TU WIEN Informatics

# On the Security of Proof-of-Stake Directed Acyclic Graph Protocols

## A Simulation-Based Approach

DIPLOMARBEIT

zur Erlangung des akademischen Grades

**Diplom-Ingenieur**

im Rahmen des Studiums

**Software Engineering/Internet Computing**

eingereicht von

**Bernhard Schachenhofer, BSc**
Matrikelnummer 01529059

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Univ.-Prof. Dipl.-Ing. Mag. Dr. techn. Edgar Weippl
Mitwirkung: Dipl.-Ing. Philipp Schindler
Dipl.-Ing. Aljosha Judmayer

Wien, 21. Jänner 2021

Bernhard Schachenhofer | Edgar Weippl

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

TU WIEN Informatics

# On the Security of Proof-of-Stake Directed Acyclic Graph Protocols

## A Simulation-Based Approach

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

**Diplom-Ingenieur**

in

**Software Engineering/Internet Computing**

by

**Bernhard Schachenhofer, BSc**
Registration Number 01529059

to the Faculty of Informatics

at the TU Wien

Advisor: Univ.-Prof. Dipl.-Ing. Mag. Dr. techn. Edgar Weippl
Assistance: Dipl.-Ing. Philipp Schindler
Dipl.-Ing. Aljosha Judmayer

Vienna, 21st January, 2021

| Bernhard Schachenhofer | Edgar Weippl |
|---|---|

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

# Erklärung zur Verfassung der Arbeit

Bernhard Schachenhofer, BSc

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 21. Jänner 2021

Bernhard Schachenhofer

# Acknowledgements

I want to say thank you to my parents. You enabled me to study at TU Vienna and supported me over all the years.

Another thank you goes to my co-advisors at SBA Research, who provided feedback and guidance to me. Aljosha, thank you for your valuable input in our discussions and for showing me a variety of different perspectives on various challenges along the way. Philipp, thank you for helping me out with all the question I came to you with and thank you for your outstanding assistance throughout the whole thesis.

# Kurzfassung

Mit Bitcoin wurde 2008 die erste Kryptowährung veröffentlicht. Mit ihr können Zahlungen getätigt werden, ohne einer zentralen Stelle wie einer Bank vertrauen zu müssen. Dies wird durch einen innovativen Konsensus-Mechanismus ermöglicht, der unter der Bezeichnung *Proof-of-Work* bekannt ist. Bitcoin ist noch immer die bekannteste und wertvollste Kryptowährung. Über die Jahre sind jedoch einige Probleme offenkundig geworden, wie z.B. der hohe Energieverbrauch und der niedriger Transaktionsdurchsatz. Um das Jahr 2016 kam eine neue Art von Protokollen auf, welche versprach diese beiden Probleme gleichzeitig zu lösen. Diese Protokolle basieren auf dem *Proof-of-Stake (PoS)* Mechanismus und verwenden *gerichtete azyklische Graphen* als Datenstruktur. Da es sich hier ebenfalls um Währungen handelt, spielt Sicherheit eine zentrale Rolle. Dies wirft die Frage auf, wie sicher Protokolle dieser Art wirklich sind.

Diese Arbeit trägt zur Beantwortung bei, indem Hashgraph im Detail analysiert wird. Hashgraph ist ein vielversprechender Vertreter dieser Protokollart. Die verschiedensten Sicherheits- und Performanceangaben des Protokolls werden mithilfen eines im Zuge der Arbeit entwickelten und veröffentlichten Simulators überprüft. Der Simulator ermöglicht es, das Verhalten des Protokolls unter vier verschiedenen (Angriffs-)Szenarien zu untersuchen. Er bietet dazu umfangreiche Konfigurationsmöglichkeiten an, welche unzählige verschiedene Verläufe ermöglichen. Akteure/Akteurinnen agieren zufällig basierend auf einem veränderbaren Parameter, der Reproduzierbarkeit gewährleistet. Der Simulator selbst verfügt außerdem über eine grafische Benutzerobefläche, wobei Ergebnisse auch als Text-Dateien für weitere Analysen exportiert werden können.

In keiner einzigen von tausenden Simulationen wurde der Konsensus-Mechanismus von Hashgraph gebrochen. Basierend auf der Tatsache das Nachrichten zur Synchronsiation beliebig schnell und an beliebige andere Knoten gesendet werden können, kamen jedoch Schwachstellen zu Tage. Diese Arbeit zeigt auf, dass es möglich ist eine widersprüchliche Transaktion zu einer bereits existierenden zu veröffentlichen und diese schneller bestätigt zu bekommen. Weiters wird belohnt, wer Synchronisationen zuerst mit ganz bestimmten Knoten durchführt, was zu einer Überlastung dieser Knoten führen kann. Die präsentierten Ergebnisse zeigen, dass diese Protokollklasse tatsächlich das Potential besitzt, zwei der grundelegenden Probleme von traditionellen Kryptowährungen zu lösen. Die Möglichkeit von Nachrichten-Spam im System kann dies jedoch zunichte machen und ist ein wichtiger Punkt in der Bewertung solcher Protokolle.

# Abstract

Bitcoin, the first cryptocurrency was published in 2008 and featured a new way of reaching consensus amongst a set of participants without the need to trust any central authority. It achieved this by introducing a novel consensus mechanism based on *Proof-of-Work*. Bitcoin is still the most valuable and famous cryptocurrency, but some limitations became more and more noticeable over the years – two of the major ones being high energy consumption and low transaction throughput. Emerging around 2016, a new class of protocols based on *Proof-of-Stake (PoS)* and using a *Directed Acyclic Graph (DAG)* as data structure, promises to solve both of these shortcomings at once. As security is crucial, carefully looking at the security properties of any new protocol family before adopting it, is necessary. This leads us to the following question: How secure are these protocols really?

This thesis contributes to answering this question by reviewing existing designs in this field and analysing Hashgraph, a promising protocol which combines a PoS mechanism with a DAG structure, in detail. We developed and published a simulator which makes it possible to look closely at the security and performance claims of the protocol under different attack scenarios. It supports simulating adversarial behaviours which we call fork, race and split attacks as well as an honest scenario for comparison. For all of them, numerous configuration options and scenario specific settings are offered. All simulations are executed in a reproducible way based on a configurable seed value. Results can be examined either in a graphical user interface or exported for further analysis.

Hashgraph proved resilient against all attempts of breaking the protocol's security over thousands of simulation runs, featuring all supported attack scenarios. Nevertheless, some weaknesses became apparent in regard to the protocol allowing everybody to perform syncs as fast as possible. Publishing a conflicting transaction to an already existing honest one and getting it finalized earlier could be shown. Additionally, the protocol encourages flooding high stake participants with syncs, leading to the possible effects of unintentional *Distributed Denial of Service* attacks. Our findings show that PoS DAG protocols have the potential to solve two of the major limitations of traditional blockchains, but before adopting them, one needs to specifically check for mechanisms to prevent message spamming or a lack thereof.

# Contents

CHAPTER 1

# Introduction

In 2008 a decentralized peer-to-peer currency system called Bitcoin [Nak08] was introduced. It laid out how a distributed consensus can be reached by a varying amount of participants without any central authority and without the need to trust any single participant. This was possible through the introduction of a revolutionary consensus algorithm which relies on the concept of *Proof-of-Work (PoW)*.

In a nutshell, transactions are grouped into blocks and each block references its predecessor, forming a blockchain. Miners extend this chain by publishing new blocks whose creation requires solving a computational intensive hash puzzle. Everybody can verify the hash puzzle result, thus a new block can be seen as a proof that work into the stability of the system was performed (PoW). If forks appear, the chain with the highest accumulated difficulty target is considered valid and provides a total order over transactions. The underlying idea is, that as long as 50 percent of the hashing power are controlled by honest nodes, one chain (the valid one) grows faster than any fork. This mechanism enables the currency to stay fully decentralized but also introduces some limitations. Two major ones are energy consumption and transaction throughput. In Bitcoin, the difficulty of the hash puzzle is adapted so that one block is published roughly every 10 minutes. Thus, adding more computing power to the network means more energy consumption while transaction throughput remains constant. The exact energy consumption can only be estimated, but a minimal value was determined by Li et al. [LLP+19] under the assumption that everybody uses the most efficient hardware available for the job. They found out that at least 23 Terawatt-Hours (TwH) were consumed to operate Bitcoin in 2018. Das and Dutta [DD20] reference an annual energy consumption of about 52 TwH for 2018. To put that into perspective, this is more than the total energy consumption of many nations worldwide. For example, Costa Rica (51,5 TwH) or Latvia (49,3 TwH)

consumed similar amounts of energy in 2018[1].

A very popular way to reduce energy consumption to a minimum is getting rid of the PoW concept as a whole. Instead of working on hash puzzles, consensus can be reached through a voting system based on stake (e.g. the amount of coins a participant owns). Many new protocols evolved around that idea, like Algorand [GHM+17] or Snow White [DPS16]. This systems are considered to be *Proof-of-Stake (PoS)* cryptocurrencies. In PoS systems, time is often divided into slots and leaders are determined in every slot who are allowed to publish a block. Then, consensus is reached on published blocks over multiple voting rounds. Voting and leader selection usually happens randomly but heavily depends on one's stake.

Through their different structure, most PoS cryptocurrencies already improve transaction throughput, the second major problem we want to consider. However, there is still a lot of room for improvement if we compare transaction throughput to fiat currency systems. For example, VISA (one of the leading payment providers) processes more than 1700 transactions per second on average[2]. In contrast, if we look at bitcoin we not only see a block creation limitation, but also limits to the block size itself. Since 2017, blocks are measured in *weight units*, effectively limiting block size to around 2-4 MB depending on the content. Also, work might be wasted on blocks which do not end on the main chain and are thus discarded. One possibility for that to happen are two blocks mined from different participants at nearly the same time as a result of the stochastic process. This all together results in 3-7 transactions per second[3].

Since about 2016, the idea of replacing the underlying chain/tree data structure with *Directed Acyclic Graphs (DAGs)* emerged, as protocols like IOTA [Pop18], Nano [LeM17] or Spectre [SLZ16] show us. Instead of referencing a single predecessor block in a chain or tree, a newly mined block usually references multiple (or all known) tips of the current graph. Then an algorithm must determine an ordering over all blocks in a decentralized manner. If two conflicting transactions occur, only the first one is considered valid. One of the underlying ideas is to allow the publication of multiple blocks at the same time, which greatly increases transaction throughput if they are all eventually incorporated into the (final) ledger.

Combining the concepts of PoS and DAGs in a cryptocurrency promises to solve two major weaknesses of current systems. However, each new mechanism, algorithm and

[1] units converted from Quad British thermal units to TwH (1 Quad Btu = 293,07 TwH), data obtained from the U.S. Energy Information Administration, accessed 13.10.2020: `https://www.eia.gov/international/overview/world`

[2] calculated using the number of 150 million processed transactions per day published on the VISA website, accessed 9.10.2020: `https://usa.visa.com/run-your-business/small-business-tools/retail.html`

[3] Bitcoin Magazine, accessed 9.10.2020: `https://bitcoinmagazine.com/what-is-bitcoin/what-is-the-bitcoin-block-size-limit`

their combination in an own protocol leads to the appearance of new attack surfaces. Additionally, there already exist many attacks on different types of blockchain systems, which might also pose a threat to any new protocol. As we are talking about a currency here, it is crucial to identify such possible weak points in these PoS DAG cryptocurrencies. And exactly that is the main motivation behind this thesis. The goal is to pick one of the most promising PoS DAG cryptocurrency and identify possible weak spots for this specific protocol. For that to work, its resilience against attacks must be probed in a structured way. Therefore, a simulator was developed which supports different attack scenarios and executes the chosen protocol in a randomized but reproducible way. Evaluating these simulations then points to possible weak spots.

The necessary steps to reach this goal are brought to paper in this thesis in the following structure: In Chapter 2 we go into detail on our research questions and the approach to answer them. Chapter 3 is about the current state of the art, where we analyze PoS DAG protocols and find answers to questions like "What are the mechanisms in traditional PoS protocols? Are there cryptocurrencies which already use DAG structures in any form and what can we learn from them? What are the most common attack surfaces, which might also be applicable to PoS DAG protocols?" by looking at related work. Chapter 4 then covers the details of how the simulator was implemented based on the chosen protocol. It also gives an overview over the supported scenarios and settings. Chapter 5 is all about evaluating and discussing the results of our simulation runs. The thesis ends with a conclusion and an outlook on possible future work in Chapter 6.

CHAPTER 2

# Research Issues and Approach

Since 2008, thousands of cryptocurrencies were launched. The website CoinMarketCap lists more than 7400 different ones and their accumulated value in US dollars lies around 362 billion dollars[1]. If we look at the top five cryptocurrencies ranked by price per unit (namely Bitcoin, Ethereum, Bitcoin Cash, Bitcoin SV and Monero), we can see that all five of them use a PoW consensus mechanisms. However, that will change in the future as Ethereum (the second largest cryptocurrency) already announced that their consensus model will be changed to PoS step by step[2]. There are multiple reasons for that switch, two of the main ones being high energy consumption and the goal of increasing transaction throughput. In regard to increasing transaction throughput, just switching to PoS does not go far enough. Therefore, Ethereum adds sharding as another measure. These changes have been in the works since 2015. All that underlines the relevance of research about combining low energy consuming consensus methods like PoS and ways to increase transaction throughput even further (e.g. by using DAG data structures). Additionally, the net worth highlights how crucial the security of such systems is.

## 2.1 Research Issues

New blockchain protocols promising better security, higher transaction throughput or novel features are presented on a regular basis. As we found out, already more than 7400 exist. Whenever a new cryptocurrency is launched or a paper describing a new concept is published, one must always ask the same question: How secure is this new piece of technology really? And that is the major research question of this thesis. How secure are PoS DAG protocols really?

[1] data from 14.10.2020: `https://coinmarketcap.com/`

[2] CoinMarketCap website, accessed 14.10.2020: `https://coinmarketcap.com/alexandria/article/a-dive-into-ethereum-2-0`

We focus on PoS DAG protocols in this thesis, as they promise to solve the two major problems of energy consumption and transaction throughput with one strike. Other known measures (e.g. sharding) can additionally be applied on top, improving the respective protocol even further. Another reason to focus on this type of solution is, that it provides the potential to keep all transactions in the underlying data structure instead of doing some work off-chain to reduce pressure on the protocol. Decentralization is one of the defining characteristics of cryptocurrencies. We do not want to loose this property while solving the open challenges.

There exist propoosals for DAG protocols using PoS which fulfill our requirements and we discuss them in detail in Chapter 3. These papers promise better transaction throughput than many other cryptocurrency protocols while claiming to be as stable and secure as them. More nuanced questions arise around these claims:

- **What are the conditions under which the claims are true?** Protocols usually define an adversarial model under which they claim to be secure. On the one hand, crucial requirements are defined for the system like a known participant size. On the other hand, there might not exist the information of how a specific implementation of such requirements can look like. Sometimes there even exists an implementation but the source code is not public. That is the reason why finding out which conditions are necessary and how they can be reached in practice are so important from a security perspective. Otherwise an actual implementation can torpedo the underlying security guarantees, which leads us to the next question.

- **Do the stated security claims hold under different attack scenarios?** The state of the art is evolving continuously and certain defence mechanisms against known attacks could have been already incorporated into the protocol in its design phase. There might also exist attacks which were not known or considered. To be able to make a statement about the security guarantees of a system, it is necessary to probe claims under different attack scenarios. Also, every combination of mechanisms or the introduction of completely new algorithms leads to some new and unique conditions. New attack surfaces might be opened up.

- **Do the stated performance claims hold even on being under attack?** When performance tests are provided, they often only consider the system under laboratory conditions without any malicious participants. These tests can provide valuable measurements for best case scenarios. However, it is also necessary to have a closer look at how the system performs during malicious activity. Protocols specially prone to attempts of slowing them down might not be suitable for the real world.

Last, it has to be mentioned that it is out of scope for this thesis to answer this questions for all existing PoS DAG protocols. Instead, a more directed approach is followed. This

thesis gives an overview over different protocols of that category and then focuses on a deeper analysis of one of the most promising candidates.

## 2.2 Methodology and Approach

In order to address the stated research issues and to determine the protocol under test in the first place, the following methodological approach was applied step by step:

- **Identification and analysis of PoS DAG protocols:** Identifying the PoS cryptocurrency protocols that use DAGs as a data structure is the first step. Candidates must be identified and compared based on how promising they are. Thereby it has to be checked what implementations already exist or are planned. Are certain protocols backed by multiple research institutions or companies of the industry? Moreover, it has to be checked if there exist claims that were already refuted by other research papers. The chosen candidate must be analysed in greater detail.
- **Literature Review:** The literature review should provide the necessary context to fully understand the PoS DAG protocols and the current state of the art around them. It includes looking at how traditional PoS systems work as well as looking at other non-PoS DAG protocols. Additionally, already existing simulators for testing the security of cryptocurrency protocols must be identified. Another part of the literature review is to identify typical attacks on PoS and PoS DAG systems or derive new ones based on the existing research. This guarantees a good knowledge foundation for a successful own development in the next step.
- **Attack scenarios and simulator development:** Through the last two steps, a deep understanding of the chosen candidate and possible adversary scenarios are given. Now, attacks must be chosen to assert the security claims of the candidate. A simulator is developed that builds up the DAG data structure in a randomized way in accordance to the protocol descriptions. The randomization must be based on a seed for reproducibility reasons. This simulation must be possible for every attack scenario.
- **Evaluation and discussion:** With the developed simulator it is now possible to validate the security claims of the candidate protocols. Moreover, the simulator helps to evaluate if other claims (e.g. regarding performance) can hold under the different attack scenarios. As a final step, the results are discussed.

CHAPTER 3

# State of the Art

This chapter discusses the current state of the art in terms of cryptocurrency protocols. They are grouped together into different categories, based on their consensus mechanism and underlying data structure. It has to be mentioned, that way too many protocols exist to discuss all of them here. However, the ones here are carefully chosen to give an overview over widely used mechanisms and under which conditions they work in a secure way. This chapter also covers different attack scenarios and other already existing simulators for cryptocurrencies.

## 3.1 PoS DAG protocols

This section contains different PoS DAG protocols which can be chosen as candidates according to the criteria defined in Chapter 2.

### 3.1.1 Hashgraph

When we talk about Hashgraph we have to differentiate between The Swirlds Hashgraph Consensus Algorithm [Bai16b, Bai16a] and Hedera Hashgraph [BHM19], the cryptocurrency which was built based on the consensus algorithm. This appears to be a very promising candidate for detailed investigations as it is backed by very big and influential companies like Google, IBM, LG and Boeing[1].

#### The Swirlds Hashgraph Consensus Algorithm

The Swirlds Hashgraph Consensus Algorithm [Bai16b, Bai16a] (from now on just referred to as Swirlds-Consensus) is a Byzantine fault tolerant algorithm which reaches consensus on a topological ordering of blocks within a DAG. We will first look at how its base data

[1] Hedera Hashgraph website, accessed 3.9.2020: `https://www.hedera.com/council`

structure looks like and how the algorithm works, before adding a PoS component to it. The base structure of Swirlds-Consensus are single chains of events per node (every participant is a node). These events also reference events on chains of other nodes and thus connect these chains, forming a single DAG. A single event can be the result of a node A telling another node B everything it knows so far. In general, an event represents some information change in the chain of a single node, like submitting a new transaction or getting new information from another node and is therefore also called gossip event. Each edge between two events of different chains is the process of syncing the information. Each of the nodes is performing this synchronization process continuously. It selects a random other node and performs a gossip-sync with it. Afterwards, another random node is chosen by both of them, leading to new information being spread exponentially fast. The whole DAG is basically a history of how the members have communicated with each other. An example for such a DAG can be examined in figure 3.1.

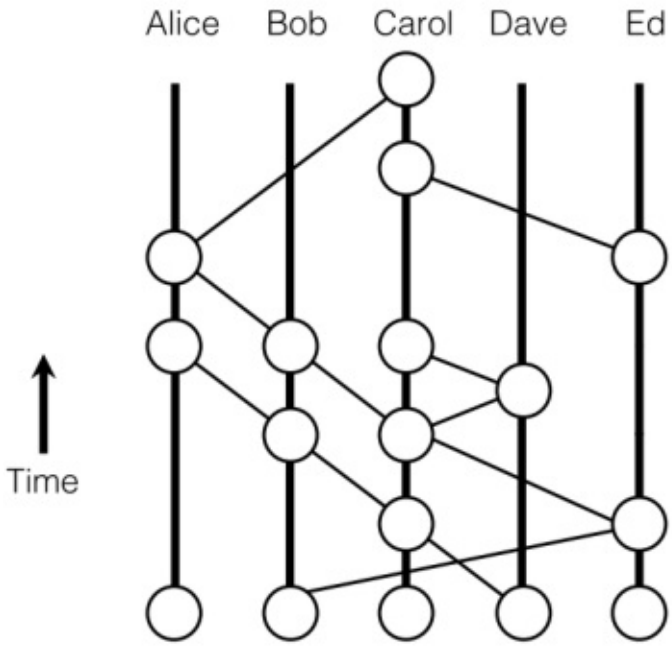


Figure 3.1: Hashgraph: gossip history as a directed graph [Bai16b]

Every edge is representing a gossip-sync, where one node is providing another node its locally built DAG. For example, when node Alice is telling node Bob about a new transaction, node Bob also gets the information when node Alice first learned about the transaction and from whom. This ultimately leads to each node being able to see the same DAG of events locally.

However, it is not enough for every node to have the same DAG locally, as it is also necessary to agree on the same topological ordering of the events. Other Byzantine

fault tolerance protocols solve that by nodes sending each other votes over potentially multiple rounds and eventually agreeing on an order. Swirlds-Consensus does the same but without sending the votes over the network. When each node sees the same base data structure, everyone can calculate how the other participants would vote and therefore reach consensus without communicating any further. It has to be noted however, that the local DAGs of nodes are not completely equal at any given time. They mainly differ in very recent events, or events that should be placed further down in the graph but were emitted later to the other participants. Additionally, they differ if a malicious node sends conflicting information to other nodes. For example, a malicious node can create a fork and communicate different events to other nodes (events of other nodes cannot be forged without their private key as all events are signed). The attack model of the protocol states that an adversary can stop messages from getting to specific nodes but eventually needs to let a message through. It is therefore only a matter of time, until a node recognizes the fork. If there exists a fork, then all involved and later events of that node are excluded from participating in the consensus mechanism. Swirlds-Consensus itself does not impose any timeout for all of this to happen. Over time, every event gets a final timestamp assigned which results in a total ordering of all events. The protocol is built in a way, that once any honest node assigned such a final timestamp to an event locally, all other honest nodes either have already assigned the same timestamp or will do that in the future. The whole process happens in multiple steps and is always triggered when a node creates or learns about new events.

Every node calculates the total order of events over multiple election rounds. First of all, a round number is assigned to each event. Starting from the genesis events, all events up each individual chain are included into the first round until events are reached that can *strongly see* $2n/3$ nodes, where $n$ is the total number of participating nodes. These events are not included in the same round anymore. An event sees all its ancestors. An event strongly sees another event, if it can see $2n/3$ different nodes which can also see the event. This process is then repeated for the remaining nodes until every node has a round assigned to it. Now, witnesses are chosen within every round. The event created first of every node in the current round is considered to be a witness. Next, for each witness it is decided if it is a famous witness. A witness of round $r$ is considered famous, if it can be strongly seen by $2n/3$ witnesses of the round $r + 1$. If the vote is too balanced, the voting procedure continues for as many rounds as necessary until a consensus is reached if a witness is famous or not. Witnesses always look at the votes of the witnesses of the round before and if they cannot decide famousness, set their own vote according to the majority of votes they recognized. In theory the vote could be too balanced forever. However, Swirlds-Consensus guarantees to come to a result with probability 1. To achieve that, coin rounds are introduced. Every $c$ rounds, if a witness cannot decide famousness, it votes randomly instead of voting with the majority of witnesses it can strongly see of the round before. Voting randomly means deriving the vote from the middle bit of the event's signature.

Once this famous witnesses are decided, a round received value and a timestamp can be calculated for every not yet decided older event. There is at most one famous witness per node chain in a round. (If there are more, both are not considered famous witnesses.) The round received value is the lowest round number in which all of these famous witnesses can see the event. To get the timestamp for an event x, every node looks with which event y it has learned the first time about x within the ancestors of the respective famous witness. The creation timestamp of y is then considered as the receiving time for event x for the node with the famous witness. The global receiving time for that event is simply the median of all that timestamps. Now, everything needed for a topological ordering of all events is known. First, events are sorted after their assigned round number. Events of the same round are then sorted by the receiving time we just calculated. If there are still ties, events are sorted by a procedure based on their signature.

The security of the whole process is tied to the parameter $n$. Every node needs to know this global value, as it enables each node to know when it sees enough of the graph to decide on the final timestamp of an event and not earlier. This ensures that once a decision about the final timestamp of an event is made by a node, all other honest nodes will agree on that as well.

In its base form Swirlds-Consensus considers every node as equal and does not give any information about how to handle changing node numbers. However, a change is proposed by introducing some kind of stake and weighting votes proportional to the stake of voters. This implies that median calculation must be adapted to calculate a weighted median of all nodes. That means that every time the phrase "more than $2n/3$ nodes" was used, it should then be "nodes owning more than $2n/3$ of the stake" instead, where $n$ represents the total number of involved stake tokens. Swirlds-Consensus gives multiple suggestions what stake could be. One would be a node's coins. Another one is stake assigned to a group of members with mutual trust which can be delegated according to a specific rule set. Anyway, the stake record can change and the protocol must be able to handle that. Once it is decided for all witnesses of a round if they are famous or not, it is clear which events of earlier rounds get a received round and a final timestamp assigned. If one of this newly finalized events contains transactions which change the stake record, the whole algorithm needs to be rerun for other events of the current and future rounds. This might change round numbers and (famous) witness status of events. Swirlds-Consensus only sketches out this handling of stake and does not provide any specific algorithm or pseudo code for it.

### Hedera Hashgraph

Hedera Hashgraph states in its whitepaper [BHM19] that it is a public network based on Swirlds-Consensus, making it to a PoS cryptocurrency using the presented DAG

structure. The mainnet went online in December 2018[2]. Hedera Hashgraph is not fully rolled out yet and still under development. It is also not open source and controlled by a government council of up to 39 leading global enterprises. This council governs the code base and should ensure stability as well as that the platform follows legal obligations and regulations. It is planned that government members serve limited times and have equal votes to avoid centralized control.
Hedera Hashgraph lies out multiple phases for scaling the cryptocurrency. First, there is a permission based setting with the government council controlling all nodes participating in the consensus algorithm and only a few coins are given out to other participants. Step by step, more trusted nodes are included into the consensus algorithm and more coins are distributed. In the end, coins should be widely distributed and everybody is able to participate in the consensus algorithm, thus replacing the permission based setting with open consensus.
The whitepaper presents mainly high level descriptions of how the protocol solves the issues of converting the Byzantine agreement Swirlds-Consensus into a functional cryptocurrency. The project itself will never be open source, but it was promised that Version 1 of the source code will be published for reviewing within 2020. This promise was kept when the source code was published in October 2020[3]. Until it was extensively reviewed, all the information must be taken with a grain of salt, as the whitepaper itself contains the following lines:

> It constitutes general information only and may be updated. It also contains forward-looking statements that are based on the beliefs and intentions of the authors, as well as certain assumptions made by and information available to them. [BHM19]

### PoS mechanism

Every node that actively participates in the consensus network can stake its own coins. Every owner of coins who does not operate an own node can proxy stake coins to node operators. All staked coins together are the total weight and every participating node has a weight according to its staked and received proxy staked coins. Users pay three different types of fees. First, they pay a *node fee* directly to the node which transmits their transaction in an event to the network. A *network fee* must be payed for validating the transaction and reaching consensus on it, which depends mostly on the transaction's file size and the number of involved signatures. Last, a *service fee* compensates nodes for ongoing burden, for example for file storage transactions. The latter two are paid into a special Hedera Treasury account and every 24 hours participating nodes receive a portion of the collected amount as reward payments. Rewards are only distributed to nodes which took part for the whole duration since the last reward payment and published an

[2]hedera website blog, accessed 7.9.2020: `https://www.hedera.com/blog/hedera-hashgraph-launches-mainnet-early-access-program`

[3]source code on Github, accessed 8.1.2021: `https://github.com/hashgraph/swirlds-open-review`

event in most rounds (there are hard limits implemented). Everybody can transfer own coins which are staked at any time, but due to the reward mechanism looses a portion of the own profit when doing that. That basically means that one can loose nothing while getting rewarded when (proxy) staking coins for a long enough time period. The idea is that most coins are staked, making it harder for an attacker to possess more than 1/3 of the stake. Additionally, mining strategies that diverge from the norm do not increase one's income, as income is spread evenly over all nodes based on stake alone.

In the Swirlds-Consensus section we noted, that the security of the system depends on the knowledge of how many participants exist in a round. Here, stake is based on coins and can change at any time. This all results in the need of mechanism to know the exact participant number at any time. This is ensured through signed state proofs. Every node maintains a copy of the state (including public keys of all participants and their coins) and updates it at the end of every round by processing all newly decided transactions. This state can be seen as an address book, containing all participants. It also references a history of former address books reaching back to the genesis address book, each signed by enough nodes of the address book before. After updating, the state is signed and gossiped to all the other nodes. Every node collects all the incoming signatures. Signatures of nodes controlling more than 2/3 of the stake must be collected for this new address book to be valid. Then, every node can prove to any client that their state is correct.

#### Performance

Due to always knowing the current state of all accounts, Hedera Hashgraph does not need to consider older rounds of which all events already got a final timestamp assigned and thus does not have to keep them in memory. This is a huge difference to most other cryptocurrencies which often need to store the whole history back to the genesis event, to be able to approve if a transaction is valid. Moreover, all published events get a final timestamp assigned eventually and thus no events are discarded as it is the case in many other cryptocurrencies. Additionally, publishing new transactions is only limited by the actual network bandwith.
Performance tests were made which claim that Hedera Hashgraph can reach consensus on every transaction under a load of 50 000 transactions per second in around 10 seconds, assuming 64 nodes are participating in 8 different regions spread all over the world. However, these tests cannot be considered indicative enough because of multiple reasons. First, all transactions were only 100 bytes big including no signatures at all. No transactions were processed, only consensus calculation was tested. Hedera Hashgraph even states, that including signatures would result in the need of processing power to validate hundreds of thousands of digital signatures per second. Also, handling Smart Contracts or file storage transactions would need way more bandwith. And last, it is planned that the network should support millions of nodes and not just 64. However, in June 2020 a company operating on the (at that time in terms of participants and functionality limited) mainnet reports a transaction throughput of 1372 per second. So,

throughputs unimaginable for other cryptocurrencies can be reached[4]. It has to be noted though, that the scenario is still far from real world conditions that can be expected once the last phase of the Hedera Hashgraph deployment is reached.

#### Functionality

Hedera Hashgraph provides four different types of functionality. First, it is a cryptocurrency with low transaction costs. Estimated fees for simple transactions lie below 1 cent per transaction. Second, Smart Contracts can be published and executed on the platform. For writing Smart Contracts the programming language Solidity is supported, which is also used by the popular PoW cryptocurrency Ethereum [Eth]. Third, a file service is included which allows users to store data for a specified amount of time and benefit from a consensus timestamp on the data. Fees heavily depend on the size of the stored data. The fourth functionality provided is a consensus service, which is not fully implemented yet. It basically provides an ordering over events and can be used by external applications to order anything. The whitepaper lists a decentralized stock market as an example, where the consensus service ensures decentralization and fairness. Fairness means that the transaction are ordered based on reaching nodes which posses more than 2/3 of the total stake first. This cannot be influenced heavily, as the distribution happens based on the gossip protocol. One can only invest in a better internet connection to transmit a transaction to the initial node faster. One can also publish the transaction to multiple nodes, but only the fastest spreading one is considered valid then. However, in that case submission fees must be payed for all of them.
As nodes participating in the consensus do not store the history, so called mirror nodes must step in for providing the results. Mirror nodes are nodes participating in the gossiping part of the Swirlds-Consensus, but they never vote and they cannot publish new transactions. Mirror nodes store either the whole graph or a filtered subset of transactions based on a topic string which can be added to every transaction. Mirror nodes can be configured/programmed individually in regard of how they handle that data. For example, it can be forwarded automatically to another application or stored in a database responding to requests from external systems via a web interface.

### 3.1.2 Nano

Nano [LeM17] is another cryptocurrency utilizing a DAG data structure. An implementation was launched in autumn 2015 and it focuses solely on being a fast and simple payment system. It does not provide any Smart Contracts support. Regarding available information, one must be careful here as the original whitepaper [LeM17] is outdated and one has to look at the living whitepaper [lw20] to get up to date information. In Nano, every account has its own blockchain where each block consists of only one transaction. Every transaction references its predecessor on the respective chain and only the owner of

[4]Hedera Hashgraph website, accessed 7.9.2020: `https://www.hedera.com/blog/adsdax-achieves-1372-cryptocurrency-transactions-per-second-on-hedera-hashgraph-with-zee-entertainment-enterprises-sets-new-industry-record`

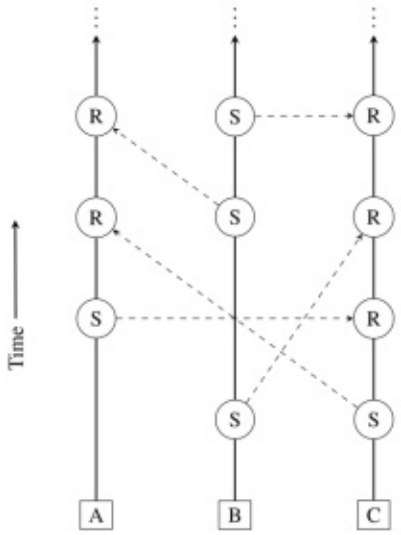


Figure 3.2: Nano DAG data structure [LeM17]

the account can add more transactions to it. To send money from one account to another, the sending account includes a *send block* into its own blockchain and broadcasts it to the network. The transaction is now considered unsettled until the receiving account adds a *receive block* to its own blockchain. The receiver thus has full control over the order of transactions within its own blockchain. The result is the DAG data structure as depicted in Figure 3.2.

Blocks are finalized by a procedure called Open Representative Voting (ORV). This is where stake comes into play. Participants vote yes or no on a block they receive based on if they consider it valid. Votes are weighted based on the stake (amount of coins) one owns. Participants can freely delegate their voting power to other participants and change their mind at any time (by issuing a special transaction). To speed up decisions, Nano only allows nodes to vote which posses 0.1 percent of the total weight or more, effectively limiting the maximum number of voting participants to 1000. Every participants finalizes or rejects blocks once they see enough valid broadcasted votes. The value of "enough" is specified locally on client side, but the default value is more than 50 percent of online vote weight. There is no further explanation available within the whitepaper what online specifically means and how one can determine this value. Once a transaction is finalized (by each node on its own), it is irreversible (also called cemented). This definition leads to some open questions, as the security of this protocol part stands and falls with how online is defined and determined. Furthermore, by only looking at these whitepapers it remains completely unclear how changing stake influences finalization. Remember, every participant can delegate its voting weight at any time by issuing a specific transaction. Furthermore, other transactions might also change the stake distribution before a vote ends.

So far we have examined the system under the assumption that everybody behaves honestly. However, a participant can try to perform a double spend by issuing transactions that reference the same parent and thus represent a fork. There exists an empty subsection for fork handling on the page of the living whitepaper about the ORV mechanic[5] which leads to more questions like "Does there exist a special fork handling or is the normal ORV voting sufficient?".

A big advantage but also a potential problem is, that there exist no transaction fees in Nano. This is an advantage for users, but leads to a lack of monetary incentive for people to host full nodes to participate in the consensus algorithm. Second, transactions can be issued without any cost, opening a huge potential for malicious activity in form of spamming. To protect against spamming, Nano forces participants to solve a PoW hash puzzle for issuing a transaction. This protection is not sufficient as hashes can be pre-computed and a processor from the year 2014 like the Intel Core i7 4790K AVX2 can compute such a hash every three seconds according to the original whitepaper [LeM17]. This *transaction flooding* attack is known and referenced in the living whitepaper, as well as multiple attacks that enable an adversary to waste resources of other participants.

To sum up, supportive to the idea of considering Nano a valid candidate is that an implementation exists and is actively used. On the other hand, many functional questions with huge impact on the system's security remain unaddressed after studying the available information in the whitepapers. Furthermore, multiple unsolved security flaws are known.

### 3.1.3 Avalanche

The authors of Avalanche introduced a family of leaderless Byzantine fault tolerance consensus protocols, each built atop of the former one, ultimately creating Avalanche [RYS+20]. It is another DAG based consensus protocol for cryptocurrencies. The paper itself lies out the consensus mechanism and although there exists a test system, it does not describe the mechanics of a fully autonomous peer-to-peer payment system. It also does not contain a specific PoS mechanism, but is still seen as a viable candidate as it explicitly states that it recommends to adopt an already established PoS mechanism for sibyl control, which it claims to be able to incorporate.

Avalanche builds upon the protocol Snowflake, whose core is the metastable voting mechanism. It is a mechanism to decide between two conflicting states. On receiving a transaction, a node asks a set of $k$ (configurable protocol parameter) randomly chosen other nodes about their opinion on the validity of the transaction. A node incorporates its own yes or no vote into the request message. Other nodes that have no opinion about this matter yet just accept whatever the querying node suggests, replying the accepted

[5] section accessed 20.12.2020: `https://docs.nano.org/protocol-design/orv-consensus/#fork-resolution`

```
1:  procedure SNOWBALLLOOP(u, col_0 ∈ {R, B, ⊥})
2:      col := col_0, lastcol := col_0, cnt := 0
3:      d[R] := 0, d[B] := 0
4:      while undecided do
5:          if col = ⊥ then continue
6:          K := SAMPLE(N\u, k)
7:          P := [QUERY(v, col)  for v ∈ K]
8:          maj := false
9:          for col' ∈ {R, B} do
10:             if P.COUNT(col') ≥ α then
11:                 maj := true
12:                 d[col']++
13:                 if d[col'] > d[col] then
14:                     col := col'
15:                 if col' ≠ lastcol then
16:                     lastcol := col', cnt := 1
17:                 else cnt++
18:                 if cnt ≥ β then ACCEPT(col')
19:         if maj = false then cnt := 0
```

Figure 3.3: Deciding validity of transactions based on the metastability [RYS+20]

value as their answer and start querying other nodes themselves. Nodes that already have an opinion reply according to it. Every node decides on a final state based on the pseudo code of the algorithm presented in Figure 3.3. Here, yes and no answers are outlined as colours red and blue. Lines 1-3 initialize the system, while line 4 shows us that the program logic of the lines below is executed in iterations until a final decisions is made. Once a node has a vote assigned, it queries $k$ other nodes for their opinion (lines 5-7). The ultimate goal is to increase a finalization counter to a value bigger than $\beta$ (the second protocol parameter) to finalize the vote, as line 18 shows us. Whenever a node receives a set of answers of which at least a fraction of $\alpha$ shows the same result, it has the chance of increasing this finalization counter (line 10). If this does not happen, the counter is reset to 0 (line 19). $\alpha$ is another protocol parameter, which must be chosen under the condition $\alpha \geq \lfloor k/2 \rfloor$. Line 19 presents, that if at least a fraction of $\alpha$ nodes reply with the same vote, another counter for the respective colour is increased. This counter is never reset. If the respective vote is different to the current opinion and its counter passes the value of the current opinion's vote counter, the opinion is updated an the finalization counter set to the value 1 (line 12-16). In the other case, where the query result supports the current opinion, the finalization counter is increased by one. The authors argue that you can choose the parameters $\alpha$, $\beta$ and $k$ in a way that gives you a highly secure instance of the protocol with safety failure probabilities below a target value $\epsilon$. The safety failure probability defines the network still changing its decision after reaching a certain bias for a specific voting result. In terms of liveness, the parameters must be chosen with a specific maximum percentage of adversarial nodes in mind. The higher this percentage, the slower the network finalizes transactions. However, an adversary controlling a bigger proportion of the network can stall termination of the mechanism with high probability.

The paper comments the transition from Snowball to Avalanche with the following words:

> Avalanche consists of multiple single-decree Snowball instances instantiated as a multi-decree protocol that maintains a dynamic, append-only directed acyclic graph (DAG) of all known transactions. [RYS+20]

Instantiating a Snowball instance means, that Avalanche creates a conflict set for each transaction (containing only one transaction, if the transaction is virtuous) and executes the algorithm we discussed for each of these sets. It does however only query the opinion of other nodes once for every conflict set. Instead of using multiple counters and question rounds for final decisions, nodes maintain multiple conflicting branches and have an opinion which of them they currently prefer. They only respond with a yes vote if the queried transaction and all its predecessors lie within their preferred branch. If a node gets a positive answer of a specific fraction of asked nodes, the transaction it asked for is considered owning a chit. The confidence of a transaction being valid can be measured by the amount of chits in its successor set. Votes on a transaction of the graph now count as votes for all other transactions on the path back to the genesis transaction. Similar to Bitocin, the deeper a transaction is inside this structure, the less likely it becomes for it to be undone. Progress of the system thus depends on transactions being attached to the tips of the DAG. However, it has to be noted here that termination is not guaranteed for conflicting transactions. If a vote does not terminate, the coins which are part of this double spend attempt are lost forever. This might yield a problem to transactions, which have the not-terminating transaction in their parent set. Avalanche allows reissuing the exact same transaction with a different parent set to solve that problem. These two transactions are not seen as conflicting. The paper mentions that clients following an adoptive parent selection strategy would be the best for the system. It does not describe that strategy in detail, nor does it provide any incentive for users to follow that strategy.

The adversarial model of the protocol states that the distribution of message delay for honest nodes must stay within the bound of the exponential distribution for the system to guarantee security. An adversary knows the state of every honest node but is not allowed to schedule or modify communication between them. Adversarial nodes may behave arbitrarily. This is a way weaker model compared to e.g. Hashgraph. The presented consensus mechanism does not impose a total order over all transactions, which usually results in not being able to support Smart Contracts (although this topic was not specifically mentioned in the paper). Moreover, transaction fees are listed as protection against flooding attacks, but details are not specified. To sum up, Avalanche presents some novel mechanisms of how consensus can be reached, but lacks details for many questions that have to be considered for a fully operational payment system.

## 3.2 PoW DAG protocols

The following subsections show how DAG data structures can also be used in PoW cryptocurrencies to increase transaction throughput.

### 3.2.1 Phantom / Ghostdag

In conventional Proof of Work (PoW) cryptocurrencies like Bitcoin, every block references a single predecessor and therefore the public consensus ledger is a chain. If an invalid block (e.g. one containing a double spend) is added to the chain, honest miners will ignore it and continue to mine a block for its predecessor. As only the longest chain is considered valid, these invalid blocks will be completely ignored. [Nak08]
Phantom [SWZ18] on the other hand generalizes this protocol to increase transaction throughput. In Phantom blocks are structured in a directed acyclic graph instead of a chain. That means, that every block can contain hash references to multiple predecessors. The protocol provides a total order over all these blocks. The order over these blocks includes an order over all transactions, as transactions are considered ordered after their appearance within the block. All this ordered transaction form the ledger and if there are transactions invalidating each other (like double spends), only the one first in order is valid and the other one is not considered part of the ledger. For such an algorithm to work, transactions of blocks from honest nodes must appear before transactions of malicious ones.
This is achieved by the following algorithm: First, the largest subset of well connected blocks is recognized which we call the largest k-cluster. Blocks are added to this set by comparing how many other existing blocks are referenced by the block and how many are not (the set of not referenced blocks is called anticone). The anticone of honestly mined blocks is usually small, as only blocks not known to the honest miner are not referenced. As every honest node immediately publishes found blocks, only blocks mined between the current time $t$ and an upper bound of the network delay $D$ are not known to the honest miner. This creates an intervall $[t - D, t + D]$ for which the proof-of-work mechanism can guarantee that the maximum number of blocks created in that time period is usually below a certain parameter $k$. Blocks which have an anticone smaller than $k$ are considered part of the well connected set. Now, a full topological order is introduced where blocks within the well connected k-cluster go first and the other ones are added last.
The underlying optimization problem is NP-hard. Therefore, there exist no efficient algorithms for solving that problem in general. The Ghostdag protocol [SWZ18] circumvents this inefficiency using a greedy algorithm for finding the k-cluster. Sompolinsky et al. use two colors in their paper to differ between blocks in the k-cluster (blue) and outside (red). The greedy algorithm constructs the blue set of the DAG by inheriting the blue set of the best tip of the graph. The best tip is the tip with the largest blue set in its past. This rule is applied recursively and therefore results in a chain through the k-cluster back to the genesis block. The blocks within this chain are the first blocks of the total global order. Afterwards, the other blocks of the k-cluster are added according to some

**Algorithm 1** Ordering the DAG

**Input:** $G$ – a block DAG, $k$ – the propagation parameter
**Output:** $BLUE_k(G)$ – the Blue set of $G$; $ord$ – an ordered list containing all blocks in $G$

1: **function** ORDERDAG($G, k$)
2: **if** $G == \{genesis\}$ **then**
3: **return** $[\{genesis\}, \{genesis\}]$
4: **for** $B \in tips(G)$ **do**
5: $[BlueSet_B, OrderedList_B] \leftarrow$OrderDAG($past(B), k$)
6: $B_{\max} \leftarrow \arg\max \{|BlueSet_B| : B \in tips(G)\}$ (break ties according to lowest hash)
7: $BlueSet_G \leftarrow BlueSet_{B_{\max}}$
8: $OrderedList_G \leftarrow OrderedList_{B_{\max}}$
9: add $B_{\max}$ to $BlueSet_G$
10: add $B_{\max}$ to the end of $OrderedList_G$
11: **for** $B \in anticone(B_{\max}, G)$ **do** in some topological ordering
12: **if** $BlueSet_G \cup \{B\}$ is a $k$-cluster **then**
13: add $B$ to $BlueSet_G$
14: add $B$ to the end of $OrderedList_G$
15: **return** $[BlueSet_G, OrderedList_G]$

Figure 3.4: Phantom algorithm [SWZ18]

topological ordering and in the end the blocks of the red set are appended. The original paper provides the pseudo code depicted in Figure 3.4 and the following description for the proposed algorithm.

> The algorithm begins with the base case where the DAG consists of the genesis block only (lines 2-3). Next, it performs a recursive call to compute the Blue sets and ordering of the past of each of the DAG's tips (lines 4-5), and inherits those of the best tip (lines 6-8). Then, the selected tip is added to the Blue set $BlueSet_G$ and to the last position in the current ordered list $OrderedList_G$ (lines 9-10). Then we iterate over $anticone(B_{max}, G)$ in some topological way which guarantees that a block is visited only after its predecessors are (lines 11-14). For every block we visit, we check if adding $B$ to the Blue set will preserve the k-cluster property, and if this condition is satisfied, we add $B$ to the Blue set (lines 9-13); either way, we add $B$ to the current last position in the list (line 14). Finally, we return the Blue set and the ordered list. Note that the recursion in the algorithm (line 5) halts, because for any block $B \in G :| past(B) |<| G |$. [SWZ18]

Ghostdag achieves a huge boost in terms of transaction throughput per second compared

to classical PoW protocols like Bitcoin by increasing the block creation rate significantly while maintaining the same security guarantees. However, values for the parameters $D_{\max}$ (the maximum network delay) and $k$ (the anticone size) must be set. These values stand in close relation to each other and the security guarantees of the whole protocol. For the sake of maintaining a proper security level it is wise to use a safety margin which is a bit larger than necessary for the parameters. And that is exactly why the protocol cannot reach optimal performance based on the current network conditions as these parameters are hard coded and are not adapted during execution.

### 3.2.2 Spectre

Spectre [SLZ16] is similar to Phantom in terms of how the ledger looks like. Again, a DAG is formed instead of a single chain, where each new block references all blocks on the tip of the graph known to the honest miner. It is assumed that an honest miner always publishes a block immediately and the order of transactions within a block corresponds to their appearance. All blocks are part oft the ledger, but when a conflict occurs between two transactions of different blocks, Spectre determines which one is valid and which one is considered invalid and therefore not part of the ledger. This is done through a pairwise voting procedure that delivers which block is considered to precede the other one. Transactions of the preceding block are considered valid.
However, a major distinction is that there exists no global order for all blocks here. The algorithm only yields a pairwise ordering and thus this ordering relation is not transitive. Also, the term *voting procedure* might be a bit misleading as no actual voting by nodes happens. Instead, each block in the graph is considered voting for one of the two blocks based on the structure of the DAG. To which block in question the vote goes is determined by a static rule set. Whichever of the two compared blocks gains more votes is considered a predecessor of the other one and therefore its transactions are considered valid.
To assign the vote of a block z to one of two conflicting blocks x and y, where all these blocks are part of the same DAG, the rules below are followed. It has to be mentioned that predecessor and successor within this rules includes blocks in a transitive relationship. Moreover, a virtual block of G is introduced, which has no successors and all blocks are its (transitive) predecessors. This block represents the aggregated vote of the entire block DAG.

1. If z is a successor of x but not y, z votes in favour of x.

2. If z is a successor of x as well as y, all votes of predecessors of z are counted and z votes together with the majority. If a tie occurs, there must be some rule to break the tie (e.g. header information, lexicographical ordering of the hashes of x and y, ...)

3. If z is not a successor of x or y, it votes the same way as the majority of its predecessors.

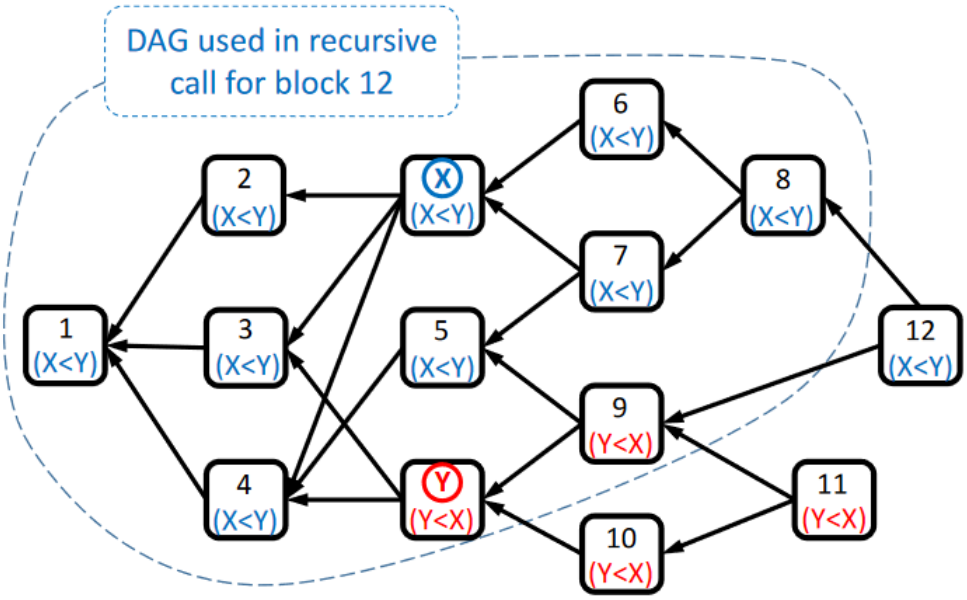


Figure 3.5: Spectre example [SLZ16]

4. If z = x or z = y, then z votes for itself to succeed any of its predecessors and to precede any block outside of that set.

5. If z is the virtual block G, it votes like the majority of blocks in the graph and therefore aggregates all votes to a single final one.

Spectre also provides an example of a voting procedure (see Figure 3.5) including the following detailed description:

> Block x and blocks 6-8 vote $x \prec y$ as they only see x in their past, and not y. Similarly, block y and blocks 9-11 vote $y \prec x$. Block 12 votes according to a recursive call on the DAG that does not contain blocks 10,11,12. Any block from 1-5 votes $x \prec y$, because it sees more $x \prec y$ voters in its future than $y \prec x$ voters. [SLZ16]

On comparing Spectre to Ghostdag, many similarities can be found between these two protocols. Nevertheless, it sticks out that Spectre features no parameters that need to be set upfront. Therefore, Spectre can adapt to the real network conditions and thus achieves higher transaction rates. This is mainly possible because Spectre does not need to create a total order over all transactions to work. Conflicting transactions are resolved by just voting on the conflicting blocks. However, some features we know from other blockchain protocols like Smart Contracts need a total order to work in all cases. Spectre does not provide that and therefore does not support such features, which is a huge drawback.

### 3.2.3 Prism

Prism [BKT+18] is a PoW based blockchain protocol, which goes a new way. Instead of having just a single block type and ordering every block in an unstructured DAG, Prism looks at what functionalities a block stands for within a cryptocurrency protocol like e.g. Bitcoin [Nak08]. Prism identifies three atomic functionalities. Before looking at them, a specific view of how the main chain (e.g. longest chain) is determined must be established. We view the process of it as being all about how a leader block is elected among all the blocks at each level (distance in number of blocks to the genesis block) of the block tree. These leader blocks then form the main chain from the last one all the way to the genesis block.

With that in mind, we can have a look at the three purposes that every block serves. First, each block adds transactions to the main chain. Second, each block stands for election to become a leader block and therefore be part of the main chain. And third, every block adds a vote to all its ancestors to become leader blocks. Prism separates this functionalities and introduces an own block type for each of them. There are stand alone transaction blocks, which only contain transactions and no references to any other block at all. Then there are proposer blocks, which include a reference to the last valid proposer block known to the miner. They form the classical chain (or tree in case of forks) and each of them also references one or multiple transaction blocks. Multiple proposer blocks are considered being a group if they are on the same level. And last, there are voter blocks, which form their own chain (or again a tree in case of forks). These voter blocks reference a single proposer block per level, making it the leader of its group. The referenced transaction blocks of the leader are used to form the final ledger. As the longest voter chain decides which transactions are included into the final ledger, the security of the system is based on the voter chain. This deconstructing process is also visualized in Figure 3.6, taken from the original Prism paper [BKT+18]. On the left side you can see how a typical PoW block structure looks like (e.g. in Bitcoin) and on the right side you can see how the same structure would look like under the Prism protocol. You can also see, that the new structure is no tree anymore. Instead it is a DAG.
It is also important to note, that there exist multiple voter chains instead of a single one. Each valid voter chain can contain only one vote per proposer chain level. The proposer block of a level which has the most votes is considered the leader block. That means, that a block being chosen as leader of its level is still pretty unstable after the first vote (the first block on any voting chain referencing the proposal block). However, as more votes are added to different voting chains or as more blocks are appended to the blocks on the voting chains referencing our proposal block, the less likely it becomes that a leader block changes. If we look at each chain on its own, the vote stabilizes at the same rate as Bitcoin (e.g. for the likelihood that a leader block changes to be below $10^{-3}$ in the face of an adversary controlling 30 percent of the hashing power, the vote has to be 24 blocks deep). However, due to the fact that there are more voter trees which have an aggregate opinion, the vote stabilizes way quicker. In Prism it is enough that each voting block has two successors in its voting chain to achieve the $10^{-3}$ security guarantee under

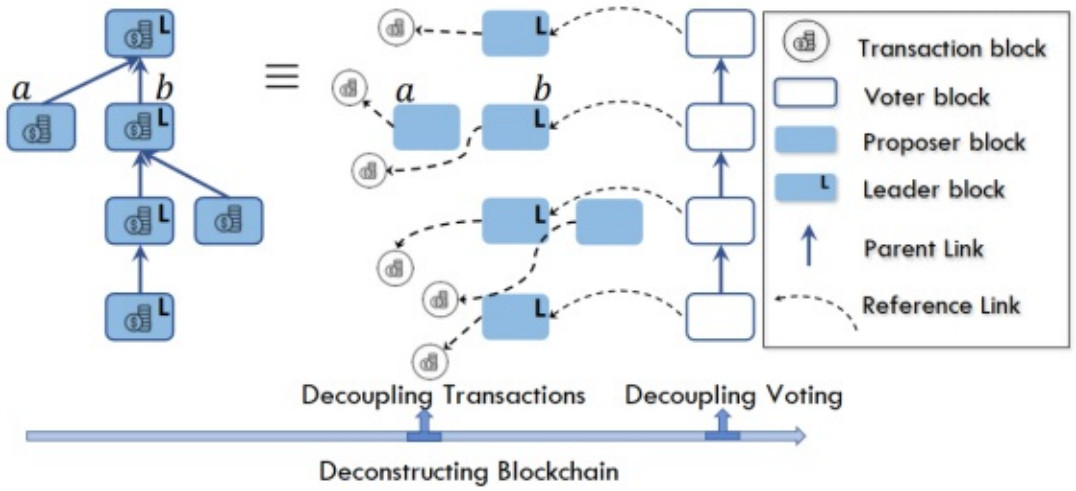


Figure 3.6: Representation of a classical PoW block structure in Prism [BKT+18]

the same adversary strength if there exist 1000 voting chains. It has to be noted here, that the PoW challenge of the protocol is designed, that there are only few proposal blocks created on the same level by honest miners.
Talking about mining, an adversary cannot target a specific voting chain directly. In general, miners mine on a transaction, a proposal and a voting block for each voting chain at the same time. First, they create a so called *superblock*, which contains information about all potentially mined blocks. Then they try to find the right nonce to solve the hash puzzle. Once a valid solution is found, the solution hash itself defines which type of block (and in case of a voting block, to which chain it belongs) a miner just successfully created.
Based on the structure described above, Prism can construct a totally ordered list of all valid transactions. This is done by just ordering all transaction blocks based on references in the lead proposal blocks of each level. It has to be noted here, that proposal blocks can also reference other proposal blocks, which are not part of the leader chain. The transactions they refer to are also included in the ordered transaction list. If two conflicting transactions occur, the one which comes first in the ordered list is considered valid and the other one will not be part of the final transaction list. The discussed procedure can be examined in Figure 3.7.
Prism has not much in common with Ghostdag and Spectre. Instead of changing the block tree structure into an DAG, it introduces different types of blocks which are structured in their own individual tree structures. Cross references between this tree structures lead to a big DAG. So, a main difference is that the DAG can be called structured to the nature of its components compared to the arbitrary growing DAGs of the other two protocols. It does however provide a total order like Ghostag does and therefore supports Smart Contracts. It is also worth mentioning, that the decoupling of functionality into different block types allows multiple ways of scaling up the transaction throughput. For

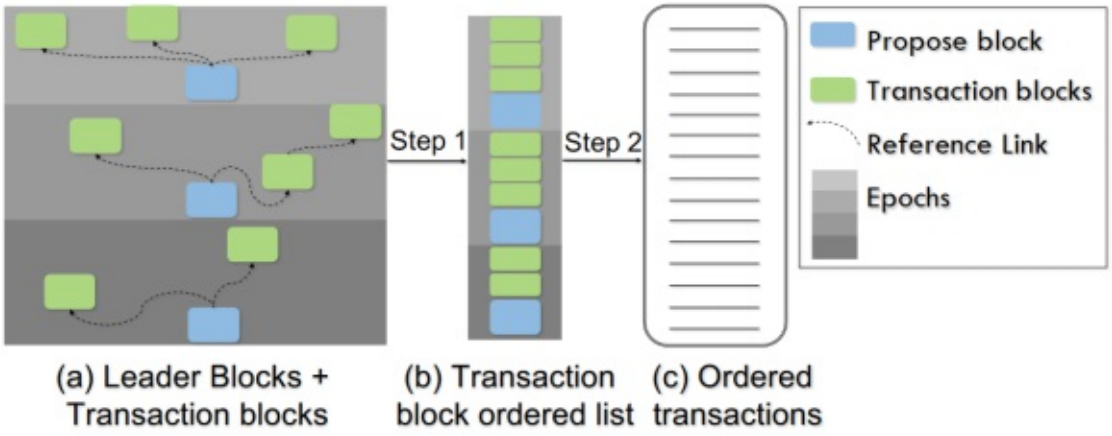


Figure 3.7: From proposal blocks to a totally ordered list of transactions [BKT+18]

example, the number of voting chains or the number of transaction blocks each proposer block is allowed to reference are levers to influence the throughput.

### 3.2.4 IOTA

IOTA is a cryptocurrency specifically designed for the Internet-of-Things (IoT) industry. It is based on the Tangle, a DAG structure for storing transactions [Pop18]. In IOTA there exist no transaction fees at all. Furthermore, every participant is equal in contrast to other cryptocurrencies which differentiate between users participating in the consensus mechanism (like miners) and others who are just using the provided services.
It all starts with a genesis transaction which sends all available tokens to a set of founder addresses. Every node can then publish transactions by approving and referencing two already existing transactions (the first transaction after the genesis transaction can only reference one). Every transaction represents a node in the Tangle and every reference is a directed edge. There are no strict rules on which former transactions have to be referenced, but there exists a reference rule set that should be followed. However, not following this rule set does not lead to any direct punishment. The system is designed in a way that following it should result in more nodes referencing one's own transaction and thus a higher chance that it will end up in the valid Tangle and not on some abandoned orphan branch. Every published transaction has a weight assigned to it and these weights play a role in determining which transactions should be referenced. Publishing transactions in IOTA is currently PoW based and weights vary based on the found hash puzzle result, as you can see in the Tangle presented in Figure 3.8. The rule set defining which former transactions should be referenced is called the tip selection algorithm, as always transactions on the tip of the graph are selected. First, a reasonably large interval of transaction somewhat away from the tips is chosen. Within this interval a number of $N$ transactions are chosen as starting point for random walkers. Each random walker just moves in direction of the current tips, always selecting a random incoming

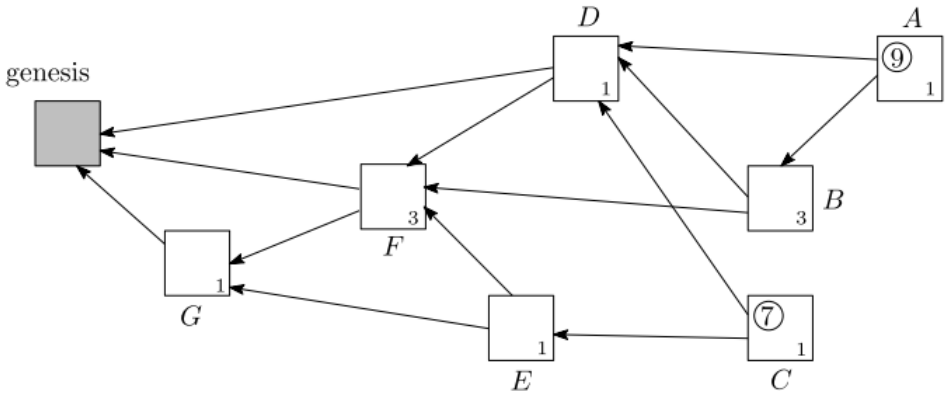


Figure 3.8: Tangle example [Pop18]

edge. The probability of taking an incoming edge over others is dependent on a chosen constant and the cumulative weight of the transaction on the other end, which ensures that tips of heavier Sub-Tangles are more likely to be reached in case of two conflicting Sub-Tangles. It has to be noted here, that the tip selection algorithm also randomly backtracks for a step. This will become important when we look at the security aspects of the system. The two tips that are reached first are chosen for the validation and referencing process. However, tips that were reached too fast should be ignored, as it is very likely that they are *lazy tips*. These are tips further back in the graph, which are not further extended. They can be created by nodes not executing the suggested algorithm, which instead just extend any transaction. Choosing such lazy tips is not a good idea, as transactions are becoming more and more final based on their own weight and the weight of the transactions that reference them directly or indirectly. As honest nodes also check transactions for validity before approving them, only valid parts of the Tangle get extended. If a conflict arises, only one Sub-Tangle gets more cumulative weight and thus the other one is abandoned. Extending a lazy tip means not getting referenced by many other nodes and is thus bringing one's own transaction in a weak position for getting strong approval.

The security of the system is based on a few factors. First, spamming transaction should be prevented by a PoW puzzle that has to be solved. This hash puzzle is implemented in the same way as in Bitcoin [Nak08], where a nonce has to be found that the hash reaches a predefined number of leading zeros. However, it has to be highlighted that the hash puzzle of IOTA is way easier to solve, as on average only $3^8$ different nounces need to be tried. Such an easy difficulty might be necessary for IoT devices with limited computational resources to participate, but it also jeopardizes the security of the system. This becomes clearer when we look at the fact, that transactions are able to have different weights assigned to them. This weight is also dependent on the hash produced by the found nonce. An malicious actor now might either use his/her own hashing power to find

high-weight nonces or just flood the network with many blocks with advantageous references. To limit the influence of finding better nonces, a rather low cap on the maximum weight is set. This should ensure that a few heavy transactions do not outweigh a large number of other transactions. However, if we look at an adversary that controls a lot of hashing power, this measures are just not enough. Vries [dV19] shows that an adversary can just publish a double-spend transaction with high weight and mine new blocks to reference it with the goal to outpace the main tangle. At the moment it is the job of a central component called Coordinator to defend against such attacks. Each transaction must be finalized by a Coordinator to be valid completely destroying the decentralization property.

Quentin Bramas shows in his work about the stability and security of the Tangle [Bra18] that with this PoW setup, the hashing power of all honest nodes must be bigger than that of an attacker. Otherwise the attacker can just create a conflicting Sub-Tangle (e.g. in private) which will be seen as the new main Tangle. He shows, that this happens independent of the tip selection algorithm. Also the underlying network (sparse vs. fully connected) makes no real difference for an attacker. Different alternative solutions are proposed by multiple papers [Pop18, Bra18]. The first approach is to ensure that the honest hashing power is as high as possible by bringing nodes to also approve transactions, even if they have no own transactions to publish. This can be achieved by adapting the tip selection algorithm to consider how active a specific node is. Thus, to get own transactions approved fast in the future, a node must always be actively approving other transactions. Therefore, empty transactions are added to the Tangle. Another suggestions is using trusted nodes. If a trusted node confirms a transaction, it is considered final. Final transactions can be seen as checkpoints as well, protecting against adversaries who control enough hashing power to rebuild large older parts of the Tangle. This solution is currently employed by IOTA (trusted nodes are the Coordinators) effectively destroying decentralization. The last proposed solution is to consider both Sub-Tangles valid in the case of a conflict. To achieve that a new transaction type named *decider transaction* must be introduced, which states which of the two transactions is considered valid. This would also ensure, that no hashing power is wasted on transactions following two conflicting transactions, as all of them will be part of the main Tangle in the future.

For the system to work it is furthermore important, that most nodes follow the suggested tip selection algorithm which is only reasonable to assume, if there exists no selfish strategy that would benefit nodes following it. Popov et al. analyzed the tip selection algorithm on exactly that matter [PSF19]. They concluded that nodes can adapt their tip selection algorithm to reference tips that the normal algorithm chooses most likely. Successfully doing that would result in one's own transactions being in the main tangle faster and more reliable. Popov et al. show in a simulation that selfish nodes outperform others by up to 25 percent on that regard. However, their calculation, proofs and simulation is based on a series of (reasonable) assumptions. They conclude that selfish strategies also implicate a higher computational cost because predicting the backtracking of honest nodes is very costly. The idea is that this additional computation cost slows

adversaries down enough, so that honest nodes are not harmed in a meaningful way. However, the question if this is enough remains unanswered.

What we have investigated so far are the basics of IOTA. IOTA is still evolving and mechanisms are changed and new ones proposed or introduced. The newest evolutionary step is called the Coordicide [PMC+20] and focuses on removing the Coordinator to achieve the decentralization of the system. For that to work numerous other mechanisms are introduced and described in detail. The paper covers topics like node identities, Sybil protection and automated peering via a reputation system (mana). This system is then further used to limit transaction rates of nodes to prevent spamming by adapting the PoW difficulty based on their transaction rate and mana value. Also, an idea is put forward that in the future the PoW puzzles (which are currently the reason of many vulnerabilities due to their low difficulty) can be replaced with verifiable delay functions (VDFs). However, research in that area is not advanced enough to answer the question, if they can ever replace the PoW puzzles while maintaining the properties and security guarantees needed.
For replacing the coordinator itself, two possible consensus algorithms are proposed. They are similar to each other in their basic idea but differ in details. The idea is to have no orphaned Sub-Tangles in case of a conflict anymore. Both branches are considered valid and nodes only vote on conflicting transaction to determine which one should be abandoned. Nodes build their own opinion on which transaction they favour by communicating with a subset of other nodes (mostly their neighbours) about their opinion over a conflict. They run a probabilistic consensus algorithm over multiple rounds to come to the same opinion over the conflict. This result spreads over the network by setting or flipping the opinion of further nodes and may collide with another result reached somewhere else locally in the graph. Opinions compete against each other until only a single opinion survives and consensus is reached.

To sum up, IOTA is a transaction fee free, DAG based, currently centralized cryptocurrency with a lot of known flaws in terms of security. Many ideas are put forward to get rid of these flaws, but one cannot say with certainty if this attempts will succeed.

## 3.3 PoS protocols

The following subsections discusses different PoS protocols which are based on other data structures than DAGs.

### 3.3.1 Ouroboros Praos

Ouroboros Praos [DGKR17] is an independent protocol based on the Ouroboros [KRDO17] PoS protocol. Therefore, knowledge about the Ouroboros protocol is needed to reason about the properties of Ouroboros Praos.
In a nutshell the Ouroboros protocol consists of epochs. Each epoch is a certain amount of

time divided into slots. For each slot there exists a participant of the protocol who acts as slot leader. This participant publishes a new block to all other participants via broadcast within his slot time. Therefore, all participants must have access to a synchronized clock to determine which slot is the current one (20s slot length is suggested). That also means that every participant gets the new block within the duration of a slot (synchronous communication setting). Honest slot leaders always extend the longest valid chain. The schedule containing the assignment of the slots to participants is publicly available at the start of each epoch. This schedule is determined by a multi-party protocol and takes the stake of the participants into account. If you have more stake (e.g. coins) you have a higher chance to become a slot leader in the next epoch. Stake is always determined at the start of an epoch and does not change until the next epoch starts.
The authors also proof their protocol secure under the following model. An adversary can spoof, inject and reorder messages (blocks it creates). That means and adversary can produce different blocks at no cost and send them to different participants when he or she is slot leader. As everybody knows who are the slot leaders of the current epoch, an adversary can try to create a fork by revealing different blocks to different upcoming slot leaders. However, the authors proof mathematically that the probability that the chain can be forked over a length of $n$ blocks drops exponentially within at least the square root of $n$. Of course an attacker who controls the majority of the stake can break the system.

Ouroboros Praos [DGKR17] is even stronger as the protocol remains secure under the circumstances that an adversary has full control over message delays within a delay of $\Delta$ slots. $\Delta$ is unknown to the protocol but all messages of honest participants reach all other honest participants within at most these $\Delta$ slots. Security of the system degrades with the increase of $\Delta$. The adversary can fully corrupt and control any participant immediately with the restriction to still have a minority of the stake.
This is achieved by exchanging the slot leader determination of Ouroboros through a kind of lottery which can be executed by every participant locally. It is based on the cryptographic concept of verifiable random functions (VRFs). A participant puts in its key and the slot number and gets an output and a proof value. Other participants can use this proof value to verify if somebody is really the slot leader. To become slot leader, the output value of the function needs to be smaller than a certain value calculated from a function which takes the own stake value as input. That change results in empty and multi-leader slots. Additionally, there is no public slot leader schedule available anymore, so an adversary does not know who the next slot leaders are. Moreover, not everybody needs to know the new block immediately in the same slot as it was published. Therefore, the protocol setting became a semi-synchronized one, resulting in shorter slot times.
Combined with the use of key evolving signature, it allows the protocol to be secure even when an adversary can corrupt any participant immediately. The term key evolving signature means that there exists only one public key, but several private keys (one is generated for each time one is a slot leader). If this generated key is deleted before broadcasting the created block, an adversary cannot publish another block for the same

slot if he succeeds in corrupting the current slot leader.
Last, the problem of bias-resistant randomness on the blockchain must be mentioned. Ouroboros Praos needs a random nonce for ensuring a stable stake distribution throughout an epoch. If data of blocks are taken to calculate that random nonce, an adversary can try to bias the function by publishing data that fits his needs if he becomes slot leader (e.g. publish a different set of transactions). Ouroboros Praos strengthens its resilience against this threat by forcing every slot leader to publish its VRF value within his created block. Just this random values are hashed to generate a new nonce.

To conclude Ouroboros Praos is a PoS protocol which is provable secure within the bounds of a model which grants an adversary a lot of power. Thus said, new blocks can be created fast without weakening the security of the blockchain. However, there can still exist only a single valid block per slot duration determined by the longest chain rule and work on the others is wasted.

### 3.3.2 Algorand

Algorand [GHM+17] is a PoS protocol based on Byzantine agreement. The general concept includes, that one or more participants are chosen to compute and publish a block to all other participants. Then a group of other participants is chosen as a committee to come to an agreement which of the published blocks will be the next valid one appended to the chain. The participants can chose themselves for the described tasks by running a local algorithm containing VRFs to determine if they are allowed to publish a block or if they are a member of the committee. Publishers are sorted by a priority value to determine which block should be chosen by the committee. If really every published block is invalid (published by a malicious user) the committee can come to the consensus to approve an empty block as chosen one. The mechanism determining who takes part in the block proposal algorithm is rather similar to Ouroboros Praos [DGKR17]. In the paper of Ouroboros Praos the authors even state Algorand as related work which they took into account before improving their own Ouroboros protocol. As in many PoS protocols, the amount of coins you own determines your stake and improves your chance to be selected for the block proposal algorithm.
If the network is quite synchronized, the committee will need about 4 rounds to come to a consensus. As the committee is not getting bigger if more participants join the protocol, the block proposal mechanism will not slow down even if millions of participants join in. However, if an adversary is involved who controls wide areas of the network and tries to split it into different synchronized chunks by delaying messages, it can take significantly more rounds to meet consensus. It has to be noted here, that there exist two types of consensus (named *final* and *tentative* in the paper). Final consensus means there cannot be a fork that invalidates the block and is usually reached if an adversary controls only a small portion of the committee and the network is strongly synchronous. Tentative consensus means that honest participants agreed on a block, but they cannot guarantee that there exists no other block in the same slot due to malicious behaviour or the network being only weakly synchronous. This can also lead to multi-block forks.

A tentative block or multi-block fork can become final in two different ways. First, if Algorand reaches final consensus on an successor block with an overwhelming probability, the predecessors are also confirmed. If that does not happen, a recovery mechanism is started periodically based on loosely synchronized clocks of the participants. Instead of proposing a block, a participant has to propose a fork which all users decide upon.
The protocol remains secure under a model of an adversary being able to immediately corrupt any participant while not excelling an amount of stake greater than a third of all existing coins. Like in nearly all Byzantine agreement protocols, a majority of two thirds of honest participants (regarding stake) are required to keep the protocol stable. If there are more rounds necessary to meet a state of consensus, every round has its own committee. That prevents an attacker from corrupting all members of the actual committee and taking control over the protocol. Even the number of committee members between each step can vary because of participants selecting themselves by computing local functions and publishing just the proof. Each committee member has to delete its key used for signing before publishing a message to ensure that a second message cannot be forged later on. Forks are very unlikely to appear within this protocol and may only occur to the activity of an adversary. That means normally there is only a single chain existing and not a tree like in other blockchain protocols like Ouroboros (Praos) or Bitcoin.

To sum up, Algorand is a cryptocurrency protocol based on the Byzantine agreement problem and it is enough to control a third of the stake to completely crash it. On the other hand nobody has to deposit coins anywhere, enabling everybody with coins to participate in the consensus process without reducing the liquidity of anyone. The speed of reaching consensus strongly depends on the underlying network. If the network is strongly synchronous (and no adversary is interfering), consensus is usually reached withing 4 voting rounds, making the protocol very fast compared to the ones we have discussed so far. If this is not the case, fast consensus cannot be guaranteed. Algorand copes with scaling by having a small committee independent of the number of participants.

### 3.3.3 Snow White

Snow White [DPS16] is another PoS blockchain protocol which focuses more on posterior corruption which has not been handled in depth in the other presented protocols. Nevertheless, the protocol is fundamentally similar to the others we have already covered.
Snow White also consists of epochs and within an epoch of slots (they are named time steps in the paper). There exists a nonce and a committee for each epoch. The nonce is calculated based on former blocks (every time somebody publishes a new block, a random nonce is added). The members of the committee are determined by a specific elect function of the blockchain protocol itself, taking into regard the coins (stake) of each participant. Stake values are updated at the start of each epoch and Snow White limits the amount of currency that can be transferred during the epoch to protect against attacks from slot leaders. Again, the committee members can calculate for themselves if they are the leader of a specific timeslot. Leaders publish new blocks for their slot, also adding a timestamp to it. A leader always extends the longest chain but never extends a

branch if it contains future timestamps.
The whole security model is based on an attacker who cannot corrupt nodes immediately. The model says that an adversary can take over every node but needs time for it. Therefore, the protocol stays secure as long as less than 50 percent of the committee and thereby slot leaders are honest at the time of publishing blocks. That also means that you have to wait till your payment transaction is in a block followed by some more blocks to be relatively sure that there will not be a fork. That is similar to Bitcoin [Nak08] and differs to protocols like Algorand [GHM+17]. It has to be mentioned that you need significant more blocks appended to a Snow White block compared to Bitcoin. We are talking about 34 to 43 percent more blocks (depending on the controlled stake/hash power of an adversary) for the same consistency failure probability. However, this is negligible in comparison to Bitcoin because of the faster block creation rate but stays relevant in comparison to the other PoS protocols.
The biggest problem identified by the authors are adversaries trying to start forks from some place back in the chain. Therefore, some safety mechanisms got implemented. First of all, every new participant has to go through a special part of the protocol where it has to contact several other participants to ensure getting the right blockchain and not a simulated one from an adversary. The same applies to participants who sleep for a longer period. Moreover, active participants do not accept blocks with too old timestamps and therefore no new whole branches, if they reach too far back in the blockchain. Also, just histories/blocks with strict increasing timestamps are accepted. Furthermore, for the committee and nonce selection an interval of old blocks are taken which are considered stable by the protocol (this is connected to the fact that older timestamps are not accepted).

All in all, it seems that each discussed PoS protocol focuses mostly on a specific thread. Snow White differs to Ouroboros Praos in terms of the adversary model. It cannot withstand an adversary who can corrupt participants immediately but on the other hand it is much more robust against posterior corruption. One major advantage of Snow White is that it works even if major parts of the committee do not respond because there is no consensus needed as it carries out the well known idea of always extending the longest chain. Snow White also sees older parts of the main chain as a kind of finalized. However, if an adversary controls more than 50 percent of the stake/committee for a long enough time, also that part of the history can be rewritten.

### 3.3.4 Hybrid Protocols

There also exist hybrid protocols which use PoS to introduce some more functionality to existing PoW cryptocurrencies or increase the security of the system. They show that the PoS concept can be used as non stand-alone functionality as well.

### Casper the Friendly Finality Gadget

Casper the Friendly Finality Gadget [BG17] is a protocol which can be used on top of current PoW blockchains. A PoW blockchain using Casper becomes a hybrid blockchain using PoW and PoS at the same time. This also offers an idea of how to switch a blockchain from a PoW setup to a PoS setup step by step. However, that would still mean to apply one or multiple hard forks to the blockchain.
Casper is not changing the block proposal mechanism of a cryptocurrency, instead it adds more functionality. A canonical chain is formed out of the existing tree created by the proposal mechanism. This is done by finalizing blocks through a set of validators. Each validator sends out votes for links containing an already justified checkpoint and a new target checkpoint. A checkpoint is justified if at least two thirds of validators (regarding their assigned weight) have published votes with the same source and target. The link from the source to the target is then called a supermajority link. If there is another supermajority link heading from the former target to a new block, the checkpoint is considered final. For performance reasons just every hundreth block becomes a checkpoint. This still means that all participants of the protocol know now, that the branch with the finalized checkpoint is the valid chain and all other forks are invalid. That implies that honest miners have to extend the longest valid chain containing the finalized checkpoint with the greatest height. The height of a block is its number if you start counting from the genesis block whereas blocks on the same layer of the tree get the same number.
Everybody can become a validator by depositing some of their coins. The proportion of one's deposited coins of all deposits of all participants is the weight of one's votes mentioned above. There are rules a validator must follow and violations can be reported by others resulting in a finders reward and the complete loss of the cheating validator's deposit. Furthermore, withdrawing deposited coins takes time (e.g. some month) to prevent validators from avoiding punishment by money withdrawal.
There are certain rules you have to follow when being a validator. You are not allowed to vote for different blocks on the same height in different branches. Moreover, you are not allowed to vote within the span of your other votes. As you have to add your signature to every of your votes, other participants can recognize violations and trigger the punishment. Furthermore, if you withdraw your deposited coins you will not get them back immediately. You have to wait a long time (e.g. some months) to prevent validators from avoiding the punishment by withdrawing all their money.

Casper introduces the idea of finalizing parts of the blockchain which we have also seen implemented in some pure PoS protocols. An attacker controlling 51 percent of the hash power can also prevent Casper from finalizing new block but cannot rewrite older validated blocks. Moreover, an attacker controlling enough stake can finalize any blocks, so attacking the system works via stake and hashing power now. The authors themselves state: "Casper remains imperfect". [BG17]

**TwinsCoin**

TwinsCoin [CDFZ17] is a stand-alone protocol combining PoW and PoS to create a more secure system. Every via PoW published block additonally has to be confirmed by participants selected in regard of their stake, so that an adversary must control a lot of hashing power and also own a huge amount of coins to be able to cause significant damage to the protocol.
The concept and implementation of the hybrid blockchain is based on the 2-hop protocol [DFZ16]. The result is a protocol with a PoW mechanism like bitcoin but with PoW blocks having 2 different states. They can be *attempting* or *successful*. If a miner finds a new block, the block's state is *attempting*. Other miners do not try to append new blocks to it but instead create other *attempting* blocks (forks).
Every participant who holds some coins owns a verification key depending on his stake [DFZ16]. A participant who wants to publish a PoS block needs to hash this verification key together with one of the PoW blocks and get a value smaller than a specific target value to be allowed to publish a PoS block. This PoS block is linked with the PoW block which immediately changes its status to *successful* (this happens implicit). Now the mining race for the next PoW block starts again. Always the longest chain is valid and forks are ignored. The difficulty of the PoW and PoS challenge are both adapted similar to Bitcoin [Nak08].
The resulting ledger structure can be seen in Figure 3.9, with the original paper providing the following explanation:

> A modified 2-hop blockchain structure. Here, dot arrows (that link to the previous successful block and attempting blocks) denote the first hops, and solid arrows denote the second hops. Green blocks $B_j^i$'s denote the successful proof-of-work blocks, $B_j^i$'s denote the attempting proof-of-work blocks, and red blocks $\tilde{B}_i$'s denote the corresponding proof-of-stake blocks. Note that the blue blocks are from the "mature blockchain". [CDFZ17]

Interesting for our purpose here is, that the resulting data structure is no tree anymore, but instead a DAG. So, this data structure can successfully be used to achieve more security instead of higher transaction throughput as well.
TwinCoins really makes it harder for adversaries to break a protocol. However, this comes at the cost of transaction throughput as the system is even slower through the additional PoS confirmation step.

## 3.4 Attack Scenarios

This section is about different types of possible attacks on blockchain systems. We will look at how they work in a general matter. As there exist so many different blockchain systems, we have to examine if the presented attacks are applicable to our PoS DAG simulation scenario. Moreover, this list cannot consist of all possible attacks through the

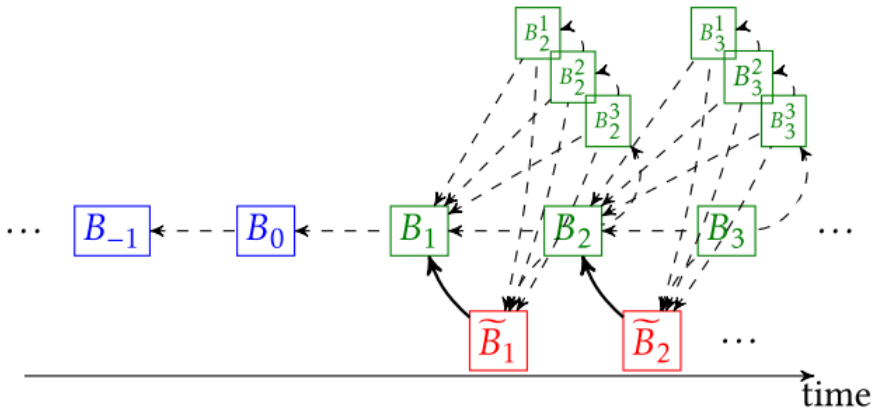


Figure 3.9: TwinsCoin ledger structure [CDFZ17]

novelty of PoS DAG systems. For each chosen candidate protocol, attacks specifically targeting the protocol must be considered. Understanding the attack scenarios in this section is crucial for being able to do that.

### 3.4.1 Double-Spend

Already the first cryptocurrency Bitcoin sees preventing a double-spend as the root problem for a working system [Nak08]. It can be best explained by the scenario of an adversary issuing a payment to buy some goods from a vendor. Once the vendor confirms the transaction and sends the goods, the adversary issues a payment to another vendor (for example on a different branch of the blockchain structure), although the funds for it are not in his/her possession anymore. That is a conflict on the blockchain, which is often resolved in PoW systems like Bitcoin or Ethereum by the longest chain rule. The adversary now needs to somehow manage that the new transaction ends up on the longest chain, effectively invalidating the old one.
A double-spend can be also seen as the goal of another attack instead of being an attack on its own. This attack type might also not be feasible, for example if a cryptocurrency guarantees final consensus for transactions, as Swirlds-Consensus [Bai16b] does.

### 3.4.2 Holding the majority

Every cryptocurrency we have looked upon so far works only if a specific proportion of the involved nodes acts honest. In the PoW protocols, 51 percent of the hashing power needs to be controlled by them and depending on the PoS system less than a third or less than 50 percent of staked coins must be controlled by an adversary. Otherwise the adversary controls the network. Depending on the protocol, this enables an adversary to perform unlimited double-spends, rewrite large parts of the history or halt the progress of the

cryptocurrency completely. Cryptocurrencies thus usually incentivize honest behaviour and/or punish malicious nodes.

### 3.4.3 Distributed Denial of Service

A *Distributed Denial of Service* attack occurs if a component gets flooded by messages from other hosts till it breaks down. Such an attack against a cryptocurrency can be effective if there exist a small set of nodes which is more important for the stability of the system [CELR17, RCa+20, Bai16b]. One of the simplest examples is that bringing down the leaders of mining pools destabilizes PoW currencies. And bringing down slot leaders in PoS systems also jeopardizes the system. The best defence is building a protocol in a way, that the stability of the system is not dependent on single nodes. PoS systems can also protect themselves by not publishing slot leaders upfront, but rather include prove mechanisms for them to show that they are indeed leaders [GHM+17, DGKR17, DPS16].

### 3.4.4 Nothing at stake

In PoS systems nodes usually get rewards based on their staked coins and their participation in the consensus algorithm. Now, think of a situation where two conflicting paths in the ledger exist. If the rewards of a node are coupled to approving transactions, a rational node has to approve both paths to be sure to get its rewards. The node can only win by following this approach, as it has nothing to loose (*nothing-at-stake*) [RCa+20, GHM+17, DPS16]. This attack can be countered by punishing misbehaving nodes, e.g. by burning their staked coins. The question why nodes should behave honest is valid in every cryptocurrency. Thus this attack must be considered for every PoS cryptocurrency, as there issuing blocks is possible nearly effortless.

### 3.4.5 Long-range attack

Another similar attack category on PoS protocols is the long-range attack [RCa+20, DPP19, LABK17, GKR18, DPS16], also known under the name *(alternative) history attack*. The general idea is to create an alternative chain containing blocks favouring oneself. An adversary can benefit by performing double-spends or creating the alternative chain in a way, which results in more fees payed to the adversary.
A very simple attack is to create an alternative branch and try to outpace the main branch. As a node's influence on block approval and creation depends on its stake, one cannot simply outpace the main chain. However, if no timestamp checks are made, an adversary can just create blocks ahead of time to create a longer chain accepted by other nodes by forging timestamps.
Another possibility (called *posterior corruption*) is to just start a fork somewhere back in the current main chain. Therefore, an adversary must have access to the private keys of nodes which together possessed enough stake at the moment when the fork starts. Nodes that possessed significant amount of stake in the past but none anymore have nothing to loose by joining such an attack. Also, users might not keep up enough security measures

to protect their private keys, when no coins are associated to their account anymore. An adversary can pick any point in time from the publication of the genesis block until now and try to control enough stake for that specific point in time. From there on, all other blocks can be rewritten easily, as generating blocks is cheap in PoS systems (this fact is known as *costless simulation*).
Another case is called *stake bleeding*. An adversary secretly prepares a fork and while doing that slows the main chain down as much as possible using his/her stake. On the secret fork, transactions from the main chain are included in a way that the stake shifts to the adversary through transaction fees. The secret fork is published once the adversary gained enough stake to outpace the main chain. This attack is very slow (we are talking about multiple years here), but can be combined with changing parts of the history.

Different countermeasures are proposed or already exist to defend against such attacks. A basic approach is to burn staked coin if misbehaviour is proven. However, this only increases the risk of an adversary in the case an attack fails. The introduction of finalization is also very common. Protocols either do not allow conflicting transactions once a block is considered final or introduce checkpoints which ensure that appearing blocks with a height lower than the checkpoint are considered invalid. Another measure are key-evolving mechanisms where for the same public key, multiple private keys are used and afterwards deleted. This ensures that alternative blocks cannot be created later on by somebody who corrupted a node and has access to its private key. A defence specifically created to counter stake bleeding, are context-aware transactions. Here, transactions reference previously mined blocks, effectively preventing them from being copied to another branch and being still considered valid. It is also possible to determine the validity of a branch based on statistics. For example, branches of attackers can be recognized by looking at the density of blocks or the amount of blocks published compared to the respective stake of the participants. Also, some radical solutions were proposed, like forcing participants of the consensus process to provide their real identity, which is automatically revealed if malicious behaviour can be proven. Another suggestion is to force participants of the consensus process to use special hardware, so called Trusted Execution Environments (TTEs) to enforce security.

### 3.4.6 Joining the network

*Weak subjectivity* is a name for the problem which arises when new nodes or nodes that have been offline for a while (re-)join the network [DPP19, DPS16]. These nodes need to get the current status from other nodes in the network. In that phase, malicious nodes can just present an alternative chain to them, because these new nodes have not enough information to differentiate between the real main chain and an alternative one. Snow White [DPS16] states that an additional trust assumption must be made to counter that. It assumes that a list of active nodes exist which can be queried by newly joining nodes. Then, the new node just asks the nodes on the list to vote if they agree on the state it received. Assuming the majority of them is honest, the new node cannot be tricked into accepting a forged chain. Another approach is presented by Coordicide [PMC+20]. Here,

peer selection happens provable randomly and every node connects to multiple others. Exploiting this situation can be considered an attack on the underlying network, as an adversary needs to make sure that newly joining nodes establish connections with him/her. However, simulating network attacks is out of scope for this work.

### 3.4.7 Eclipse attack

Another network layer vulnerability is the *Eclipse attack*, which was mainly studied for PoW blockchains like Bitcoin or Ethereum [HKZG15, CELR17, MHG18]. In a nutshell, the underlying peer-to-peer network mechanisms of the respective protocol are used to make a node aware of as many nodes as possible of a node set the adversary controls. Then, once the targeted node restarts (or is forced to restart) it connects to adversary controlled nodes with a high probability (exact number depending on the scale of the attack). The adversary then has successfully split the node from the network and can launch follow up attacks, like letting the node mine on a fork or not forwarding published transactions. This attack is also possible for non PoW protocols. Possible countermeasures minimizing the risk are adapting the mechanisms of how clients overwrite and choose their known peers locally, reducing the attack window after reboots, disabling of incoming connections, whitelisting, anomaly detection or anchoring connections so that they are not erased on a restart.
For our purpose it is important to consider if an Eclipse attack is possible and if the answer is yes, what effects it has on the whole protocol if an adversary can control what messages single nodes receive.

### 3.4.8 Selfish mining

*Selfish mining* is another way for an adversary to increase gains on the blockchain [ES13, RCa+20, DPP19, CELR17]. Selfish mining can have different forms with the simplest one being to hide found blocks as long as possible and privately mine appending blocks in a PoW system. Once the danger of loosing rewards for the privately mined blocks becomes imminent, the blocks are published. In a more general way, selfish mining just means a deviation from the protocol to increase one's own gains. That can also happen in PoS protocols as it mainly depends on the incentive structure of the protocol. Once such a vulnerability in a protocol is found, it can usually only be fixed by changing or extending the mechanisms of the protocol itself.

### 3.4.9 Grinding attack

*Grinding attacks* are possible whenever a source of randomness is needed. In several PoS protocols it is the case that some source of randomness is needed to e.g. determine slot leaders [KRDO17, DGKR17, DPS16]. If this randomness is taken from the blockchain itself, an adversary can bias it in a way that benefits herself/himself when publishing blocks. Snow White [DPS16] counters that by using randomness from an older block interval where nothing is known about future block leaders. Another tactic is to use

hashes produced by VRFs when determining block leaders as source of randomness. This decreases the possibility to inflict bias.

## 3.5 Simulators

DAGsim [ZWH18] is a simulator for DAG protocols focusing mainly on performance and scalability. It does not support malicious nodes and only provides a working implementation for IOTA and an incomplete version of SPECTRE. The source code is publicly accessible on GitHub [6].
SimBlock [AOK+19] focuses on the network side of a blockchain. It is a general simulator for blockchains using the PoW system and not designed in a way to simulate a specific implementation. Network metrics like block propagation time can be examined under different configuration settings. The source code is availabe on GitHub [7].
There exist two similar frameworks called BlockSim [AvM20, FC19] which are based on models for different blockchain layers. One needs to programmatically define the models for each layer. For example, it has to be defined how the network behaves, which incentive structure exists and how consensus is reached. Then simulations are executed based on these models. Both frameworks focus on PoW blockchains and there exist models for Bitcoin and Ethereum only.

To the best of our knowledge, there exist no simulators which can be used or easily extended to evaluate attacks on (PoS based) DAG systems.

[6] DAGsim on GitHub, accessed 14.9.2020: `https://github.com/IC3RE/DAGsim`
[7] SimBlock on GitHub, accessed 14.9.2020: `https://github.com/dsg-titech/simblock`

# CHAPTER 4

# Simulator

In the course of this thesis, a simulator for Hashgraph (one of the most promising PoS DAG protocols) was created. Hashgraph was chosen because of a multitude of reasons. First of all, it defines its security under a very powerful asynchronous adversarial model. Moreover, to the best of our knowledge no major security issues are known yet. A test network of an actual implementation already exists, as well as a plan of how to convert that into a fully functional cryptocurrency. It is furthermore backed by various influential companies worldwide. The other candidates on the other hand state their security claims under weaker adversarial models or a series of relevant security flaws are already known.

This chapter goes into detail on how the developed simulator is structured and what functionality it offers. We will also discuss what assumptions have been made in cases, where the underlying consensus protocol does not provide enough information necessary for simulating real world attack scenarios.

The source code of the simulator, developed during the work on this thesis, is published on Github[1]. Feel free to extend it and use it for your simulations.

## 4.1 Simulation modes

Our simulator offers a wide range of different configuration possibilities for every simulation run via parameters. Additionally, the simulator supports two different simulation modes for all attack scenarios. Some configuration parameters apply to every mode and some only affect a specific one. Furthermore, the modes greatly affect how many actions every participant performs and in which order they are executed as well. The simulator has its own internal clock and time progresses differently between these modes. One

[1] source code of the simulator, published under MIT license on Github: `https://github.com/BSchachenhofer/dagsim`

must therefore first understand the difference between these two modes before having a closer look at the implementational details of the consensus mechanism and configuration parameters of the simulator.

The first mode is called *Equal Sync-Times*. Here, every participant performs a single action before its the next participant's turn. The turn order is permuted randomly after every round. Once every participant performed an action, the time of the simulator progresses by a configurable value resulting in every action taking the same amount of time. This mode enables perfect control about the maximum actions of participants and limits what action sequences are possible (e.g. honest participants cannot act multiple times in a row). This enables better evaluation for attack scenarios where certain measurable conditions need to be enforced, like an adversary being able to perform an action multiple times compared to honest nodes.

The *Random Sync-Times* mode is the other option offered by the simulator. Here, every action takes a random amount of time based on the exponential distribution around configurable mean values. Every participant plans one action ahead and the action's finish time is calculated. Then, whatever action has the lowest finish time is executed first. The simulator's internal clock is set to the finish time of the action and the participant plans its next action. The action patterns of this mode are more unpredictable. For example, it can happen that one participant performs five smaller gossip-syncs in a row before any other participant acts, provided that their actions take longer. The advantage of this mode is that it represents real world conditions better and is thus more suitable for performance comparisons between different scenarios (e.g. looking at confirmation times).

## 4.2 Structure of the Simulator

The simulator itself is configurable via a graphical user interface which can be examined in Figure 4.1. One can alter general simulation settings at the top of the window as well as mode specific configurations of the current sync mode. Modes can be changed by ticking the respective radio box on the top left of Figure 4.1. The following general parameters can be configured independently of the chosen mode:

- **Participants** Defines the number of nodes present in the simulation. Every node takes part in the consensus algorithm.
- **Seed** Many actions performed during a simulation scenario need some sort of randomness (like execution order permutation or choosing an action). To ensure reproducibility of simulation runs, every random action is based on the configured seed. This determinism is not only important for being able to verify the results by reproducing them, but also for other purposes like debugging. It is hardly possible to locate and fix implementational bugs, if the simulator yields different results

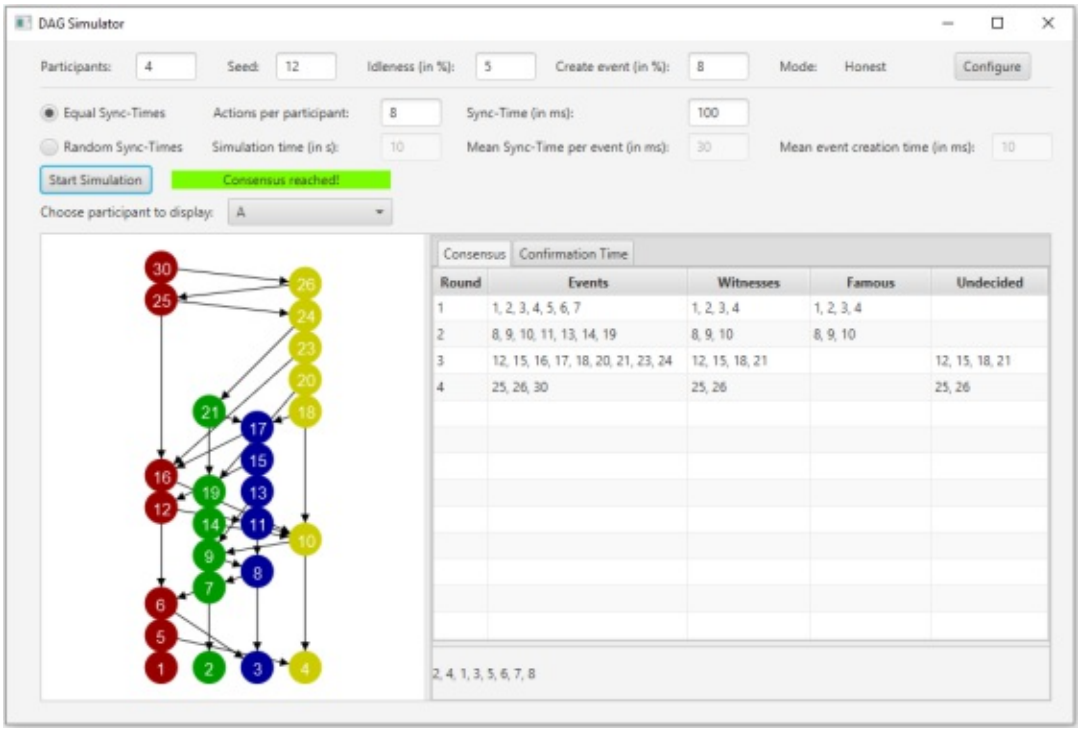


Figure 4.1: User-Interface of the simulator

for the same input parameters. Having a deterministic simulator made it possible that bugs can be found, reproduced and fixed. Any positive integer number can be chosen as seed.

- **Idleness and Create event probabilities** Whenever it is a participant's turn to act, one of three actions is performed. A participant can either perform a gossip-sync, create an event or remain idle. Actions are chosen randomly with a certain probability. These probability values can be configured. One can set an integer value defining the probability of idleness and creating an event. The remaining difference to one hundred percent gets assigned to the gossip-sync action. Be aware that probabilities are checked in the following order: idleness, create event, gossip-sync. If one enters a value greater than a hundred in the idleness text field, all actions the simulator will perform will be idleness actions.

  Performing a gossip-sync is the default action a participant performs most of the time according to Swirlds-Consensus [Bai16b]. Thus, the probability of choosing that action should be high. A participant picking this action chooses another participant and syncs all known events to the other participant, creating a new event at the receivers end during this process.

Creating a new event locally enables a participant to incorporate transactions into the hashgraph and communicate them to other participants when initiating its next gossip-sync. Transactions can always be incorporated into the event created when receiving a gossip-sync. This action makes sense when a node has not received such gossip-syncs in a while, but wants to communicate new transactions. Therefore, it makes sense to set the probability level for this action to a lower value.

The idleness action represent a node that involuntarily does not participate in the protocol for a short period of time. Reasons for that can be manifold and range from simple network failures to gossip-syncs that were discarded by the recipient. That such things happen should not be the norm and if they do they sometimes justify creating a new event locally (e.g. in case of a network failure). Therefore, it is appropriate to assign the lowest probability value to this type of action.

In addition to these general parameters, one must also specify some mode dependent parameters. The necessary settings for the *Equal Sync-Times* mode can be examined below.

- **Actions per participant** This parameter sets the duration of the simulation run by defining how many actions each participant performs. Every participant performs one of the three actions (gossip-sync, create event or idleness) before the procedure is repeated with permuted execution order.
- **Sync-Time (in ms)** This value defines the duration of performing one action (like a gossip-sync). The internal clock of the simulator starts at this value and is incremented after each round of performed actions. It has to be noted here, that every node deviates from this clock randomly by up to plus or minus a third of the defined value. The idea behind this mechanic is to bring the simulator more into line with real world conditions, where clocks might not be exactly synchronous. Moreover, this is also a necessary step to ensure different timestamps on events within the same round of actions. In general, if any final consensus timestamp values are still the same, the simulator orders events based on their event number, as no signatures for resolving ties exist.

Also, the *Random Sync-Times* mode operates based on some user provided parameters. The following values must be set:

- **Simulation time (in s)** Sets the internal simulation time. Participants perform actions (and therefore progress the internal simulation clock) until an action exceeds this time limit. This will be the last action that is executed and the simulator stops and shows the results then.
- **Mean event creation time (in ms)** One can influence the duration of the action where an event is created locally via this parameter. The duration is acquired by

picking a random value from an exponential distribution with this parameter as mean value. It also influences gossip-sync durations because of the gossip-sync receive event creation.

- **Mean Sync-Time per event (in ms)** We discussed that participants can perform one out of three actions. This parameter influences the duration of two of them. First, it influences the duration of gossip-syncs. For every gossip-sync a random value of an exponential distribution is calculated. This parameter represents the mean value of this exponential distribution. The resulting duration is then multiplied with the number of events that the receiver did not know before this sync. This ensures that syncing times scale with the transmitted amount of data, just as in the real world. An event creation time is calculated next and added (representing the creation of the gossip-sync receive event), resulting in the total sync time.

  Second, the idleness action is also influenced by this value. We discussed before that idleness can occur when a gossip-sync gets abandoned by the receiver. However, there might exist other reasons for idleness as well. Thus, the simulator calculates idleness the same way as it calculates the gossip-sync durations, but instead of using the unknown node count as factor, a random number from an exponential distribution with a constant mean value is taken. Also, no event creation is included into the calculation of the final duration value. This calculation changes reflect that idleness represents not only failed gossip-syncs, but also other events like e.g. network connection problems. Nevertheless, using the mean sync time of an event as a factor in the calculation ensures that on average the values do not deviate in an unrealistic way (e.g. it prevents extreme cases like the average gossip-sync duration for ten nodes being one hundred milliseconds but the average idleness time being five seconds). To prevent such an unrealistic deviation, one must also choose the right constant mean value for the node count factor. It makes sense to choose a smaller double-digit value, to keep the probability of extreme cases low. For example, we set this value to fifteen for our simulations. If one argues that this mean count value limits the idleness duration too much in any direction, one can still change the idleness action probability to increase or decrease the overall idleness time of a simulation run.

With these values defined, a simulation run can be started. Once such a run is completed, one can immediately see if the underlying consensus protocol was broken by looking at the coloured label next to the *Configure* button. This automatic check compares the internal graph representations of all participants and reports inconsistencies. It does not only focus on final consensus timestamps, but also on inconsistencies of not yet decided events. For example, it also considers different rounds assigned to the same event as breaking, as such states can lead to different final consensus timestamps once more events are added to the data structure. More detailed results are displayed in the area below that label.

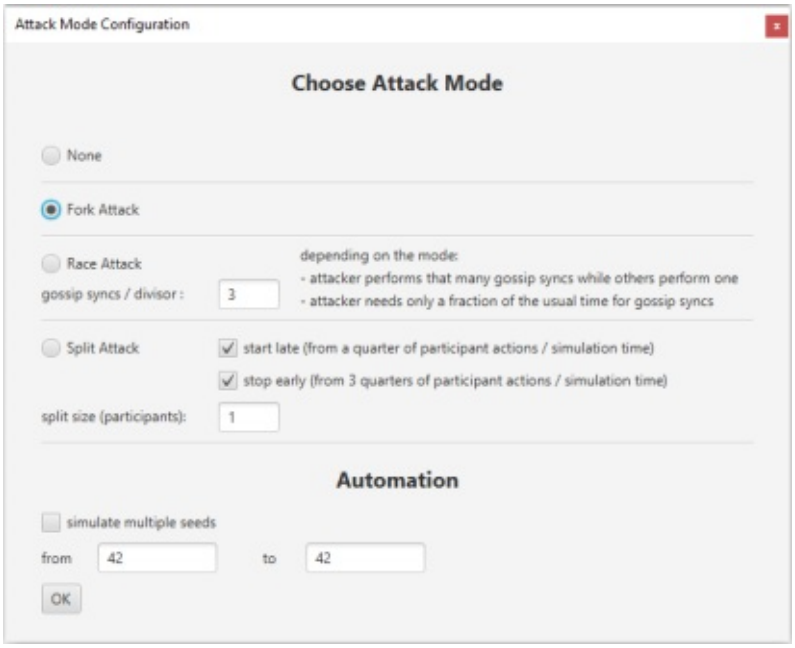


Figure 4.2: Simulator attack scenario configuration

On the left side of Figure 4.1, a graphical representation of the DAG structure can be examined. By clicking on any point of the graph view with the mouse, one centers the view on this point. Zooming in and out using the mouse wheel is possible as well. It has to be noted here, that this is mainly useful for low participant and action numbers. In the right part of the user interface, information about the consensus algorithm is displayed in a table. One can see to which round every event belongs to, if it is a witness or even a famous one, as well as the witnesses for which famousness is not yet decided. Exactly below that table, a final consensus order for all decided events is displayed. If one is interested in the duration between event creation and reaching final consensus, one can examine these values by clicking on the *Confirmation Time* tab of the consensus result table. This tab also contains the round received and final consensus timestamp values for all finalized events. All results are displayed from the point of view of a specific node. This node can be changed using the dropdown above the result area.

The simulator supports four different scenarios, which can be chosen and customized by clicking on the *Configure* button. The configuration options can be seen in Figure 4.2. The following attack modes are available:

- **None** Every participant acts honestly and follows the consensus protocol.
- **Fork Attack** Participant A creates a fork after everybody performed two actions (Equal Sync-Times mode) or one second of the total simulation time passed (Random Sync-Times mode). From that point on, the malicious participant communicates

only the left side of the fork to the first half of other participants and only the right side to the other half on performing a gossip-sync. Also, on incoming gossip-syncs only the respective side that would usually be transmitted to the sender is extended. The malicious node is excluded from the automatic consistency check.

- **Race Attack** In this mode, participant A behaves differently and performs only gossip-syncs instead of random actions, once the attack was triggered. This scenario simulates an adversary observing a specific transaction in the last event of participant B (like a transaction closing a lucrative stock trade) and issuing the same one with the goal to get it incorporated into the final ledger before the original transaction. Both of these transactions are highlighted in the graph view by their shape being a square instead of a circle. Once the adversary issues the copied transaction, it does not accept any gossip-syncs from participant B to slow down the spread of the original event. In the Equal Sync-Times mode the attack gets triggered after 3 action rounds passed and participant A starts syncing to as many other participants during one time period as configured in the text field. Sync targets are not chosen randomly. Instead, others are contacted in an ordered manner based on their name. The attack starts after one second in the Random Sync-Times mode. Here, the duration acquired for every gossip-sync action is reduced by dividing it with the provided value in the text field. It has to be noted here, that gossip-syncs not providing the receiver with any new information are never performed by the simulator independent of the scenario, as they bring no benefit to anybody (neither honest nodes nor the attacker).
- **Split Attack** No participant acts malicious, as this attack simulates the impact of attacks on the underlying network. The network is split into two parts and gossip-syncs are only possible between nodes within the same part. The amount of participants per part can be configured via the total participant number and the split size text field. There are also checkboxes which determine if the network split occurs during the whole simulation run or if it starts and ends earlier.

What you can also see at the bottom of Figure 4.2 is the configuration for the automation mode. Here, a range of seeds can be defined and a simulation run is performed for every seed value without displaying any results in the user interface. Instead, the configuration and the information if consensus was reached or if the protocol broke is written to a text file. Also, both result tables (including scenario related extra information like event numbers of race victims and malicious events) are saved as text files for every participant in the CSV format with semicolons as delimiters.

## 4.3 Technical Details

The source code of the simulator is written in Kotlin, using Maven as dependency management tool. The most important libraries are graphstream for handling and visualizing the DAG data structures and JavaFX for building the frontend.

The simulator was programmed using Java SDK 12.0.2 and Apache Maven 3.6.2, which are necessary to compile and run it. All dependency versions can be looked up in the pom file of the published source code.
The simulator supports logging of very detailed information during simulation runs. By default, logging is configured only for warning or error messages. This can be extended to debug and info messages by setting the respective value in the *log4j.properties* file in the *resources* directory of the source code. Be aware that the performance of the simulator can decrease when changing this setting.

## 4.4 Implementation of the Consensus Algorithm

In Chapter 3 we have stated, that Hashgraph consists of the Hedera Hashgraph cryptocurrency [BHM19] and the underlying Swirlds-Consensus algorithm [Bai16b]. The simulator is an implementation of Swirlds-Consensus, directly following the description and pseudo code provided in the paper. As the simulator is about specific attack scenarios applicable in the real world, it was sometimes necessary to take parts of the whitepaper into account. Additionally, own assumptions had to be made in certain other cases. A general overview of all assumptions and the resulting implementational details is given below. Serious implications on specific simulation scenarios are discussed directly in the respective evalutation part in Chapter 5.

### 4.4.1 Gossip-Syncs

Swirlds-Consensus presents a simple gossip-sync mechanism, where one participant tells another one everything it knows. The receiver records that by creating a new event and a directed edge to its own and the senders last event. The paper states that optimizations or deviations are possible (e.g. syncing back immediately or syncing with multiple participants at the same time) but explicitly states that the simple version is sufficient. Therefore, the simulator sticks to this version.

### 4.4.2 Number of participants

Swirlds-Consensus assumes that the current number of participants is known by every node which is taking part in the consensus algorithm. Hedera Hashgraph suggests an address book in form of signed state proofs to achieve this. The simulator just assumes that such a secure mechanism exists. At simulation start, all participants know the total number of other participants. This number never changes during a single simulation run, as Swirlds-Consensus provides no mechanism of how to add or remove participants.

When talking about Hedera Hashgraph, we are talking about a PoS protocol. However, there exists no variable stake in Swirlds-Consensus, there is only a list of necessary modifications for supporting variable stake available. Thus, there is no stake in our simulator. Every participant is considered equal and possesses the same stake. No new participants can emerge, which also eradicates the need for recalculations of event states

(like round numbers or (famous) witness assignments). If an event gets a value assigned during the course of a simulation run, the value can be consider final immediately.

### 4.4.3 Coin Rounds

The frequency of coin rounds for voting must be decided in any implementation of Swirlds-Consensus and the paper itself does only offer an example value of 10 in the functional form of the consensus algorithm in the appendix. No arguments about suitable values are brought forward, so this value of 10 was also used for the simulator.

### 4.4.4 Simplifications

Swirlds-Consensus presupposes that there exist cryptographical hash functions as well as secure digital signatures. They must be used to prevent any adversary from modifying messages undetected or pretending to be another participant. The simulator itself does not use any hashing or digital signature algorithms, but acts in a way as if these mechanisms are implemented. So, no messages are tampered or sent in the name of another participant.

Additionaly, the simulator does not represent individual transactions. The smallest data entity is an event and this is sufficient, as an order over all events implies an order over single transactions anyway.

### 4.4.5 Performance

Swirlds-Consensus makes no assumptions about the speed gossip spreads or how fast progress in terms of finalized events is made. However, the Hedera Hashgraph whitepaper does state the following:

> Initially, we anticipate that the Hedera network will be able to process 10,000 cryptocurrency transactions per second. Consensus latency is measured in seconds, not minutes, hours, or days. [BHM19]

Therefore, the simulator supports the measurement of confirmation times over all execution scenarios and modes based on individually configurable parameters. It has its own simulation clock, which is incremented depending on the chosen mode.

However, the simulation runs are not optimized in terms of performance. Thus, execution times vary greatly from seconds to minutes or hours, depending on the configured participant numbers and simulation time or action per participant parameter. The consensus algorithm always operates on the full DAG structure and does not throw away the events of already decided rounds, as suggested by the Hedera Hashgraph whitepaper. Therefore, during each gossip-sync the whole data structure is handed over from one to another participants. Although Swirlds-Consensus presents ideas of what to transmit during gossip-syncs to reduce the required bandwith as much as possible, these optimizations are not necessary as there is no real network communication.

### 4.4.6 Forks

Swirlds-Consensus states that a participant can misbehave by creating a new event, which does not reference one's own last event as parent, but one that already has another child. This leads to a fork and the two branches that result from it can now be independently communicated to other participants. This can lead to inconsistencies in the view of other participants onto the total DAG structure. As no votes are sent over the network and every node can provide a proof for all finalized events themselves, it must be ensured that the different views do not result in confirmation of events of both branches. Otherwise double spends are possible. Swirlds-Consensus mitigates that possibility through the following definitions:

> **Definition 5.6.** An event $x$ can see event $y$ if $y$ is an ancestor of $x$, and the ancestors of $x$ do not include a fork by the creator of $y$.
> **Definition 5.7.** An event $x$ can strongly see event $y$ if $x$ can see $y$ and there is a set $S$ of events by more than 2/3 of the members such that $x$ can see every event in $S$, and every event in S can see $y$. [Bai16b]

Swirlds-Consensus does not define any further consequences for participants which misbehave. Participants which caused a fork can still continue performing gossip-syncs and due to cryptographic hash functions and signatures, their gossip-syncs can still be valuable. Figure 4.3 shows such a case, where participant B learns about event 1 via attacker M during the gossip-sync that creates event 7 although M created a fork through events 4 and 5. Thus, the simulator displays all received events for every participant no matter if they represent a fork or not. These events are also included in future gossip-syncs. On receiving events via gossip-sync, the simulator checks if a fork occurred. From

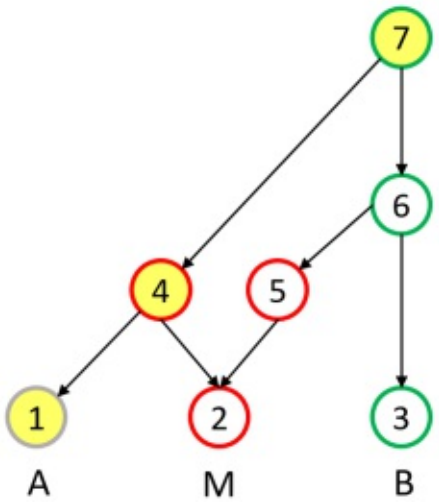


Figure 4.3: Example of a forking participant's gossip-syncs being still valuable

that moment on, all events which have both fork roots in their ancestor set cannot see

them and all their successor events in regard to the consensus algorithm. This influences event confirmation, as final timestamps are only assigned after all famous witnesses of a round have an event in their ancestor set. To decide if witnesses of a round are famous, the concept of strongly seeing is used which also implies seeing as definition 5.7 shows us. The node producing the fork still follows these definition in the simulator. However, in theory it could also completely deviate and thus its consensus and confirmation time data in the respective tables can be seen as irrelevant.

# CHAPTER 5

# Discussion

This chapter covers the evaluation and discussion of the simulation results. The simulator itself offers a wide range of possible settings for every simulation scenario and operational mode. However, to make the impact of attacks comparable even across different attacks, we define the following set of base configurations first.

**Equal Sync-Times:**

- **4-EST:** 4 participants; Actions per participant: 30; Sync-Time: 100ms; Idleness: 5%; Create event: 8%
- **10-EST:** 10 participants; Actions per participant: 30; Sync-Time: 100ms; Idleness: 5%; Create event: 8%
- **20-EST:** 20 participants; Actions per participant: 30; Sync-Time: 100ms; Idleness: 5%; Create event: 8%

**Random Sync-Times mode:**

- **4-RST:** 4 participants; Simulation time: 10s; Mean Sync-Time per event: 30ms; Mean event creation time: 10ms; Idleness: 5%; Create event: 8%
- **10-RST:** 10 participants; Simulation time: 20s; Mean Sync-Time per event: 30ms; Mean event creation time: 10ms; Idleness: 5%; Create event: 8%
- **20-RST:** 20 participants; Simulation time: 40s; Mean Sync-Time per event: 30ms; Mean event creation time: 10ms; Idleness: 5%; Create event: 8%

Each configuration has its own short name for direct reference in the following sections. The short name indicates the *Sync-Times* mode and how many participants are involved. The participant numbers were chosen to be 4 (to represent the minimal number of participants necessary, so that the system can proceed if one participant acts malicious), 10 and 20 (to see how malicious actions impact the system if the proportion of participants acting honest gets bigger and bigger). Keep in mind that Swirlds-Consensus [Bai16b] considers each participant equal in terms of consensus voting weight (like everybody possessing the same amount of stake) and that more than two thirds of the participants must be honest for the system to work properly. In terms of action probabilities, 5% were chosen for idleness and 8% for event creation for all configurations. Optimally, events should not be created too often, as there should be enough incoming gossip-syncs which include event creation anyway. Therefore, values below 10% make sense, which includes the chosen value of 8%. Idleness should occur even less often, as idleness can sometimes be seen as a reason for creating an event locally (e.g. in case of a network outage one might want to immediately communicate own transactions before waiting for incoming gossip-syncs). Therefore, 5% was chosen which together with the idleness probability results in gossip-syncs being chosen as an action with a probability of 87%.

The parameters *Actions per participant* and *Simulation time* influence the amount of created events during a simulation and were chosen high enough, that events get actually confirmed. Otherwise, consensus cannot be broken as it was not reached. As you can see, the number of actions remains stable for the *Equal Sync-Times* setting as it was chosen high enough to lead to confirmed events. It has to be noted here, that in this mode performing a gossip-sync takes the same time no matter how many events are synced. However, the simulation time of the *Random Sync-Times* setting increases with higher participant numbers. This is necessary as the total number of actions in this mode depends on the duration of each action. More participants means more events which leads to bigger gossip-syncs (the most time consuming operation) resulting in less actions overall. By increasing the simulation time one can ensure that events get actually confirmed. The next parameters are the respective action times. For the *Equal-Sync-Times* mode the given duration only affects total confirmation times. 100ms were chosen because of simplicity reasons (every participant can perform up to 10 actions per second). The *Mean Sync-Time per event* value for the *Random Sync-Times* mode was configured at 30ms to prevent the sync-time difference between gossip-syncs from getting too big. For example, a mean value of 100ms for the random value of the exponential distribution would lead to bigger differences in values one can expect, especially as the value gets multiplied with the amount of events that are synced. The *Mean event creation time* value was set to 10ms to fullfil the assumption that a local action (creating an event with signatures) takes less time than sending the same event over the network to another participant.

All of these configurations were executed for the seeds 1 to 1000 for every attack scenario. This results in thousands of simulation runs for every attack scenario, which provides a good data basis for evaluations. Whenever additional or divergent simulations were

performed for a specific attack scenario it is explicitly mentioned. The simulated results of the attack scenarios (like confirmation times) are often compared to the honest case. The following sections often compare aggregated confirmation time values to show the impact of an attack scenario on the overall performance. These values are always acquired in the same way. First, a mean confirmation time is calculated over all existing events independently for every participant. These values differ between participants, depending on the structure of the underlying DAG of the simulation run. This is done for every seed, leaving us with a thousand confirmation times per participant. Again, the mean value of these thousand confirmation times is calculated for every participant. Now, we can set these values in comparison to assess what impacts one can expect.

## 5.1 Fork attack

A fork attack in our context means, that an adversary creates two blocks referencing the same parent block instead of one. From that moment on till the end of the attack, the adversary communicates one block (and any successor blocks) to a subset of all participants. The other block (and any successor blocks) is only communicated to the remaining participants. There is no interference in the communication between honest participants.

For that behaviour to be an actual attack on a protocol, it must be the case that participants are only allowed to create one block in a time period or per parent block. In classical PoW chains, this is not necessarily an attack as the same can happen as a result of the stochastic process.

The goal of this attack is to trick different participants into accepting conflicting transactions, so that a successful double spend can be performed. However, if this is not possible by protocol design another goal of this attack can be to break the protocol completely, impair other participants or slow progress down.

### 5.1.1 Attacking Hashgraph

Swirlds-Consensus describes how a malicious user can create a fork and defines how the protocol works under this scenario. However, it does not provide any details on what this means for the malicious as well as the honest participants. We can gain some insight in that topic when simulating multiple fork attack scenarios.

On choosing the fork attack, participant A acts malicious and creates a fork after every participant performed two actions or one second of total simulation time passed (dependent on the configured mode). Participant A still acts randomly, but its actions differ. When transmitting a gossip-sync to the first half (rounded up) of other participants, only the left side of the fork is communicated. The event created on receiving a gossip-sync from these participants is also only attached to the left branch. The same

goes for the other participants and the right branch. When the malicious participant creates an event, one is created on both branches.

One can now look at what happens to events of the forking participant as well as the honest ones. Then, answers for the following questions can be derived: How are events of different branches finalized and what does excluding them from the seeing mechanic of the consensus algorithm imply? Does this observed behaviour have any impact on posterior corruption scenarios, where adversaries take over private keys of former participants? Are there any impacts on other participants recognizable (e.g. Do confirmation time values change? Does the consensus mechanism break at any point?)

### 5.1.2 Evaluation

First, look if the consensus mechanism of the protocol can be broken using a fork attack. Therefore, each of the default configuration was simulated under the fork attack mode a thousand times with seed values from 1 to 1000.

The consensus mechanism did not break in any of these 6000 simulation runs and the protocol remained stable. On investigating some of them in more detail, one can observe that the forking participant usually has two witnesses per round in our simulation scenarios. An event becomes a witness if it got a higher round number assigned than its self parent (that happens if it strongly sees more than two thirds of the current round's witnesses). As events on the two fork branches have two different self parents, having two witnesses can happen (and does happen regularly if both branches are equally active). It has to be noted here, that the following definition of Swirlds-Consensus is thus misleading:

> **Definition 5.8.** A witness is the first event created by a member in a round. [Bai16b]

It is possible that there are two "first" events in case of a participant forking. However, that has no impact on the correctness of the protocol as the consensus mechanism specifically depends on *unique famous witnesses*, meaning a famous witnesses for which no other witness exists in that round. This is also explicitly stated in the respective definitions and the pseudo code in Swirlds-Consensus. As discussed in Chapter 4, once an event has a fork in its ancestor set, it cannot see the forking events and their self-children in regard to the consensus mechanism. This results in none of the forking participant's witnesses becoming famous and thus banning it from the consensus mechanism. This can be examined starting with round three in the consensus table of the simulation run with four participants depicted in Figure 5.1. Nevertheless, consensus can be reached as the other three participants act honest. The coins of the malicious participant are not lost and all transactions it issues are still incorporated into the ledger the same ways as before. It just does not participate in the consensus mechanism and thus gains no share

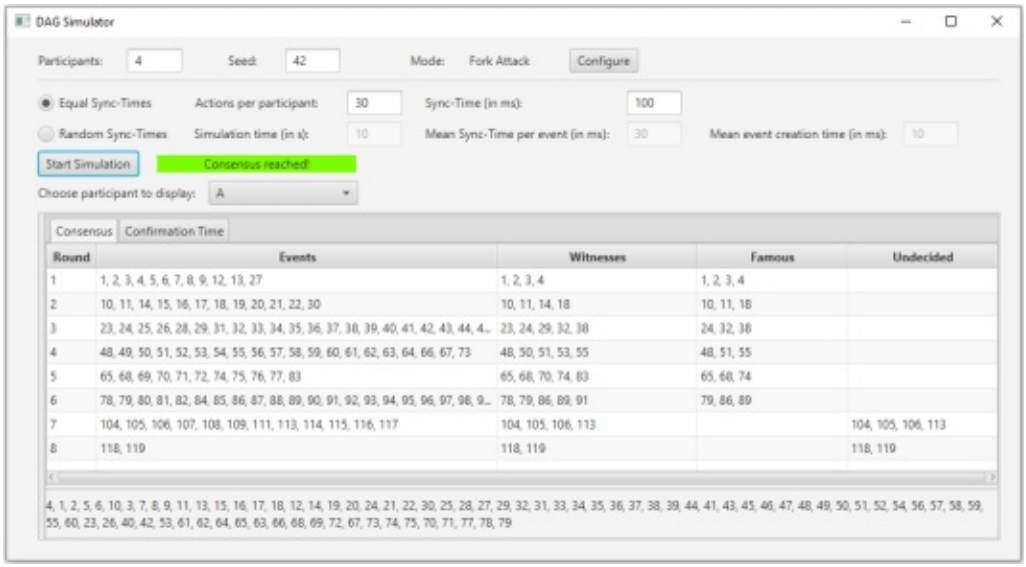


Figure 5.1: Witnesses of forking participant excluded from consensus mechanism

of paid transaction fees. However, it is no problem to transfer all funds to a different account and participate in the consensus mechanism again.

It is however also possible that a forking participant still has famous witnesses as long as the fork is not known to most other participants and there exists no other witness of the same participant in the current round. Just imagine a nothing-at-stake scenario where an adversary gets control over the private key of an honest participant once the honest one does not posses any stake anymore. The adversary could then start a fork somewhere back in time. Once the forking event is communicated to other participants, the respective gossip-sync event gets the current round assigned. Thus it is not possible that conflicting witnesses pop up in the past. Furthermore, conflicting transactions are resolved by considering the one with the lower confirmation timestamp as valid. As long as more than two thirds of the participants act honest, later appearing forking events cannot achieve a lower timestamp as already confirmed events on the former existing branch. It might even be impossible to start forks for too old events if we look at the Hedera Hashgraph whitepaper [BHM19], which allows participants to drop already decided older rounds to save resources. These mechanics are also effective defence mechanisms against any posterior corruption attacks.

To see if a fork attack has any impact on the confirmation times of events, we compare the average confirmation times of the honest executions with the fork scenario times. As the *Random Sync-Times* mode represents real world conditions better than the *Equal Sync-Times* mode, we use its configurations for the comparison. Table 5.1 covering the four participant case shows us, that every participant needs on average 20 to 25 percent more time to confirm an event. This high increase makes sense, as a quarter of the

| | A | B | C | D |
|---|---|---|---|---|
| **Honest 4-RST (ms)** | 4519 | 4498 | 4512 | 4503 |
| **Fork 4-RST (ms)** | 5421 | 5595 | 5576 | 5546 |
| **Difference (%)** | 19.96 | 24.39 | 23.58 | 23.16 |

Table 5.1: Average confirmation times 4-RST honest vs. fork

| | A | B | C | D | E | F | G |
|---|---|---|---|---|---|---|---|
| **Honest 10-RST (ms)** | 14439 | 14459 | 14443 | 14526 | 14458 | 14492 | 14468 |
| **Fork 10-RST (ms)** | 15259 | 15170 | 15213 | 15201 | 15191 | 15233 | 15196 |
| **Difference (%)** | 5.68 | 4.92 | 5.33 | 4.65 | 5.07 | 5.11 | 5.03 |

Table 5.2: Average confirmation times 10-RST honest vs. fork

| | A | B | C | D | E | F | G |
|---|---|---|---|---|---|---|---|
| **Honest 20-RST (ms)** | 34795 | 34718 | 34834 | 34804 | 34773 | 34779 | 34831 |
| **Fork 20-RST (ms)** | 35571 | 35399 | 35288 | 35479 | 35303 | 35357 | 35454 |
| **Difference (%)** | 2.23 | 1.96 | 1.30 | 1.94 | 1.52 | 1.66 | 1.79 |

Table 5.3: Average confirmation times 20-RST honest vs. fork

participants (and thus stake) is excluded from the consensus mechanism, which slows it down. When we look at tables 5.2 and 5.3, we can see that this slowdown decreases based on the number of participants. It hast to be noted here, that tables 5.2 and 5.3 only show values for the first seven participants for the sake of readability. Anyway, for ten participants confirmation times are slowed down by around 5 to 6 percent on average. Looking at the values for 20 participants one can observe that the confirmation time increase there lie around 1.3 to 2.3 percent on average. This leads us to the conclusion, that the impact of a fork attack on the overall performance of other participants in terms of confirmation times is significant for very low numbers. Nevertheless, it becomes more and more negligible the more the ratio of malicious nodes to honest nodes shifts in the favour of the honest ones. Comparing all three tables shows us that the average confirmation times also increase based on the total number of participants. One can thus conclude that the system acts faster the fewer participants there are, while at the same time getting more vulnerable to attacks in regard to performance numbers.

## 5.2 Race attack

What we understand by a race attack in the context of cryptocurrencies is an adversary being able to get transactions faster confirmed/finalized than honest users. We see that

such an attack can become an issue when we look at Ethereum's ERC-20 token standard[1]. This standard enables the creation of tokens representing financial assets (e.g. shares of a company, a fiat currency, or a new currency on top of Ethereum). These tokens can be traded via decentalized exchanges[2]. Users can freely place sell or buy orders which can lead to scenarios where you can buy a certain token at a low price and immediately sell it for a higher one, leading to profit without any risk. Whoever finds a (possibly malicious way) to boost one's own transactions, profits at the cost of honest users. It even becomes worse, as the necessary transactions can be publicly seen once they are published. This leads to the possibility of an adversary publishing a conflicting transaction to get the profit instead. As Ethereum is still PoW based, the transaction landing on the chain with the highest accumulated difficulty target will be considered valid. However, an adversary starts the race behind the honest user in such a scenario. If there exists a way for an adversary to increase the chance of outpacing the honest user in either case, race attacks are possible. Daian et al. [DGK+19] published a paper on what possibilities exist to do exactly that on decentralized exchanges for Ethereum. They show that there exist various bots that even compete against each other to close arbitrage deals found directly on decentralized exchanges (normal race) or received through transactions of other participants (frontrunning).

This topic is highly relevant, as Hedera Hashgraph explicitly references stock markets as an use case for their ordering service.

### 5.2.1 Attacking Hashgraph

The fairness model of Swirlds-Consensus states, that events are ordered based on the mean timestamp of participants first hearing about the transaction, once it reached a significant fraction of the community. The protocol defines no restrictions on how many gossip-syncs one is allowed to perform in a certain time period and there exists no punishment for flooding the network with gossip-syncs to other participants. On the contrary, the protocol actually states the following in regard to participants trying to do exactly this:

> Similarly, Bob could gain an advantage over Alice by buying more bandwidth, so that his gossips reach more people, faster. If he has 8 times the bandwidth of Alice, so that he can send his transaction initially to 8 members in the time Alice sends to 1, then he can gain an advantage of the time of about 3 gossip syncs. This is not considered a failure. If his message actually reaches the world before hers, then he should have the credit for it. This is similar to the current stock markets, where companies spend large sums of money for slightly faster connections, in order to reach the central server faster. [Bai16b]

[1] ethereum website, accessed 20.11.2020: `https://ethereum.org/en/developers/docs/standards/tokens/erc-20/`

[2] there exist multiple different exchanges which handle such trades via Smart Contracts, one example being EtherDelta (accessed 20.11.2020): `https://etherdelta.com/`

Considering that Hedera Hashgraph introduces stake and everybody knows how stake is distributed (because for the protocol to work, everybody needs to know the total participant/stake count), this can become a problem. Just imagine participants with different processing capabilities and everybody is flooding the network with as many gossip-syncs as possible to get own transactions confirmed faster. Weaker participants (or participants with less bandwith) might have problems to process all incoming gossip-syncs. But worse, why would any participant choose gossip-sync partners randomly as intended? A better strategy would be to always contact participants with higher stake first, increasing the load for them even more.

However, if we look at how network fees are spread, we see that faster confirmation times or publishing more transactions does not influence one's earnings. As long as participants exceed the minimum activity treshhold, they get coins based on their proportion of stake. They do however get directly payed for including transactions into the DAG structure. It is imaginable, that users prefer to publish their transactions to participants who ensure faster confirmation times. This might provide more incentive for participants to perform as many gossip-syncs as fast as possible. However, to argue about that, one needs to know what confirmation time differences can be expected.

This data can be acquired using our simulator. We need to compare confirmation times between faster and slower participants under different scenario configurations (changing participant numbers or gossip-sync times) to look at how effective race attacks are, as well as finding out if the difference is relevant enough to be able to influence users decision to whom they publish their transactions. Additionally, the very relevant question, if an adversary is able to outpace an event issued shortly before by another participant, needs to be evaluated as well. This can be done by comparing the *round received* and *final consensus timestamp* values of the two competing events, which the simulator provides.

### 5.2.2 Evaluation

We first evaluate how realistic it is for an adversary to win a race based on how many more gossip-syncs it is able to perform in comparison to an honest participant. We use the *Equal Sync-Times* mode (configurations 4-EST, 10-EST and 20-EST) for that, as it enables us to exactly configure the amount of gossip-syncs the adversary performs compared to the honest participants. Additionally, the adversary does not accept any gossip-syncs from the victim once the attack started, to slow down the spread of the original transaction. That and the fact that the adversary creates an event at the start of the attack result in a difference between an honest run and a race attack with gossip-sync value 1. Again, participant A takes on the role of the adversary. We do not simulate the scenario with multiple adversarial participants, as this would make the scenario more complex without increasing its meaningfulness. Having two adversary for example would only increase the amounts of gossip-syncs with malicious intent that are performed at the same time. However, our simulator already enables a scenario configuration where the adversary is syncing to all

other participants at once (or an equivalent reduction of gossip-sync durations). This already represents the scenario where the adversary possesses the maximum advantage and adding further adversarial participants does not increase that. It only increases the amount of malicious stake, which can already be varied indirectly via participant numbers.

Figure 5.2 shows how likely it is for an adversary to win a race, when its event is published shortly after the original event was published. Although simulation runs were executed for all seeds, some of the runs do not result in the competing events being finalized when the simulation ends. For example, EST-20 with gossip-sync value 9 results in 993 races from which 851 were won by the adversary. No bars are displayed for EST-4 starting with gossip-sync value 4 and EST-10 for gossip-sync value 19, as the maximum number of unique gossip-sync targets is lower. The best case for an adversary is already reached when syncing to all other participants at once. Further gossip-syncs do not provide any advantage for the adversary and are thus not performed by the simulator. As expected, syncing to more participants than an honest participant increases the chance of winning a race independent of the number of participants. This is not surprising due to the exponential nature of events spreading as receivers forward them within their own gossip-syncs. This mechanic amplifies any advantage a single participant might have. We can also examine the trend, that the impact of being able to perform more gossip-syncs turns out higher the more participants exist. The reason being, that it is not enough that all participants can see your event. In the best case at least two more rounds of famous witnesses must be present for your event to get finalized. Sure, if both events have the same round received value, the median timestamp of reaching the participants is taken, which puts the adversary at an advantage. We also assume, that the adversary wins all ties which are broken by a mechanism based on the signature to give him even more power. However, if there are only four participants, an honest participant needs only two gossip-syncs, so that more than two thirds of the participants know about its event. It is very realistic that this happens before two more rounds of famous witnesses occur. The evaluation results of Figure 5.2 also reflect that so far, as that the highest success rates occur under the EST-20 simulation. Syncing to three out of ten/twenty participants is more effective than syncing to all out of four participants. Interesting is also, that massively increasing the number of syncs leads to only a small rise in success probability, as the EST-20 numbers show us. Syncing to all other participants instead of only half of them (gossip-sync value 19 instead of 9) only increases the success probability by 1%. Success probabilities for only ignoring gossip-syncs from the victim lie between 35 to 50 percent. That means that even if you cannot outperform your victim in terms of gossip-syncs, you still have a reasonable chance of success. It has to be noted though, that the adversary in the simulation runs does not take any idleness or create event actions anymore and that Figure 5.2 shows that the chance of success decrease by fifteen percent between the EST-4 and EST-20 scenario even under these advantageous circumstances. One can assume from the data we have, that this number decreases even further if more participants join. The question how low it would be in the end for thousands of participants as intended by Hedera Hashgraph remains open

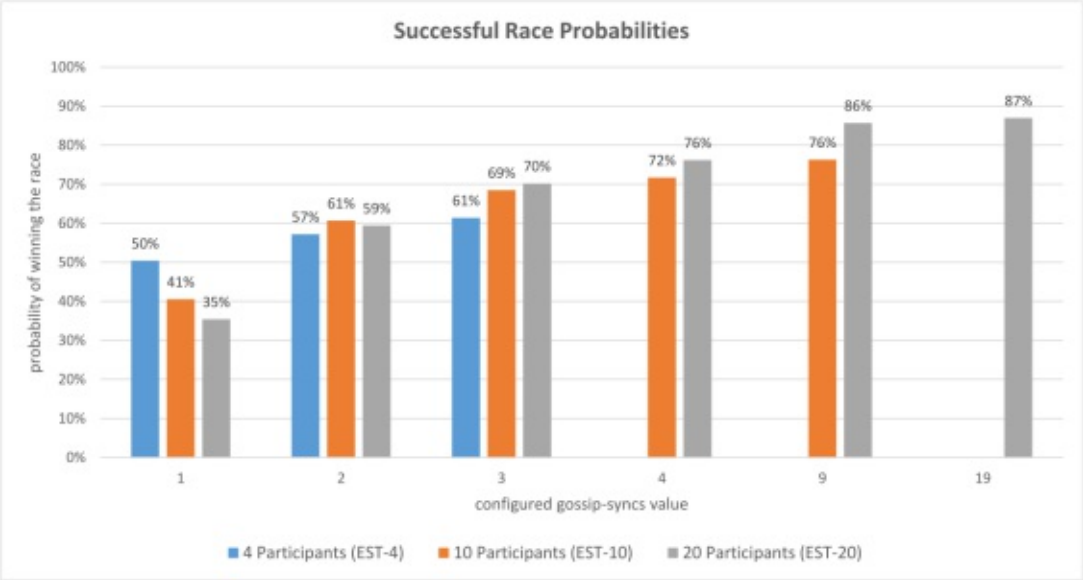


Figure 5.2: Probability of adversary winning a race based on number of gossip-syncs and number of participants

for future work. All in all, we can clearly see that already syncing twice as often sets the chance of successfully frontrunning another transaction to around 60% which makes Hashgraph vulnerable to such attacks.

Next, we look on the effects of this attack on the overall confirmation times. Table 5.4 shows us that average confirmation times for different race speeds decrease only slowly. Syncing to all participants is 285ms (or 15.4%) faster for the EST-10 setting and 305ms (or 12.7%) faster for the EST-20 setting compared to the honest values. The race attack with gossip-sync value 1 being slower than the honest case for EST-4 can be explained by the different behaviour of the racing participant. It does not accept gossip-syncs from the victim, which leads to a little slowdown in confirmation times. On the other hand, the adversarial participant does not perform any idleness or event creation actions anymore, which leads to a decrease in average confirmation times. As one can see in Table 5.4, the combination of these two effects turns out in favour of the adversarial participant with rising participant numbers. You can see all that data visualized in Figure 5.3. Also, we examined the average confirmation times for the racing participant, as its confirmation times profit the least from its own speed. The racing participant must still wait for gossip-syncs of other participants to confirm all events and it does not accept gossip-syncs of its victim. So, this performance increases in average confirmation time are also valid for all the other participants. Looking at the data, the confirmation times between participants vary in a range of around 20ms (EST-20), 10ms (EST-10) and 5ms (EST-4).

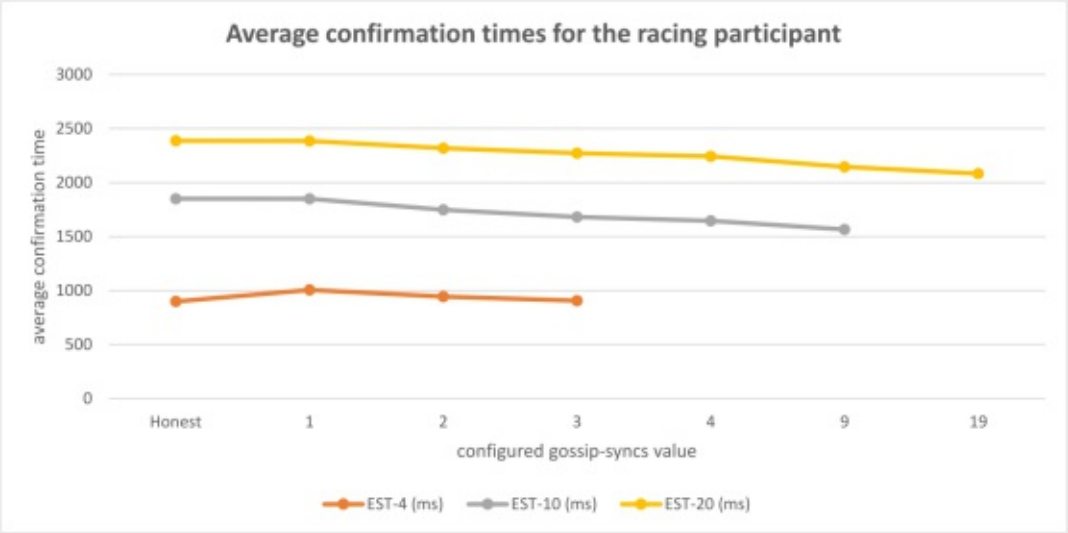


Figure 5.3: Average confirmation times for racing participant visualized

| | Honest | 1 | 2 | 3 | 4 | 9 | 19 |
|---|---|---|---|---|---|---|---|
| **EST-4 (ms)** | 899 | 1006 | 944 | 906 | | | |
| **EST-10 (ms)** | 1851 | 1850 | 1748 | 1682 | 1647 | 1566 | |
| **EST-20 (ms)** | 2387 | 2386 | 2318 | 2271 | 2244 | 2146 | 2083 |

Table 5.4: Average confirmation times for participant A under different race speeds

All in all, we can conclude that a single participant can have an impact on total confirmation times. The percentage rate of this influence decreases with rising participant numbers. This can be traced back to the fact that a certain entanglement between the other honest participants must be given to fulfill the criteria of strongly seeing. However, the adversary cannot influence how fast the other participants perform their gossip-syncs. A more significant decrease in confirmation times only occurs, if more participants sync faster, as one can verify by decreasing the parameter affecting gossip-sync time.

Our results state that race attacks are possible and thus there exists an incentive for users to send their transactions to the participants which spread their transactions faster over the network. This incentive however only applies for transactions where order is important, as total confirmation times vary insignificantly. When we introduce stake, spreading over the network faster means reaching as much stake as fast as possible. Thus, for transactions where order is important, participants have a strong incentive to first (or only) sync to other participants possessing more stake, effectively excluding participants with lower stake partially from the consensus mechanism by not sending them gossip-syncs. High stake participants might receive more requests than they can process, which can be compared with a DDoS attack on them, rendering them useless.

The possibility of this attack thus endangers the functionality of the whole protocol, as taking out stake slows down the protocol until it cannot achieve progress at all, as we will see in the next attack we discuss.

## 5.3 Split attack

A split attack partitions the nodes of a cryptocurrency in (at least) two parts, resulting in nodes only being able to communicate with other nodes of the same part. Cross-sectional communication is prevented.

To achieve this, the attack targets the underlying network of the cryptocurrency. Because decentralization is a major property of most cryptocurrencies, the protocols usually rely on a peer-to-peer network structure. Nodes know a few other nodes and communicate directly with them. This peer-to-peer structure is built on top of the Internet, leading to more attack surfaces. Apostolaki et al. [AZV17] describe that the Internet consists of multiple networks called Autonomous Systems and that there is de-facto only the Border Gateway Protocol (BGP) used to regulate the packet flow between them. Adversaries can perform BGP hijacks by forging BGP routing information to isolate a certain amount of nodes of a cryptocurrency. This effectively splits the peer-to-peer network into two disjoint components. Looking at attacks on different systems in the past, Apostolaki et al. state, that it can take hours of work to resolve such attacks once they are detected. It has to be noted here, that network partitioning like that can also happen without any malicious intent behind it, which makes it even more important to study its impact on cryptocurrencies. The threat was analyzed in depth for Bitcoin, but it was also mentioned that the research carries important lessons for other peer-to-peer network based cryptocurrencies operated atop of the Internet. Thus, such attacks must also be considered for Hashgraph.

### 5.3.1 Attacking Hashgraph

Our simulator enables us to simulate split attacks where all participants are divided into one of two different groups. The split size can be configured and thus we have to consider two different scenarios. First, we can have a look at the case where the split results in none of the two parts having more than two thirds of the total stake. This results in no progress in terms of event finalization. Nevertheless, we can investigate if and how the DAG data structure changes under such an attack.

The more interesting case is when the split does not prevent progress, as one of the parts possesses more than two thirds of the stake. Here, one can find out how event confirmation numbers are influenced based on a combination of split size and total participant count.

### 5.3.2 Evaluation

For Swirlds-Consensus to confirm events it is per definition necessary that more than two thirds of the participants can communicate with each other. If just some participants cannot communicate with each other for a certain amount of time (e.g. through a split attack), the DAG data structure grows slower. That is because less gossip-syncs can be performed. This then leads to the creation of less gossip-sync receive events. Moreover, we have two distinct graphs in the different split parts if the two third mark is not exceeded in any part and the split condition holds since the start of the simulation. Once the split is resolved (simulatable by ticking the configuration checkbox that the partition ends after three quarters of total actions/simulation time), gossip-sync time increases for the first syncs after the split is resolved. Again, we use the *Random Sync-Times* mode for our evaluation to be closer to real world conditions. It always calculates gossip-sync times based on the amount of unknown events which would have to be transferred. However, calculating it that way when a gossip-sync cannot be performed would seriously distort the simulation results. Gossip-sync times would increase the longer the split lasts, which would lead to less overall actions. To mitigate that distortion, a participant performs an idle action whenever its random action would have been a gossip-sync to an unreachable participant.

To evaluate the effects of a split attack, we will look at the amount of confirmed events instead of average confirmation times. We prefer that over average confirmation time, as the latter would not account for events not confirmed through the split situation. Not confirmed events are not taken into account for that value, because the last events of every simulation cannot have a confirmation time assigned. This holds true independent of the simulation scenario, simply because there exist no future famous witnesses that could decide their final consensus timestamps. Figure 5.4 presents the number of confirmed events based on split sizes. That data was derived from simulations where the split condition ended after three quarters of the simulation time, to give the participants the chance to confirm more events before the simulation time is over. The diagram shows us, that for RST-20 no events are confirmed even one value before the critical split size is reached. Please keep in mind, that for RST-4 at least three participants, for RST-10 at least 7 participants and for RST-20 at least 14 participants need to be in the same part to confirm events. If the split size is increased high enough, this condition will be given again (e.g. split size 1 and 3 are equivalent for RST-4). Figure 5.4 only presents split sizes until no confirmation is possible. It also shows us that confirmation numbers decrease slower on runs with more participants. Nevertheless, we see that nearly no events are confirmed once the split size approaches one third of the participants. That implies, that a split attack severely slows down confirmation even in the part that can still confirm events. Progress is completely halted in the other one. For RST-20 at split size 2 (representing 10% of the total participant count), confirmed events dropped by 23%.

This is not optimal, but not critical to the security of the system. In comparison, in PoW cryptocurrencies like Bitcoin [Nak08] or Ethereum [Eth] a split would mean that

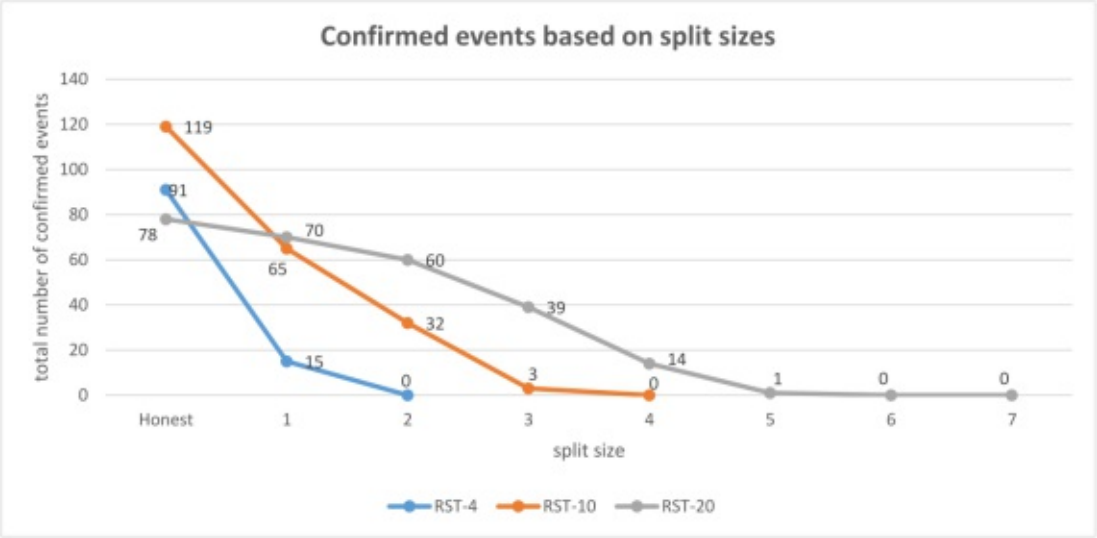


Figure 5.4: Confirmed events based on split sizes

a fork occurs and the different split parts mine on different branches. An adversary that controls how the network is split might have enough hashing power in one of the parts to dominate it. Other protocols like Ouroboros Praos [DGKR17] explicitly state that they stay secure as long as an adversary has control over message delays until a certain upper limit. So, security problems may occur due to a split attack if it lasts for long enough. On the other hand, cryptocurrencies like Algorand [GHM+17] might also be mainly delayed in finalizing blocks.

## 5.4 Balancing attack based on split attack

The split attack enables an adversary to stall event confirmations completely for a limited amount of time (as long as the split prolongs). The split in this attack is rather static. The network is partitioned and participants in different parts cannot communicate with each other until it is resolved. Swirlds-Consensus states, that it stays secure under a scenario where an adversary can delete or delay messages, as long as it is given that if honest members repeatedly send messages to one another, the adversary must eventually let one through. One can see this scenario as an advanced split attack, where the adversary dynamically defines which participants are cut off from each other and which ones can communicate. But what about liveness under this scenario? As the adversary can delay messages without any bounds, he or she can effectively define exactly how the DAG structure looks like. Is it possible to generate it in a way, so that events are never confirmed (given that an adversary must continuously extend the DAG structure)? In Swirlds-Consensus events are finalized if they are seen by all famous witnesses of a round. So, the ultimate goal is to stall the process of deciding which witnesses are going to be famous. This voting process is based on what the witnesses of the respective fol-

lowing round can see. So, lets look at the possibilities to sabotage this process step by step.

First, we have to discuss if an adversary can connect events in a way that a next round is never reached. An event only gets a higher round number assigned if it can strongly see more than two thirds of the witnesses in the current round. This can be written mathematically as $\lfloor 2n/3+1 \rfloor$ with $n$ being the total participant count. $\lfloor 2n/3+1 \rfloor$ also denotes the minimum number of honest participants (by assumption). An adversary cannot prevent reaching the next round by definition, as messages between honest participants must be eventually let through. Witnesses then vote for the famousness of the witnesses of the round before based on if they can see it or not. An adversary with the described power can define the outcome of every vote by building the DAG structure accordingly. Now, every witnesses of the next round has the power to finalize the decision if it strongly sees more than two thirds of the witnesses of the round before. However, this condition is always true for every witness because of the way round numbers are assigned. The first event getting a higher round number assigned certainly fulfills this condition, as we established above. Other events in this round either go through the same process if they have two parents of the round before, or are themselves children of an event that already fulfills this criteria. Thus, all witnesses strongly see the witnesses of the round before.

We established that an adversary cannot prevent this votes from happening, but can completely control the outcome. Witnesses see votes from at least $\lfloor 2n/3+1 \rfloor$ participants and only finalize a famousness vote based on the majority vote if it is mathematically impossible to flip their decision with the votes they cannot see. If they cannot finalize their decision, they just vote with the majority and hand over to the witnesses of the next round. So, an adversary can prevent finalization of famous witnesses by building the DAG in a way that votes are too balanced to ever decide. For that case, Swirlds-Consensus introduced coin rounds. Basically, all $c$ rounds voting is not done based on previous votes that a witness can see, but instead on the middle bit of the deciding witnesses' signature. These votes are thus out of the control of the adversary, as these events are created by the honest participants themselves. An adversary can however set the votes for stake controlled by him or her. The adversary can still determine which $\lfloor 2n/3+1 \rfloor$ votes each witness of the next round can strongly see, enabling him or her to include his own votes in every of these decisions. If we assume that the adversary controls the maximum number of participants (or stake) possible so that $\lfloor 2n/3+1 \rfloor$ participants are still honest, in the worst case all random honest votes must be equal to tip the balance in one direction. Looking at the example of $n = 10$ participants, we can see that $\lfloor 2n/3+1 \rfloor = 7$ nodes are honest and that is the same number of equal votes an witness must see to decide famousness. The adversary controls the remaining 3 votes and can include them in any way in the votes that witnesses see. No matter if just one or all three votes are included, witnesses strongly seeing only 7 witnesses of the former round cannot finalize famousness. Remember, if they cannot decide famousness, they vote themselves based on the majority of votes they see and leave the decision to the witnesses of the next

round. As the adversary controls only three of the seven votes taken into account, all witnesses of the coin round (even the one's of the malicious participants) are considered to vote with the honest majority. Next round the adversary has nothing to balance, as all votes are the same and the famousness of the witnesses in question is decided. It has to be noted here, that progress in terms of event confirmation is only made if witnesses become famous and thus only one coin round result ensures progress.

The probability for a coin round to cause the confirmation of events is thus $0.5^{\lfloor 2n/3+1 \rfloor}$. The probability of that happening is pretty low as the numbers for ten participants ($0.5^7 = 0.0078$) and a thousand participants ($0.5^{667} = 1.63e-201$) show us. In theory, progress is guaranteed with probability one even for that case. In practice, the low probabilities render the protocol useless under this attack scenario.

## 5.5 Findings relevant to PoS DAG protocols in general

If we look at the PoS DAG protocols we have discussed and the findings of the simulation runs (especially the ones of the race attack), we can see that one problem appears over and over again. Participants which spam certain types of messages can cause trouble in most of the protocols. Therefore, one lesson learned is, that having no mechanism in place to prevent uncontrolled spamming of messages can lead to security of liveness weaknesses, potentially endangering the whole system. Thus, whenever designing or looking at such protocols from a security perspective, one should specifically pay attention to mechanisms preventing spamming or a lack thereof.

CHAPTER 6

# Summary and Future Work

Traditional PoW cryptocurrencies like Bitcoin can be seen as decentralized payment systems. Their potential is however limited by some challenges, two of the major ones being high energy consumption and low transaction throughput. Over the years, solutions for these problems were proposed, like PoS systems for reducing the energy consumption our exchanging the underlying data structure with DAGs to increase transaction throughput. Some protocols emerged which combined these two principles to solve both problems at once.

In this thesis we investigated Hashgraph, one of the most promising of these PoS DAG protocols. We tested its resilience against multiple different attacks by creating a simulator and evaluating the protocol over thousands of simulation runs. A learning was, that a fork attack does not compromise its security. To be more concrete, not a single one of thousands of simulation runs broke the protocol, independent of the simulation scenario. So, the major security claims hold true although the protocol allows for very powerful adversaries. Adversaries are allowed to delete or delay all messages as long as they eventually let one through if it was repeatedly sent by an honest participant. However, an adversary with these powers can balance votes of participants in a way, that there is only a very, very small chance of the protocol to progress every few rounds. Although progress is guaranteed with probability 1, this causes a liveness problem for all practical purposes.

That being said, some additional weaknesses became apparent when performing race attacks. Hashgraph states that the protocol only benefits from all participants performing many gossip-syncs as fast as possible, but the test results point to a different conclusion. The protocol's design allows for frontrunning other transactions, meaning publishing the same transaction after having seen a transaction of another participant to e.g. close a lucrative deal. The simulation results show, that frontrunning attacks are possible and

depending on the speed advantage are successful in about 57 to 82 percent of the time (dependent on the set configurations of the simulation scenario). This possibilities result in some cases in an incentive structure for for users to send their transactions to the fastest participants. Participants are (as intended) encouraged to perform gossip-syncs as fast as possible. While the latter decreases overall transaction times, it can also endanger the system. Considering that communicating first with higher stake owning participants increases the chance of lower final consensus timestamps, there exists an encouragement to mainly perform gossip-syncs to them. That can result in participants which own less stake being ignored and partially excluded from the consensus mechanism, as well as participants possessing more stake being flooded with messages. The latter can have similar effects to a DDoS attack on the main contributing participants if these cannot handle the amount of messages.

Evaluating split attacks also showed us, that one can expect less events getting confirmed when the network is partitioned. This ultimately increases average confirmation times and lowers transaction throughput. So far, Hashgraph only provided performance metrics under conditions that are not representative for the planned end stage of the system. Nevertheless, goals of thousands of confirmed transactions per second were presented. What we learned examining the split attack results is, that performance numbers will be way lower if the network is partitioned independent of how the actual performance numbers of the final system will look like. Additionally, progress may be temporarily stalled completely if the network partitioning happens in a way that no single part contains more than two thirds of the total stake.

In addition to these findings, the simulator itself is another major contribution of this thesis. Its source code is published under an open source licence to enable everybody to perform their own simulations. The simulator operates in two modes that differ greatly in how actions are performed. It also offers a wide range of different configuration possibilities like participant numbers, simulation time, action probabilities and the respective action durations as well as specific attack scenario details. Results can either be investigated in a graphical user interface or exported in the CSV format for automated analysis. All simulation runs are randomized in a reproducible way by a configurable seed value.

The simulator supports investigating further aspects of Hashgraph or even functions as a template for similar programs for other cryptocurrencies. It can be extended, so that additional attack scenarios can be explored in future work. It would also be interesting to look at the results of the same scenarios for thousands of participants, by increasing the simulators performance to enable such simulations within reasonable timespans. Apart from extending the simulator, further research is necessary around frontrunning attacks. Daian et al. [DGK+19] categorized frontrunning as a security threat to Ethereum because of additional income sources for miners (in which Smart Contracts played a big role) through certain behaviour like transaction order optimization. Hashgraph also supports

Smart Contracts, but is built on top of a very different consensus mechanism. The question remains open if frontrunning poses a similar threat to PoS DAG systems like it does to PoW chains like Ethereum. Another interesting question for future work is, if some sort of rate limiting or punishment is necessary to prevent gossip-syncs from turning into DDoS attacks on the participants which possess the most stake.

All in all, our simulation scenarios showed that Hashgraph's consensus protocol is very resilient against attempts of breaking it. We have however found some weaknesses that incentivize undesirable behaviour or enable exploitation by an adversary. We can conclude this thesis by stating that Hashgraph clearly shows that PoS DAG protocols can solve two of the major problems of traditional cryptocurrencies. Nevertheless, it is not a perfect protocol yet and further investigations and improvements are advisable.

# List of Figures

# List of Tables

# Bibliography


[AOK+19] Yusuke Aoki, Kai Otsuki, Takeshi Kaneko, Ryohei Banno, and Kazuyuki Shudo. Simblock: A blockchain network simulator, 2019. version 2, https://arxiv.org/abs/1901.09777.

[AvM20] Maher Alharby and Aad van Moorsel. Blocksim: An extensible simulation tool for blockchain systems, 2020. https://arxiv.org/abs/2004.13438.

[AZV17] Maria Apostolaki, Aviv Zohar, and Laurent Vanbever. Hijacking bitcoin: Routing attacks on cryptocurrencies. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 375–392, 2017. https://ieeexplore.ieee.org/document/7958588.

[Bai16a] Leemon Baird. Hashgraph consensus: Detailed examples. Swirlds website, 2016. https://www.swirlds.com/downloads/SWIRLDS-TR-2016-02.pdf.

[Bai16b] Leemon Baird. The swirlds hashgraph consensus algorithm: Fair, fast, byzantine fault tolerance. Swirlds website, 2016. https://www.swirlds.com/downloads/SWIRLDS-TR-2016-01.pdf.

[BG17] Vitalik Buterin and Virgil Griffith. Casper the friendly finality gadget, 2017. version 4, https://arxiv.org/abs/1710.09437.

[BHM19] Leemon Baird, Mance Harmon, and Paul Madsen. Hedera: A public hashgraph network & governing council. Hedera Hashgraph website, 2019. version 2.0, https://www.hedera.com/hh_whitepaper_v2.1-20200815.pdf.

[BKT+18] Vivek Kumar Bagaria, Sreeram Kannan, David Tse, Giulia C. Fanti, and Pramod Viswanath. Deconstructing the blockchain to approach physical limits. *CoRR*, abs/1810.08092, 2018. version 4.

[Bra18] Quentin Bramas. The stability and the security of the tangle. HAL archives, 2018. version 2, https://hal.archives-ouvertes.fr/hal-01716111v2.

[CDFZ17] Alexander Chepurnoy, Tuyet Duong, Lei Fan, and Hong-Sheng Zhou. Twinscoin: A cryptocurrency via proof-of-work and proof-of-stake. Cryptology ePrint Archive, Report 2017/232, 2017. version 20170314:072227, https://eprint.iacr.org/2017/232.

[CELR17] Mauro Conti, Sandeep Kumar E, Chhagan Lal, and Sushmita Ruj. A survey on security and privacy issues of bitcoin. *CoRR*, abs/1706.00916, 2017. version 3, https://arxiv.org/abs/1706.00916.

[DD20] Debojyoti Das and Anupam Dutta. Bitcoin's energy consumption: Is it the achilles heel to miner's revenue? *Economics Letters*, 186, 2020. https://www.sciencedirect.com/science/article/pii/S0165176519302526.

[DFZ16] Tuyet Duong, Lei Fan, and Hong-Sheng Zhou. 2-hop blockchain: Combining proof-of-work and proof-of-stake securely. Cryptology ePrint Archive, Report 2016/716, 2016. version 20170416:032904, https://eprint.iacr.org/2016/716.

[DGK+19] Philip Daian, Steven Goldfeder, Tyler Kell, Yunqi Li, Xueyuan Zhao, Iddo Bentov, Lorenz Breidenbach, and Ari Juels. Flash boys 2.0: Frontrunning, transaction reordering, and consensus instability in decentralized exchanges. *CoRR*, abs/1904.05234, 2019. http://arxiv.org/abs/1904.05234.

[DGKR17] Bernardo David, Peter Gaži, Aggelos Kiayias, and Alexander Russell. Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake protocol. Cryptology ePrint Archive, Report 2017/573, 2017. version 20171115:001835, https://eprint.iacr.org/2017/573.

[DPP19] Evangelos Deirmentzoglou, Georgios Papakyriakopoulos, and Constantinos Patsakis. A survey on long-range attacks for proof of stake protocols. *IEEE Access*, 7:28712–28725, 2019. https://ieeexplore.ieee.org/document/8653269.

[DPS16] Phil Daian, Rafael Pass, and Elaine Shi. Snow white: Robustly reconfigurable consensus and applications to provably secure proof of stake. Cryptology ePrint Archive, Report 2016/919, 2016. version 20200726:223311, https://eprint.iacr.org/2016/919.

[dV19] Lucas de Vries. Iota vulnerability: Large weight attack performed in a network. 30th Twente Student Conference on IT, 2019. http://essay.utwente.nl/77602/1/30-TScIT_paper_28.pdf.

[ES13] Ittay Eyal and Emin Gün Sirer. Majority is not enough: Bitcoin mining is vulnerable. *CoRR*, abs/1311.0243, 2013. version 5, http://arxiv.org/abs/1311.0243.

[Eth] Ethereum. Ethereum white paper. introduced and maintained on github. accessed on 9.9.2020: https://ethereum.org/en/whitepaper/.

[FC19] Carlos Faria and Miguel Correia. Blocksim: Blockchain simulator. In *2019 IEEE International Conference on Blockchain (Blockchain)*, pages 439–446, 2019. self published by author: https://static.carlosfaria.pt/file/personal-assets/papers/blocksim-blockchain-simulator.pdf.

[GHM+17] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. Cryptology ePrint Archive, Report 2017/454, 2017. version 20170924:210956, https://eprint.iacr.org/2017/454.

[GKR18] Peter Gaži, Aggelos Kiayias, and Alexander Russell. Stake-bleeding attacks on proof-of-stake blockchains. In *2018 Crypto Valley Conference on Blockchain Technology (CVCBT)*, pages 85–92, 2018.

[HKZG15] Ethan Heilman, Alison Kendler, Aviv Zohar, and Sharon Goldberg. Eclipse attacks on bitcoin's peer-to-peer network. In *Proceedings of the 24th USENIX Security Symposium*, 2015.

[KRDO17] Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. volume 37. The 37th International Cryptology Conference, 8 2017. version 20170407:061833, https://eprint.iacr.org/2016/889.

[LABK17] Wenting Li, Sebastien Andreina, Jens-Matthias Bohli, and Ghassan Karame. Securing proof-of-stake blockchain protocols. pages 297–315, 09 2017. https://www.researchgate.net/publication/319647471_Securing_Proof-of-Stake_Blockchain_Protocols.

[LeM17] Colin LeMahieu. Nano: A feeless distributed cryptocurrency network, 2017. https://content.nano.org/whitepaper/Nano_Whitepaper_en.pdf.

[LLP+19] Jingming Li, Nianping Li, Jinqing Peng, Haijiao Cui, and Zhibin Wu. Energy consumption of cryptocurrency mining: A study of electricity consumption in mining cryptocurrencies. *Energy*, 168:160–168, 2019. https://www.sciencedirect.com/science/article/pii/S0360544218322503.

[lw20] Nano living whitepaper. Nano - digital money for the modern world. Nano Website, 2020. https://docs.nano.org/what-is-nano/living-whitepaper/.

[MHG18] Yuval Marcus, Ethan Heilman, and Sharon Goldberg. Low-resource eclipse attacks on ethereum's peer-to-peer network. Cryptology ePrint Archive, Report 2018/236, 2018. https://eprint.iacr.org/2018/236.

[Nak08] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. Bitcoin website, 2008. https://bitcoin.org/bitcoin.pdf.

[PMC+20] Serguei Popov, Hans Moog, Darcy Camarg, Angelo Capossele, Vassil Dimitrov, Alon Gal, Andrew Greve, Bartosz Kusmierz, Sebastian Mueller, Andreas Penzkofer, Olivia Saa, William Sanders, Luigi Vigneri, Wolfgang Welz, and Vidal Attias. The coordicide. IOTA website, 2020. https://www.iota.org/foundation/research-papers.

[Pop18] Serguei Popov. The tangle. IOTA website, 2018. version 1.4.3, https://www.iota.org/foundation/research-papers.

[PSF19] Serguei Popov, Olivia Saa, and Paulo Finardi. Equilibria in the tangle. *Computers & Industrial Engineering*, 136:160–172, 2019. https://www.sciencedirect.com/science/article/pii/S0360835219304164.

[RCa+20] Gabriel Antonio F. Rebello, Gustavo F. Camilo, Lucas C. B. Guimar aes, Lucas Airam C. de Souza, and Otto Carlos M. B. Duarte. On the security and performance ofproof-based consensus protocols, 2020. publication RCG20b of the State University Rio de Janeiro: https://www.gta.ufrj.br/publicacoes/.

[RYS+20] Team Rocket, Maofan Yin, Kevin Sekniqi, Robbert van Renesse, and Emin Gün Sirer. Scalable and probabilistic leaderless bft consensus through metastability, 2020. version v2, https://arxiv.org/abs/1906.08936v2.

[SLZ16] Yonatan Sompolinsky, Yoad Lewenberg, and Aviv Zohar. Spectre: A fast and scalable cryptocurrency protocol. Cryptology ePrint Archive, Report 2016/1159, 2016. version 20180115:092032, https://eprint.iacr.org/2016/1159.

[SWZ18] Yonatan Sompolinsky, Shai Wyborski, and Aviv Zohar. Phantom and ghostdag: A scalable generalization of nakamoto consensus. Cryptology ePrint Archive, Report 2018/104, 2018. version 20200202:095456, https://eprint.iacr.org/2018/104.pdf.

[ZWH18] Manuel Zander, Tom Waite, and Dominik Harz. Dagsim: Simulation of dag-based distributed ledger protocols. Cryptology ePrint Archive, Report 2018/1062, 2018. https://eprint.iacr.org/2018/1062.