Tagion Whitepaper

Theis Simonsen
ts@decard.io

Carsten B. Rasmussen
cr@decard.io

Coauthor:
Benjamin Hilton
bbh22@cam.ac.uk

September 11, 2023

Original Version: August 1, 2018

Abstract

This paper describes the implementation of a Distributed Ledger Technology (DLT) system which solves the **3 aspects of the trilemma** better than any current system. The system distinguishes itself technically in *4 key ways*: The consensus protocol, the data storage method, the swapping algorithm, and its use of time-based staking. The system uses the *Hashgraph algorithm* as its asynchronous Byzantine Fault Tolerant (aBFT) consensus protocol. The protocol's asymptotic communication complexity is quadratic (theoretically optimal), making the effective throughput of the system significantly greater than blockchain alternatives. Moreover, the protocol has deterministic finality, enabling the system to order activity and transfer the byzantine proof to the storage layer. Finality also greatly enhances the **scalability** of the system, when combined with the second distinguishing feature: The *DART*. The Distributed Archive of Random Transactions (DART) saves data in a hash invariant database, allowing the system to be stateless. Lastly, the system uses *swapping* to achieve **decentralisation**, while maintaining **security** by using *time-based staking*. Tagion isn't just another DLT; it's a fundamentally new infrastructure that is scalable, secure, decentralised, fair, and sustainable.

Contents

1 Introduction

The landscape of Distributed Ledger Technology (DLT)-systems is continuously growing. Although most systems share notable similarities, they often differ in their prioritisation of the trilemma: How to achieve scalability, decentralisation, and security. *"The scalability trilemma stands in the way of blockchain* [DLT] *fulfilling its potential as a technology to change the world"*[2]. Most blockchains are secure and decentralised but lack scalability. In contrast, layer-two systems prioritise scalability, but in doing so lose the byzantine proof and thus their security. Other systems choose a leader-based system achieving both security and scalability but losing the decentralisation by choosing a single leader. Tagion is a DLT-system that makes none of these compromises, and claims to solve the trilemma better than any current system.

Many other systems which claim to solve the trilemma make idealised assumptions about the network: maximum response times, trusting a single leader, unlimited bandwidth, or are simply infeasible to implement in practice. Tagion makes none of these assumptions. Tagion is asynchronous, leaderless, has asymptotically minimal message complexity, and can be implemented in practice as this paper describes.

Tagion distinguishes itself from other systems in 4 key ways; the most fundamental one being the asynchronous Byzantine Fault Tolerant (aBFT) protocol it is built upon - the *Hashgraph algorithm*. By basing Tagion on the *Hashgraph algorithm* Tagion solves the trilemma in a way not possible in blockchain systems. While this consensus protocol has many advantages, it does not fully solve the trilemma by itself. On its own, it's a permissioned system with no fast way for computers outside the Hashgraph to read information; so neither scalable nor decentralised. Therefore, Tagion has built solutions around the Hashgraph algorithm to utilise its advantages and rectify its weaknesses.

These solutions are Tagion's 3 other unique features: *the Distributed Archive of Random Transactions (DART)*, *swapping*, and *time-based staking*. Each feature enhances the Hashgraph to achieve a specific part of the trilemma. The next section describes the *Hashgraph* in its atomic broadcast protocol, the foundation for solving all 3 parts of the trilemma. The following 3 sections outline how Tagion achieves each part of the trilemma using its unique features: **Scalability** with the *DART*, **decentralisation** through *swapping*, and **security** via *time-based staking*.

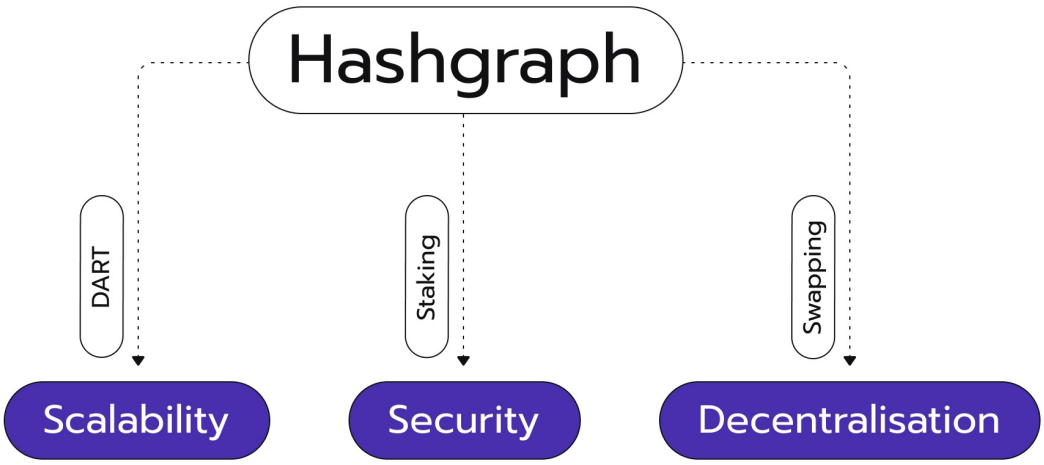

Figure 1: How Tagion uses its *4 distinguishing features* to solve **the trilemma**.

2 The Atomic Broadcast Protocol

The fundamental thing that makes Tagion stand out is its usage of a Hashgraph as the aBFT consensus algorithm, invented by Leemon Baird [1]. The following section gives a brief overview of how the Tagion utilizes the Hashgraph, to build an Atomic Broadcast Protocol (ABP).

2.1 The Hashgraph

The Hashgraph consists of a fixed number of nodes communicating with each other to reach consensus on the ordering of events. Information (such as a transaction) can enter the system through any node. Nodes continuously and randomly communicate with other nodes telling them all the information they do not have; in this way information propagates throughout the network. All shared information is signed by nodes, so a node cannot deceitfully lie about what other nodes have said.

2.1.1 Gossip about gossip

Apart from just gossiping information to each other, nodes also share the complete history of gossip - who talked to whom in what order. All nodes gossip about the history of gossip. Thus every node eventually learns of the complete history of communication; even for nodes they never directly communicated with. Every node locally stores the history of *hashed* communication in a *graph* - the *Hashgraph*.

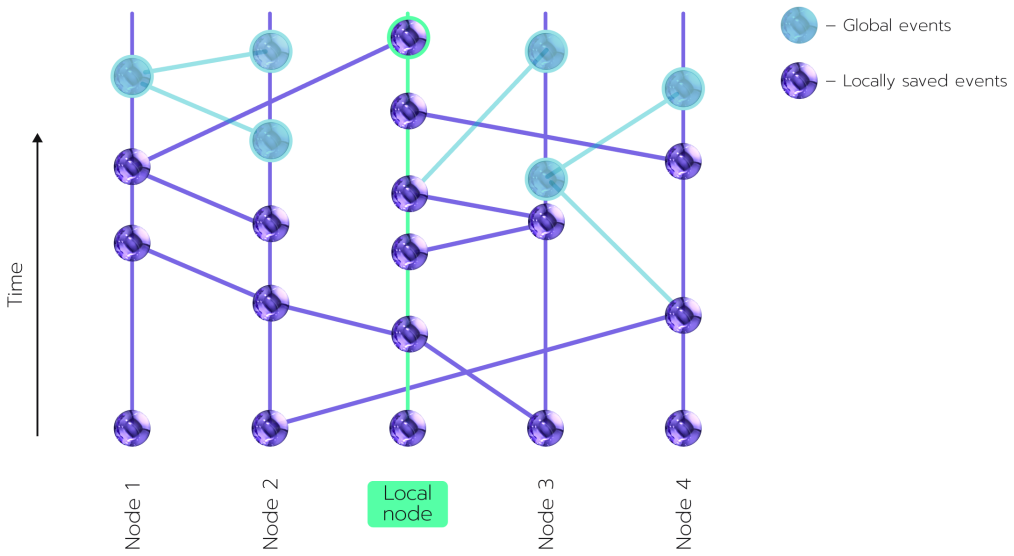

Figure 2: An example of the Hashgraph stored locally by a node. The transparent lines and circles represent events the local node is not *yet* aware of.

Above is a diagram showing an example of a Hashgraph stored locally by a node. Each vertical line represents a computer node and each circle an event: Some information entering

the system or being shared. The diagonal lines connecting events represent a node gossiping to another, that is, sharing the history of communication the other did not already know. It is important to note that this is not all the gossip happening in the network, just the gossip that our local computer node is aware of because other nodes have gossiped about it. The transparent lines and circles are gossip our local node is not *yet* aware of because no node has gossiped the information to it. This information is not *yet* saved in the local Hashgraph. So every node's local Hashgraph will differ at the top of the Hashgraph (no node is aware of all the newest gossip), but with time, all gossip will be added to every local Hashgraph.

2.1.2 Byzantine Fault Tolerance

The Hashgraph consensus protocol reaches byzantine agreement on the ordering of events. This protocol is aBFT, meaning that it makes no assumptions about communication delay and is tolerant of up to $\frac{1}{3}$ byzantine failures. So, even in scenarios with arbitrarily long communication delays and $\frac{1}{3}$ of the nodes sending contradictory information to disrupt the network, the system would continue to function. This makes the system very robust even under extreme circumstances which is essential in decentralised, distributed systems. A protocol where every node eventually agrees on the ordering of information is called an Atomic Broadcast Protocol (ABP). Tagion uses the original Hashgraph protocol described by Leemon Baird to reach consensus, but differs in how it implements the ABP on top. The following 2 sections describes Tagion implements a new, fairer way to order events, and a way to send information, to achieve minimal communication complexity.

2.2 Fair Ordering

Tagion uses the Hashgraph to order all information passing through the system. The Hashgraph doesn't just reach byzantine agreement on event ordering; it is constructed around how to ensure a genuinely fair ordering of events. But what is a fair ordering? A naive approach would be to say the information is ordered according to the moment the information was added to any node. This would be problematic if a node doesn't share the event for a long time: When it finally shares it the network would have to insert it in the past potentially invalidating information added since. A fairer approach is to order an event according to when the network as a whole knows of the event. The Hashgraph does this through *famous witnesses*. A node is defined to become a witness [1] if it is communicating actively in the network and has received messages from most of the network recently. A witness further becomes famous, if it shortly afterwards has sent messages to most of the network. A node thus both needs to talk and listen to become a famous witness. Tagion uses these famous witnesses to represent the network as a whole. If the network is running efficiently every node will be a famous witness. An event is then ordered according to the average point of when the famous witnesses first learned of this event. Abstractly, an event is ordered according to the average point that the network as a whole heard of the event. Not only does the Hashgraph order all events in the network tolerant to byzantine failures; it also does this in a fair way. [2]

[1] It is actually a specific event from a node which is defined to be a witness event. For a technically precise definition we refer to: [1]

[2] For a more precise and technical definition of famous, witness, and ordering we refer to our documentation and the SWIRLDS Hashgraph paper

2.3 The Wavefront Protocol

So far, we have described nodes gossiping the history of communication the other doesn't know. Sending exactly what the other doesn't know is a technically difficult task. A simple solution is to simply send all information, but this would result in much duplicate information being shared across the network. This would greatly increase the communication complexity and become a bottleneck for the system. Tagion achieves this, while maintaining a asymptotic quadratic communication complexity which is the theoretical minimum, by using its Wavefront Protocol.

The Wavefront Protocol is used to exchange information between two nodes ensuring the minimal amount of overhead communication. Each created event has an altitude one higher than the one below it. Thus the altitude can unambiguously refer to a specific event by a certain node. Each node encodes the information it knows into a wave; the newest event it locally knows from each other node. The wavefront doesn't contain the newest information but rather numbers, the altitudes of the events representing it. By receiving the wavefront of another node you know exactly what it knows and can send just the right amount of information to the node. In the diagram below, node A and node B each have some amount of the newest gossip in their local Hashgraph. They each have a wavefront just covering all the events they locally know of. The Wavefront Protocol has three states: tidal wave, first wave, and second wave.

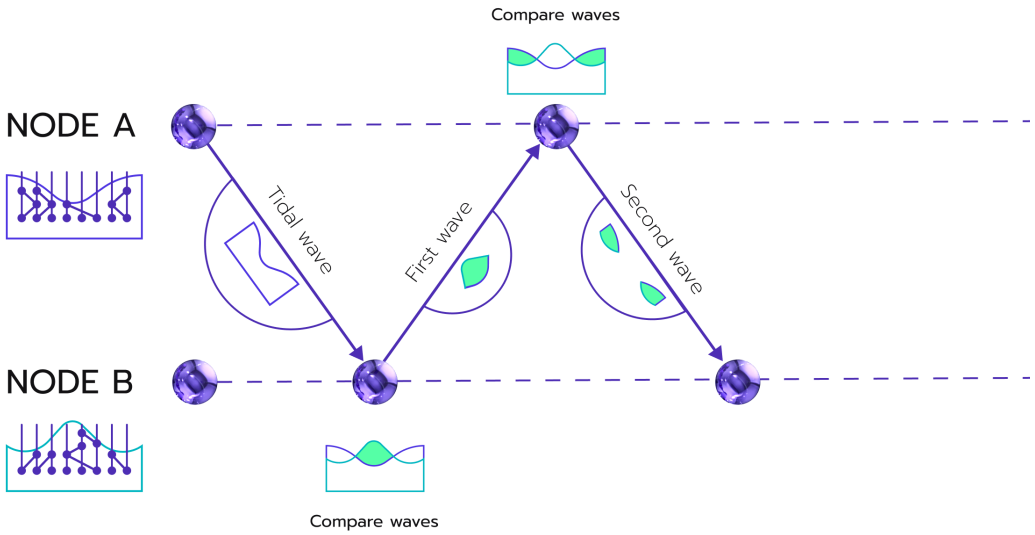

Figure 3: How the Wavefront Protocol works. Node A and B each has a wavefront representing what they know. By sharing and comparing their waves they can send exactly what is necessary.

1. Node A selects random Node B and sends its wavefront representing what it knows. This is called a tidalwave.
2. Node B receives a tidal wave from Node A, representing exactly what A knows. Node B compares its own wavefront with the tidal wave from node A, and sends back all events which are in front of the wave of Node A; that is, tell node A exactly all information it

does not know already. This is called a first wave.

3. Node A receives a first wave from Node B and saves all the new information to its local Hashgraph. Reasoning from the information node B sent, node A can deduce the wavefront of node B. Node A then compares its own wavefront with the wavefront of node B. It then sends node B with a second wave, sending all the information node B did not know already. Node B saves all of these to its local Hashgraph.

2.4 Communication Complexity

The Wavefront Protocol allows node A and node B to share exactly what is necessary and (almost) nothing else. Since Tagion is an asynchronous system delays can cause conflicting states to occur, which is fixed by nodes sending a breaking wave resetting the states. Delays can, in theory, also cause duplicate information to be sent, but this would rarely occur in practise. The Wavefront Protocol thus allows the Hashgraph protocol to be very close to the theoretically minimal communication complexity. The only overhead information that the Hashgraph shares is the tidal waves, signatures, breaking waves, and occasional duplicate information; all negligible compared to the actual information flowing through the system. As the network increases in size and throughput, the overhead becomes insignificant, achieving asymptotically optimal communication complexity. This allows Tagion to efficiently reach agreement and *write* information to the system. The next section describes how Tagion efficiently *reads* information from the system to achieve scalability.

3 Scalability

The following section describes how Tagion achieves scalability. A Hashgraph allows the system to add information quickly and with minimal communication. To read data, Tagion utilises the deterministic finality of the Hashgraph to make the system stateless. To do this efficiently Tagion uses the DART.

3.1 Writing Information

As mentioned in the previous chapter, the Hashgraph sends close to the minimum amount of information necessary. This reduces the possibility of communication bandwidth becoming the bottleneck of the system. This, in combination with the Hashgraph consisting of a fixed number of nodes, allows both consensus and ordering to happen with great speed - much greater than in any decentralised blockchain. In total, it is fast to *write* data into the system.

3.2 Reading Information

A system which reaches agreement fast does little good if it cannot communicate what it has agreed upon effectively. We discuss the limits of blockchain systems and how the Hashgraph overcomes these by using the DART and deterministic finality.

3.2.1 The Problem with Blockchain Systems

For blockchains to be fast, forking must happen often, resulting in one not being able to blindly trust the newest broadcasted block. A node wanting to read the state of the system must continually decide which blocks to trust and which to disregard, so significant effort is required to read the system's current state. Probabilistic finality is at the heart of this problem: A blockchain never becomes fully confident that a state will be accepted in the network, only more and more confident with time. Either new states are broadcasted often enough that there is doubt about the validity of the state (forking), or rarely enough that it limits the system's speed. The FLP-theorem[3] proves that no aBFT system can achieve *liveness* and finality; said differently, no aBFT system can *ensure progress* while simultaneously achieving finality (a guarantee that progress will not be overridden). Thus, blockchains cannot achieve finality - it is mathematically impossible. This lack of finality limits the system's scalability.

3.2.2 The Hashgraph's Solution - Finality

The Hashgraph as a consensus protocol prioritises finality above liveness. Initially, it might sound problematic that Tagion cannot be sure of progress, but it can, in fact, be certain that progress eventually happens with a 100% probability. Instead of being certain of progress and probabilistically sure it will not be changed (like blockchains), Tagion is probabilistically sure of progress and certain it will not be changed. Having finality comes with significant benefits:

3.2.3 Using Finality to Achieve Read-scalability

Having finality in the Tagion system allows for scalability in a way not possible in systems without it. When nodes in the Tagion network agree on a new state of the system, they push it to all listeners along with a signature from all nodes. Due to finality, all nodes can trust the broadcasted state just by verifying the signature. This makes the system stateless, meaning that any node only needs to remember the current state of the system. This is in contrast

to blockchain systems where the history of states defines the consensus; making it a necessity to store the history of the system. This increases hardware requirements to participate in the system, resulting in centralisation as a side effect. Since Tagion is stateless, it avoids these problems. It is thus easy to read the state of the system *if* we have an efficient and effective way to continuously verify the broadcasted signed states. To do this Tagion uses the DART, a database which retains the byzantine proof from the consensus layer.

3.3 DART

The DART is how Tagion stores the data in the network. The DART handles removal and addition of data, and computing *the bullseye* to represent a signature of the system. The main problem the DART solves, is fast recomputation of the bullseye after some parts of the database changes.

3.3.1 Structure

The DART is a database storing archives, a flexible data structure that can hold different types of files. At its core, the DART is a Distributed Hash Table (DHT), which means that archives are stored based on its hash-key. The hash of an archive thus determines where the archive is stored. The database can be represented as a circular structure: A dart-board with branches of data growing from the center. The more archives that are stored in the system, the more branches the DART creates. The following describes the DART through the proceeding example[3].

[3]For explanatory purposes, the example shows the DART with rim 1, 2, and 3. In practise, the first 2 rims are kept clear of archives and store only branches, and the DART also contains many more rims. The shown hashes are also 6 hexadecimals long, where as real hashes are 64 hexadecimals long.

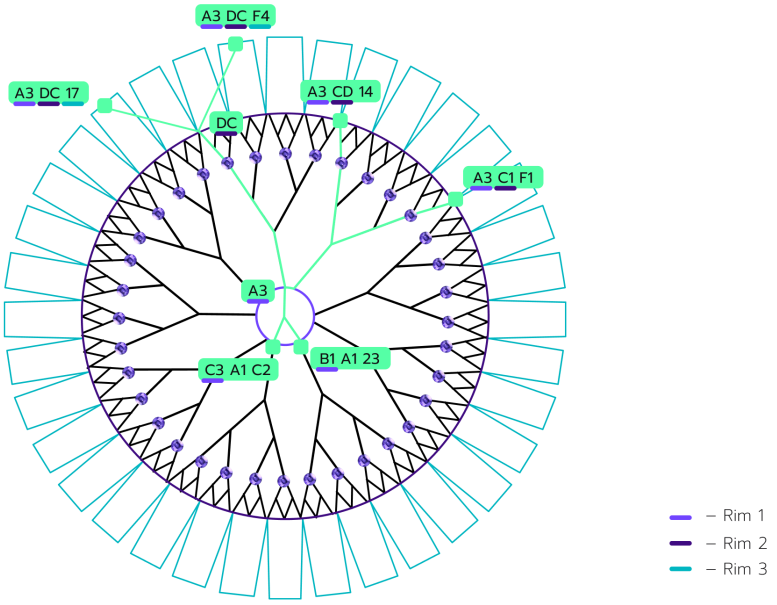

Figure 4: The structure of the DART database

The DART consists of multiple rims, the number of rims increasing with the total number of archives. The first 2 hexadecimals of the archive's hash determines where in rim 1 it is stored. If the first 2 hexadecimals are unique (e.g. C3 or B1 on the figure) this archive is stored at rim 1. On the other hand, if multiple archives share the first 2 hexadecimals of their hash (e.g. A3), then the dart creates a branch at A3 and stores this data on the next rim (e.g. A3-CD-14 and A3-C1-51). If multiple archives *also* share the third and fourth hexadecimal we create a further branch (e.g. DC). Each branch has at most 256 combined archives and subbranches. The DART adds and removes data in a way that ensures that the DART always has the minimum number of branches and rims. The more rims in the DART the more compute it takes to retrieve data, so it is essential to keep the number of rims minimal. The DART can maximally store 32 rims; using just 29 of these it could store every atom in the Milky Way in individual archives. So, storage capacity of the DART will never become a problem. Tagion also uses the DART to generate Unpredictable Deterministic Random (UDR) data, used to agree on truly random choices.

3.3.2 State of the System

The reason for using this structure is that it makes recomputing the signature of the system, the bullseye, very efficient. The DART can be viewed as a Sparse Merkle Tree (SMT). Each branch computes its hash based on all the hashes of its subbranches/archives (of which there are up to 256). To compute the bullseye, one needs to compute the hashes from the outermost rim and compute hashes inwards until one reaches the innermost rim: The bullseye. In the provided example one would first compute the hash of the archives in rim 3. Then we would compute the hash of the archives in rim 2, where the hash of branch DC is computed by combining the hash of A3 DC 17 and A3 DC 54. Finally it computes the hash of archives in rim 1 (where the hash of branch A3 is based on the hashes of branch DC, archive A3 CD 14, and archive A3 C1 F1), and then combines these to the bullseye. During the computation of the bullseye, one saves the

calculated hashes. This is the *crucial* step which allows the DART to recomputate the bullseye quickly. When the DART changes and one needs to compute a new bullseye, it is only necessary to recompute the hashes of the branches which changed, which will be a tiny fraction of the system in practise. For example, consider if only the archive B1-A1-23 changes. Computing the bullseye only requires hashing this single archive and combining rim 1 to get the bullseye, since all other hashes are saved from the last bullseye calculation. In this way one can continuously, efficiently calculate the bullseye: A signature of the entire system.

3.4 A Stateless Scalable System

Once consensus is reached in the Tagion system, nodes broadcast the state and its signed bullseye. Passive nodes, or any individual wanting to know the state of the system, can subscribe to these broadcasts. Using the DART, receivers can verify the bullseye quickly, with minimal computation. This results in a stateless system where we don't need to save previous states of the system. This allows for deletion of data (as we don't need to store data from a previous state) and results in storing less data overall. Not only does this make the system scalable, it also decreases the hardware requirements for participating in the network. Thus, the barrier to participate in the system is lower, increasing the degree of decentralisation. It is important to note that our system is open to the possibility of storing past states. This might be important for some actors who want to use the full history. In practise few nodes would store the entire history, while most nodes would only maintain the current state. In blockchain systems one is forced to save the history; in the Tagion system one has the *option* of saving it *only* if deemed necessary. In the next section we discuss how Tagion utilises *swapping* to achieve a fully decentralised system.

4 Decentralisation

So far the system we have described might appear very centralised: we have a fixed number of nodes gossiping to decide the state of the network. This is a permissioned system, no one can join the network without permission. To decentralise the network, it is essential to make the system permissionless allowing anyone to join the network. To achieve this, Tagion uses *swapping* to dynamically rotate nodes in and out of the Hashgraph.

4.1 Node Pools

To ensure scalability as the system grows, Tagion maintains a constant number of nodes in the Hashgraph. While such a network consisting of a fixed set of nodes offer greater decentralisation than standard systems, it can never claim true decentralisation. True decentralisation eliminates central points of control, distributing power across a continuously changing network of participants. The network thus needs to be permissionless. Rather than adjusting the size of the Hashgraph as nodes want to join and leave the network, Tagion employs the notion of an active and a passive pool. The active pool is of constant size, and it is purely nodes within this pool that participate in the Hashgraph consensus protocol. In contrast, the passive pool does not influence the consensus, and dynamically adjusts in size to accommodate nodes joining or leaving the system. The passive pool allows a permissionless way to join the network.

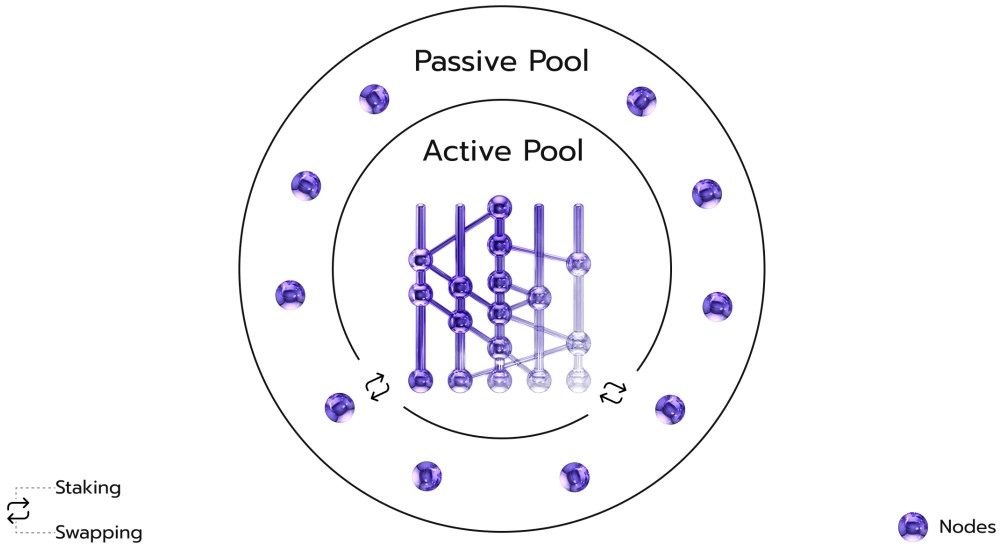

Figure 5: Tagion is made up of 2 pools: an active pool of constant size where the consensus protocol takes place, and a passive pool of dynamic size.

4.2 Swapping

As the network runs, Tagion continuously swaps nodes between the active and the passive pool. A random node from the active pool is swapped with a random one from the passive pool after a

fixed number of rounds (e.g. every 10^{th} round). The DART is used to agree on an Unpredictable Deterministic Random (UDR) choice. Optimally, nodes would have a precise agreement on when swapping occurs. But, due to the asynchronicity of the network, this cannot be guaranteed. If any node is significantly far behind the rest of the network, it might still receive messages from a node that the others have determined to be swapped out. The active pool is thus a local property, which might differ between nodes, though this would occur rarely in practise and for short durations. From a technical standpoint, leaving the active pool is straightforward, while joining is a slightly more complicated process. In rounds where a swap occurs, the nodes save the new active pool, and a set of *foundations events* in the DART. A joining node thus has a foundation it can built its own local Hashgraph on top of, and will quickly catch up to the rest of the active pool. The precise details for how nodes are *randomly* chosen to be swapped is described in section 5. Security.

4.3 A Permissionless System

In summary, to avoid power accumulating on a small set of nodes, Tagion continuously swaps nodes between the passive and the active pool. Thus, power distribution is always changing, removing central points of control. Furthermore, Tagion's utilisation of 2 pools makes the system permissionless: any node can, without permission, join the network through the passive pool. These 2 attributes makes the Tagion a fully decentralised system.

5 Security

The Hashgraph is an aBFT protocol tolerant to $\frac{n}{3}$ byzantine faults. The following section describes how this tolerance is kept when swapping nodes between the active and passive pool, and the incentive structure assuring security by using *time-based staking*.

5.1 Time-Based Staking

To join the Tagion Network through the passive pool, participants must stake a minimum fixed amount of Tagion tokens (TGN). Staking refers to investing some amount of TGN in the network for a predetermined amount of time[4]. This ensures node operators have a vested interest in the network, and also deters malicious actions since penalties can be enforced through slashing (reduction in staked TGN). The amount of TGN staked, combined with the staking time, determines ones chance of joining the active pool. This is contrast to being swapped out of the active pool, where the probability is uniform, and does not depend on your stake. This ensures that while nodes with more TGN participate in the Hashgraph more frequently, their voting power remains neither stronger nor more enduring than nodes with fewer TGN. This is contrast to most staking systems, where the amount of staked TGN determines the weight of your vote. Tagion's staking guarantees that consensus isn't unduly skewed by stake magnitude, promoting a decision-making ethos that prioritises equality and fairness.

5.2 Time and Amount

Your staked amount of TGN fundamentally determines your chance of being selected as an active node. The probability of joining the active pool increases linearly with every TGN staked. This method is preferred over a diminishing returns model. The latter could incentivise operators to launch several nodes, compromising network transparency by suggesting the system is more decentralised than it truly is. To foster enduring commitment among node operators and promote long-term stability within the network, Tagion utilises seniority. Those who maintain their stakes for prolonged periods, while continuously supporting the network throughout, are granted heightened seniority. This seniority enhances the probability of joining the active pool.

[4]It is also possible to restake ones staked TGN after this predetermined period ends. This, though, comes with the cost of losing some seniority (defined in a later section)

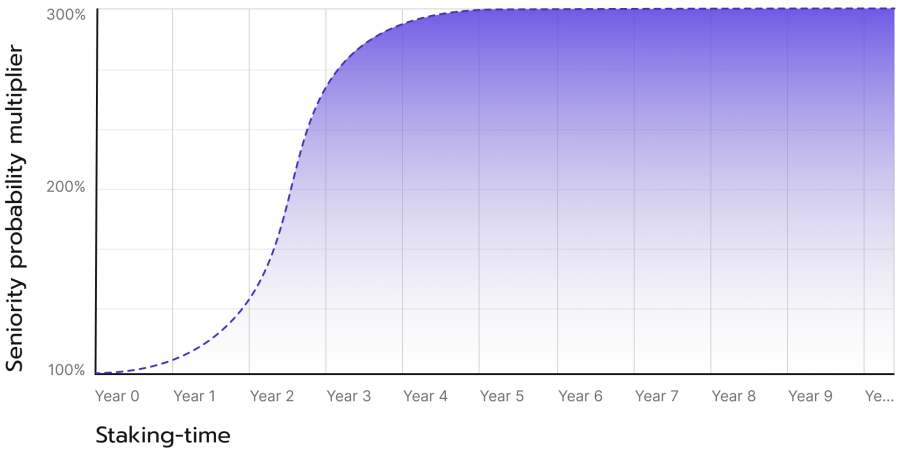

Figure 6: How seniority affects the probability of becoming an active node

The seniority probability multiplier follows an increasing S-curve. Initially, the probability of joining the active pool is purely determined by the amount of TGN staked. During the first 15 months, seniority gradually increases having some influence on probability. The following 15 months seniority increases more rapidly, effectively doubling the probability of joining the active pool through the first 30 months. Afterwards, the growth of seniority gradually diminishes and plateaus after 60 months at which point ones probability has tripled. This only holds if one has actively participated in the network throughout ones staking period. If a node regularly fails to communicate and engage in the network, it will be penalised through slashing. By making it plateau, it sets an upper cap on accumulated seniority, ensuring that power isn't disproportionately concentrated, keeping the decentralisation of the system. In summary, time-based staking fosters long-term engagement amongst node operators.

5.3 Staking Rewards

When nodes are swapped into the active pool during their staking period, they have the possibility of earning staking rewards. Each epoch, a random node is selected and receives such a reward. To incentivise fast communication and engagement, nodes receiving a reward are only chosen among famous witness nodes (an algorithmic trait defining nodes engaging actively). Thus nodes are encouraged to both receive and send data to make the Hashgraph to run as fast and smoothly as possible. To further increase this incentive, nodes only receive their rewards when they are chosen to leave the passive pool. This motivates continuous engagement throughout the time in the active pool, and nodes willingly swapping back out of the active pool. The rewards can either be paid out to the node operator, or added to the nodes currently staked TGN. To discourage operating many smaller nodes, staking rewards are increased slightly for nodes with more staked TGN. Otherwise, splitting ones stake into 2 pools of half the size, would result in better rewards,

since it allows one to have multiple nodes in the active pool simultaneously.

5.4 Sybil Attacks

Establishing a minimum staking threshold is a strategic move to curb the threat of Sybil attacks, where a single entity spawns multiple nodes to gain undue influence over the network. If the staking threshold is set too low, a malicious actor with many TGN might flood the system with nodes, posing a risk to the consensus process. To illustrate: When the staking limit is 10 MTGN (mega TGN, 1 million TGN), an individual holding 250 MTGN can only deploy 25 nodes. This is far from sufficient to pose a threat to the system. Conversely, if the staking threshold was lowered to 1 MTGN, they can deploy up 250 nodes. By elevating the staking requirement, the network limits the potential for single entity dominance, ensuring a more secure and decentralised system. However, if set too high, the threshold may dissuade individuals with fewer TGN from participating in the validation process, effectively centralising the network. The staking threshold thus needs to be large enough to prevent sybil attacks, but small enough to not dissuade smaller actors.

5.5 Proof of Byzantine Fault Tolerance [Coming Soon]

Coming soon...

6 System Architecture and Incentive Structure

The previous sections have described how Tagion solves the trilemma by utilizing its unique features. While the previous sections have delved into specific parts of the system, the following section gives a broader overview of the system as a whole.

6.1 Tagion from a User Perspective

If a user wants to use the Tagion system, the message must be communicated to one of the many nodes of the Tagion system. It is possible to build a relay system on top of the network, to make sure user messages are evenly distributed. A message can be anything from a transaction to a smart contract (a piece of executable code)[5]. The user must prove that its message is valid; that he owns the money he is using, or owns the file he is deleting. To do this the user submits a signature together with the message.

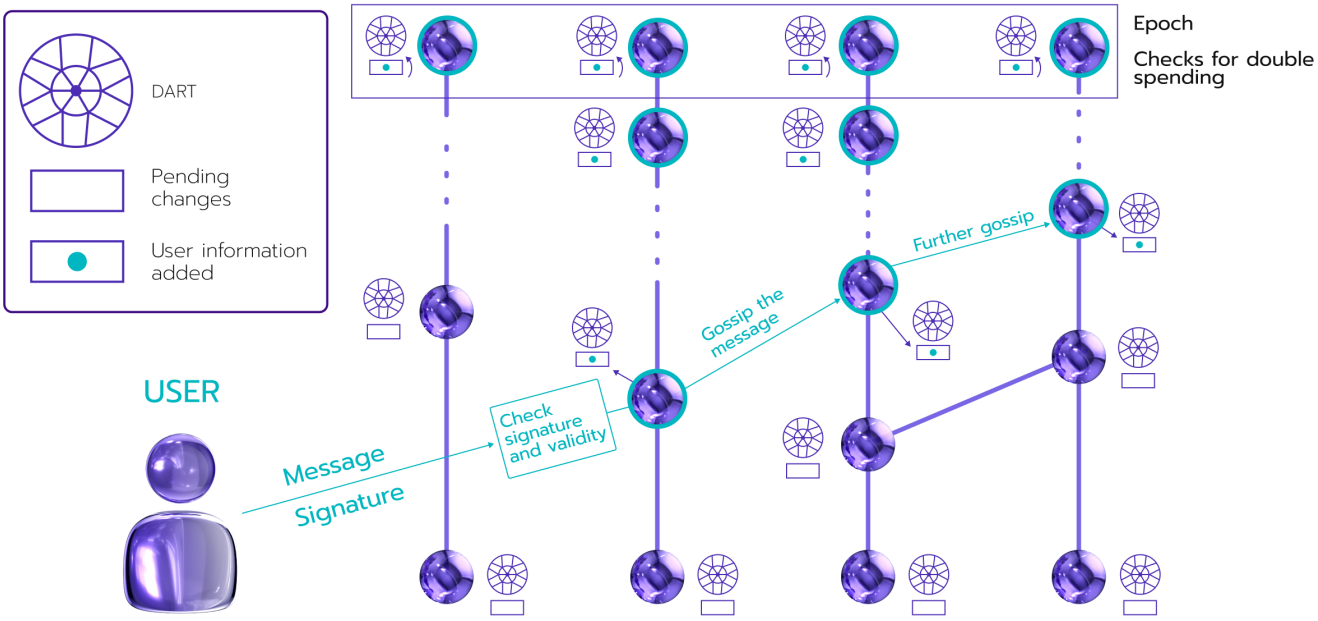

Figure 7: How a message flows from the user, until there is consensus about it in the network.

When a node receives a message it checks that the message is valid, and that the signature gives the correct permissions. If the node forwards an invalid message it will be penalized by the rest of the system, so it is incentivised to check the validity of the users message. When this check has been done, the node saves the user message in pending changes. As the network runs,

[5]All things in the network are technically smart contracts. A transaction is just a piece of code telling the system to move tokens from A to B.

gossip about the message spreads throughout the network. Whenever the message reaches a new node it does the same validity checks, and adds the message to its local pending changes. After some time, the message has spread to all nodes. When an epoch happens where every famous witness (actively participating) node has heard of the event, the event is permanently ordered with respect to other events. This allows the system to make a final check for double spending (that you have not spent the same bill twice) before it permanently writes the message from the pending changes into the DART. At this point the message has achieved finality: The message is registered, ordered, and is mathematically certain to never be overridden. The following two sections zooms in on what happens on a low level from a node perspective. Firstly, when nodes receive messages, and secondly, when epochs occur.

6.2 Node POV: Receiving Information

Nodes can receive information in two ways: Either from gossip inside the Hashgraph, or from user input. Gossip is transferred through the Wavefront Protocol described in section 3. Scalability, and user input is just sent directly to the node. Either way, the received data is handled the same. The node passes the information through the execution pipeline. This module reads the input of the different smart contracts from the DART. It uses this to check the validity of the provided signature, and the passed message is valid in general. Once the messages has passed all the tests, they are executed in the Tagion Virtual Machine (TVM). This virtual machine executes the smart contracts and adds the output of these to the pending changes. Note how the storage layer of the system is separated into a structure (the DART) outside the consensus layer.

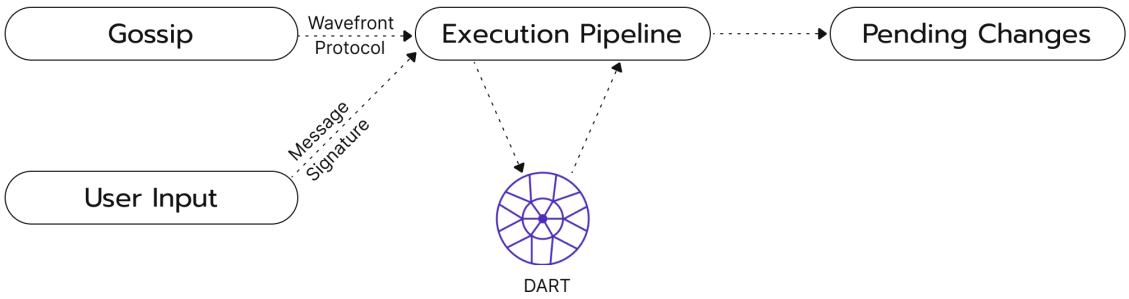

Figure 8: A technical close-up of how nodes handle receiving information.

6.3 Node POV: Epoch and Consensus

Epochs occur continuously as the consensus protocol runs, and is when events are finally and irreversibly agreed upon. Once an epoch occurs in the consensus protocol (the Hashgraph) this epoch is passed into an ordered execution. This module orders all the events contained in this epoch, and checks for double spending. It then passes all valid outputs to the DART. The DART deletes all inputs of the executed smart contracts, and writes all outputs of the smart contracts. At this point, all valid smart contracts have been executed on the DART. Once this has been done, a bullseye representing the state of the system is computed. This bullseye is then signed and gossiped to the rest of the nodes. This allows nodes to check their local DART against others, making sure the system is running smoothly, or figuring out if it has made a local error itself.

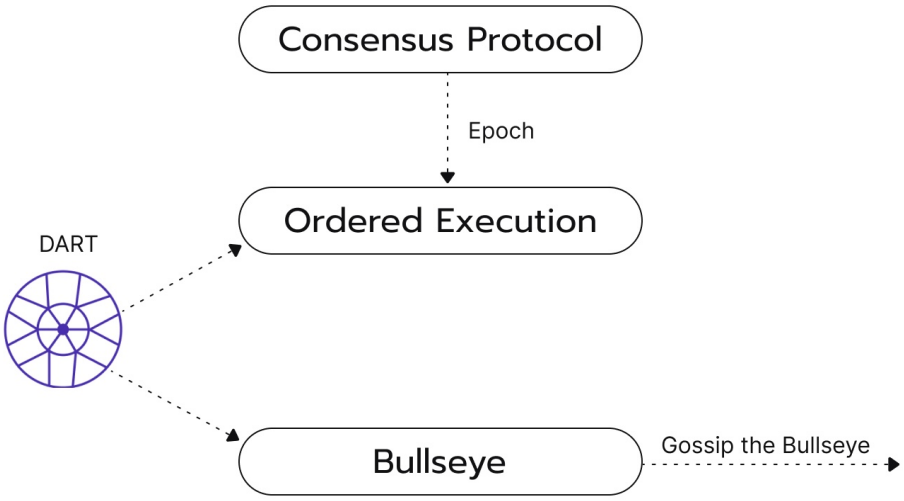

Figure 9: A close-up of what happens when an epoch occurs.

6.4 Comparison to Blockchain Systems

Blockchain systems fundamentally tie the consensus layer to the storage layer. The longest chain of stored data is what defines the consensus. Tagion is fundamentally different. In the Tagion system, the consensus algorithm and the storage layer are distinct parts. The consensus occurs in the Hashgraph, and the data is stored in the DART. This structure allows us to make the consensus layer, and the storage layer, separately much faster than they could be when jumbled together. This is what really allows Tagion to achieve scalability, without compromising on any parts of the trilemma.

Glossary

bullseye The merkle root of the DART, that is, a singdle hash representing the state of the entire Tagion system. 10, 11, 19

finality An attribute of a DLT-system where once a message is ordered its order never changes. A weaker form of (deterministic) finality, is probabilistic finality where the probability of the order changing goes toward 0. 8

liveness An attribute of a DLT-system that ensures the system continues to operate and make progress, even in the presence of faults or adverse conditions. 8

trilemma The challenge of simultaneously achieving security, decentralisation, and scalability in a DLT system. 1, 3, 17, 19, 20

Wavefront Protocol The communication protocol Tagion uses to achieve asymptotically minimal communication complexity. 6, 7, 20

Acronyms

aBFT asynchronous Byzantine Fault Tolerant. 1, 3–5, 8, 14

ABP Atomic Broadcast Protocol. 4, 5

DART Distributed Archive of Random Transactions. 1, 3, 8–11, 13, 18–20

DLT Distributed Ledger Technology. 1, 3, 20

SMT Sparse Merkle Tree. 10

TGN Tagion tokens. 14–16

TVM Tagion Virtual Machine. 18

UDR Unpredictable Deterministic Random. 10, 13

List of Figures

References

[1] LEEMON BAIRD. The swirlds hashgraph consensus algorithm: Fair, fast, byzantine fault tolerance. *SWIRLDS TECH REPORT*, 2016.

[2] Binance. What is the blockchain trilemma? *Binance*, 2022. Accessed: 2023-08-28.

[3] Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM (JACM)*, 32(2):374–382, 1985.