongly preferred by the nodes of the network, by declaring both invalid we are on the safe side. On the other hand, it would not be a good idea to *always* declare them invalid. Indeed, if we do this, then a malicious actor would be able to exploit it in the following way: First, he places a legitimate transaction, e.g., to buy some goods from a merchant. When he receives the goods, he publishes a double-spending transaction as above in the hope that *both* would be canceled by the system, and so he would effectively receive his money back (or at least take the money away from the merchant). To avoid this kind of development, it would be desirable if the first transaction (payment to the merchant) which, by that time, have probably gained some confidence from the nodes, would stay confirmed, and only the subsequent double-spend gets canceled.

A special feature of the protocol of [68] is that it makes use of a sequence of random numbers which are either provided by a trusted source [53] or generated by the nodes themselves using some decentralized random number generating protocol (provided that the proportion of the adversarial nodes is not too large) by leveraging on cryptographic primitives such as verifiable secret sharing, threshold signatures, cryptograhic sortition or verifiable delay functions, see e.g., [17, 65, 73, 75, 44, 10]. More specifically, a reasonable approach could be to use a variant of the *drand* protocol[23] (already used by other projects such

[21] Loosely speaking, a significant majority is something statistically different from the 50/50 situation; for example, the proportion of 1-opinion is greater than ϕ for some fixed $\phi > 1/2$.

[22] Again, this is a loosely defined notion; a supermajority is something already *close* to consensus, e.g., more than 90% of all nodes have the same opinion.

[23] https://github.com/dedis/drand

as *The League of Entropy*[24]) as detailed in a post on *iota.cafe*[25].

It is important to observe that, even if from time to time the adversary can get (total or partial) control of the random number, this can only lead to delayed consensus, but he cannot convince different honest nodes of different things, i.e., safety is not violated. Also, it is not necessary that really *all* honest nodes agree on the same number; if most of them do, this is already enough for the protocol (that is, the specific task of random number generation does not require any sort of "strong consensus"). Similarly a random number is also not required for every round, see Fig. 17 in [16].

6.1.1 The base protocol

We present here only some key elements of the proposed protocol and refer the interested reader to [16] and [68] for more details. In order to define FPC we have to introduce some notation. We assume the network to have n nodes indexed by $1, 2, \ldots, n$.[26] Every node i has an opinion or state. We note $s_i(t)$ for the opinion of the node i at time t. Opinions take values in $\{0, 1\}$.

At each (discrete) time step each node chooses k random nodes C_i, queries their opinions and calculates

$$\eta_i(t+1) = \frac{1}{k_i(t)} \sum_{j \in C_i} s_j(t),$$

where $k_i(t) \leq k$ is the number of replies received by node i at time t and $s_j(t) = 0$ if the reply from j is not received in due time. Note that the neighbors C_i of a node i are chosen using sampling with replacement and hence repetitions are possible.

As in [16] we consider a basic version of the FPC introduced in [68] in choosing some parameters by default. Specifically, we remove the cooling phase of FPC and the randomness of the initial threshold τ. Let $U_t, t = 1, 2, \ldots$ be i.i.d. random variables with law Unif$([\beta, 1-\beta])$ for some parameter $\beta \in [0, 1/2]$. The update rules for the opinion of a node i is given by

$$s_i(1) = \begin{cases} 1, & \text{if } \eta_i(1) \geq \tau, \\ 0, & \text{otherwise,} \end{cases}$$

and for $t \geq 1$:

$$s_i(t+1) = \begin{cases} 1, & \text{if } \eta_i(t+1) > U_t, \\ 0, & \text{if } \eta_i(t+1) < U_t, \\ s_i(t), & \text{otherwise.} \end{cases}$$

[24] https://www.cloudflare.com/leagueofentropy/

[25] https://iota.cafe/t/distributed-random-number-generator/243

[26] This assumption is only made for sake of a better presentation; a node does not need to know every other node in the network. While the theoretical results in [68] are proven under this assumption, simulation studies [16] indicate that it is sufficient if every node knows about half of the other nodes.

Note that if $\beta = 0.5$, FPC reduces to a standard majority consensus. The above sequence of random variables U_t are the same for all nodes, see also the discussion in the previous section.

We introduce a local termination rule to reduce the communication complexity of the protocols. Every node keeps a counter variable `cnt` that is incremented by 1 if there is no change in its opinion and that is set to 0 if there is a change of opinion. Once the counter reaches a certain threshold `l`, i.e., `cnt` $\geq$ `l`, the node considers the current state as final. The node will therefore no longer send any queries but will still answer incoming queries. In absence of autonomous termination the algorithm is halted after `maxIt` iterations.

6.1.2 Byzantines adversaries

We consider the "worst-case" scenario where adversarial nodes can exchange information freely between themselves and can agree on a common strategy. In fact, we assume that all Byzantine nodes are controlled by a single adversary. In [68], three different kinds of these Byzantine adversaries are described and analyzed in more detail:

- *Cautious* adversary: any adversarial node must maintain the same opinion in the same round, i.e., respond the same value to all the queries it receives in that round;
- *Semi-cautious* adversary: adversarial node will not give contradicting responses, however if it suits them they may not respond to a node;
- *Berserk* adversary: an adversarial node may respond different opinions to different queries in the same round.

Since the berserk attack strategy allows more degrees of freedom in the possible responses of the adversary, it is the most problematic one.

6.1.3 Theoretical analysis

We present some theoretical results on FPC. In [68] it is proven that, when the protocol parameters are suitable chosen, the protocol finalizes in a consensus state in finite time with high probability. We state a consequence of Theorems 4.1 and 6.2 in [68].

Theorem 1. *Let q be the proportion of adversarial nodes. Then, for any choice of $\varepsilon > 0$, and*

1. *for any cautious adversary with $q < \frac{1}{2}$,*
2. *for any semi-cautious adversary with $q < \frac{1}{1+\varphi} \approx 0.38$[27], or*
3. *for any berserk adversary with $q < 1/3$,*

[27] Interestingly, the golden ratio $\varphi = \frac{1+\sqrt{5}}{2}$ appears here.

there exist, for sufficiently large n, parameters $k, \beta,$ `l` and `maxIt` *such that the protocol finalizes in a consensus state with probability of at least $1 - \varepsilon$.*

The above results come with bounds on the values of the different parameters. Moreover, since these bounds are not yet optimal, simulations studies were performed in [16] on various explicit cautious and berserk adversarial strategies. For instance, two strategies are presented in the study, where the attack aims for an agreement failure in the protocol: the cautious inverse voting strategy (IVS) and the berserk maximal variance strategy (MVS). In the IVS, the adversary transmits at time $t+1$ the opinion of the minority of the honest nodes of step t. In the MVS, the adversary waits until all honest nodes received opinions from all other honest nodes. The adversary then tries to subdivide the honest nodes into two equally sized groups of different opinions while trying to maximize the variance of the η-values.

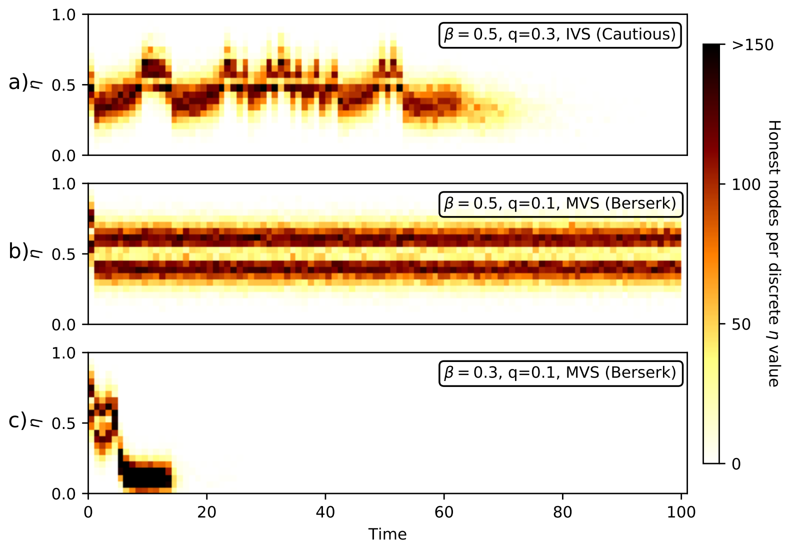


Fig. 5: Evolution of the received mean opinions η of the number of undecided nodes. We consider a network of $n = 1000$ nodes containing qn malicious nodes and a quorum size of $k = 21$. The initial average opinion p_0 equals the initial threshold $\tau = 2/3$.

Fig. 5 compares the effects of the IVS and the MVS on the histogram evolution of the nodes' received mean opinions η. In Fig. 5a) and b) the randomization of the threshold is turned off. We show that the IVS is less efficient and the adversary struggles to keep the opinions split. On the other hand the berserk MVS strategy performs much better. Fig. 5c) shows that when the random threshold is activated, the protocol performs well even against the berserk attack.

Furthermore, optimisation studies show that an optimum range for the random threshold can be found, that maximizes the resilience to adversary nodes.

For example, Fig. 6 shows the agreement rate[28] under a berserk MVS attack, against the random threshold window β and the proportion of adversary nodes q.

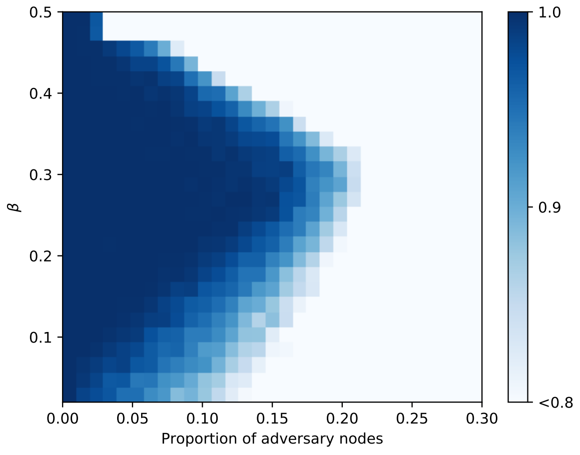


Fig. 6: Agreement rate against the parameter β that decides the randomness of the threshold, and the proportion of MVS-adversarial nodes. We consider a network of $n = 1000$ nodes containing qn malicious nodes and a quorum size of $k = 21$. The initial average opinion p_0 equals the initial threshold $\tau = 2/3$.

Figs. 5 and 6 show a worst case scenario in the following sense. The performances of the FPC are much better if the initial average opinion p_0 is not close to the initial threshold $\tau = 2/3$. It is also worth to note, that an increase of the quorum size k leads to higher agreement rates and allows FPC to withstand higher proportion of adversarial nodes.

6.1.4 manaFPC

FPC in its Vanilla version described in [68] is not robust against Sybil attacks since an adversary could deploy an excessively large number of nodes, thus inflating the value of q. In Section 3.2 Sybil protection is implemented by *mana*. We change FPC such that a node is queried proportional to its mana and do allow to query a node multiple times. As multiple queries don't increase the message complexity (a node just counts the opinion multiple times without sending multiple queries), this allows creating a bigger quorum using the same communication overhead. In turn, a larger number of samples increases the safety of the protocol. In situations with heterogeneous mana distributions,

[28]The agreement rate is the rate at which the protocol concludes with all nodes having the same opinion.

nodes with high mana might be queried by a high number of nodes. Such high-mana nodes are therefore incentivized to gossip their opinions or to publish them on the tangle as a data transaction.

To further decrease the message overhead when gossiping their opinions, nodes can apply monotonicity rules and compress their statements. For example, if a transaction is liked that contains two (liked) conflicts in its past, it would be sufficient to inform with the opinion on the former transaction. We dub these selected opinions, which effectively vote on the entire future or past of a transaction, state-votes.

6.1.5 Berserk detection protocol

Since berserk strategies are the most severe attacks, the security of the protocol can be improved if berserk nodes can be identified. We, therefore, propose a detection mechanism that is based on the record of nodes' responses. In the proposed berserk detection protocol nodes exchange information about the opinions received in the previous rounds. A subsequent analysis of this information can then reveal the berserk behavior. Upon discovering malicious voting patterns, nodes would gossip the proofs, such that all of the other honest network participants drop the berserk node.

The Protocol. We allow that a node can ask a queried node for a list of opinions received during the previous round of FPC voting. We call such a list v-list and we may request for it in several ways. For example, the full response message to the request of a v-list and the opinions could be comprised of the opinion in the current round and the received opinions from the previous round. We do not require nodes to apply this procedure for every member of the quorum or every round. For instance, each node could request it with a certain probability or if it has the necessary bandwidth capacity available. Furthermore, we can set an upper bound on this probability on the protocol level so that spamming of requests for v-lists can be detected. We denote this probability that an arbitrary query request includes a request for a v-list by p.

A more formal understanding of the approach is the following: assume that in the last round a node y received k votes, submitted by nodes $z_1, ..., z_k$. If a node x asks y for a v-list, then y sends votes submitted by $z_1, ..., z_k$ along with the identities of $z_1, ..., z_k$ but without their signatures. This reduces the message size. Node x compares the opinions in the v-list submitted by y with other received v-lists. If x detects any trace of a suspicious behavior it will ask the node y to send it the associated signatures that would prove the malicious behaviour. Having collected the proof the honest node gossips the evidence to the network and the adversary node will be dropped by all honest nodes. Since a single evidence for berserk behaviour is sufficient, further evidence does not yield any additional benefit. Therefore, in order to prevent spam, once a node propagated a proof for a given node to be berserk, it is not required to forward additional proofs even though the content may be different.

Expected number of rounds before detection. To test how reliable this detection method is and what the communication overhead would be, we carry out the following calculations. We are interested in the probability of detecting a berserk adversary since the inverse of this value gives us an estimation of how many rounds are required to detect malicious behaviour. If the number of rounds is sufficiently small the protocol allows for fast detection of a berserk attacker.

Let us consider the following scenario: Among n nodes there is a single berserk node. In the last round, the adversarial node was queried k times, which is the expected number of received query requests (if we pick nodes with uniform probability). Furthermore, it sends kf votes with opinion 0 and $(1-f)k$ votes with the opinion 1 to different nodes in the network. Let us call the first group G_0 and the second one G_1.

The probability that an honest node x requests a v-list from a node from the group G_0 equals pfk/n and $p(1-f)k/n$ for G_1. Note that the events of receiving v-lists from G_0 and G_1 are not independent. The probability that x receives v-lists that allow for the detection of the berserk node equals

$$P(x \text{ receives opinion from } G_0 \text{ and } G_1) = \frac{p^2(1-f)fk^3(k-1)}{n^2} + \mathcal{O}\Big(\frac{k^6p^3}{n^3}\Big).$$

Then using the approximation $(1-\varepsilon)^a = 1 - a\varepsilon + \mathcal{O}(\varepsilon^2)$ the probability that some node detects the berserk behaviour equals

$$P(\text{Some node detects malicious node}) = \frac{p^2(1-f)fk^3(k-1)}{n} + \mathcal{O}\Big(\frac{k^6p^3}{n^2}\Big).$$

For example, in a system with $n = 10000$, $k = 30, p = 0.1$ and $f = 1-f = 0.5$ the detection probability is ≈ 0.2. Assuming that the full FPC voting for a conflict takes about 15 rounds, berserk nodes can be detected within one FPC voting cycle with high probability.

Improvements: opinion history comparison. In the analysis presented above, nodes compare opinions from a single, previous round. However, we can increase the efficiency of the protocol when the nodes compare opinions from more than just the last round, i.e. nodes may compare entire histories of opinions rather than only the current and last opinions. By history, we mean a list of all held opinions by a given node in consecutive rounds. For example, if a node is in the m-th round on voting on a particular conflict, its history consists of m bits and each of them corresponds to the opinion in one of the m rounds. When such a node is asked to give its current opinion it would respond with the m bits in the message.

Note that nodes could compare the histories of opinions of different sizes. For example, when a node x receives the voting history from the same node y in the m_1th and m_2th rounds ($m_1 < m_2$) it could try to find traces of the berserk behavior on the overlapping m_1 rounds. With this method, the effectiveness of the berserk detection increases substantially. An obvious drawback of this approach would be that, in order for such a protocol to work, we would require each node queried in FPC to send its entire opinion history (on a particular conflict). This opinion history would grow with each round of voting by one bit. Moreover, nodes would need to allocate some memory to keep both the history of their own opinions and the history of the opinions of other nodes. However, the additional communication cost is not very high.

6.2 Cellular consensus

The second implementation discussed in this paper is *cellular automata* (CA). The CA approach, also known as majority dynamics, has originally been developed as a formal description of the Ising model [48] and is extensively studied, see e.g., [1, 9, 42, 58, 60, 63, 76, 79]. More recently, it is also adopted in DLTs projects [55]. One of the biggest advantages of the CA-based techniques over other consensus algorithms is the opportunity to achieve a very high level of parallelism. This advantage alone is a sufficient incentive for deeper studies. Our proposed CA implementation brings the following novel key properties:

- Every node acts as a cellular automaton [22] that, in the presence of conflicts, changes its opinion only based on the state of its direct neighbors and always adopts the majority opinion.
- The set of neighbors of a node does not change during one run of the consensus algorithm. In this case, the reorganization mentioned in Section 4 must only happen for different runs, i.e., different conflicts.
- When evaluating the opinions of neighbors, nodes will require a "proof" that includes the opinions of the neighbors' neighbors. This will allow nodes to monitor each others' behavior and prevents a node from lying independently of its neighbors.
- Misbehaving neighbors, i.e., neighbors that hold an opinion that is inconsistent with this proof, will be dropped immediately. This information is then also broadcasted to the network for other nodes to verify and mark that corresponding node as malicious and prevent future connection attempts.

At the beginning of each round, every node sends a "heartbeat" of its current status. This includes its signed current opinion, as well as the opinions of each of its neighbors from the previous round, each signed by the issuing node. Since the previous opinions of the neighbors cannot be faked, every node receiving this heartbeat can validate that the current opinion is indeed correct and follows the rules of the consensus mechanism.

We formalize the above ideas in the consensus protocol described in the next subsection.

Algorithm 5: Send heartbeat

```
Function heartbeat(Node i, Round m):
    foreach neighbor $j \in N_i$ do
        send opinion $X_m(i)$ to neighbor $j$
        foreach $j' \in N_i \setminus \{j\}$ do
            send opinion $X_{m-1}(j')$ to neighbor $j$
```

Model Suppose that there is a network composed of n nodes, and these nodes need to come to a consensus on the value of a bit. For clarity, we assume in the following that each node is directly connected to k neighbors, through the autopeering mechanism described in Section 4. The set of neighbors of node i is denoted by N_i. The autopeering mechanism is a key factor for the security of this approach, since a malicious node should not be able to select or influence its neighbors.

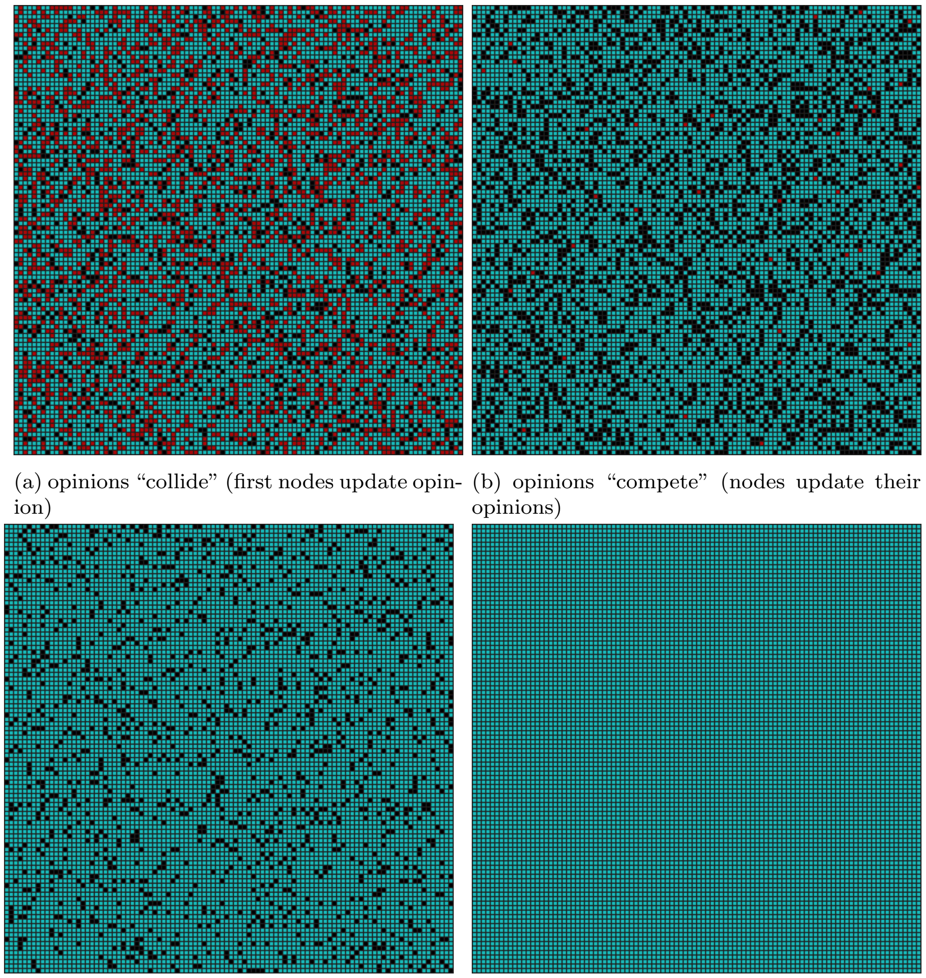
During each stage of the algorithm, each node holds an opinion on the value of the bit. The opinion can be either 0, 1 or $\perp$, depending on whether the node prefers 0, 1 or none at all. The opinion of node i in round m is denoted by $X_m(i) \in \{0, 1, \perp\}$. We further assume, that each node i has the initial opinion $X_0(i) \in \{0, 1\}$.

The protocol depends on the following parameters:

- $k \in \mathbb{N}$, number of (initial) neighbors of each node.
- $M \in \mathbb{N}$, maximum number of rounds.
- $\ell \in \mathbb{N}$, the number of consecutive rounds with the same opinion after which it becomes final.
- $p : \{0, \dots, k\} \to \mathbb{R}_{\geq 0}$, monotonically increasing weight function that maps the number of neighbors to a weight. This penalizes nodes having fewer than k neighbors.

Algorithm Each node i knows the opinions of its neighbors $j \in N_i$ as well as the opinions of all their neighbors N_j. This is assured by a broadcasting step where all the opinions are signed in such a way that the originating nodes as well as broadcasting node are unforgeable. This is formalized in Algorithm 5.

The consensus mechanism is a CA where a node uses the opinions of its neighbors to update its own state. When the majority of neighbors support either 0 or 1, the node adopts that opinion. If none of these opinions has a majority, it adopts $\perp$, i.e., none of them. As long as we assume that the set N_i is known at least for all of its neighbors i, any node can use these simple rules to validate whether the reported opinion of neighbor i is consistent with the opinions of all nodes in N_i. The overall consensus mechanism is more formally illustrated in Algorithm 6 and an illustration of the CA process for an example scenario can be found in Fig. 7.



(a) opinions “collide” (first nodes update opinion)

(b) opinions “compete” (nodes update their opinions)

(c) opinions “compete” (nodes update their opinions)

(d) a single opinion “survives” (network reaches consensus)

Fig. 7: Visualization of the CA consensus process: Each square corresponds to a node connected to random neighbors. Initially, the two conflicting transactions are propagated through the network. Then, nodes are consistently adapting their opinions (0: red, 1: cyan, $\perp$: black) before eventually coming to consensus.

Algorithm 6: Cellular consensus

```
foreach node $i$ do
    Send initial opinion $X_0(i)$ to neighbors $N_i$

for $m \leftarrow 1$ to $M$ do
    foreach node $i$ do
        foreach neighbor $j \in N_i$ do
            if $X_{m-1}(j)$ is inconsistent wrt $X_{m'}(j')$ for
             $j' \in N_j, m' < m - 1$ then
                // drop neighbor $j$
                $N_i \leftarrow N_i \setminus \{j\}$
        if node $i$ finalized then
            $X_m(i) \leftarrow X_{m-1}(i)$
        else
            // adopt majority opinion
            $total \leftarrow \sum_{\{j \in N_i\}} p(|N_j|)$
            if $\sum_{\{j \in N_i | X_{m-1}(j)=0\}} p(|N_j|) > \frac{total}{2}$ then
                $X_m(i) \leftarrow 0$
            else if $\sum_{\{j \in N_i | X_{m-1}(j)=1\}} p(|N_j|) > \frac{total}{2}$ then
                $X_m(i) \leftarrow 1$
            else
                $X_m(i) \leftarrow \perp$                          // cancel all

        heartbeat($i$, $m$)

        if opinion $X(i)$ did not change in the last $\ell$ rounds then
            mark node $i$ finalized
```

7 Tip selection

7.1 Importance and Motivation

The Tangle is a data structure built in accordance with the following rule:

> *In order to join the Tangle, a transaction has to validate two existing transactions.*

The validation of a transaction is a procedure that verifies whether an address owns the tokens spent[29]. If transaction y validates transaction x, we say that y *directly* approves x. Furthermore, if there is not a directed edge between the transactions x and y, but there exists a directed path between them, then we say that y *indirectly* approves x.

[29]The actual validation process in IOTA is more complex, and we invite the interested reader to visit https://docs.iota.org for more information.

Although the Tip Selection is not part of the conflict resolution in our current approach to the Coordicide, it still plays many important roles in the Tangle. It keeps the Tangle growing in an scalable and decentralized way. This is facilitated by the users being the ones that validate other transactions, such that the flow of new transactions keeps continuously cementing older ones. Furthermore, this happens in a way that, as more users are using the network, the more secure and fast finality can be achieved, since the total number of approvals of a transactions increases much faster. We also want to emphasize that the influence of the Tip Selection on properties of the Tangle has been studied recently, e.g., [26, 37, 54, 69].

It is interesting to notice how the tip selection is one of the main ingredients that makes the Tangle a network of *continuous consensus*. This is in the sense that "young" transactions have less transactions validating them and hence are prone to induce a small divergence of the ledger state among the nodes. As more and more new transactions approve a transaction, the more the nodes become aware of this transaction, increasing the level of consensus over time. For a network using probabilistic consensus such as with the Tangle, this is especially important since a higher quantity of approvals translates in an ever smaller probability of different views among the nodes, and this chance can be made as small as we need it to be.

The original IOTA white paper originally uses only a TSA based on a biased random walk to determine the tips to approve. This TSA comes with its own limitations, such as its computational complexity and that it needs to orphan branches of possibly honest transactions to resolve conflicts. With the new consensus mechanism being independent from the TSA, a much faster algorithm may be used to select tips and incentivise good behavior for the network.

In the rest of this section, we provide an overview of our current candidate for the default TSA. It is important to stress that, since the TSA is not (and cannot be easily[30]) enforced, the choice of the particular TSA is, *ultimately*, up to the node's owner. Therefore, the actors will choose their TSAs in a *reasonable* way, as argued in [67]. Because of this intrinsic freedom of choice, and also because the "space" of all possible TSAs is enormous, it is crucial to have all reasonable options on the table. We are absolutely not obliged to be ever content with a particular version of TSA; instead, our vision is that, similarly to the human society itself, the system will continue evolving. Hence, we believe our current candidate to be an appreciated upgrade over the random walk based TSA and we expect the future input we receive to help us to improve the algorithm even more.

7.2 (Almost) URTS on a subset

In this subsection[31] we aim to describe a tip selection rule which is a reasonable modification of the URTS (i.e., Uniformly Random Tip Selection) algorithm

[30]It is, however, still possible to perform a statistical analysis on the node's choices in order to try figuring out what sort of TSA it is using.

[31]see also https://iota.cafe/t/almost-urts-on-a-subset/234

considered in the original whitepaper [66].

Selecting tips uniformly at random has advantages: every (non-lazy) tip gets eventually picked up with probability 1; also, at least in the situation when all nodes seek to select tips, the URTS is a Nash Equilibrium (knowing that other nodes select tips with equal probabilities, a given node has nothing to gain from deviating from this behavior). Unfortunately, the URTS has the issue of not demotivating lazy-behavior. One possibility for solving this would be to only consider URTS on the subset on non-lazy tips, but this is too strict in the sense that if a lazy connection happens by any unexpected situation, the node would need to make an reattachment of the transaction. In this proposal we aim to improve the URTS in a way that it will demotivate such unwanted behaviors, while still keeping the possibility of promoting lazy transactions a viable option (therefore, reducing the number of reattachments).

The proposal for this new TSA is that, instead of choosing uniformly a "good" tip, we check all non-lazy approvals from the subset of good tips, and select one of those approvals uniformly at random. The selected tip for the TSA will be the tip from where the selected approval was originated. Observe that it is natural why this algorithm demotivates lazy behavior (since a lazy connection cannot be selected, a lazy tip has at most half the probability of a non-lazy tip), while keeping the possibility of lazy transactions to be promoted and approved if their issuer is willing to.

Now, let us describe the algorithm, starting by defining what we consider a "good" tip. Consider a specific node j. Then for any transaction v, let us denote by

- $t(v)$ the objective (signed) timestamp of the transaction;
- $r_j(v)$ the time of solidification of the transaction for this node (i.e., the time when a transaction and all of its history is received by the node j).

Fix positive constants C_1, C_1', C_2 (which are not very large) and M (which is large). Suppose that the node j considers v a tip, and v approves v_1 and v_2. Then, we define the *weight* $w_j(v)$ of the transaction v for the node j to be null (hence, being a "bad" tip) in any of the following scenarios:

- if
$$t(v) \notin [r_j(v) - C_1, r_j(v) + C_1']$$
(i.e., the transaction's timestamp is either too much to the past or too much to the future);
- if
$$\max_{i=1,2}(r_j(v) - t(v_i)) > M,$$
(this is in order to avoid approving transactions that reference something "too old").

Otherwise (if the weight $w_j(v)$ was not set to 0 according to the above rules) define

$$k_j(v) := \#\{i = 1, 2 : r_j(v) - r_j(v_i) \leq C_2\}$$

to be the number of "nonlazy approvals" that v did. Then, it would be natural for the jth node to set $w_j(v) := k_j(v)$. For the tip selection, the node chooses v with a probability proportional to $w_j(v)$.

We may consider further modifications of this method, which are meant to make it more resilient, for example against deliberate attachments to non-tips that are not very old. For two transactions u, v where v approves u, we introduce the sibling number $s_j(v, u) \in \mathbb{N}$ (again, with respect to the node j) in the following way. If $v_0, \ldots, v_k$ approved u directly and the node j received them in that consecutive order, then $s_j(v_i, u) = i$.

Let $f : \mathbb{N} \to [0, 1]$ be a nonincreasing function with $f(1) = 1$ (for example, $f(m) = \alpha^{m-1}$ for some $\alpha \in (0, 1]$). Then, set

$$\begin{aligned} w_j(v) \quad = \quad & f(s_j(v, v_1))\mathbf{1}\{r_j(v) - r_j(v_1) \leq C_2\} \\ & + f(s_j(v, v_2))\mathbf{1}\{r_j(v) - r_j(v_2) \leq C_2\}. \end{aligned} \tag{10}$$

Again, we then just chose v with probability proportional to $w_j(v)$ for tip selection. To explain why this should be closer to a Nash equilibrium, recall the main idea of [69]: if some "greedy" actors start favoring a smaller subset of tips, they would create a competition between themselves (see Fig. 4 in [69] which gives an idea why "completely greedy" strategies cannot be advantageous for the nodes). Therefore, they would be penalized if the weights are defined as in (10). This tip selection should also be more resilient against the aforementioned attack (when an actor attaches transactions to non-tips, the sibling number of such transactions typically increases).

8 Conclusion

In this paper, we outline our approach for the Coordicide project. In particular, we describe our main ideas around the consensus mechanism, tip selection, security and protection against attacks, spam control and autopeering; all of which provide the building blocks that are crucial for the Coordicide project. Our proposal towards the path to Coordicide is now well-defined, and we are currently implementing and evaluating various options in our prototype code *GoShimmer*.

Acknowledgements

We thank Jon Crowcroft, Michael Huth, Emil Matejka, Robert Shorten, Jonas Theis, Roman Vitenberg, and Olaf van Wijk for comments and suggestions on the previous version of this paper.

References

[1] Mohammed Amin Abdullah and Moez Draief. Global majority consensus by local majority polling on graphs of a given degree sequence. *Discrete Applied Mathematics*, 180:1 – 10, 2015.

[2] Marcos K. Aguilera and Sam Toueg. The Correctness Proof of Ben-Or's Randomized Consensus Algorithm. *Distributed Computing*, pages 371–381, 2012.

[3] Vidal Attias and Quentin Bramas. How to choose its parents in the tangle. In Mohamed Faouzi Atig and Alexander A. Schwarzmann, editors, *Networked Systems*, pages 275–280, Cham, 2019. Springer International Publishing.

[4] Vidal Attias, Luigi Vigneri, and Vassil Dimitrov. On the decentralized generation of thersa moduli in multi-party settings, 2019.

[5] Sven Banisch, Tanya Araújo, and Jorge Louçã. Opinion dynamics and communication networks. *Advances in Complex Systems*, pages 95–111, 2010.

[6] P. C. Bartolomeu, E. Vieira, and J. Ferreira. IOTA Feasibility and Perspectives for Enabling Vehicular Applications. In *2018 IEEE Globecom Workshops (GC Wkshps)*, pages 1–7, Dec 2018.

[7] Luca Becchetti, Andrea Clementi, Emanuele Natale, Francesco Pasquale, and Luca Trevisan. Stabilizing Consensus with Many Opinions. In *Symposium on Discrete algorithms*, pages 620–635. SIAM, 2016.

[8] Michael Ben-Or. Another Advantage of Free Choice (Extended Abstract): Completely Asynchronous Agreement Protocols. In *ACM Symposium on Principles of Distributed Computing*, pages 27–30, 1983.

[9] Itai Benjamini, Siu-On Chan, Ryan O'Donnell, Omer Tamuz, and Li-Yang Tan. Convergence, unanimity and disagreement in majority dynamics on unimodular graphs and random graphs. *Stochastic Processes and their Applications*, 126(9):2719 – 2733, 2016.

[10] Dan Boneh, Joseph Bonneau, Benedikt Bünz, and Ben Fisch. Verifiable Delay Functions. In *Annual International Cryptology Conference*, pages 757–788. Springer, 2018.

[11] Dan Boneh and Matthew Franklin. Efficient generation of shared RSA Keys. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 1294(4):425–439, jul 1997.

[12] Allan Borodin and Ian Munro. *The Computational Complexity of Algebraic and Numeric Problems*. North-Holland, 1975.

[13] Gabriel Bracha. Asynchronous Byzantine Agreement Protocols. *Information and Computation*, pages 130 – 143, 1987.

[14] Quentin Bramas. The Stability and the Security of the Tangle. In Vincent Danos, Maurice Herlihy, Maria Potop-Butucaru, Julien Prat, and Sara Tucci-Piergiovanni, editors, *International Conference on Blockchain Economics, Security and Protocols (Tokenomics 2019)*, volume 71 of *OpenAccess Series in Informatics (OASIcs)*, pages 8:1–8:15, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[15] James Brogan, Immanuel Baskaran, and Navin Ramachandran. Authenticating health activity data using distributed ledger technologies. *Computational and Structural Biotechnology Journal*, 16:257 – 266, 2018.

[16] Angelo Capossele, Sebastian Mueller, and Andreas Penzkofer. Robustness and efficiency of leaderless probabilistic consensus protocols within byzantine infrastructures, 2019. In arXiv, eprint 1911.08787.

[17] Ignacio Cascudo and Bernardo David. SCRAPE: Scalable Randomness Attested by Public Entities. In *International Conference on Applied Cryptography and Network Security*, pages 537–556. Springer, 2017.

[18] Claudio Castellano, Santo Fortunato, and Vittorio Loreto. Statistical physics of social dynamics. *Reviews of Modern Physics*, page 591, 2009.

[19] Miguel Castro and Barbara Liskov. Practical Byzantine Fault Tolerance and Proactive Recovery. *ACM Transactions on Computer Systems*, pages 398–461, 2002.

[20] T. F. Chiang, S. Y. Chen, and C. F. Lai. A Tangle-Based High Performance Architecture for Large Scale IoT Solutions. In *2018 1st International Cognitive Cities Conference (IC3)*, pages 12–15, Aug 2018.

[21] Dah-Ming Chiu and Raj Jain. Analysis of the increase and decrease algorithms for congestion avoidance in computer networks. *Comput. Netw. ISDN Syst.*, 17(1):1–14, June 1989.

[22] Edgar F. Codd. *Cellular Automata*. Academic Press, 1968.

[23] Colin Cooper, Robert Elsässer, and Tomasz Radzik. The Power of Two Choices in Distributed Voting. In *International Colloquium on Automata, Languages, and Programming*, pages 435–446. Springer, 2014.

[24] Colin Cooper, Robert Elsässer, Tomasz Radzik, Nicolas Rivera, and Takeharu Shiraga. Fast Consensus for Voting on General Expander Graphs. In *International Symposium on Distributed Computing*, pages 248–262. Springer, 2015.

[25] M. Corless, C. King, R. Shorten, and F. Wirth. *AIMD Dynamics and Distributed Resource Allocation*. Advances in Design and Control. Society for Industrial and Applied Mathematics, 2016.

[26] Andrew Cullen, Pietro Ferraro, Christopher King, and Robert Shorten. Distributed Ledger Technology for IoT: Parasite Chain Attacks, 2019. In arXiv, eprint 1904.00996.

[27] Ivan Damgård and Gert Læssøe Mikkelsen. Efficient, robust and constant-round distributed RSA key generation. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 5978 LNCS:183–200, 2010.

[28] Kundan Dasgupta and M. Rajasekhara Babu. *A Review on Crypto-Currency Transactions Using IOTA (Technology)*, pages 67–81. Springer Singapore, Singapore, 2019.

[29] Luca De Feo, Simon Masson, Christophe Petit, and Antonio Sanso. Verifiable Delay Functions from Supersingular Isogenies and Pairings. Cryptology ePrint Archive, Report 2019/166, 2019., 2019.

[30] G. De Roode, I. Ullah, and P. J. M. Havinga. How to Break IOTA Heart by Replaying? In *2018 IEEE Globecom Workshops (GC Wkshps)*, pages 1–7, Dec 2018.

[31] Vassil Dimitrov, Luigi Vigneri, and Vidal Attias. Smooth operator – the use of smooth integers in fast generation of rsa keys, 2019.

[32] John R. Douceur. The Sybil Attack. In *International Workshop on Peer-to-peer Systems*, pages 251–260. Springer, 2002.

[33] Cynthia Dwork and Moni Naor. Pricing via Processing or Combatting Junk Mail. In *Advances in Cryptology*, pages 139–147, 1993.

[34] Robert Elsässer, Tom Friedetzky, Dominik Kaaser, Frederik Mallmann-Trenn, and Horst Trinker. Rapid Asynchronous Plurality Consensus. *arXiv preprint arXiv:1602.04667*, 2016.

[35] Giulia Fanti, Nina Holden, Yuval Peres, and Gireeja Ranade. Communication Cost of Consensus for Nodes with Limited Memory. *arXiv preprint arXiv:1901.01665*, 2019.

[36] Paul Feldman and Silvio Micali. An Optimal Probabilistic Algorithm for Synchronous Byzantine Agreement. In *Automata, Languages and Programming*, pages 341–378. Springer, 1989.

[37] Pietro Ferraro, Christopher King, and Robert Shorten. IOTA-based Directed Acyclic Graphs without Orphans, 2018. In arXiv, eprint 901.07302.

[38] B. C. Florea. Blockchain and internet of things data provider for smart applications. In *2018 7th Mediterranean Conference on Embedded Computing (MECO)*, pages 1–4, June 2018.

[39] Bryan Ford and Rainer Böhme. Rationality is Self-Defeating in Permissionless Systems, 2019.

[40] Tore Kasper Frederiksen, Yehuda Lindell, Valery Osheter, and Benny Pinkas. Fast Distributed RSA Key Generation for Semi-Honest and Malicious Adversaries (Full Version).

[41] Roy Friedman, Achour Mostéfaoui, and Michel Raynal. Simple and Efficient Oracle-Based Consensus Protocols for Asynchronous Byzantine Systems. *IEEE International Symposium on Reliable Distributed Systems*, pages 228–237, 2004.

[42] Bernd Gärtner and Ahad N. Zehmakan. Majority model on random regular graphs. *Lecture Notes in Computer Science*, pages 572–583, 2018.

[43] Daniel Germanus, Stefanie Roos, Thorsten Strufe, and Neeraj Suri. Mitigating eclipse attacks in peer-to-peer networks. In *2014 IEEE Conference on Communications and Network Security*, pages 400–408. IEEE, 2014.

[44] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In *Proceedings of the 26th Symposium on Operating Systems Principles*, pages 51–68. ACM, 2017.

[45] Carmit Hazay, Gert Læssøe Mikkelsen, Tal Rabin, Tomas Toft, and Angelo Agatino Nicolosi. Efficient RSA Key Generation and Threshold Paillier in the Two-Party Setting. Technical Report 2, 2019.

[46] Ethan Heilman, Alison Kendler, Aviv Zohar, and Sharon Goldberg. Eclipse attacks on bitcoin's peer-to-peer network. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 129–144, Washington, D.C., August 2015. USENIX Association.

[47] Peter Ince, Joseph K. Liu, and Peng Zhang. Adding confidential transactions to cryptocurrency iota with bulletproofs. In Man Ho Au, Siu Ming Yiu, Jin Li, Xiapu Luo, Cong Wang, Aniello Castiglione, and Kamil Kluczniak, editors, *Network and System Security*, pages 32–45, Cham, 2018. Springer International Publishing.

[48] Ernst Ising. Beitrag zur Theorie des Ferromagnetismus. *Zeitschrift für Physik A Hadrons and Nuclei*, pages 253–258, 1925.

[49] Hatem Ismail, Daniel Germanus, and Neeraj Suri. Detecting and mitigating p2p eclipse attacks. In *2015 IEEE 21st International Conference on Parallel and Distributed Systems (ICPADS)*, pages 224–231. IEEE, 2015.

[50] V. Jacobson. Congestion avoidance and control. In *Symposium Proceedings on Communications Architectures and Protocols*, SIGCOMM '88, pages 314–329, New York, NY, USA, 1988. ACM.

[51] Tomáš Janečko and Ivan Zelinka. Impact of Security Aspects at the IOTA Protocol. In Ajith Abraham, Sergey Kovalev, Valery Tarassov, Vaclav

Snasel, and Andrey Sukhanov, editors, *Proceedings of the Third International Scientific Conference "Intelligent Information Technologies for Industry" (IITI'18)*, pages 41–48, Cham, 2019. Springer International Publishing.

[52] Frank P. Kelly, Akhil K. Maulloo, and David K. H. Tan. Rate Control for Communication Networks: Shadow Prices, Proportional Fairness and Stability. *Journal of the Operational Research Society*, pages 237–252, 1998.

[53] John Kelsey, Luís T. A. N. Brandão, Rene Peralta, and Harold Booth. A reference for randomness beacons: Format and protocol version 2. Technical report, National Institute of Standards and Technology, 2019.

[54] Bartosz Kusmierz and Alon Gal. Probability of being left behind and probability of becoming permanent tip in the Tangle v0.2, 2018. https://assets.ctfassets.net/r1dr6vzfxhev/6FMwUHOb4WIyi6mm8oWWgY/8f1d7b30f7b652098a5e68b6634c63df/POLB-02.pdf.

[55] NKN Lab. NKN: a Scalable Self-Evolving and Self-Incentivized Decentralized Network, 2018. https://www.nkn.org/doc/NKN_Whitepaper.pdf.

[56] Leslie Lamport. Password authentication with insecure communication. *Communications of the ACM*, 24(11):770–772, 1981.

[57] Frederik Mallmann-Trenn. *Probabilistic Analysis of Distributed Processes with Focus on Consensus*. PhD thesis, PSL Research University, 2017.

[58] Manuel Marques-Pita and Luis Mateus Rocha. Conceptual structure in cellular automata - the density classification task. In *ALIFE*, 2008.

[59] Abdul Wahab Memon, Mahad Barlas, and Waqas Mahmood. Zenith certifier: A framework to authenticate academic verifications using tangle. 2018, 05 2018.

[60] Elchanan Mossel, Joe Neeman, and Omer Tamuz. Majority dynamics and aggregation of information in social networks. *Autonomous Agents and Multi-Agent Systems*, 28(3):408–429, Jun 2013.

[61] Till Neudecker and Hannes Hartenstein. Network layer aspects of permissionless blockchains. *IEEE Communications Surveys & Tutorials*, 21(1):838–857, 2018.

[62] Hong-Li Niu and Jun Wang. Entropy and recurrence measures of a financial dynamic system by an interacting voter system. *Entropy*, pages 2590–2605, 2015.

[63] Massimo Ostilli, Eiko Yoneki, Ian X.Y. Leung, Jose F.F. Mendes, Pietro Lió, and Jon Crowcroft. Statistical mechanics of rumour spreading in network communities. *Procedia Computer Science*, 1(1):2331 – 2339, 2010. ICCS 2010.

[64] Krzysztof Pietrzak. Simple Verifiable Delay Functions. In *Innovations in Theoretical Computer Science Conference*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

[65] Serguei Popov. On a Decentralized Trustless Pseudo-Random Number Generation Algorithm. *Journal of Mathematical Cryptology*, pages 37–43, 2017.

[66] Serguei Popov. The Tangle, 2018.

[67] Serguei Popov. IOTA: Feeless and Free. *IEEE Blockchain Technical Briefs*, 2019.

[68] Serguei Popov and William J. Buchanan. FPC-BI: Fast probabilistic consensus within byzantine infrastructures. *CoRR*, abs/1905.10895, 2019.

[69] Serguei Popov, Olivia Saa, and Paulo Finardi. Equilibria in the Tangle. *Computers & Industrial Engineering*, 136:160–172, 2019.

[70] Piotr Przybyła, Katarzyna Sznajd-Weron, and Maciej Tabiszewski. Exit probability in a one-dimensional nonlinear q-voter model. *Phys. Rev. E*, 2011.

[71] Michael O. Rabin. Randomized Byzantine Generals. In *Annual Symposium on Foundations of Computer Science*, pages 403–409. IEEE, 1983.

[72] Alexander Raschendorfer, Benjamin Mörzinger, Eric Steinberger, Patrick Pelzmann, Ralf Oswald, Manuel Stadler, and Friedrich Bleicher. On IOTA as a potential enabler for an M2M economy in manufacturing. *Procedia CIRP*, 79:379 – 384, 2019. 12th CIRP Conference on Intelligent Computation in Manufacturing Engineering, 18-20 July 2018, Gulf of Naples, Italy.

[73] Philipp Schindler, Nicholas Stifter, Aljosha Judmayer, and Edgar Weippl. HydRand: Practical Continuous Distributed Randomness. Cryptology ePrint Archive, Report 2018/319, 2018.

[74] Diego Stucchi, Ruggero Susella, Pasqualina Fragneto, and Beatrice Rossi. Secure and effective implementation of an iota light node using stm32. In *Proceedings of the 2nd Workshop on Blockchain-Enabled Networked Sensor*, BlockSys'19, page 28–29, New York, NY, USA, 2019. Association for Computing Machinery.

[75] Ewa Syta, Philipp Jovanovic, Eleftherios Kokoris Kogias, Nicolas Gailly, Linus Gasser, Ismail Khoffi, Michael J. Fischer, and Bryan Ford. Scalable Bias-Resistant Distributed Randomness. In *IEEE Symposium on Security and Privacy*, pages 444–460, 2017.

[76] Mieko Tanaka-Yamawaki, Sachiko Kitamikado, and Toshio Fukuda. Consensus formation and the cellular automata. *Robotics and Autonomous Systems*, 19(1):15 – 22, 1996. Evolutional Robots.

[77] Liang Wang, Ben Catterall, and Richard Mortier. Probabilistic Synchronous Parallel, 2017.

[78] Benjamin Wesolowski. Efficient verifiable delay functions. Cryptology ePrint Archive, Report 2018/623, 2018. https://eprint.iacr.org/2018/623.

[79] Dietmar Wolz and Pedro P. B. de Oliveira. Very effective evolutionary techniques for searching cellular automata rule spaces. *J. Cellular Automata*, 3:289–312, 2008.

[80] Zhiyi Zhang, Vishrant Vasavada, Randy King, and Lixia Zhang. Proof-of-authentication for private distributed ledger. 2019.