Department of Informatics
Technical University of Munich

TECHNICAL UNIVERSITY OF MUNICH

DEPARTMENT OF INFORMATICS

MASTER'S THESIS IN INFORMATICS

Analysis of Practical Permissionless PoS-based Consensus

Christian Kilb

Technical University of Munich

Department of Informatics

Master's Thesis in Informatics

Analysis of Practical Permissionless PoS-based Consensus

Analyse von Praktischem Zulassungsfreiem PoS-basiertem Konsensus

Author:	Christian Kilb
Supervisor:	Prof. Dr.-Ing. Georg Carle
Advisor:	Richard von Seck, M. Sc.
	Filip Rezabek, M. Sc.
Date:	January 15, 2022

I confirm that this Master's Thesis is my own work and I have documented all sources and material used.

Garching, January 15, 2022

Location, Date

Signature

Abstract

In presence of the increasing demand for more energy-efficient alternatives to Proof-of-Work consensus in the context of blockchain, Proof-of-Stake is a popular and promising consensus approach alternative. While Proof-of-Stake has theoretical advantages, evaluations of concrete implementations are sparse in practice. An exemplary Proof-of-Stake blockchain with real monetary transactions is Algorand. It comes with a new Byzantine Agreement consensus protocol and aims to provide decentralization, scalability and security at the same time and thereby solving the blockchain trilemma. This thesis contributes a practical analysis of the Algorand Proof-of-Stake realization. Specifically, the Algorand network infrastructure and consensus operation are evaluated. Based on a review of core aspects of the Algorand blockchain, a measurement system is designed and implemented with an own Algorand node at its core that is used for data collection. The practical evaluation of Algorand shows that the goals of performance, scalability and security can be considered fulfilled. However, the goal of decentralization is only achieved partially due to open issues with the Algorand reward model and relay core network. An exploration of solution ideas shows there is potential for improvement.

Zusammenfassung

In Gegenwart von steigendem Interesse an energieeffizienten Alternativen zu Proof-of-Work-Konsensus im Kontext von Blockchain ist Proof-of-Stake ein populärer und vielversprechender Ansatz. Während Proof-of-Stake theoretische Vorteil bietet, gibt es nur wenige Untersuchungen von konkreten Implementierungen in der Praxis. Eine exemplarische Proof-of-Stake-Blockchain mit realen Geldtransaktionen ist Algorand. Mittels eines neuartigen Byzantine-Agreement-Konsensus-Protokolls beabsichtigt Algorand die Ziele Dezentralisierung, Skalierbarkeit und Sicherheit gleichzeitig zu erfüllen und damit das Blockchain-Trilemma zu lösen. Diese Masterarbeit leistet als Beitrag eine praktische Analyse der Proof-of-Stake-Implementierung von Algorand. Im Spezifischen werden die Netzwerkinfrastruktur und der Konsensus-Betrieb von Algorand evaluiert. Basierend auf einer Untersuchung von Kernaspekten der Algorand-Blockchain wird ein Messsystem entworfen und implementiert. Zentraler Baustein dieses Systems ist ein Algorand Knoten, welcher zur Datensammlung verwendet wird. Die Evaluierung von Algorand in der Praxis zeigt, dass die Ziele Performanz, Skalierbarkeit und Sicherheit als erfüllt betrachtet werden können. Das Ziel von Dezentralisierung ist jedoch nur teilweise erfüllt auf Grund derzeit ungelöster Probleme im Hinblick auf das Belohnungsmodell und Basiskommunikationsnetz von Algorand. Eine Erkundung von Lösungsansätzen zeigt, dass weiterhin Potenzial zur Verbesserung existiert.

Acknowledgments

I would like to thank Prof. Dr.-Ing. Georg Carle for giving me once again the opportunity to write my thesis at the chair of network architectures and services. My gratitude also goes to my advisors Richard and Filip. Thanks to Richard for guiding me through my thesis yet another time and thanks to Filip for providing a third perspective. The continuous feedback was very helpful. I did enjoy the thesis journey and found the topic very interesting.

Contents

Chapter 1

Introduction

Blockchains are the foundation of cryptocurrencies. A cryptographically linked list of blocks, each containing a set of transactions, represents a public ledger. In a distributed permissionless blockchain system, a decentralized set of participants operate nodes, propose blocks and validate them. To find distributed agreement on the ledger state and new blocks, an appropriate consensus protocol is needed. Proof-of-Work is the original approach for permissionless cryptocurrencies to agree on new blocks. It was introduced by Bitcoin [1] and involves solving computationally expensive puzzles. Due to the high energy usage, demand for alternatives has arisen, with Proof-of-Stake being the most popular one. In Proof-of-Stake, the power of participants in the network is proportional to their amount of owned stake instead of their owned computing resources. Popular Proof-of-Stake approaches are Ethereum 2 [2] and Cardano [3]. Concepts that are commonly found in such approaches are stake bonding and delegation. That means, participation in the consensus protocol can require to lock in tokens or delegate the token power to another party. Algorand [4] is another Proof-of-Stake blockchain that avoids these restrictions. Additionally, it aims at solving the "blockchain trilemma" of unifying the three properties decentralization, scalability and security.

While Proof-of-Stake consensus offers advantages in theory, analysis results of concrete blockchain realizations are sparse. The goal of this thesis therefore is to analyze one specific Proof-of-Stake implementation in practice: the one of Algorand. Main objectives are to research the consensus protocol operation and infrastructure of Algorand in order to understand how the theoretical advantages of Proof-of-Stake translate into practice. The Algorand blockchain aims to provide as main features decentralization, security, scalability and performance. In this thesis, a subset of these measurable Algorand goals is analyzed. Additionally, existing challenges are identified and solution ideas explored.

The research questions to be addressed with this thesis are:

RQ1 To what extent have the ideas proposed by Algorand been put into practice?

RQ2 In a practical Algorand node setup, do the experimental observations match the performance, scalability and security expectations set by Algorand?

RQ3 What open issues can be identified when evaluating the Proof-of-Stake realization of Algorand?

In order to answer these questions, core aspects of the Algorand Proof-of-Stake implementation must be understood first. Reviewed for example are the general goals of Algorand, the structure and dynamic of the Algorand network, the exchanged messages and the message propagation mechanism. By setting up and operating an own Algorand node, data about the Algorand network and consensus operation can be collected and evaluated. A combination of these practical observations with the theory of Algorand then allows to draw conclusions about the posed research questions.

The thesis is structured in the following way: First, background information is given in Chapter 2 about blockchain, Proof-of-Work, Proof-of-Stake and Algorand. Then, the Proof-of-Stake realization of Algorand is analyzed in various ways in Chapter 3. Algorand goals are outlined, measurable data is identified, open Algorand issues are discussed and possible solution approaches presented. In Chapter 4, a measurement setup is designed with an own Algorand node at its core. Details about its implementation are given in Chapter 5. The collected data is evaluated in the subsequent Chapter 6. At the end of the thesis, related work and future work is outlined in Chapter 7 and 8 respectively before the conclusion is drawn in the last Chapter 9.

Chapter 2

Background

This chapter first gives an overview of the background on blockchains and Proof-of-Work consensus. It then elaborates on Proof-of-Stake and describes popular implementations. Finally, a more in-depth background is given about the Algorand blockchain.

2.1 Blockchain

Blockchains are the foundation of cryptocurrencies, where they are used as distributed public ledger [5]. A blockchain is a cryptographically linked list of blocks, each containing a list of transactions. Block headers contain the hash of the previous block, a combined transaction list hash in form of a Merkle tree [6] root hash and other block metadata. A decentralized blockchain data structure is operated by a set of nodes participating in the blockchain. They keep track of the blockchain, verify the validity of transactions and blocks and can participate in the periodic formation of new blocks. The way nodes agree on these new blocks is defined by the consensus protocol that the blockchain uses. A concept of distributed agreement on the ledger state is central to the security of the system and protects against double spending of currency.

2.2 Proof-of-Work

Proof-of-Work is the consensus approach used in the first blockchain Bitcoin [1], which was introduced in 2008, and has since seen widespread adoption in other blockchains. In this model, proposers of new blocks have to solve computationally expensive cryptographic puzzles. So called *miners* compete in the production of new blocks. The first miner to find a new valid block by solving the computational puzzle is then rewarded with

tokens. This gives participants influence on the blockchain in proportion to their invested computational resources.

With this Proof-of-Work approach, a new block at the end of the blockchain is not immediately final. It could be annulled, if the chain *forks* before that new block and the network pursues the competing chain fork that does not contain the one block. As forks could always occur, blocks never reach a truly final state. However, the probability that a block stays valid and can be effectively treated as final increases with each new block appended to the corresponding fork of the chain. In case of forks in the blockchain, the network chooses to follow the longest chain, e.g. the one with most invested computing power [1]. Next to blockchains with probabilistic Proof-of-Work consensus finality, blockchains with immediate block finality are also possible by utilizing a Byzantine fault-tolerant consensus protocol [7].

Due to its high computational cost, Proof-of-Work has been criticized for its high energy usage and has created a demand for other blockchain consensus protocols. As of 4th November 2021, Bitcoin is estimated to consume between 48 TWh and 186 TWh of energy per year, which equals the energy consumption of entire countries [8].

2.3 Proof-of-Stake

Proof-of-Stake is an alternative consensus approach that is gaining popularity [9, 5]. The concept was introduced in 2012 by the cryptocurrency Peercoin [10]. In Proof-of-Stake, the influence of users on the blockchain is directly proportional to their monetary stake in the system. Voting power on new blocks therefore scales with the owned amount of that cryptocurrency. As no cryptographic puzzles have to be solved in Proof-of-Stake, the energy cost is much lower. A lower computational cost can also lead to lower transaction fees. Typical problems that Proof-of-Stake models have to solve are the Nothing-at-stake problem and Sybil attacks. As consensus participation does not come with significant computational costs, participants could vote for multiple competing blocks in the same round. This could lead to multiple blockchain forks to be supported simultaneously, which is undesirable. Possible solutions for this problem are a penalty mechanism or a non-fork guarantee. Similarly, due to low computational costs, an attacker could create multiple Sybil identities and participate with them in the consensus protocol. The mechanism should therefore be designed in such a way that an attacker cannot gain an undesirable advantage by operating multiple identities.

[1] `https://ethereum.org/en/developers/docs/consensus-mechanisms/` (Accessed: 2021-11-05)

2.3.1 Taxonomy

The creators of the Algorand blockchain identify three subtypes of Proof-of-Stake blockchain approaches [1]. In the *Bonded* Proof-of-Stake approach, consensus protocol participants are required to lock in some of their tokens as collateral. A penalty mechanism then discourages undesirable or malicious behavior in the protocol. The *Delegated* Proof-of-Stake approach features a more centralized voting process. Users delegate their voting power to a smaller subset of participants. These delegates then vote on behalf of the users. In the *Pure* Proof-of-Stake model, every user is given the ability to participate in the consensus protocol without having to lock in their tokens or delegate their votes. Algorand classifies itself as *Pure* Proof-of-Stake blockchain.

2.3.2 Popular Implementations

As of 26th October 2021, the most popular Proof-of-Stake implementations in terms of market capitalization are Ethereum 2, Cardano, Polkadot, Avalanche and Algorand, according to Coinbase [11].

Ethereum 2

Ethereum [2, 12] is a Proof-of-Work blockchain launched in 2015 that is currently in a multi-step process of transitioning to a Proof-of-Stake consensus mechanism. The consensus protocol of Ethereum 2 follows a Bonded Proof-of-Stake model. Users have to lock in some of their stake to be able to participate in the protocol as *validators*. New blocks are proposed and verified by randomly chosen validators. Good behavior in the network is incentivized by rewarding validators as well as with a *slashing* mechanism, which causes users to lose their locked in stake if malicious behavior is detected. The protocol is secure if more than two thirds of stake is controlled by honest users. In the long term, the Ethereum 2 blockchain is planned to consist of multiple *shard chains* and a coordinating *beacon chain* to further increase the performance and scalability of the network.

Cardano

Cardano [3] is a Proof-of-Stake blockchain launched in 2017 that uses a consensus protocol called Ouroboros. It follows a Delegated Proof-of-Stake model. Users delegate their stake to *stake pools* in order to indirectly participate in the consensus protocol. Stake pools participate in the consensus with the corresponding combined stake. Stake pools are randomly chosen to generate new blocks based on the amount of total stake

[1] `https://www.algorand.com/technology/pure-proof-of-stake` (Accessed: 2021-10-24)

in their pool. In exchange, the pool operators as well as the pool members are given rewards to incentivize participation. For security, Cardano requires that at least 51 % of stake is controlled by honest users. As stake pool operation comes with significant technical requirements, the blockchain is only decentralized in a limited way.

Polkadot

Polkadot [13] is a blockchain launched in 2020 that focuses on interoperability with other blockchains for cross-chain token and assert transfer. Similar to Ethereum 2, Polkadot pursues a multi-chain sharding approach with one main *Relay Chain* and multiple *parachains*. They classify their consensus protocol as *Nominated* Proof-of-Stake, which can be seen as a mix of Delegated and Bonded Proof-of-Stake. Users in the network can take on the role of a *validator* or *nominator*. Validators operate nodes and actively participate in the consensus protocol. Nominators elect and support validators. A validator forms a *validator pool* together with the corresponding nominators. Both validators and nominators are required to lock in some of their stake as collateral to participate in the protocol. Consensus participation yields rewards in return. Misbehavior is penalized with a *slashing* mechanism, which causes validators as well as supporting nominators to lose their stake. The amount of rewards a validator pool receives is independent of its associated stake. The motivation behind this incentive strategy is to increase the decentralization of the network. A minimum stake requirement for validator pools helps at protecting against Sybil attacks. The protocol is secure if more than two thirds of elected validators are honest.

Avalanche

Avalanche [14, 15] is a Proof-of-Stake blockchain launched in 2020 with a unique consensus approach based on *metastability*. The protocol is leaderless and operates without block proposers. Instead, every participant or *validator* maintains their local view of which transactions to accept or reject. To align their transaction acceptance decision with the network, a random *sub-sample* of other validators is queried and asked for their opinion. Participants then adopt the majority opinion of the corresponding responses. This process repeats multiple times and converges to a certain preference until an opinion-threshold is reached, at which point the transaction acceptance decision is treated as final. To protect the network against Sybil attacks, validator nodes have a minimum stake requirement. Additionally, the amount of stake of a validator influences the probability of being queried by others, so that high-stake validators are queried more often. Validators are also required to lock in some of their stake. “Sufficiently correct and responsive” validators are then rewarded for their participation. There is however no slashing mechanism that causes stake loss in case of misbehavior. Users are also able

to participate in the protocol indirectly by becoming a *delegator*. Delegators lock in some of their tokens to support a validator node and receive a portion of the validator rewards in exchange. In total, Avalanche can be seen as lightly Bonded Proof-of-Stake approach, as tokens need to be locked in, but without the risk of losing them due to a penalty mechanism. The Avalanche protocol includes a tunable security parameter. In its current configuration, Avalanche is supposed to tolerate attacker powers of up to 80 % at a negligible probability of failure.

2.4 Algorand

Algorand [16, 4] is a cryptocurrency that implements a Proof-of-Stake consensus algorithm. It is being developed at the Massachusetts Institute of Technology since 2017 and launched in 2019. Algorand comes with a new Byzantine Agreement consensus protocol and aims to solve the "blockchain trilemma" of unifying the three properties decentralization, scalability and security. In its current operation, the blockchain handles around 1 million transactions per day and confirms new blocks as final within 5 seconds [17].

2.4.1 Gossip Network

The Algorand blockchain is run by a distributed set of Algorand nodes that connect to each other to form a peer-to-peer overlay gossip network [18]. Over this network, the peers exchange gossip messages with broadcasts. A specific subset of nodes called *relay nodes* build the core of the gossip network. Their main purpose is to relay gossip messages and to help other nodes to join and synchronize with the network. The remaining nodes are *non-relay nodes*, which are also referred to as *client nodes* in this thesis. Relay nodes connect to other relay nodes and accept incoming connections from other nodes. Client nodes only connect to relay nodes. Figure 2.1 visualizes the Algorand network structure in a simplified way with a set of relay nodes (R) and client nodes (C). By default, each node currently maintains 4 outgoing connections to different relay peers. The list of domain names and IP addresses of these relay nodes is publicly available via DNS as SRV records. This relay node list is used by the nodes to know to which relay peers they can connect to. Currently, the Algorand corporation controls the corresponding DNS zone and therefore has control over which node is allowed to be a relay node.

The Algorand blockchain gossip network that processes transactions with real monetary value is called *MainNet*. This main network is however not the only one. There is also an Algorand test blockchain in the so called *TestNet*. There, blockchain applications can be tested before deploying them to the MainNet. The TestNet has its own set of relay nodes, but otherwise operates similarly to the main network. New Algorand protocol

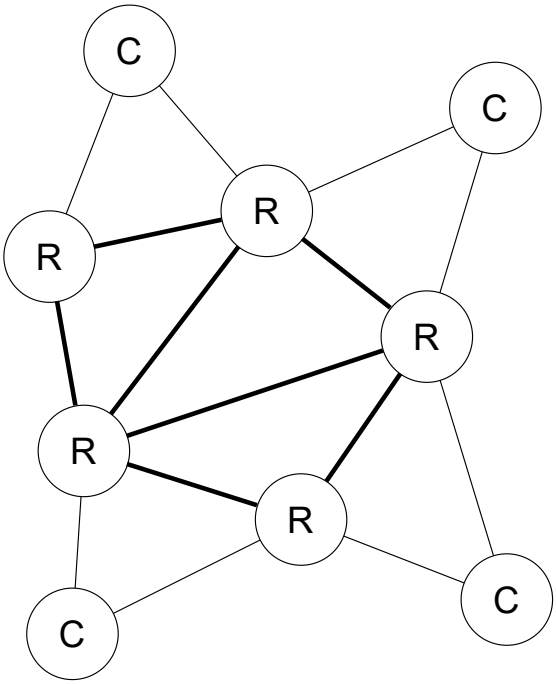

Figure 2.1: Algorand network structure (simplified)

versions are tested in yet another network: the *BetaNet*. It is also possible to operate private networks with an own set of relay nodes which are independent of the official set of relay nodes. The foundation of each network is the initial genesis block, in which the initial Algo token distribution is specified.

2.4.2 Consensus Mechanism

Algorand implements a decentralized Byzantine Agreement Proof-of-Stake consensus protocol [19, 16, 20]. It assumes that the supermajority of stake ($> 2/3$) is controlled by honest users. In exchange, it guarantees partition tolerance and safety, such that if honest users come to an agreement, they all agree on the same value with overwhelming probability. Forks can therefore occur only with negligible probability. In case of network partitioning, the protocol assures asynchronous safety and is able to recover from partitions. Liveness is guaranteed if the network is not partitioned and honest messages are received by other honest users within some bounded delay.

In the context of the Algorand blockchain, the consensus protocol is used to agree on the next block. Each *round* in the protocol produces a new block. At the beginning of a round, proposals for new blocks are being made as first *step* of the round. Afterwards, the users vote on the proposals in two subsequent voting steps. Proposals and votes are broadcasted over the gossip network. In the *soft vote* step, the proposals made in the

first step are narrowed down to the one proposal with highest priority. The priorities of proposals are pseudo-random values derived from the proposal messages and are protected against manipulation by an attacker. In the subsequent *cert vote* step, that one proposal is then confirmed as the accepted proposal and becomes a new block. For each vote to be successful, a supermajority of around 75 % must be reached. If the vote for a new block was unsuccessful, another attempt is started as new round *period*. Rounds therefore consist of one or more periods and periods consist of the described proposal and voting steps.

To assure scalability of the protocol, only a subset of all users is eligible to propose new blocks and vote on the proposals in each round. The set of eligible users is called the *committee*. After each step in the protocol, the committee changes and a new pseudo-random subset of users is selected to be on the committee for the next step. This assures decentralization of the protocol.

In each step, the users can check for themselves whether they have been selected as committee member without network interaction. That way, the members stay unknown to the network until they have submitted their proposals or cast their votes. This self-selection approach prevents targeted attacks against the committee. To implement this idea, the cryptographic concept of *Verifiable Random Functions* (VRFs) [21] is used. Given a publicly known *sortition seed* value and the user's private key, a user can check their committee membership and compute a proof for it. Should the user be part of the committee and send a consensus message, this proof would be sent alongside the message. When other users receive a proposal or vote, they are able to verify the committee membership proof, using the public seed and the public key of the sender. The pseudo-random proof data is also used to assign priorities to each proposal. Users can find the sortition seed in the block of the previous round. Block proposers advance this seed value pseudo-randomly by deriving a new seed from the previous seed, the round number and their private key.

The probability for a user to be selected as committee member is proportional to the stake owned by that user. Members of the committee however do not all have the same voting power by default. The *weight* of a proposal or vote is yet another output of the cryptographic sortition mechanism. Vote weights are expressed in the same unit as stake and specify which portion of own stake is eligible for voting. Higher stake therefore translates into a higher expected voting power.

The expected size of the committee, which is the expected amount of total sub-stake eligible for voting, and the voting majority threshold are hard-coded values that have been mathematically calculated with probability theory. The probability to be selected

as committee member is dynamically adjusted each round based on the current amount of participating online stake. More online stake in circulation leads to a lower selection probability in order to meet the expected committee size number on average. The actual committee size in each round is therefore a probabilistic value. The developer documentation [18] is inaccurate in their description of this mechanism, as they suggest in one place that the voting threshold was a dynamic value: "[...] the network uses the online/offline status of an account to calculate block vote thresholds" [1].

In order to participate in the consensus protocol, one has to own stake and operate an Algorand node. The node has to run in participation mode, which requires the creation of a participation key and its registration with the network. A node can be configured to vote on behalf of multiple accounts by registering one participation key for each account. Committee membership checks and vote transmissions are then handled separately for each account.

2.4.3 Reward Model

In Algorand's current reward model, all regular wallet accounts with a stake of at least 1 Algo regularly receive rewards in proportion to their stake [22, 18, 20]. This gives new users an incentive to join Algorand. Whether they participate in the consensus protocol or run a client or relay node does not influence their gained rewards in any way.

Algorand accounts can be in three different on-chain states. The default state is *Offline*, which indicates the account is not participating in the consensus protocol but is still eligible for rewards. Should the account participate in the consensus protocol, it is in the *Online* state, where it is also eligible for rewards. Next to these two regular account states, there is the *NotParticipating* state, in which the account does not participate in the consensus protocol and is also not eligible for rewards. The special *FeeSink* and *RewardsPool* accounts maintain this *NotParticipating* state for example. Accounts can choose to enter the *NotParticipating* state, but cannot exit it.

Two special Algorand addresses exist that are used in the reward model. The *FeeSink* receives all transaction fees. This transfer of fees to the sink address happens implicitly and automatically whenever the blockchain is extended by a new block. The other special Algorand account is the *RewardsPool*. From this pool, token rewards are distributed to all the regular accounts as specified and implemented by the reward distribution protocol. The *RewardsPool* cannot make ordinary transactions.

[1] `https://developer.algorand.org/docs/run-a-node/participate/` (Accessed: 2021-10-25)

The *FeeSink* can only make transactions to the *RewardsPool*. When looking at the transactions made from the *FeeSink* however on the Algorand Blockchain Explorer [17], no outgoing transactions from the *FeeSink* ever occurred. Additionally, the amount of Algos in the *FeeSink* seems to match the total amount of transaction fees of all historic transactions. There is also no automatic transfer between these two accounts implemented. The transaction fees collected by the *FeeSink* are therefore effectively taken out of circulation. It can be assumed that Algorand controls the *FeeSink* account and did not yet choose to add these fees back into circulation, which is however inconsistent with the source code comment "The RewardsPool accepts periodic injections from the FeeSink [...]" [20].

Instead, the *RewardsPool* is populated every 500 k blocks with newly minted Algos [1] by the Algorand Foundation. Over the course of the next 500 k blocks, the tokens in the *RewardsPool* are linearly distributed as rewards. A reward rate variable is calculated that specifies how many MicroAlgos should be distributed as rewards on average per block to all eligible accounts. Wallets with higher stake receive a proportional higher amount of rewards. As optimization, the received rewards are not actually added to the account until the account is updated, for example due to a transaction from or to this account. This means that there are no automatic rewards on the pending rewards. To receive the maximum amount of rewards, it is therefore necessary to assure regular account updates, with empty transactions for example. When calculating the frequency of such updates, the transaction fees have to be taken into account. With this reward distribution model, performance takes priority over usability when looking at the reward maximization strategy.

Figure 2.2 visualizes the current reward model and its flow of Algorand tokens.

2.4.4 Protocol Upgrades

The Algorand blockchain features a consensus protocol upgrade mechanism to enable modifications to the protocol itself [20]. While Algorand Inc. develops their Algorand node software and designs and implements changes to the protocol specification, the decision to accept protocol changes is a decentralized one. Similar to how the participation nodes vote on new blocks, they also vote on protocol upgrades directly on-chain.

Every block proposer has the ability to propose a new protocol version and start a voting period, unless one is already ongoing. In the voting period, subsequent proposers vote either in favor or against the proposed version. Every new block therefore represents one

[1] Algos minted at genesis, controlled by Algorand, but not in circulation so far

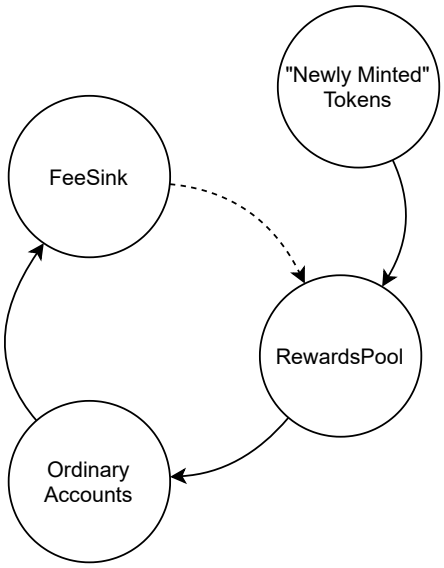

Figure 2.2: Current Algorand reward model

protocol version vote. The voting period is currently set to 10 k blocks, which is around 12 h real time with a block time of around 4.4 s. For the vote to be successful, 90 % of votes must be in favor of the change. If the vote was successful, a transition period of up to one week starts that allows nodes to apply the protocol changes before they become active. After the voting period has concluded, a new protocol upgrade proposal with subsequent voting period can be made by any proposer.

Operators of Algorand nodes signal their support of protocol changes by upgrading the Algorand software that is running on their nodes. The node software votes in favor of protocol changes if it implements the proposed protocol specification. Node operators should be aware that if they configure automatic Algorand software upgrades, they will by default vote in favor of new protocol versions.

Chapter 3

Analysis

This chapter first gives an overview of the goals of the Algorand blockchain, so that the progress of Algorand towards certain goals can be evaluated later. For the purpose of effective measurements, it then describes which Algorand node data sources are available for evaluation and how relay nodes work specifically. Based off a review of the Algorand design properties and source code, a first set of core issues is identified. Finally, solution ideas for these issues are presented and explored.

3.1 Algorand Goals

On their websites (as of 19th October 2021) [4, 22], Algorand describes the general capabilities and goals of their blockchain.

Algorand claims to be the "[...] first blockchain platform to solve the trilemma of decentralization, scalability, and security" [1].

3.1.1 Decentralization

The Algorand blockchain is intended to be decentralized both on network as well as consensus level. It is described as "[...] entirely decentralized, which means there is no powerful central authority or single point of control" [2]. On consensus level, the permissionless Proof-of-Stake consensus protocol allows users to join the system and participate with modest hardware resources and low computational costs. On network

[1] `https://www.algorand.com/technology/algorand-network-architecture`

[2] `https://www.algorand.com/technology/technical-advantages`

level, decentralization is achieved by maintaining a diverse set of node operators with different backgrounds and geo-locations. Algorand promises that "[a]ny user is free to register as a relay or participation node" [1]. While the blockchain is advertised as "truly decentralized network", Algorand hints at this goal not being reached yet with the phrase "path to decentralization" [2]. This is confirmed on another website of Algorand [3], where their plan of "responsible decentralization" is outlined.

3.1.2 Security

The Algorand blockchain needs to provide defense against attackers on consensus level as well as network level. Algorand's Byzantine Agreement consensus protocol guarantees security as long as more than two thirds of stake is controlled by honest users. The random, self-selected and initially secret consensus committee design offers protection against targeted attacks. The protocol is able to tolerate and recover from network partitions. As the Algorand source code follows the open source development model, it can be independently verified.

3.1.3 Scalability

Scalability is important on the network level and consensus level. Over time, more nodes might join the network, more users choose to participate in the consensus protocol and the frequency of transactions could increase. The consensus committee approach with a limited number of self-selected participants in the committees assures scalability on the consensus protocol level. An efficient gossip network communication model enables scalability on the network level.

3.1.4 Performance

New transactions are confirmed by the Algorand blockchain within a few seconds. The non-fork property of the Algorand protocol allows new blocks to be finalized quickly. The relay nodes in the gossip network provide a performant core to the network so that consensus messages are propagated quickly and efficiently to participating peers. The Proof-of-Stake nature of the protocol keeps the computational costs low, thus offering a energy-efficient blockchain solution.

---

[1] `https://www.algorand.com/technology/core-blockchain-innovation`

[2] `https://www.algorand.com/technology/algorand-protocol`

[3] `https://algorand.foundation/algorand-protocol/network`

3.1.5 Rewards

Owning stake is currently incentivized by Algorand with financial rewards so that new users join the blockchain [1]. While not explicitly mentioned in the rewards section of the Algorand website, a proper reward model for consensus participation and node operation can be seen as desireable implicit goal of the Algorand blockchain, as it is necessary for the long-term stability of the system and has a great effect on its decentralization.

3.1.6 Evolution

Modifications to the protocol and therefore a long-term evolution of the system are enabled by a protocol upgrade voting mechanism.

3.2 Algorand Node Data Sources

In order to perform measurements with an Algorand node and evaluate the gossip network, the available data sources must be understood first. A running Algorand node process produces data in multiple ways.

The most information about the node internals can be gained from the main log file called `node.log`. This log gives insights about the peer connections, consensus operation, network performance and more.

The Algorand `carpenter` tool is able to read the log file and visualize the progress of the consensus operation. It provides live information about consensus events such as the acceptance of a proposal or a reached voting committee threshold.

Algorand nodes also produce telemetry events. Node operators can choose to automatically send the telemetry data to Algorand on an opt-in basis. Telemetry events can also be redirected to an own telemetry server or to the main `node.log` file. Common telemetry events are peer connection change events and node heartbeats.

A local Algorand node can additionally be queried actively over a REST API, which is enabled by default. This API [2] mainly provides on-chain information such as Algorand accounts or blocks, but can also be used to post new transactions. The performance of this API can be enhanced with an optional indexer, which would make use of a connected database.

---

[1] `https://www.algorand.com/technology/algorand-protocol`

[2] `https://developer.algorand.org/docs/reference/rest-apis/algod/v2/` (Accessed: 2021-10-31)

As most insights about network and consensus operation can be gained from the main `node.log` file, it is the data source of central interest. The Algorand node log file provides detailed live information about node internals, node status changes, peer connection changes, consensus agreement progress and network performance. Each log entry is formatted as JSON and accompanied by a timestamp. The level of detail can further be increased by adjusting the log level in the node settings file.

3.2.1 Network Log Events

Out of the network change log events, `ConnectPeer` and `DisconnectPeer` events are the most relevant. They are generated each time a new outgoing or incoming peer connection is established or an existing connection is terminated. Outgoing connections are created between the local node and relay nodes. Incoming connections from other client or relay nodes to the local node can occur only when running in relay mode.

An example `ConnectPeer` log entry is shown in Listing 3.1. It contains network details about the newly connected peer such as IP address, host name and port. It also specifies whether the new connection is an incoming or outgoing one. The instance name of the peer can be observed as well.

Listing 3.1: Example `ConnectPeer` log entry

```
{
    "details": {
        "Address": "34.126.180.253",
        "HostName": "b768f568-affc-43d0-bd38-2d5b0a54eaa4:sing-alg-testrelay-2.
            prod.purestake.tech",
        "Incoming": false,
        "InstanceName": "g2/tVQo0Po5Y2wni",
        "Endpoint": "singaporeg-algorand-test-2.algorand-testnet.network:4161"
    },
    "file": "telemetry.go",
    "function": "github.com/algorand/go-algorand/logging.(*telemetryState).
        logTelemetry",
    "instanceName": "VDXe4dsfQmWTJHDL",
    "level": "info",
    "line": 212,
    "msg": "/Network/ConnectPeer",
    "name": ":4161",
    "session": "",
    "time": "2021-07-10T19:37:16.293571+02:00"
}
```

`DisconnectPeer` log entries contain similar network address information. An example is shown in Listing 3.2. Additionally, a reason for the disconnect is given.

Listing 3.2: Example `DisconnectPeer` log entry

```
{
    "details": {
        "Address": "34.127.77.138",
```

```
4           "HostName": "110524a6-55e2-44f0-9b99-e2dff81819f8:org-alg-testrelay-1.
                prod.purestake.tech",
5           "Incoming": false,
6           "InstanceName": "Q0qSM49fAu+wMF29",
7           "Endpoint": "oregong-algorand-test-1.algorand-testnet.network:4161",
8           "MessageDelay": 592246872,
9           "Reason": "LeastPerformingPeer"
10      },
11      "file": "telemetry.go",
12      "function": "github.com/algorand/go-algorand/logging.(*telemetryState).
            logTelemetry",
13      "instanceName": "VDXe4dsfQmWTJHDL",
14      "level": "info",
15      "line": 212,
16      "msg": "/Network/DisconnectPeer",
17      "name": ":4161",
18      "session": "",
19      "time": "2021-07-10T19:37:15.746036+02:00"
20  }
```

The full list of disconnect reasons can be found in the Algorand source code file `network/wsPeer.go` [20]. An understanding of these reasons is achieved by exploring the source code in more detail. Table 3.1 explains the different disconnect reasons. Most disconnect types are associated with read or write errors and timeouts. An intentional node shutdown or restart results in a `DisconnectRequest` reason.

Reason	**Explanation**
DisconnectRequest	Other peer disconnects intentionally
BadData	Malformed peer message received
ReadError	Receiving message from peer failed
WriteError	Sending message to peer failed
IdleConnection	No communication with peer for some time (currently 5min)
SlowConnection	Outgoing message transmission takes too long (currently 25sec)
DisconnectStaleWrite	Outgoing message transmission not initiated in time (currently 25sec)
CliqueResolving	Random outgoing peer disconnected if agreement protocol did not make progress for some time (currently 5min)
LeastPerformingPeer	Periodic disconnect of slowest outgoing peer (according to relative message delay)

Table 3.1: Peer disconnect reasons [20]

The `LeastPerformingPeer` disconnect reason stands out. It is associated with a peer performance monitoring mechanism. This is implemented in the source file `network/connPerfMon.go` [20]. An Algorand node continuously monitors the network

performance of outgoing peer connections. Around every 5 min, the currently least performing peer is disconnected and replaced with a new random peer. The purpose of this mechanism is to optimize the peer connections over time.

The deciding performance metric is the relative message delay of incoming messages from the monitored outgoing connections. Under ideal network conditions, each gossip message such as votes or transactions is received from each outgoing connection once. The node tracks the timestamps of when certain messages have been received from each peer. The difference of the message arrival time and the time that message has been received first is then taken as relative message delay. Aggregated over many gossip messages within the monitoring time span of around 5 min, each outgoing peer is assigned a total message delay performance metric. The least performing peer to be disconnected is then the one with highest aggregated message delay. After the peer replacement, a new monitoring cycle is started. From the perspective of the disconnected peer, a `DisconnectRequest` reason can be observed.

When running a node in relay mode, only a subset of outgoing peers is subject to this performance monitoring and peer replacement mechanism. Half of the outgoing connections are excluded from the mechanism and stay connected independent of their performance. The peer to disconnect is instead chosen from the list of non-persistent peers. A reason for this special relay node behavior could not be found in the source code or developer documentation.

The least performing peer network performance monitor of the node also periodically produces log entries that contain the recently observed performance metrics of the outgoing peers. Available as information are the relative message delay values and the first message percentages by outgoing peer.

3.2.2 Agreement Log Events

Consensus protocol progress is logged in form of agreement events. For example, the start and conclusion of rounds is logged as `RoundStart` and `RoundConcluded` event. Most agreement log entry types are related to consensus proposal, block and vote messages. These messages trigger multiple log events while being processed by the node. Certain error conditions are logged as well, such as a `StepTimeout`. The complete list of agreement log entry types can be found in the source file `logging/logspec/agreement.go` [20].

Whenever a new proposal, soft or cert vote message is received, it is checked and processed in multiple ways before being accepted. During the vote processing, the vote is associated with a node-internal state that specifies the processing progress. When reaching certain internal states, corresponding log entries are generated. The processing flow is visualized

in Figure 3.1. Internal vote events are written in italic font, triggered log events in brackets. Votes start in the state *votePresent*. A first series of checks determine whether the vote should be filtered out, in which case a `ProposalRejected` or `VoteRejected` log message would be generated. Filtered out are for example messages that are duplicates or are too old. If the vote is malformed or fails cryptographic verification, it is rejected as well. Only if it passes all checks, the vote is accepted and a `ProposalAccepted` or `VoteAccepted` log message is created. In case the vote causes a protocol threshold to be reached, a corresponding `ThresholdReached` event is generated.

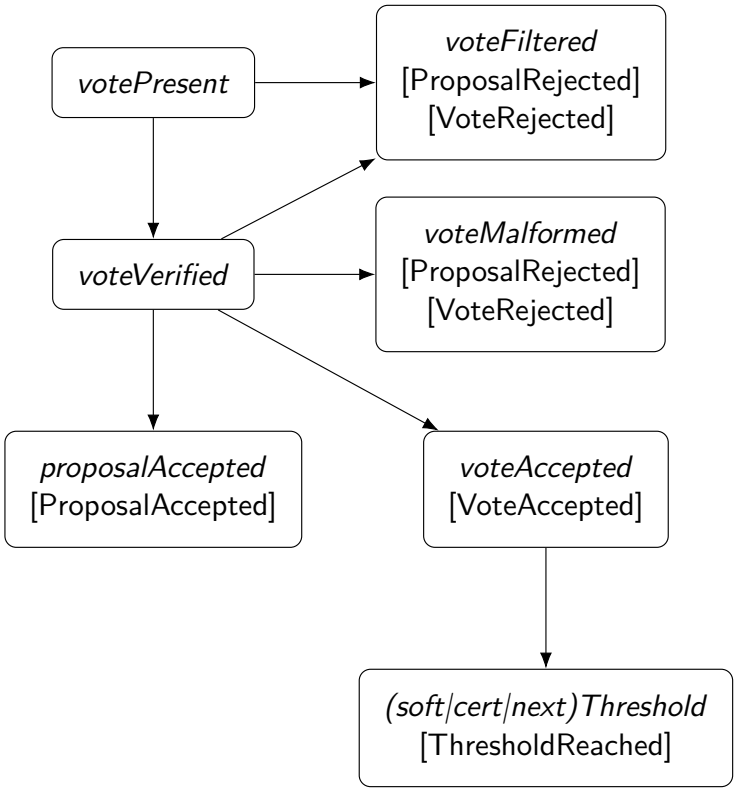

Figure 3.1: Node-internal vote processing events and associated log message types

Received block messages progress through a similar processing system. The corresponding flow is visualized in Figure 3.2. Block messages start in the *payloadPresent* state. The block first passes through a set of initial checks before either reaching the pipelined state or being rejected. It is also cryptographically verified. On success, it is accepted and a `BlockAssembled` log event is generated. The committable state is reached by the block of the current round that receives a threshold of votes.

The agreement message processing event log entries contain information about the sender of the message in form of an on-chain address and specify the block hash associated with

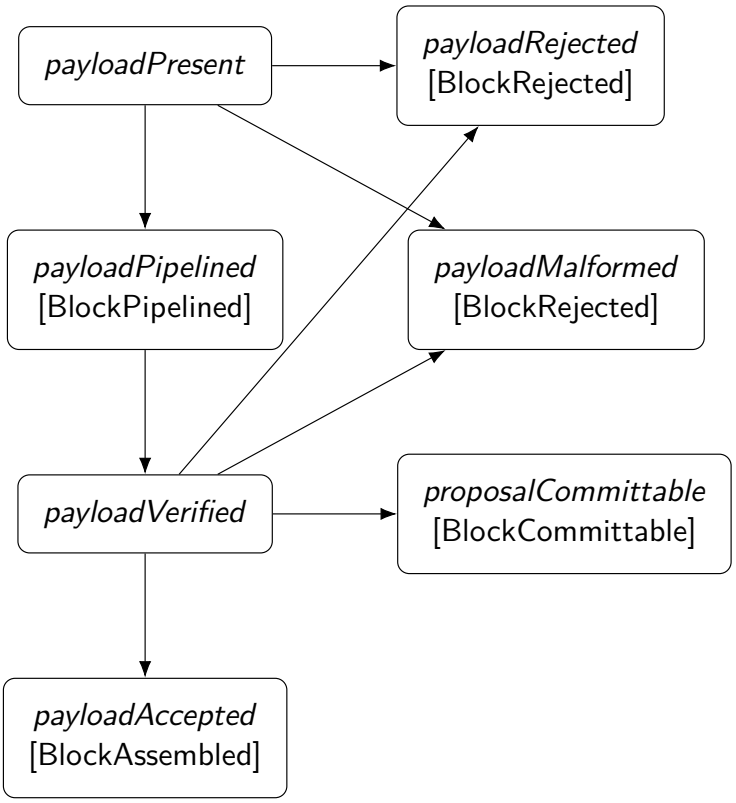

Figure 3.2: Node-internal block payload processing events and associated log message types

the message. Round, period and step numbers are also given for both the local node state and the message state. Soft and cert vote log entries also include the corresponding vote weights. An example `VoteAccepted` log entry is shown is Listing 3.3.

Listing 3.3: Example VoteAccepted log entry

```
{
    "Context": "Agreement",
    "Hash": "WUVGHUQFNTBHMO5NTK2RV33IZVKKRUTMJ26OUTTCGSEPGVVLLQIA",
    "ObjectPeriod": 0,
    "ObjectRound": 13937454,
    "ObjectStep": 1,
    "Period": 0,
    "Round": 13937454,
    "Sender": "UXRF2ESMUPBVTPUWYN33ESMQ2MKSYO73C52PQ24UIHZOALJJBROUKRQV5M",
    "Step": 1,
    "Type": "VoteAccepted",
    "Weight": 1,
    "WeightTotal": 128,
    "file": "trace.go",
    "function": "github.com/algorand/go-algorand/agreement.(*tracer).
        logVoteTrackerResult",
    "level": "info",
    "line": 477,
    "msg": "vote accepted for {{} 0 4
        H5UNRBJ2Q6JENAXQ6HNTGKLKINP4J4VTQBEPK5F3I6RDICMZBPGNH6KD4
        WUVGHUQFNTBHMO5NTK2RV33IZVKKRUTMJ26OUTTCGSEPGVVLLQIA
```

```
        BFG6RADPDMCVBNMB4IE4WJ5B2RBZN44OXLXWZTJ57AM2ZVGBKZFQ} at (13937454, 0, 1)
        ",
    "time": "2021-05-21T09:23:24.846864Z"
}
```

3.2.3 Heartbeat Log Events

Another notable log entry type is the node `Heartbeat`. Every 10 min, a node produces a heartbeat telemetry event. It contains a long list of metrics in form of key-value pairs related to node and network performance as well as information about the running node software version. The performance metrics are aggregated values collected by the node since its startup. Especially useful for the analysis of the gossip network are the values that specify the number of messages and bytes sent and received grouped by gossip message type. An example key-value pair set for these network metrics is given in Listing 3.4.

Listing 3.4: Example Heartbeat network metrics

```
"algod_network_message_received_AV": "13215558",
"algod_network_message_received_PP": "779915",
"algod_network_message_received_TX": "1182020",
"algod_network_message_received_total": "15272967",
"algod_network_message_sent_AV": "9905150",
"algod_network_message_sent_PP": "564436",
"algod_network_message_sent_TX": "884998",
"algod_network_message_sent_total": "11392192",
"algod_network_received_bytes_AV": "8320815240",
"algod_network_received_bytes_PP": "1719914857",
"algod_network_received_bytes_TX": "348390632",
"algod_network_received_bytes_total": "10711543000",
"algod_network_sent_bytes_AV": "6236672604",
"algod_network_sent_bytes_PP": "1233936608",
"algod_network_sent_bytes_TX": "257718815",
"algod_network_sent_bytes_total": "7730577400",
```

3.3 Algorand Relay Nodes

As of 5th November 2021, 101 Mainnet and 8 Testnet relay nodes are listed in the official Algorand DNS relay records. Relay nodes are enumerated in form of SRV records managed by Algorand and can be listed with the Linux console commands shown in Listing 3.5.

Listing 3.5: Relay list fetch command

```
dig -t SRV _algobootstrap._tcp.mainnet.algorand.network
dig -t SRV _algobootstrap._tcp.testnet.algorand.network
```

3.3.1 Relay vs. Client Nodes

Relay and client nodes only differ in their behavior minimally. Basically, relay nodes are just client nodes that additionally accept incoming peer connections and forward messages. While both client and relay nodes could participate in the consensus protocol in theory, Algorand recommends only client nodes to do so [1]. No specific reason is given, but it can be suspected that security concerns are a motivation behind this recommendation. Network addresses of relay nodes are publicly available, which makes relays more susceptible to attacks.

A node is configured to operate as relay node by specifying a `NetAddress` in the node configuration options. This causes the node to listen for and accept incoming peer connections on the given address and port. Relay mode also implies the configuration options `Archival`, `EnableLedgerService` and `EnableBlockService`. As a result of these options, the relay node stores all blocks since genesis locally and allows other peers to query for blocks and catchpoints. The forwarding of gossip messages is another core functionality enabled by relay mode. Gossip messages are relayed to other peers with a broadcast mechanism.

More subtle behavior changes are triggered by relay mode as well. For example, relay nodes subscribe to special gossip messages of type `CompactCertSig` by informing incoming peers of this interest. Additionally, the least performing peer evaluation mechanism performs slightly differently.

3.3.2 Broadcast Mechanism

If a message is scheduled for broadcast, it is by default sent to all other connected peers. In case the message is one that was received from another connected peer and scheduled for relaying, it would not get sent back to the sender peer, but forwarded to all other connected peers. The broadcast mechanism is implemented in such a way that certain peers take priority over other peers, so that the prioritized peers would be the first recipients of the broadcasted message. In a broadcast, the message is first sent over outgoing relay peer connections and to peers that are connected and declared as `PriorityPeers` in the node configuration. The order in which the other incoming peers receive the broadcast message depends on their `prioWeight`, which equals the value of their on-chain Algorand accounts. When a node connects to a peer and has a participation key configured, this key (or the one with largest stake in case of multiple keys) is used in a challenge-response mechanism to proof the account value of the node

[1] `https://developer.algorand.org/docs/run-a-node/setup/types/` (Accessed: 2021-11-13)

to the other peer. Broadcast messages are sent to incoming peers with larger stake first. Incoming peers which do not announce their account values, either because they do not participate in the consensus or they disabled this with the configuration option `AnnounceParticipationKey`, therefore receive the broadcasted message last. There is an additional configuration option `BroadcastConnectionsLimit`, which is disabled by default, to set a maximum number of peers that receive the broadcast.

Should the configuration option `EnableOutgoingNetworkMessageFiltering` be enabled, which it is by default, an interactive filtering mechanism is activated that reduces the load on the network by reducing unnecessary duplicate transmissions of large messages. When a node receives a message from one peer with length equal to or greater than a threshold value (currently 5000 B), it notifies the other peers with a broadcast about the hash of this message. The other peers then temporarily remember this hash value. Before sending a large message, the other peers compare the hash value of the message with the set of filter hashes. If there is a match, the message is not sent out to the peer, as that would be an unnecessary duplicate transmission.

Nodes similarly keep track of the hashes of vote and transaction messages that they have received recently. Should they detect an already seen incoming vote or transaction message, it would be dropped and not processed further.

Malformed, old and unverified transaction, proposal and vote messages are not relayed by nodes. The same applies to duplicate and equivocated votes, so that only one vote is relayed per consensus participant per consensus step.

Additionally, the propagation of proposal votes and blocks is optimized with a proposal priority mechanism. The priority of a proposal is derived from its sortition hash. The proposal with lowest sortition hash value has the highest priority and is expected to be the "winner" of the round period. When relaying proposal vote messages, it is therefore only necessary to forward proposals with a priority higher than all previously seen proposal vote priorities of that period. Proposal votes with a lower priority are dropped instead of relayed. The proposal priority mechanism impacts the transmission of block messages as well, as only the ones associated with the currently highest priority proposal votes are accepted and relayed. This reduces the amount of unnecessary transmissions of comparably large block messages and saves bandwidth, as lower priority blocks are not in a position to "win" the round period.

3.3.3 HTTP Headers Exchanged in Handshake

During the connection establishment of peers, a series of HTTP headers is exchanged. Some of these might be interesting to collect and analyze.

UserAgentHeader: When a node connects to a relay node with the intention of joining the gossip network, it transmits its user agent string to the relay. An example user agent is "algod/2.8 (dev; commit=9bd17e9f; 0) linux(amd64)". It contains Algorand software version, operating system and processor information.

GenesisHeader: During the handshake, the connecting node and the relay compare their Algorand network identifier, which is "mainnet-v1.0" in case of the Mainnet. This prevents for example a Testnet node from accidentally connecting to a Mainnet node. While the relay transmits its network identifier in form of the genesis header, the connecting node includes that information as HTTP path variable.

NodeRandomHeader: To make sure that a node does not accidentally connect to itself, a header with a random 10 B long encoded value is exchanged in both connection directions. The random value is determined during the node start and stays constant while the node is running. The connection attempt fails if both nodes sent the same random value to each other.

ProtocolVersionHeader: To allow nodes to check whether they are compatible with each other on a protocol level, they exchange their gossip protocol version in both ways. The current protocol version number is 2.1.

ProtocolAcceptVersionHeader: The connecting client announces to the relay a list of supported gossip protocol versions. Version 2.1 is the only currently supported one. Nodes verify during the handshake whether their protocol versions are compatible.

InstanceNameHeader: Each node is associated with an instance name identifier string. Instance names are shortened hash values and are derived from the node data directory path and a node- or machine-wide consistent globally unique identifier. Instance names can be used to differentiate multiple nodes that are operating with the same public IP address. Nodes exchange their instance name in both directions. The header is not transmitted if telemetry, either to Algorand or to the local log, is deactivated.

TelemetryIDHeader: Nodes exchange their telemetry globally unique identifier in both ways. The instance name header is partially derived from this value. The telemetry identifier header is not transmitted if telemetry, either to Algorand or to the local log, is deactivated. The header value can be used for logging purposes.

AddressHeader: An address header is exchanged in both directions during the node handshake. Regular client nodes leave their address header empty. Relay nodes specify their address there, as configured in the node configuration file fields `PublicAddress` or `NetAddress`. Neither client node nor relay node currently make direct use of this address header.

PriorityChallengeHeader: Nodes participating in the consensus protocol can announce their stake to their outgoing relay peers. Relays can then assign nodes with higher stake a higher broadcast priority, so that they receive messages slightly earlier. The participating node is requested by the relay to proof its stake during the connection establishment. The priority challenge header contains a challenge from the relay in form of a 32 B encoded string that the participating node has to sign.

3.4 Open Algorand Issues

The analysis of Algorand made two core design issues and a small set of technical issues apparent.

3.4.1 Design Issues

During the analysis of the Algorand blockchain, the relay admission and the reward model stood out as main issues of Algorand.

Relay Admission: Currently, the list of relay nodes is distributed as DNS SRV records. The corresponding DNS zone is controlled by Algorand. They can decide which relays are admitted and under which conditions. This centralized relay control contradicts the original goal of Algorand to provide full decentralization.

While the relays listed in the SRV records are officially labeled as *bootstrap* nodes and node operators have the ability to specify custom relay addresses, the default bootstrap relays still build the core of the network. During the research, no alternate public relay list became apparent, so most nodes are expected to use the default relays.

In the short term and while growing the Algorand ecosystem, it might be a valid approach to maintain central control over the relay nodes to assure a performant and stable core network. In the long term however, the relay admission should probably be a more decentralized process.

Reward Model: Another main open problem of Algorand is the lack of a long-term reward model. In its current model, owning stake is rewarded as well as relay node

operation, but not consensus participation. The rewards are supplied by Algorand with tokens they allocated at genesis for this purpose [1].

In 2022, Algorand is planning to replace passive stake rewards with *governance* rewards [2]. The community governance program allows stake owners to lock in tokens in order to gain the ability to voice their opinion as part of a voting committee regarding general ecosystem and policy questions of Algorand.

There are currently no apparent long-term plans for consensus participation rewards. Similarly, it is unknown if or how Algorand wants to decentralize the reward distribution for relay node operators. Both of these points are however essential for a stable and decentralized blockchain in the long-term.

3.4.2 Software Issues

During the research of Algorand's implementation on source code level, some technical issues with the official Algorand software have been identified.

Software Dependencies: Currently, the Algorand software is built with an unsupported version of the Go programming language. Go is used in version 1.14 [3], which is unsupported since 16th February 2021 with the release of Go 1.16 [4]. Using unsupported dependencies is a security risk, as the outdated software might be left in a vulnerable state.

Signed Releases: The security of Algorand software releases and deployments could be improved with signed releases. Providing SHA checksums and GPG signatures would allow users to verify the downloaded software. Automatic upgrades could be integrity-protected in a similar way. Without signed releases, users have to trust that the server they downloaded the software from did not tamper with the binaries.

Reproducible Builds: Making the official Algorand software builds reproducible would allow users to verify that the distributed binaries match the official source code. If the checksums of self-compiled binaries differ from the official ones, users cannot be sure that the source code has not been tampered with in case of the official binaries.

---

[1] `https://algorand.foundation/governance/algo-dynamics` (Accessed: 2021-11-10)

[2] `https://algorand.foundation/gov-faq` (Accessed: 2021-11-10)

[3] `https://github.com/algorand/go-algorand/blob/master/go.mod` (Accessed: 2021-10-21)

[4] `https://golang.org/doc/devel/release#policy` (Accessed: 2021-10-21)

Source Code Quality: The Algorand source code could be polished with the help of appropriate development tools. Currently, warnings generated by Go tools and advanced development environments are sometimes ignored. As warnings can potentially translate into runtime errors, these should be addressed. Using an automatic typo checker could also increase the source code quality. Defining a maximum source code line length would furthermore increase the readability.

3.5 Consensus Participation Rewards

Currently, there is no direct monetary motivation to participate in the consensus protocol. Operating a participating node does not yield any on-chain rewards. Stakeholders are indirectly motivated to participate due to the desire to maintain a supermajority of honest voters for the sake of protocol liveness. Would only conspiring malicious actors participate, transactions could be denied arbitrarily, as these actors would be able to decide which transactions to include in new blocks proposed and voted on by them.

Stakeholders should however be actively motivated to participate in the protocol with monetary rewards. Nodes are able to prove their participation by being chosen as committee member and broadcasting a proposal or vote into the network. Past committee members can only be verified through on-chain information. Only "winning" block proposals and votes from the *cert vote* step are available on-chain. Limiting participation rewards to proposers of "winning" blocks and certification voters enables a seamless integration into the blockchain catchup mechanism. New nodes need to be able to verify and apply reward distributions of past rounds when synchronizing the blockchain. The complete participation reward solution idea is therefore to reward the proposer of the "winning" block and the certification voters each round with a portion of the transaction fees from that round. Alternatively, the rewards could also come from the RewardsPool account instead of directly from the transaction fees.

3.6 Relay Decentralization

Decentralization of relay nodes refers to decentralizing the decision of who is allowed to operate a relay node. Currently, Algorand Inc. controls the list of relay nodes through DNS records. Decentralizing control would require moving away from such DNS records, as they are controlled by one party.

One goal in the decentralization of relay nodes is the protection against Sybil attacks. These can occur if creating many relays is somewhat free. Therefore, creating a relay must be associated with some cost.

Ideally, relay nodes should also fulfill a certain performance level. Currently, Algorand nodes already monitor the performance of their relays and frequently replace the connection to the relay with least performance. Because of this mechanism, it might not be necessary to enforce a certain minimum relay performance.

One additional problem that needs to be addressed is bootstrapping new nodes into the network. New nodes first have to learn the network address of at least one peer. Currently, this is realized with a DNS query to the SRV relay records. Algorand nodes also need to be able to synchronize with the network, learn about old blocks as well as the current state of the blockchain. Relay nodes currently operate in archival mode and serve the full blockchain to peers to help them join and synchronize with the network.

In a decentralized relay setup, new nodes should not have to rely on certain DNS records to find initial peers. Algorand Inc. could continue operating their SRV list, but nodes should have the ability to choose different bootstrapping peers. The feature to choose specific peers already exists, so this subproblem can be considered solved.

Regarding the catchup, if any node could register itself as relay node, there would be no immediate guarantee that these nodes indeed operate in archival mode and serve the whole blockchain. A peer would only learn about the lack of catchup data when actively querying for such data. Should a peer get into such a situation where a relay node is not able to provide catchup data, the peer could disconnect such a relay and re-attempt the query with another relay. Another possibility is for certain parties, such as Algorand Inc., to operate dedicated archival servers as fallback mechanism in case no reliable relay can be found immediately.

3.6.1 Relay Incentives

Another main requirement in the decentralization of relays is an appropriate incentive model. Operators of relay nodes should be compensated for their expenses. Otherwise, there would only be little motivation to run a relay node. Currently, Algorand Inc. contracts relay runners and rewards them with “newly minted” tokens. At some point in the future, all tokens will be in circulation and the transaction fees will have to somehow translate into relay runner rewards. This raises the question of who will be eligible for how much relay rewards. Some minimum requirements will have to be defined for a node to count as relay node. The main purpose of relay nodes it to relay messages to other nodes. Those in control of the reward distribution need to be able to evaluate whether relay nodes indeed fulfilled this purpose. Currently, Algorand Inc. receives telemetry data from nodes that allows them to evaluate the operation of relay nodes. Based on telemetry performance metrics, a judgement about the amount of earned rewards can

be made. Reliable nodes with consistent high performance deserve more rewards. Fully decentralizing such a relay reward mechanism is a challenge. When designing a solution to this problem, an appropriate catchup mechanism has to be included to allow new nodes to verify past relay reward distributions.

Distributed Relay Performance Monitoring

Relay nodes cannot provide direct proof about their performance to the network. Telemetry data from the relays themselves is not trustworthy. Nodes could directly proof that they have received certain messages, but not that they have relayed them. Directly connected peers have the ability to monitor the performance of connected relays. This feature is already implemented and actively used for another purpose, but could be reused here. Each peer could publish the monitored performance of its relays in some way, for example by gossiping it through the network. Every peer would then have the ability to make an educated performance evaluation of relays that are not directly connected by aggregating the different performance perspectives. For this approach to work, nodes must not be incentivized to lie about the performance of their neighbors. Performance gossip messages would have to be signed by the monitoring peer to prevent manipulation by relay nodes. Participation keys could be used to create such signatures and also assign the performance data a weight. Such weight is important to prevent Sybil attacks. Without weight, a spam of performance messages can manipulate the perceived performance of certain nodes. When aggregating received performance data, observations by high-stake nodes could be assigned a higher level of trustworthiness. A disadvantage of such a performance data sharing mechanism is a decrease in network level privacy, as participation keys could be more easily associated with network location. If a client node publishes a list of connected relays associated with their on-chain identity, an attacker would learn that they need to target these specific relays if they want to attack that identity. As security measure, nodes might only want to publish a subset of the full connected relay list to maintain at least one secret relay connection. An implementation of this mechanism should also consider that relay nodes are motivated not to relay messages that disapprove of the own performance. Since nodes maintain connections to multiple peers at the same time, relays cannot easily censor bad messages about themselves.

Reward Assignment

How relay rewards have been distributed over time is information that needs to be part of the blockchain in order to allow new nodes to reproduce and verify this assignment during their catchup. If performance data was stored on-chain, rewards could be implicitly assigned each block. Another approach would be to store the reward assignment as special

transaction in the block. The proposer of a new block would then have to include such a transaction to form a valid block. Either way, during the execution of the consensus protocol steps, the voters must be able to validate the relay reward assignment. They must check whether the proposed distribution matches the decentralized performance metrics.

One idea is that the proposer of a new block is given the responsibility of assigning the relay rewards of that round. If the proposer could do this without restrictions, they would just be motivated to assign the rewards to themselves. The proposer could therefore be required to include the signed performance metrics that they have received from other peers in order to justify their chosen reward assignment. The voters could then verify this assignment. This approach has two problems though. First, the proposer could leave out certain relays from the rewards list to assign more rewards to the included relays. Second, nodes would be motivated to conspire and lie about each others performance metrics during the decentralized performance evaluation part. Validators cannot easily tell which claimed performance level is the accurate one. Nodes could claim that colluding peers have outstanding performance while others have a poor one. Associating each performance claim with the observer's stake weight probably does not provide sufficient performance confidence. Such weight system would allow high-stake actors to conspire to allocate a great portion of relay rewards for themselves.

3.6.2 Sybil Protection Idea: Minimum Relay Stake

Next to the incentive mechanism, the protection against Sybil attacks is left as one main open problem. One possible idea to prevent Sybil attacks is to require relay nodes to be associated with some minimum amount of stake. Creating more relays would then require ownership of proportionally more stake. Peers would verify the stake associations of the relays.

A realization of this idea needs to assure the same stake cannot be used to operate multiple relays at the same time. This would be possible if the stake was verified during the connection attempt by asking for a stake proof, as multiple relays could then use the same stake proof with different clients. Stake therefore has to be strongly associated with a specific node. This association could for example be saved in the blockchain itself.

It should also not be possible to easily rotate the same stake over multiple nodes, depending on how the verification mechanism works. Imagine the following scenario: A client connects to relay R1 and verifies the stake requirement successfully. Then R1 rotates its stake to R2 and modifies the association of the relay stake. Now the same client could connect to R2 successfully as well. The client could however check if the two

relays stakes are the same and then reject the second connection attempt. But what if R1 rotated its stake to another relay R3 instead. The client would not immediately learn about this change unless it frequently verifies the stake association of actively connected relays. However, if the stake association was stored in the blockchain itself, the client would learn about changes to relay stakes each round and can immediately react to them, e.g. by disconnecting relays that lost their relay status.

Example relay creation:

1. Acquire a minimum of x Algos
2. Configure node
3. Associate this minimum stake with the own node address (e.g. with an on-chain transaction, similar to account status changes)

Example connection attempt:

1. Learn about relay addresses (e.g. by looking at new on-chain relay info)
2. Verify that the address is currently associated with a minimum relay stake
3. Connect to relay

The relay stake association could be implemented on-chain similarly to participation status information. Each on-chain address can be in an online or offline state. Similarly, each on-chain address could carry an extra field that contains the network address of the associated relay node. To achieve relay status, the account would then only need to maintain some minimum stake.

The stored network address should be a combination of IP address and port instead of domain name and port. The problem with domain names is that they can be easily rotated and pointed to other IP addresses. That would violate the strong association between on-chain address and relay node.

3.6.3 Reward Assignment with Path Proofs

If the decision over the relay reward assignment was in the hands of the block proposers, they could be required to provide some proof to the subsequent voters of the same round so they can reproduce the chosen reward assignment and comprehend its fairness. One new concept that could be introduced for this are *path proofs*. Gossip messages could be associated with the path they take through the network. Each node along the path would add their signature to the gossip message as proof they were part of the path. Client nodes that receive gossip messages could then see which relay nodes were

involved in the forwarding process. If the client node was chosen as block proposer in a certain round, they can use this routing data to infer the relay nodes they assign the relay rewards of the round to. The block proposal gossip message could then include the necessary path proofs, so voters can validate the assignment.

As concrete example, imagine some client creates a new transaction and is about to send it to its peers. Before doing so, it adds the corresponding next hop IP address to each of the transaction messages and signs these with the own participation key. As next hop relay node, the claimed next hop IP address would first be checked by comparing it with the own IP address. Then, the relay extends the existing path proof with the subsequent next hop IP address and signs the whole path chain with its participation key. When relaying the transaction, the path proof is included as attachment to the transaction message. The whole signed forwarding concept recursively continues until all nodes in the network have seen the transaction. Every node then knows which path through the network the transactions have taken in the form of a sequence of signed IP addresses.

Figure 3.3 shows an example message flow of a transaction with attached path proof data from one client to another client via two relays. The proof chain data consists of a sequence of node identities and a list of corresponding signatures. Each hop adds one identity and one signature to the proof data.

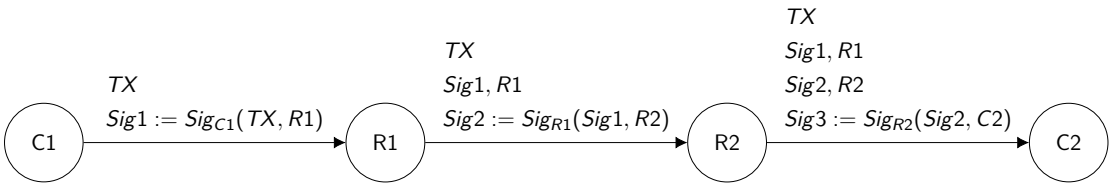

Figure 3.3: Path proof chain example

The reward assignment could then be managed in the following way: Each round, a portion of the transaction fees of this round is available for distribution as relay rewards. The block proposer is given the power to assign these rewards to relay nodes as special transaction in the proposed block. To prevent the proposer from assigning all rewards to a colluding relay node, the proposer is mandated to back up their assignment with path proofs. For example, each transaction added to the proposed block could be required to be associated with a matching path proof. Each relay node in the path proof is then eligible for rewards. Including more transactions in the block increases the total transaction fees of the block and therefore the distributed relay rewards. Even when colluding with a relay node, the block proposer is incentivized to include many transactions in the block to increase the available rewards. At the same time however, other relay nodes are also added to the rewards list and receive rewards, which is a

positive side effect for the fairness of the distribution, as the proposer is incentivized to reward not only colluding relays but implicitly also unassociated relays. The actual distribution of relay rewards could then be derived from the included path proofs. Each relay could receive rewards in proportion to how often their IP address appears in the path proofs of a block proposal. Alternatively, a portion of fees of one transaction could directly translate into rewards for only the relays on the path of that transaction, instead of first aggregating all fees and relays and then distributing the rewards. Either way, the proposer cannot arbitrarily influence the reward distribution, but has to base it on a set of path proofs.

With a default fanout factor of 4, the block proposer receives each transaction 4 times and has therefore 4 path proofs to choose from. To incentivize relays to offer good performance, the proposer should choose the path proof of the first received transaction message out of the 4. The message latency measurement mechanism can be used once again here to track the best paths and reward the relays along these paths accordingly.

Next to transaction message path proofs, proposals, votes or other gossip messages could also be associated with such proof mechanism. This might however not be necessary. Relay nodes would still be incentivized to forward messages besides transactions due to the least performing peer performance evaluation mechanism. Should a relay forward only transactions, peers would detect the absence of other messages in the form of an infinite message latency and would then disconnect the not performing relay. The relay however desires to maintain many connections so they are included in as many transaction path proofs as possible to increase the probability of earning rewards.

While the path proofs would not have to be part of the blockchain, the gossip network bandwidth requirements would increase. First of all, the transaction message size would increase, as the path proof would be included as attachment. Second, the block proposal message size would increase, as it would have to include path proofs of the finalized transactions. As optimization for the block message size, path proofs could maybe only mandated for a portion of the transactions. This would however create more room for cherry-picking transaction path proofs to optimize relay collusion profits.

Another aspect to consider with path proofs is network privacy. If message paths were known to the nodes, the network topology and the location of peers in the network would be more exposed.

With the outlined design, relay nodes would also be at least partially motivated to help other node synchronize and catchup with the network, since synchronized transactions can be associated with a path proof. Should the new client then start participating in the consensus and become a block proposer, the helping relay node would be included in

the relay list of the transaction path proofs and receive rewards. The bandwidth cost of bootstrapping a client with the full blockchain might however not be worth these potential rewards, so another incentive for serving the full chain might be needed.

Path Proof Idea Problems and Improvements

Path proofs as described above are not actually fully able to provide proof about which relays forwarded a certain message. The sender of a message is able to attest the first hop. The receiver at the end of the relay chain is able to attest the last hop and can learn the first hop with an appropriate signature from the sender. What happens between relay nodes in the middle of the forwarding chain is however not clearly visible to the receiver at the end. Two consecutive relay nodes in the chain could collude and add an arbitrary amount of other colluding relays to the middle of the path signature chain, even if the message was not forwarded by the added relays. By adding more colluding nodes to the path, a greater fraction of rewards can be claimed if the message gets chosen as path proof eligible for rewards.

In the example shown in Figure 3.4, imagine that relay nodes R1, R3 and R4 cooperate for profit. Now a transaction propagates from C1 to C2 over different paths. Relays R3 and R4 could then claim that Path2 consists of the relay sequence R3, R1, R4 by using a copy of the participation key of cooperating relay R1, even though R1 is not actually on the path. Should the client at the end of the path then become block proposer and choose Path2 as the path eligible for rewards, R1 would be in the included rewards list.

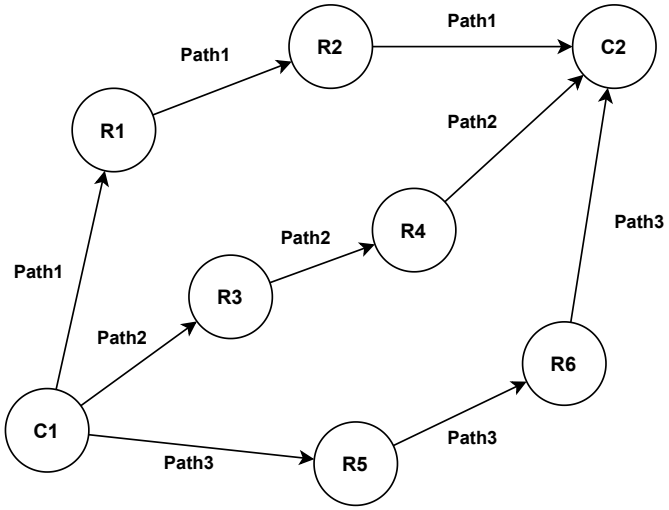

Figure 3.4: Gossip network path alternatives example

Since only the first and last relay hop of a message can be verified by the proposer, only those two relays of the chain could be made eligible for rewards. A proposer could then

still collude with one of its neighboring relays and always prefer paths including that relay instead of the other neighbors, but would still be required to also reward relays at the other end of the path. Those relays would be less likely in a colluding relationship with the proposer due to their distance.

Path Proof Concept in Existing Research

In 2018, Ersoy, Ren, Erkin, and Lagendijk [23] already developed a very similar version of the previously described path proof reward mechanism. In their approach, intermediate relay nodes are eligible for rewards. They attempt to solve the intermediate node Sybil attack outlined above by letting the block proposer choose to reward the shortest paths. A transaction message for example is received multiple times by the block proposer. The path eligible for rewards would then be the shortest one. This motivates the intermediate nodes to refrain from performing a path-lengthening Sybil attack.

Block proposers do however not have the ability to proof that a certain path is the shortest they have seen. They therefore could choose to reward a longer path if that one includes more conspiring peers than the other paths. The lack of shortest-path proof also makes the Sybil attack possible again. Two connected and conspiring relays could collude with the block proposer and launch a Sybil attack that is then intentionally approved by the block proposer to increase the portion of gained rewards.

Rewarding the shortest path may also set suboptimal performance incentives, as the shortest path is not necessarily the fastest path.

3.6.4 Related Work about Incentive Mechanisms

Another blockchain peer-to-peer network incentive approach was discussed by He, Li, Cheng, Liu, Yang, and Sun in 2018 [24]. In their model, relay nodes are rewarded by the initial sender of the message if they can prove the message has reached its destination. Each node provides a signed but pending micro payment to their successor node on the path to the target node. Nodes send back signed acknowledgements to their predecessor nodes on the path in exchange. Relays can then claim their pending rewards by proving message delivery with the signed acknowledgements. The described incentive mechanism however requires that each message has a known destination, thus making the model not directly applicable to the broadcast-based gossip network of Algorand.

In general, incentive mechanisms for peer-to-peer networks can be classified as payment-based or reputation-based [24, 25]. In payment-based schemes, providing services in the network is financially rewarded by the users or some other entity. This is for example the case in the current relay reward model of Algorand and the own proposed Algorand incentive strategies. Reputation-based incentive designs are an alternative approach.

There, users in the network keep track of some reputation values of other users. The provided service level can then be set relative to the reputation of the user, so that high reputation leads to high received service level. Reputation can be earned by following certain behavioral rules of the network. A problem with a reputation-based approach is that any service operation costs are not covered with financial rewards. Users in the network just invest resources to provide services to each other for mutual benefit, which can be enough motivation by itself.

Chapter 4

Design

With the achieved understanding of how relevant aspects of an Algorand node are implemented, a measurement plan can be derived. This chapter first summarizes, which information is available when operating a client or relay node. Then the measurement approach and the core components of the measurement system are explained.

4.1 Observable Metrics

Based on the review of design and implementation of Algorand, a set of measurable metrics can be identified.

Operating a client node gives access to the Algorand network. Gossip messages that are broadcasted through the network can be observed such as proposals, votes and transactions. The gossip network performance can be evaluated from the client's perspective. A client node also provides insights about the consensus operation, even when not actively participating in the consensus protocol. Client nodes can furthermore be used to analyze relay nodes by connecting directly to them and observing the network, for example in terms of relay performance. Relay host information can be derived from the public relay list as well such as the geographical location of nodes or any hosting organizations.

In order to evaluate the relay core network in more detail, it is necessary to operate a public relay node that is listed in the official relay records. By maintaining a relay node with incoming connections from clients and other relays, a first hand perspective on the core network can be gained and peer connectivity behavior and relay performance requirements can be evaluated. As client nodes only connect to relay nodes and are

otherwise invisible in the network, operating a public relay would enable an analysis of active client nodes in the Algorand network. For example, the activity of clients, their network locations and their user agents could be evaluated.

More information can be learnt from incoming peer connections than from outgoing ones. The `UserAgentHeader` of incoming peers is for example available for analysis.

Should the incoming peer participate in the consensus, then the peer priority challenge response mechanism, unless disabled by the peer, can detect this and information about the peer's participation key and stake can be learnt.

Incoming peers can apparently not be identified with certainty as node operating in relay mode. Receiving gossip messages with a variety of sender addresses is a relay indicator, however. Presence of the peer's IP address in the SRV records is another relay indicator.

Multiple incoming connections from different nodes running on the same IP address can be differentiated by the `InstanceNameHeader`.

4.2 Measurement Approach

To gain access to the outlined observable metrics, the initial plan is to setup and operate a client as well as a public relay node for both Mainnet and Testnet. The relay node would run continuously and passively collect data. The client node would be pointed to specific relays in turn to measure them one by one.

4.2.1 Client Measurements

Algorand client nodes only maintain outgoing connections to other relay nodes. The Mainnet and Testnet relay peers listed in the SRV DNS records can be measured with a client node.

In a *targeted* measurement setup, the local client node would be configured to connect to exactly one of the relays for some time. During the measurement, metrics such as network performance and other data can be observed in the client log file that provide insights about the measured relay peer. After the measurement time has elapsed, the node would be reconfigured to connect to another relay peer. The components of the client measurement setup and their interactions are visualized in Figure 4.1.

A measurement controller first fetches the list of relays from the Algorand DNS SRV records. It then points the local client node to one of the listed relays. The client node connects to this relay peer and no other peer to avoid interference. The controller makes a note of the measurement in a database. During the measurement, the client

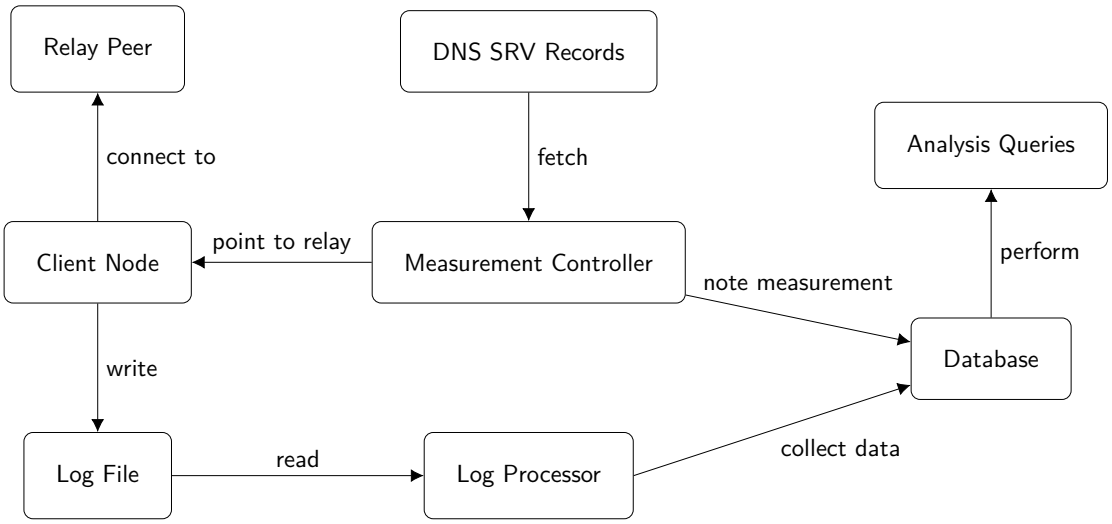

Figure 4.1: Client measurement setup

node produces measurement data as part of a log file. After some time, the controller rotates the current relay target, so that all relays are measured one by one over time. A log processor reads the log file, extracts relevant measurement data and stores that in a database. In the end, the collected data in the database can be evaluated with appropriate analysis queries.

Next to targeted measurements, the local Mainnet and Testnet client nodes can also be measured during *normal* operation in a similar fashion. During such normal operation, the client nodes are not directed to connect to one or multiple specific peers.

4.2.2 Relay Measurements

A relay node maintains incoming peer connections from client nodes and other relays in addition to the outgoing connections to other relays. Other Mainnet and Testnet clients can therefore only be measured with a relay node that is listed in the SRV DNS records.

To setup and operate a public relay node, it is therefore necessary to be added to these SRV records controlled by Algorand. On their website, they claim that "[a]ny user is free to register as a relay or participation node" [1]. This sets the expectation that everyone is able to setup a relay node and simply have the address added to the relay records.

[1] `https://www.algorand.com/technology/core-blockchain-innovation` (Accessed: 2021-11-12)

In summer 2021, Algorand started a relay runner pilot program to invite more organizations and individuals to join the Mainnet relay network [1,2]. Relay node applicants were however required to meet certain conditions. From a technical point, significant hardware requirements were mandated to assure the node offers a high level of performance. Additionally, a reliable service level was expected. Relay nodes were also required to enable telemetry and keep the running Algorand software up to date. Finally, signing a matching legal contract was given as necessity for joining the relay network.

While these conditions did not fulfill the expectations of "[a]ny user is free to register as a relay or participation node", an application to join the Mainnet relay network was still pursued with the intent of performing measurements and collecting data with the relay node. Unfortunately, the contract conditions were found to be too restrictive, as collected data would be labeled as confidential and rendered unusable for evaluation.

The unsuccessful negotiations with Algorand void the plan of operating a relay node and investigating the gossip network from the perspective of a relay node. Therefore, the focus is put onto data that can be measured with a client node.

4.3 Localnet Possibilities

The Algorand software supports the creation and operation of a private local Algorand network. A *localnet* consists of a user-defined number of relay and client nodes and genesis information that specifies the initial stake distribution and consensus participation state. A running localnet is a private blockchain similar to the Algorand Mainnet and Testnet. A private localnet is especially useful for experiments with Algorand that are independent of the real Algorand blockchain.

For example, scalability aspects of Algorand can be investigated by running local networks with various node counts. With a fixed amount of relays and an increasing amount of clients, the scalability complexity of Algorand with regards to the client count could be researched in practice. Similarly, the impact of stake distribution across clients on the network could be analyzed by manually specifying appropriate initial stake distributions, for example perfectly equal or with only a small set of powerful voters.

---

[1] `https://algorand.foundation/news/community-relay-node-running-pilot` (Accessed: 2021-11-12)

[2] `https://algorand.foundation/relay-node-runner-pilot-faq` (Accessed: 2021-11-12)

When setting up a localnet, enough hardware resources must be provided to support all the running local nodes or the localnet liveness can decrease and impact experiment results negatively. For small local networks, a single machine might be sufficient. For larger ones, a distributed node setup might be necessary.

While a localnet setup with non-trivial experiments would provide some additional insights on the scalability of Algorand, in the context of this thesis it is left as future work due to time constraints.

Chapter 5

Implementation

The designed measurement system is implemented on a Linux machine. As programming language, Python is used for the measurement orchestration and Rust for performant node log data processing. The collected data is stored in a PostgreSQL database.

5.1 Node Setup

The Algorand node measurement system is set up on a Linux virtual machine running Debian. The machine is equipped with 8 vCPU, 128 GB RAM, a 2 TB SSD and a 1 Gbit/s duplex network link.

On this machine, four Algorand nodes have been installed: one client node and one relay node for both Mainnet and Testnet. Client nodes are more lightweight than relay nodes, as they do not have to maintain a full archival copy of the blockchain. The configured relay nodes are not listed in the public relay SRV records, so no other peers actively connect to these nodes, but they still operate in relay mode.

The synchronization of the Mainnet node in archival mode took 10 days. In that time, 15.1 m blocks have been fetched from the network at a bandwidth of around 4 Mbit/s. 643 CPU hours were used for the synchronization. Once completed, 560 GB of disk space was occupied by the Mainnet blockchain.

With 40 h of real time, synchronizing the Testnet archival node was significantly faster. 14.7 m blocks were downloaded at a bandwidth of around 15 Mbit/s, using a total of 160 CPU hours. After the synchronization, the Testnet blockchain consumed 200 GB of disk space.

5.2 Node Configuration File Settings

The installed Algorand nodes operate with configuration file settings that are slightly different from the default settings. For all installed nodes, the logging verbosity is increased and the log file path modified for optimal interaction with the log processing service. For the local relay nodes, a `NetAddress` is specified to enable the relay mode and so that the nodes are reachable by other peers, given the firewall allows the incoming connections. For targeted measurements with local client nodes, the DNS and peer phonebook settings are modified. All other configuration options are left in their default state.

5.3 Peer Control

When performing targeted measurements, a mechanism is needed to point the node to a specific relay in some way. There are multiple possibilities to achieve this goal, some directly supported by an Algorand node.

Peer phonebook configuration file: The most straightforward way of pointing a node to a certain relay is to list that relay as sole peer in the `phonebook.json` configuration file. When a node process is started, it reads this peer phonebook file and initializes the internal relay list with the given peer addresses. To make sure this custom list is not overwritten with relay addresses fetched from the DNS SRV records, the `DNSBootstrapID` configuration option must be set to an empty string. That way, the node will only connect to peers listed in the custom phonebook. In a targeted measurement setup, this phonebook would only contain one relay address. A disadvantage of this peer control approach is that the node needs to be restarted for phonebook changes to take effect, which might not be a problem for targeted measurements.

Source code patch: Another idea would be to patch the peer selection strategy at Algorand source code level and then run a modified node binary version. This approach would be the most flexible, as the code can be modified arbitrarily. In the context of relay operation, running a node with modified sources might however not be possible.

DNS interception: By default, the list of relays is fetched from the DNS SRV records. By intercepting and manipulating the DNS response to only list the desired relays, peer control could be achieved. This would require the by default enabled DNSSEC to be deactivated in the node configuration options. One helpful configuration option in this approach is `DNSBootstrapID`, which is set to "<network>.algorand.network" by default.

By operating a custom DNS zone, e.g. “algorand.internal”, and replacing the default search domain, SRV requests initiated by the node could be redirected to an own DNS server. The custom zone could then be configured to only return the desired relays.

Intentional disconnects or restarts: The last peer control idea is the most disruptive. The node could be forced to connect to new peers by disconnecting existing peers on the network level or by restarting the node entirely. One problem with this approach is the randomness of peer selection, as no specific new relay peer could be guaranteed. Additionally, intentionally disconnecting peers or restarting the node would have a negative effect on the performance metrics of the node. This might especially be a problem in the context of relay operation.

Various details about the peer replacement behavior of a node are also relevant for the peer control problem: The `GossipFanout` configuration option can be used to influence the number of outgoing peer connections. Currently, a node-internal network management thread checks every minute if outgoing relay peers disconnected or are to be disconnected and replaces them with new random relay peers. Depending on the chosen peer control solution, this delay needs to be taken into account. A peer that just got disconnected can be immediately chosen as the replacement peer. A node maintains at most one outgoing connection to a specific peer. The node internal list of known relays is replaced completely whenever the node performs the periodic SRV record fetch, unless disabled with the appropriate configuration file setting. Periodic disconnects based on the least performing peer evaluation mechanism are not triggered if the `GossipFanout` peer number is not reached.

The peer phonebook approach is the chosen peer control strategy for targeted measurements. It is a directly supported feature and frequent node restarts with a client node are not as problematic as with a relay node.

5.4 Node Source Code Extensions

Should more node internal data be desired than available through the default log messages, the node source code could be patched and a modified binary version could be deployed locally. Two of such source code modifications have been implemented.

The node heartbeat event provides periodic node metrics as log entry. To extend the amount of provided heartbeat metrics, a `CustomHeartbeat` log event is introduced that is logged alongside the default heartbeat event. A custom heartbeat currently only contains a `network_big_message_received_total` metric. *Big messages* are defined as messages with a size of at least 5 kB, which is equal to the message size threshold

that triggers `MessageSkip` gossip responses as explained in the broadcast mechanism analysis section.

Gossip messages include information about the sender in form of an on-chain account address. Network address information is however not included by gossip messages. On-chain participation addresses are not publicly mapped to network addresses or vice versa either. This makes tracking the origin of messages on IP level difficult. What can be tracked however is from which neighboring peer certain messages are received. For this, log entries regarding consensus proposals and votes are extended by a `PeerAddress` field, which specifies the IP address of the corresponding neighbor peer.

5.5 Observed Metrics

Algorand nodes produce a large amount of insightful logging information. A relevant subset of log entry types are extracted by the implemented log processing service and forwarded to the database. In summary, the following log entries are collected:

- Peer connect and disconnect events
- Votes
- Heartbeats
- Relative performance metrics about outgoing peers
- Local node start and stop events
- User agents of incoming peers

Relay addresses are collected as well, but directly from the DNS SRV records and not from the node log files. The exact details about the extracted log data and its storage format are explained in the following section about the database schema.

5.6 Database Schema

The collected measurement data is stored in a database. PostgreSQL 13 is used as database system.

Each local Algorand node is assigned a unique node id. The assignment between node identifiers and custom human readable node names is saved in the database table `node_names`, as shown in Table 5.1.

Observed peer connection events are stored in the `peer_connects` table. These events are associated with a specific local node as well as a timestamp and include network

Field	Type
node_id	smallint
node_name	text

Table 5.1: Table `node_names`

address information about the connected peer. User agent strings are also stored if provided by the peer. The concrete table format is shown in Table 5.2.

Field	Type
id	bigint
node_id	smallint
log_time	timestamptz
address	text
host_name	text
incoming	boolean
instance_name	text
endpoint	text
user_agent	text

Table 5.2: Table `peer_connects`

Peer disconnect events are saved in a separate `peer_disconnects` table, but in a format similar to the connect events. Information about why disconnects occurred is also available. Table 5.3 lists the concrete disconnect table fields.

Node heartbeat telemetry events contain a long list of mostly performance-related metrics that have been collected and aggregated since the start of the node. Due to the extensive nature of the heartbeat metrics, they are stored as JSON in the `heartbeats` Table 5.4. This `metrics` field contains key-value pairs, mapping metric names to metric values. The `custom` field indicates whether the stored heartbeat refers to the default heartbeat event or the custom heartbeat event which is associated with a different set of metric key-value pairs.

Nodes continuously monitor and compare the network performance of their outgoing relay peers so that the least performing relay can be periodically replaced. The relative message delay is the deciding factor in that mechanism. Nodes expect to receive each transaction for example from each of their outgoing relays. When receiving a transaction for the first time, they start a timer and measure the delay until that transaction is received from the other peers as well. Over time, each relay is then associated with an aggregated delay value. Nodes also observe from which of the relays they receive certain

Field	Type
id	bigint
node_id	smallint
log_time	timestamptz
address	text
host_name	text
incoming	boolean
instance_name	text
endpoint	text
reason	text
message_delay	bigint

Table 5.3: Table `peer_disconnects`

Field	Type
id	bigint
node_id	smallint
log_time	timestamptz
metrics	jsonb
custom	boolean

Table 5.4: Table `heartbeats`

messages first most often. After staying connected to a set of relays for some time, the node can tell that for example x percent of messages have been seen first from relay node n. These first message percentages are extracted from the log file alongside the message delays and stored in the `monitored_latencies` Table 5.5.

Field	Type
id	bigint
node_id	smallint
log_time	timestamptz
endpoint	text
message_delay	bigint
first_msg_percentage	smallint

Table 5.5: Table `monitored_latencies`

The measurement controller makes note of measurements in the `measurement_info` Table 5.6. Measurements are associated with their time frame and a text tag to differentiate measurements of different types. A tag can for example include information about the measured Algorand network and whether the local node was running in normal

operation or was targeted to a certain relay. In case of targeted measurements, the host and port of the peer is also stored in the `measurement_info` table.

Field	Type
id	bigint
measurement_tag	text
node_id	smallint
start_time	timestamptz
stop_time	timestamptz
peer_host	text
peer_port	integer

Table 5.6: Table `measurement_info`

The `votes` Table 5.7 stores the observed proposal and agreement votes. Each vote is associated with its consensus round, period and step as well as the voted-for block hash, the voter address and the weight of the vote. In case of a patched node binary, the network address of the neighboring peer the vote was received from is also available as data point. Due to the relatively high frequency of incoming votes, measures have been taken to optimize the space usage of the `votes` table. Specifically, the voted-on block hash, the voter address and the peer address are only stored as shorter reference pointers in the `votes` table.

Field	Type
id	bigint
node_id	smallint
log_time	timestamptz
round	int
period	smallint
step	smallint
vote_id	bigint
voter_id	bigint
weight	int
peer_id	int

Table 5.7: Table `votes`

The voted-on block hashes and voter on-chain addresses are stored in the `voting_strings` Table 5.8 instead. There, these voting strings are saved in a deduplicated format for storage space efficiency.

Field	Type
id	bigint
string	text

Table 5.8: Table `voting_strings`

Similarly, the vote peer network addresses are deduplicated in the `peer_addresses` Table 5.9.

Field	Type
id	int
address	text

Table 5.9: Table `peer_addresses`

The list of relay nodes is periodically fetched from the corresponding DNS zone managed by Algorand. Should new relays be found on that list, they would be added to the locally maintained list of known relay nodes in form of the `srv_relays` Table 5.10. Each relay is associated with an Algorand network name, host and port information as well as a timestamp of when the relay was first and last seen in the SRV records.

Field	Type
id	bigint
network	text
relay_host	text
relay_port	integer
first_seen	timestamptz
last_seen	timestamptz

Table 5.10: Table `srv_relays`

5.7 Services

The implemented system consists of a variety of systemd services. They depend on other services such as **`postgresql.service`** and **`postfix.service`**.

Measures have been taken to increase the security of the system by applying systemd confinement options to the custom services. By following the principle of least privilege and limiting the permissions of the services, the attack surface is reduced. For example, network access has been restricted for services that are not supposed to interact with the

network. Additionally, write access to the file system has been greatly restricted to only necessary files and directories. Access to privileged operations has also been limited.

algorand@.service: The `algorand` service wraps each of the various local Algorand node processes in a service. This improves the node administration and enhances the security of the system by allowing the described confinement options to take effect. Since Algorand nodes are internet-facing applications in the case of relay nodes, this is especially relevant.

measurement-controller.service: The main objective of the measurement controller is to point the local client nodes that are used for measuring to certain relay peers that should be measured. Both the Algorand Mainnet and Testnet relay nodes are scanned in parallel with corresponding local client nodes. The measurement controller supports two types of measurements: *targeted* and *normal*. In a targeted measurement, the client node is pointed to exactly one specific relay peer, so that it only maintains a network connection with that peer. In a normal measurement, the client decides for itself to which peers and to how many peers it connects to, as if it was running normally without interference. The measurement results for these two scenarios can then be evaluated separately. The duration of measurements is set to one hour. The measurement controller continuously performs measurements. Once one relay peer has been measured for one hour, the controller points the client node to the next peer. The controller also alternates between targeted and normal measurements, so that every five targeted ones are followed by one normal one. The list of existing relay peers is fetched from DNS periodically. In its choice of the next peer to measure, the controller tries to prioritize peers have not yet been measured so often. A list of performed measurements is stored in the database for reference, associated with their time frames and the measured peers.

node-log-processor.service: One core part of the Algorand node analysis system is the node log processing service. Its task is to read the log files, extract and parse relevant entries and forward them to the database. It is implemented in the Rust programming language to achieve a high performance level. An asynchronous multi-threaded program design allows it to process the log files of all locally running Algorand nodes simultaneously. Log files and log entries of each node are processed sequentially to make the service crash-tolerant and provide consistency guarantees. After a restart or crash, the service is able to check the latest entry timestamp in the database and continue processing log entries with more recent timestamps. That way, no log entries are skipped or added to the database twice. To increase the efficiency of the sequential processing pipeline, queries are batched before being sent to the database. The log

processing service is not only able to process existing log files, but also offers a live processing functionality, which enables it to read new entries as soon as they are written by the node. This allows the service to run continuously and stay up-to-date with the logs.

srv-record-importer.service / *.timer: Once a day, the SRV record importer fetches the current list of relay nodes in the Mainnet and Testnet from DNS. In the local database, a list of known relay nodes is stored. Each entry is associated with the host name, port and a timestamp of when the relay was first seen. The SRV record importer compares the fetched relay list with the local relay list and then adds any new relays to the database. Old entries are preserved, even if no longer present in the DNS records. The local list of relay nodes can be used to observe which relay nodes are approved and added over time by Algorand.

node-update-checker.service / *.timer: The node update checker service runs daily and compares the current local Algorand software release version with the newest version. If an update is available, an email notification is sent to the administrator(s). Updates are not applied automatically to give time to review the release notes and check for incompatibilities with the custom fork of Algorand.

db-backup.service / *.timer: The database backup service is triggered every two days. It creates a backup of the database and deletes the oldest backup if the maximum backup count has been surpassed.

status-email@.service: The status email notification service informs the administrator(s) about failures of the other services above via email. It includes information about which service failed and why by adding a portion of the service log to the mail. The service makes use of a Postfix mail client installed on the system.

Chapter 6

Evaluation

The implemented system is used to evaluate the Algorand network in practice in multiple ways. The relay core network is one central aspect of interest, especially its state of decentralization and performance level. Evaluated as well is the traffic in the Algorand gossip network, the occurring message types, their sizes and occurrence frequencies. Observations are made regarding the performance of the network, its stability and security. Another point of focus are the voting committees at the center of the Algorand consensus protocol. The structure of these committees is analyzed, specifically the number of participants, the voting unanimity level and the committee power distribution. Based on the evaluation results, conclusions about the Proof-of-Stake realization of Algorand can be drawn and posed research questions answered.

6.1 Relay Domains

As of 28th October 2021, the Algorand relay SRV DNS records list 101 Mainnet and 8 Testnet relay domains. To evaluate this list, the relay domains are first resolved to their IP addresses with regular DNS queries. Next, the third-party web service *ipinfo.io* [26] is consulted to map the relay IP addresses to autonomous system host organizations and countries. The resulting relay IP information dataset is then evaluated in the following.

6.1.1 Relay Host Organizations

Figure 6.1 and Table 6.1 show, how Mainnet relays are distributed among autonomous system host organizations. The vast majority of relays are running on cloud services. 45 of 101 relays are hosted by Amazon and 22 by Google. Two thirds of relays are running on just these two cloud providers. Considering the aspect of decentralization,

this distribution is suboptimal. The next two most popular used cloud services are Cloudflare and OVH, with 4 and 3 relays, respectively. The remaining 27 relays are well distributed among 24 host organizations.

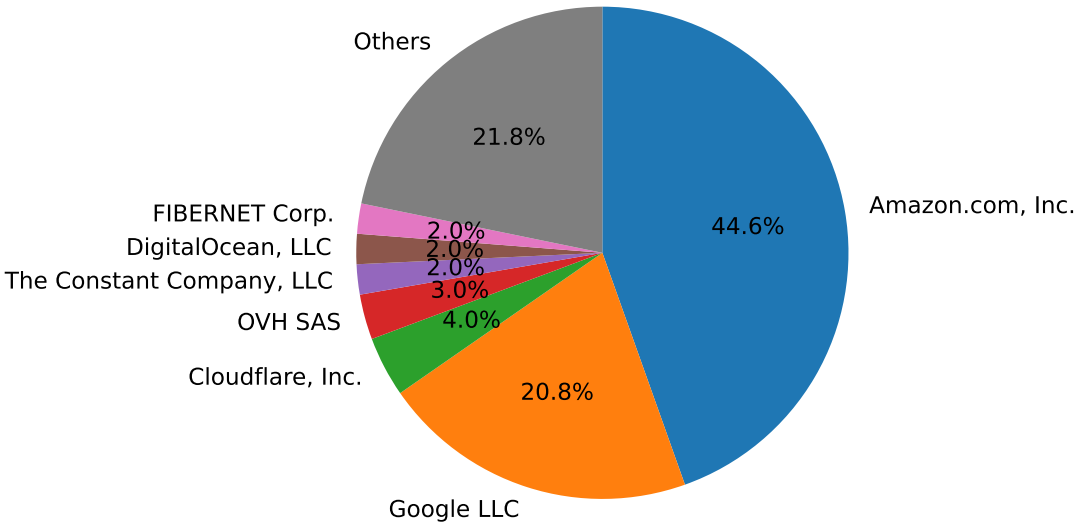

Figure 6.1: Mainnet relay organizations pie chart

Most relays are hosted by Amazon. To estimate how expensive it is to run a relay node, the technical relay requirements given by Algorand [1] are entered into the Amazon Web Services pricing calculator [27]. The minimum machine requirements are given as: 8 vCPU, 16 GB RAM, a 1 TB SSD, a 1 Gbit/s network duplex link and 5 TB of monthly outbound traffic. A matching Amazon EC2 instance located in Germany and reserved for one year is estimated to cost 690 USD per month. Network traffic makes up a majority of these costs.

Most of the Testnet relay nodes, 5 out of 8, are running on the Google cloud. The remaining relays are hosted by Amazon, China Unicom and Cloudflare. This distribution is shown in Figure 6.2 and Table 6.2.

Considering the majority of relays in Mainnet and Testnet are hosted by only two providers, the Algorand relay core network is only decentralized in a limited way.

[1] `https://algorand.foundation/news/community-relay-node-running-pilot` (Accessed: 2021-11-09)

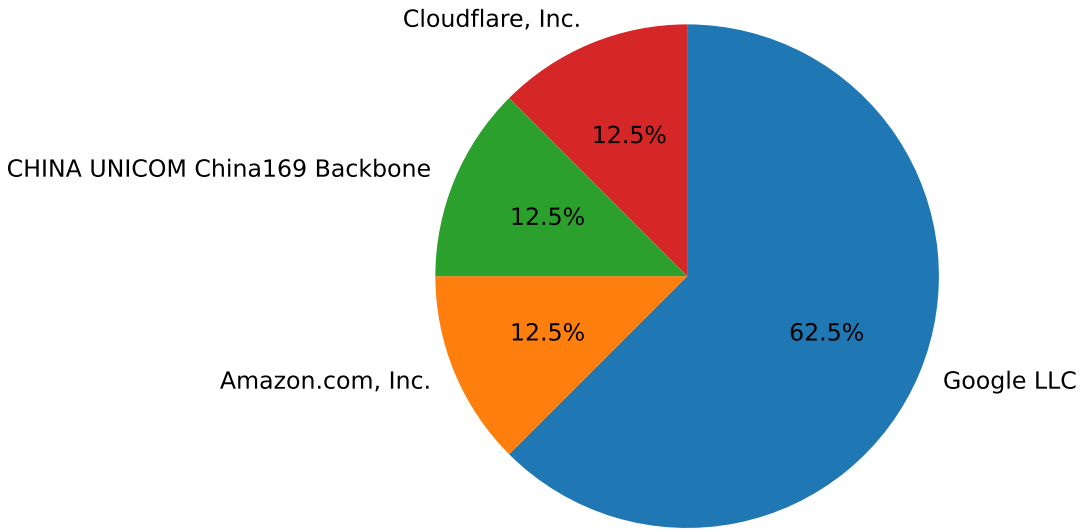

Figure 6.2: Testnet relay organizations pie chart

6.1.2 Relay Countries

Figure 6.3 visualizes how Mainnet relay nodes are distributed over the world. Darker country colors represent a large amount of located relays. Countries with light colors contain less relays. Uncolored countries are associated with no relays.

Table 6.3 shows the country distribution for Mainnet relays. One third of relays are located in the United States. Singapore and Ireland host 12 % of relays, respectively. 9 relays are located in Canada and Japan each. In total, around 75 % of Mainnet relays are hosted in 5 countries. The remaining 25 % of relays are distributed among 14 countries.

The 8 Testnet relay nodes are located in 4 countries. This is shown in Figure 6.4 and Table 6.4. Once again, the most present country are the United States.

Algorand relays are distributed over many different countries, which has a positive effect for the decentralization of the system.

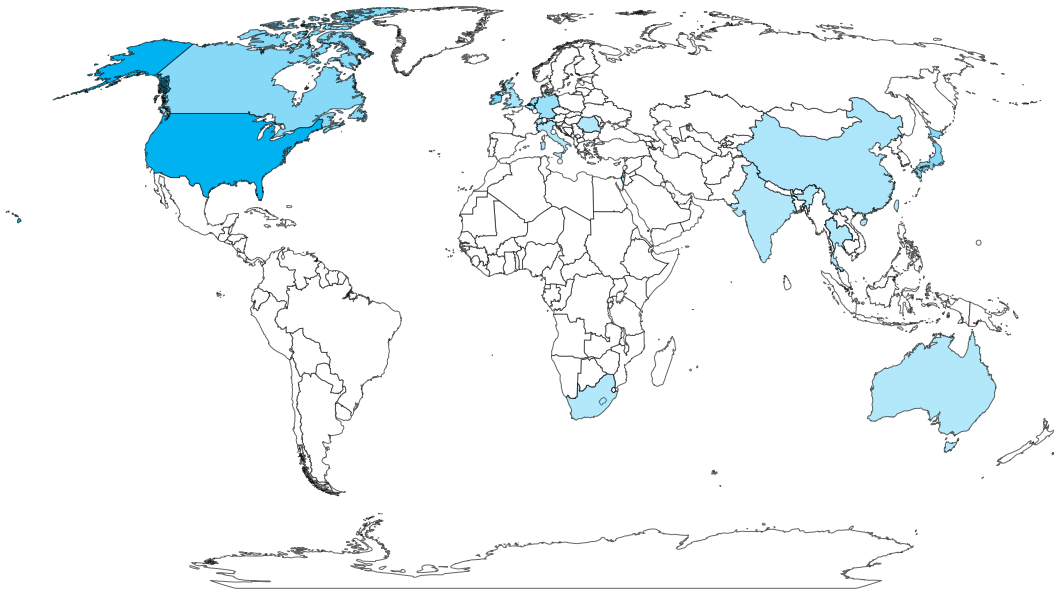

Figure 6.3: Mainnet relay locations world map

6.1.3 Relay Runner Pilot Program

As part of the relay runner pilot program, a set of new relays were admitted to Mainnet [1]. In Algorand's announcement, they counted 20 new relays.

To evaluate the added relays in more detail, a look at the SRV records is taken. The implemented measurement system fetches the current list of relays from the Algorand DNS SRV records once a day. Listed relay domain names are stored in the database together with a timestamp.

As baseline for the analysis, the relay list from before the pilot program is taken. 98 relays were listed on DNS level in the middle of September 2021. The first new relay domains were added on 29th September. In total, 21 new relay records have been added until 13th November. Additionally, 10 domains were removed from the list in that time frame. On 19th November, a total 109 Mainnet relay domains were listed in the records.

These numbers do not fully match the official ones specified by Algorand. 21 instead of 20 new domains were found. A larger number of relays is in principle positive. However,

[1] `https://algorand.foundation/news/new-algorand-relay-node-running-pilot-now-live` (Accessed: 2021-11-19)

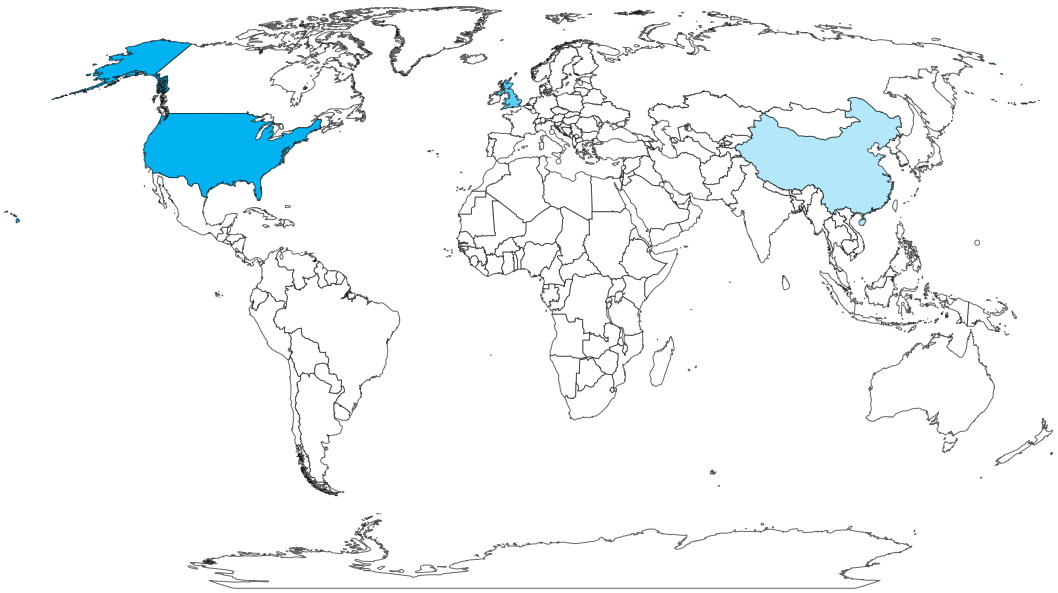

Figure 6.4: Testnet relay locations world map

10 other domains were removed at the same time. Therefore, the total number of relays effectively only increased by 11.

Details about the 21 newly observed relays are listed in Table 6.5. There, each new relay is mapped to their canonical domain name, reduced to the second level domain, as well as the corresponding hosting organization and country. The organizations and countries appear to be fairly mixed, which has a positive effect on decentralization. One notable exception to this is the relatively high occurrence of the United States as country. The evaluation also shows that the resolved domain "probsttech.com" appears twice. This indicates that one program applicant joined the network with two relay nodes. Algorand might only count these two entries once, which would align the number of observed new relays with the announced number of 20. In their announcement, Algorand also outlined the geographical distribution of the new nodes. The countries listed in the table do not fully align with the official location description. For example, Australia is listed in the announcement, but is not part of the table. A possible explanation for this difference is that the used geolocation resolution service from the third party *ipinfo.io* [26] did not pinpoint the location of the given IP addresses with full accuracy.

While many individual parties might operate their own relay node, the central control of Algorand regarding the admission of relays greatly limits the decentralization of the

core network. The relay runner pilot program is described by Algorand as "[...] the first step on the path to bring Relay Nodes to the same level of full decentralisation as the Algorand blockchain's consensus participation nodes" [1].

[1] https://algorand.foundation/news/new-algorand-relay-node-running-pilot-now-live (Accessed: 2021-11-26)

Host organization	Relay count
Amazon.com, Inc.	45
Google LLC	21
Cloudflare, Inc.	4
OVH SAS	3
The Constant Company, LLC	2
DigitalOcean, LLC	2
FIBERNET Corp.	2
Digiweb ltd	1
GigeNET	1
Isomedia, Inc.	1
IUCC - Israel InterUniversity Computation Center	1
Jisc Services Limited	1
LANSOFT DATA SRL	1
Linode, LLC	1
Massachusetts Institute of Technology	1
Microsoft Corporation	1
ScaleMatrix	1
SimplerCloud Pte Ltd	1
SUNY at Stony Brook	1
SWITCH	1
Tencent Building, Kejizhongyi Avenue	1
University of California at Berkeley	1
University of Waterloo	1
BroadbandONE, LLC	1
China Education and Research Network Center	1
CHINA UNICOM China169 Backbone	1
Comintech Corp	1
Consortium GARR	1
Cox Communications Inc.	1

Table 6.1: Mainnet relay host organizations distribution

Host organization	Relay count
Google LLC	5
Amazon.com, Inc.	1
CHINA UNICOM China169 Backbone	1
Cloudflare, Inc.	1

Table 6.2: Testnet relay host organizations distribution

Country	Relay count
United States	33
Singapore	12
Ireland	12
Canada	9
Japan	9
Germany	5
Netherlands	5
Northern Ireland	3
Belgium	2
China	2
South Africa	1
Switzerland	1
Israel	1
India	1
Italy	1
Romania	1
Thailand	1
Taiwan	1
Australia	1

Table 6.3: Mainnet relay country distribution

Country	Relay count
United States	3
Northern Ireland	2
Singapore	2
China	1

Table 6.4: Testnet relay country distribution

Reduced relay domain	Host organization	Country
bixin.com	Cox Communications Inc.	United States
algorand-mainnet.network	Tencent Building, Kejizhongyi Avenue	Thailand
bdnodes.net	Google LLC	United States
algorand-mainnet.network	Amazon.com, Inc.	Singapore
a-wallet.net	Linode, LLC	Japan
anodezero.com	OVH SAS	Canada
node-ops.com	The Constant Company, LLC	United States
algo.sysolnet.com	GigeNET	United States
algorand-mainnet.network	LANSOFT DATA SRL	Romania
tesseractops.io	Datacamp Limited	Brazil
algorand-mainnet.network	The Constant Company, LLC	United States
probsttech.com	FIBERNET Corp.	United States
probsttech.com	FIBERNET Corp.	United States
soton.ac.uk	Jisc Services Limited	Northern Ireland
algorand-mainnet.network	FOP Samoilenko Igor Olegovich	Ukraine
blockaffinity.com	DigitalOcean, LLC	India
bdnodes.net	Amazon.com, Inc.	Japan
sofia-relay.com	Neterra Ltd.	Bulgaria
algorand-mainnet.network	Aruba S.p.A.	Italy
hypernetlabs.io	Cloudflare, Inc.	United States
nighthawkapps.com	Oracle Corporation	Canada

Table 6.5: Relay runner program domain additions

6.2 Gossip Network Performance

The measurements performed with the local client Mainnet and Testnet nodes are used to analyze the performance of the Algorand gossip network as a whole and of individual relay nodes. Measurements are separated into *normal* and *targeted* ones. Observing a node during its operation in its default state is labeled as normal measurement. To evaluate individual relays, the local client is instructed to connect to only specific relays one at a time. These are called targeted measurements.

During a measurement, metrics regarding the exchanged gossip messages are collected. Available for analysis are for example the type of gossip message, the message sizes, bandwidths and message rates. The results of the targeted measurements also enable a relay performance comparison.

6.2.1 Normal Measurement Results

Over a duration of around 6 weeks, 200 *normal* one hour long Mainnet measurements have been conducted with the client node. In this time, around 174 m gossip messages were received in total. This corresponds to around 115 GB of data. The incoming average bandwidth resolves to 1.53 Mbit/s. 291 gossip messages were received per second on average. The vast majority of received messages are of type *AgreementVote*, *MessageDigestSkip* and *Txn*. Table 6.6 shows the statistics of received gossip messages by message type. *ProposalPayload* messages are received less often, but still frequently. Other message types are only encountered more rarely, by at least two orders of magnitude.

The most frequently received messages are votes with a message rate of 125/s. Message skip requests are observed almost as often with a message rate of 111/s. Half as frequently (51/s), transactions are received. Around 2 proposed blocks are encountered per second. The other observed message types are related to the startup and catchup of nodes. They are therefore received only infrequently.

Messages that contain block information are the largest observed messages. They contain around 30 kB of block data on average. Votes and transactions are medium sized gossip messages, with around 600 B in size. Other message types are only a few dozen bytes small on average.

One surprising measurement result is the unexpectedly high rate of message skip requests. If a relay node receives a large message with a size of at least 5 kB, it informs its neighboring peers about the digest of this message as part of a skip request. The neighbors can then save bandwidth by skipping a duplicate transmission of the large message. The gossip network statistics observed with the local client node show that

proposal payloads are the only frequent messages that exceed an average message size of 5 kB. However, these are received at a rate that is around 50 times lower than the one of skip requests. Relay nodes apparently receive large messages at a much higher rate than the local client. A reason for this could not be identified with just the client measurement data. A closer look at the data additionally showed that the unusually high rate of skip requests is not an effect local to certain relay nodes but can be observed across the whole network with some variance. To identify the cause for this behavior it might be necessary to operate an own relay node.

Msg tag	**Total msg received**	**Total bytes received**	**Avg msg received per sec**	**Avg bytes per msg**
AV	75 349 223	47 459 338 592	125.476	630
MS	66 939 231	2 275 933 854	111.471	34
TX	30 855 760	18 613 410 897	51.383	603
PP	1 436 231	46 051 960 081	2.392	32 065
TS	15 724	454 064 528	0.026	28 877
MI	3005	141 235	0.005	47
UE	473	32 637	0.001	69
VB	26	674 681	0.000	25 949

Table 6.6: Mainnet normal measurement statistics about received gossip messages

Similar to the Mainnet, the Testnet has also been measured. 200 *normal* measurements were conducted over a time span of 6 weeks. The gossip network activity in the Testnet was found to be much smaller than in the Mainnet. 46 m messages were received in total with an aggregated size of 31 GB. The incoming bandwidth aggregates to 424 kbit/s with a message rate of 76/s. The detailed statistics grouped by message type are shown in Table 6.7. The number of block proposals per second is similar to the one of Mainnet. However, the voting frequency is less than half in comparison. Message sizes by message type are also very similar with the exception of block proposals. As the amount of transactions received in Testnet is around five times smaller, the sizes of block messages are lower by a similar factor.

6.2.2 Targeted Measurement Results

1089 targeted one hour long Mainnet measurements have been performed over a time span of two months. In this time, around 253 m gossip messages were received in total, combining to 176 GB of data. The message sizes by gossip message type are unsurprisingly very similar to the results of the normal measurements. As the local client node was only connected to one specific relay node in the context of targeted

Msg tag	Total msg received	Total bytes received	Avg msg received per sec	Avg bytes per msg
AV	28 665 159	18 048 236 652	47.745	630
MS	9 900 874	336 629 716	16.491	34
TX	5 390 300	3 249 880 363	8.978	603
PP	1 604 153	9 269 381 533	2.672	5778
TS	63 886	871 955 039	0.106	13 649
MI	2991	140 577	0.005	47
UE	21	1449	0.000	69

Table 6.7: Testnet normal measurement statistics about received gossip messages

measurements instead four nodes in the case of normal measurements, the message rates are comparably lower by a factor of four. Detailed statistics are shown in Table 6.8.

Msg tag	Total msg received	Total bytes received	Avg msg received per sec	Avg bytes per msg
MS	109 919 911	3 737 276 974	33.618	34
AV	99 100 538	62 418 951 436	30.309	630
TX	40 772 554	24 216 084 251	12.470	594
PP	2 289 873	73 144 355 195	0.700	31 943
TS	802 392	13 376 501 813	0.245	16 671
UE	2040	140 760	0.001	69
MI	1198	56 306	0.000	47
VB	147	3 732 932	0.000	25 394

Table 6.8: Mainnet targeted measurement statistics about received gossip messages

Similar results can be seen in Table 6.9 for the 1461 Testnet targeted measurements.

In contrast to normal relay measurements, targeted ones directly enable a performance comparison of the measured relays. As deciding comparison metric, the average received bandwidth is used.

Ordered by performance, the relay comparison results for Mainnet are plotted in Figure 6.5. The evaluation shows that the vast majority of the measured relays provide a quite similar bandwidth of around 500 kbit/s. Some outliers can be observed both on the top and bottom. Only less than 10 % of relays provided a rather low performance level.

Msg tag	Total msg received	Total bytes received	Avg msg received per sec	Avg bytes per msg
AV	52 348 864	32 960 073 096	11.936	630
MS	13 285 220	451 697 480	3.029	34
TX	7 930 259	4 498 754 154	1.808	567
PP	3 033 650	14 600 291 227	0.692	4813
TS	384 922	4 906 327 228	0.088	12 746
MI	1503	70 641	0.000	47

Table 6.9: Testnet targeted measurement statistics about received gossip messages

The results for the Testnet relays, visualized in Figure 6.6, show a very consistent level performance for all relays. They provided a bandwidth of around 100 to 110 kbit/s.

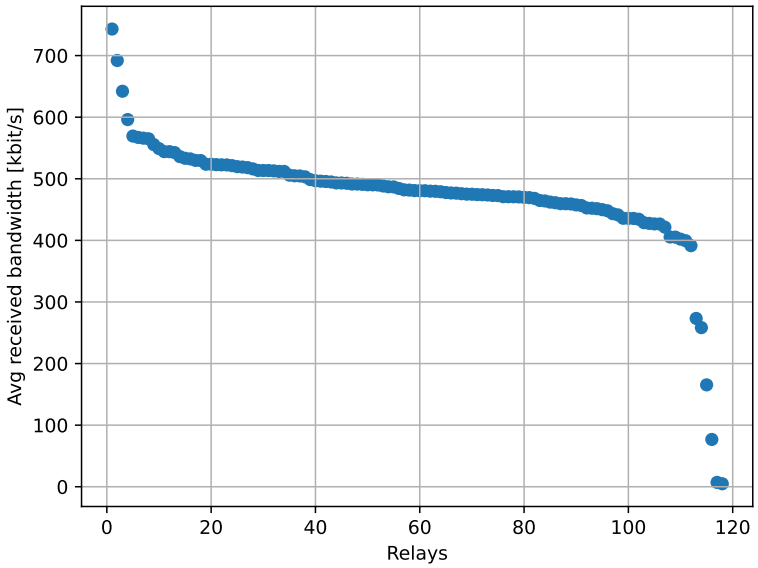

Figure 6.5: Mainnet relay performance comparison

In summary, the performance of relays can be described as quite consistent. Client nodes currently need to handle traffic of around 1.5 Mbit/s in Mainnet, which does not appear as a high node resource requirement. The goal of low barrier client operation without high-speed network link can be seen as fulfilled.

6.2.3 Scalability Modeling

To draw more conclusions about the scalability of the Algorand gossip network, core aspects of it are modeled. Based on derived formulas regarding different theoretical network metrics, the scalability of gossip message broadcasts can be analyzed.

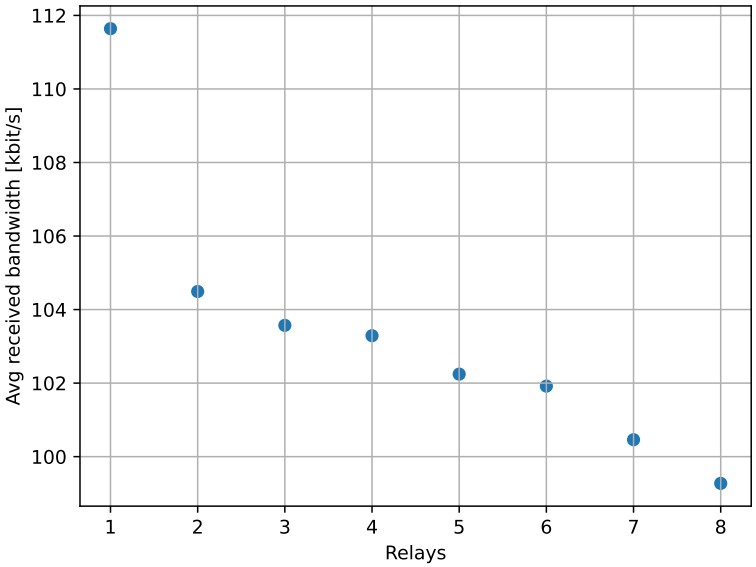

Figure 6.6: Testnet relay performance comparison

The gossip network consists of a relay core network and clients that are connected to this core network. The network can be modeled as a graph. The total number of nodes in the network is given by:

$$|N| = |Clients| + |Relays| \tag{6.1}$$

The node $fanout$ factor, which is equal to 4 by default, specifies the outgoing node degree $deg_{out}(n)$ of node n. The total amount of edges in the network can be calculated with:

$$|E| = fanout \cdot |N| \tag{6.2}$$

The incoming node degree $deg_{in}(n)$ for client nodes n is 0. The incoming degree for relay nodes is a probabilistic value and depends on the number of clients and relays in the network. Every edge in the network, which can connect either a client node or a relay node with another relay node, increments the total incoming node degree of the network by one. Every relay node consumes a fraction of this total degree. For a sufficiently large network, the average incoming relay node degree can then be calculated with:

$$avg(deg_{in}(n)) = \frac{|E|}{|Relays|} = \frac{fanout \cdot |N|}{|Relays|} \tag{6.3}$$

The average degree of relay nodes is therefore:

$$avg(deg(n)) = deg_{out}(n) + avg(deg_{in}(n)) \tag{6.4}$$

$$= fanout + \frac{fanout \cdot |N|}{|Relays|} \tag{6.5}$$

Vote Broadcasts

A vote message is first sent by the voting client to its neighboring relay peers. The relays then forward the message to all other peers. Relays forward the same vote only a single time and ignore any received duplicates. This continues recursively until all nodes in the network have seen the vote, assuming a perfectly synchronous and honest network. The vote touches each link between peers at least once during its broadcast. Each link between client and relay node observes the vote exactly once, as the clients do not relay any messages back to other connected relays. In the core relay network, the vote message can travel over the same link more than one time. This can be the case if two relays simultaneously inform each other about a certain vote. When a relay node receives the vote for the first time, it forwards it to all other peers. This causes traffic on $deg(n) - 1$ neighboring edges. To calculate the total amount of edges the vote is sent across, the average relay degree minus one is multiplied by the number of relay nodes. The initial message transfer from the voting client to its peers has to be added separately. A single vote therefore results in a total amount of vote message transmissions across the whole network of:

$$count(msg) = fanout + |Relays| \cdot (avg(deg(n)) - 1) \tag{6.6}$$

$$= fanout + |Relays| \cdot (fanout + \frac{fanout \cdot |N|}{|Relays|} - 1) \tag{6.7}$$

When taking the size of the vote message into account, $count(msg) \cdot size(msg)$ bytes total network load is generated across the whole network by a single vote. Each relay node sends $(deg(n) - 1) \cdot size(msg)$ vote bytes to its peers. Each client node except the voting client receives $fanout \cdot size(msg)$ bytes. The incoming number of bytes to relay nodes is once again a probabilistic value that depends on the number of incoming peers.

To calculate the required bandwidth per node, the network load in bytes per vote has to be combined with the frequency of vote messages. The vote frequency depends on the average round time and the number of voting peers. In an ideal scenario, the protocol executes voting steps 0 through 2 exactly one time before completing the round. When looking at the traffic generated by a specific voting step, the committee size and the stake distribution has to be considered. Each participating client node votes on behalf of

one or multiple accounts. Each account can be randomly chosen as committee member with a chance proportional to their stake. The number of votes in one step is equal to the number of committee members in that step. The committee nodes are responsible for generating the initial vote traffic of that specific step. The committee size, which is different for each of the three voting steps and is expressed in the unit of stake tokens, is an upper bound for the number of committee members. This bound is reached if all for voting eligible tokens are owned by different accounts. The corresponding lower bound for the committee size is 1, which is met if all eligible tokens are owned by the same account. To calculate the average voter count per step, information about how stake is distributed among participating accounts is necessary. This data is in theory publicly available on the Algorand blockchain. With that information, the traffic generated by each relay node n in a single voting step can be estimated with:

$$vote\_traffic(n, step) \tag{6.8}$$

$$= count(committee\_members(step)) \cdot (deg(n) - 1) \cdot size(msg) \tag{6.9}$$

$$\approx count(committee\_members(step)) \cdot (deg_{out}(n) + avg(deg_{in}(n)) - 1) \cdot size(msg) \tag{6.10}$$

$$= count(committee\_members(step)) \cdot (fanout + \frac{fanout \cdot |N|}{|Relays|} - 1) \cdot size(msg) \tag{6.11}$$

$$= count(committee\_members(step)) \cdot (fanout + fanout \cdot (1 + \frac{|Clients|}{|Relays|}) - 1) \cdot size(msg) \tag{6.12}$$

$$= count(committee\_members(step)) \cdot fanout \cdot (2 + \frac{|Clients|}{|Relays|} - \frac{1}{fanout}) \cdot size(msg) \tag{6.13}$$

This formula shows that the vote traffic scales with linear complexity in four factors: the number of voters in the committee, the fanout factor, the number of client nodes in the network and the vote message size. As the number of voters is bounded and the fanout and message size factors can be treated as constants, the vote traffic generated by each relay node effectively scales linearly with the total number of client nodes in the network. An increase in the number of relays in contrast lowers the vote traffic per relay, as clients are more distributed among the relays.

Evaluating the vote traffic formula for the *soft* and *cert* voting steps and aggregating the result produces the outgoing vote traffic per relay for one round:

$$round\_vote\_traffic(n) = \sum_{step\,=\,1}^{2} vote\_traffic(n, step) \tag{6.14}$$

Finally, this number can be combined with the average round duration to estimate the total outgoing voting network bandwidth for one relay per round:

$$outgoing\_voting\_bandwidth(n) = \frac{round\_vote\_traffic(n)}{avg(round\_duration)} \tag{6.15}$$

In summary, a linear scalability with the number of clients appears as decent network property. More clients do cause an increased network load on relay nodes, but only in linear proportion.

Proposal Broadcast

Proposals take on the form of two separate messages: a small proposal vote and a large block message. Compared to regular vote messages, proposals are subject to additional filter mechanisms: the proposal priority mechanism and the special filter messages of type *message skip*. These aspects make modeling the proposal broadcast more challenging.

At the start of a round, the proposal committee assembles new blocks and starts transmitting the small proposal votes that contain the block hashes followed by the actual blocks as separate message. Proposal votes are forwarded in the network similarly to regular votes. They however compete with each other due to the priority mechanism. A proposal vote is only forwarded by a relay if the relay has not already seen another proposal with higher priority. If a relay sees the proposal with highest priority first, it will not forward any other proposals in that round. In the other extreme case, the relay sees the proposal with lowest priority first, followed by the one with second lowest priority, and so on until it has seen and forwarded all proposal votes of that round. The actual number of relayed proposals is a partially random value and depends on the structure of the relay network, the location of the proposing clients and message latencies. This makes it difficult to formalize an equation for the average number of relayed proposals.

As the actual block messages are larger than the proposal announcements, they propagate slower through the network and use more bandwidth. Blocks are also affected by the proposal priority mechanism. They are only forwarded if the relay has seen the corresponding proposal vote and no other proposal with higher priority. Additionally,

block transmissions are also optimized by the message skip filter mechanism. Whenever a node receives a large enough block, it notifies its peers about the hash of the block message. These neighboring peers then avoid sending the same block to the peer that sent the filter request message. In the optimal case, the block message with highest priority is only received once by each node. If all nodes learn about the highest priority proposal before the first block is sent, only the winning block is propagating through the network. Creating a model for the block propagation is challenging due to the many unpredictable latency variables and the unknown network structure.

However, some conclusions about the scalability of proposal broadcasts can still be drawn. First, proposal traffic does not scale worse than linear regarding the factor of client node count. Without the proposal broadcast optimizations, transmissions of proposals behave similarly to those of regular votes. And votes were shown to scale linearly with the number of client nodes. Second, the size of proposals is an additional factor and not a constant as in the case of votes. Proposal block messages scale linearly with the number of transactions in the block. An increased transaction frequency leads to an increase in proposal traffic size.

6.3 Connection Stability

The stability of gossip connections is one performance factor of the Algorand network. Stable connections with relay nodes are a sign of a reliable network. Algorand nodes usually maintain their connections with other performant peers consistently. The least performing peer evaluation mechanism however causes a node to replace the slowest peer periodically. Other than that, peers only disconnect each other in error cases or if they are intentionally stopped or restarted.

To evaluate the stability of the relay network, all disconnects are observed between the local nodes running in relay mode and the outgoing relay peers. For each disconnect, its reason is tracked. In Table 6.10, the observed disconnect reason distribution for Mainnet is displayed. 97 % of disconnects can be attributed to the least performing peer replacement mechanism, which periodically triggers around every 5 min independent of whether there are errors or other stability problems. In only one case, the connection was cut because the remote peer shut down. Some form of connection problem was the reason for the other disconnects. Assuming a performance evaluation based disconnect every 5 min, these other unstable kinds of disconnects are encountered around every 2 h 42 min. For the Testnet, a very similar disconnect reason distribution can be observed in Table 6.11. As nodes maintain multiple relay connections at the same time, periodic

disconnects due to performance or infrequent disconnects due to errors or other problems are unproblematic. In total, the relay connections appear quite stable.

Disconnect reason	Count	Percentage
LeastPerformingPeer	26 821	96.87 %
ReadError	616	2.22 %
DisconnectStaleWrite	122	0.44 %
SlowConnection	115	0.42 %
WriteError	6	0.02 %
IdleConnection	5	0.02 %
DisconnectRequest	1	0.00 %
CliqueResolving	1	0.00 %

Table 6.10: Mainnet disconnect reasons

Disconnect reason	Count	Percentage
LeastPerformingPeer	25 450	97.04 %
ReadError	410	1.56 %
SlowConnection	229	0.87 %
DisconnectStaleWrite	95	0.36 %
WriteError	37	0.14 %
DisconnectRequest	4	0.02 %
CliqueResolving	1	0.00 %

Table 6.11: Testnet disconnect reasons

6.4 Network Security

During the research of the Algorand implementation, some observations about its network security have been made. They are described in the following, grouped by their associated security goal.

Confidentiality: Gossip messages are exchanged between nodes unencrypted. Their contents are therefore exposed to network observers along their message paths. While exchanged messages are usually intended to be available to the whole network, certain metadata such as user agent or stake information in message headers could be worth protecting. Support for TLS is implemented in theory and requires the configuration of TLS certificates by relay operators, but its usage in practice could not be observed.

Integrity & Authenticity: A lack of TLS encryption does not directly affect the integrity and authenticity of core consensus messages. Proposals, votes and transactions

are cryptographically signed separately with corresponding participation or wallet keys. However, not all gossip messages are protected with a signature. An attacker with favorable network position can therefore still inject for example forged message skip requests, possibly causing a disruption or disconnects in the network.

Availability: Algorand relay nodes validate every message before forwarding them to other peers. This validation mechanism protects the network from amplification flooding attacks, as malicious messages would be directly dropped at the first relay hop. When receiving a malformed message from a peer, that peer is immediately disconnected. If a message is a duplicate or no longer relevant for other reasons, it is just filtered and not forwarded. A rate limiting mechanism further reduces the impact of connection attempts from the same source with high frequency. In case of an ongoing denial of service attack caused by network overload, the priority peer mechanism additionally attempts to forward messages at least to connected outgoing relays and high-stake clients. With all these mechanisms in place, the security goal of availability has been extensively addressed.

Privacy: The IP addresses of relay nodes are publicly available through DNS. The ones of clients however are only known to the directly connected relays and not to the whole network. Additionally, Algorand has access to client information through relay node telemetry messages. Gossip messages can transmit on-chain addresses of participants through the network, but not IP addresses. As stake is usually associated with client nodes and not relays, the privacy of client IP addresses leads to an increase of stake security. Possible non-relay attackers are unable to easily locate and target client nodes. Additionally, client nodes operate without open ports, which further increases their security.

6.5 Consensus Voting Committees

The *committees* are a central part of the Algorand consensus protocol. They are responsible for proposing and voting on new blocks. To evaluate these committees, all accepted votes observed by the local nodes running in relay mode were collected over a time period of 8 weeks from September to November 2021.

This time period covers around 1 m rounds of the Algorand protocol. Around 3 m distinct committees were observed. As the protocol consists of the three main steps of proposal vote, soft vote and cert vote, this number aligns with the round count. In the 1 m rounds, over 3 m individual block proposal votes were observed. In the Mainnet, 473 distinct consensus participants could be identified. In contrast, 33 distinct voters were found

in the Testnet. The participants were responsible for 149 m Mainnet and 56 m Testnet votes in total. Table 6.12 summarizes these general committee metrics.

Metric	**Mainnet**	**Testnet**
Distinct rounds	1 071 726	1 107 844
Distinct committees	3 214 811	3 322 989
Distinct votes	149 235 060	56 184 437
Distinct voters	473	33
Distinct proposals	3 471 844	3 351 644

Table 6.12: General observed voting committee metrics

In an ideal case, the three main committees immediately find consensus about a new block in one protocol period. Should there be insufficient agreement, a subsequent period in the same round would be initiated to find consensus. In the 1 m observed Mainnet rounds, more than one period was required to determine a new block in only 45 cases. In the Testnet, all observed rounds completed in one period. These numbers signal a very high level of protocol liveness.

Due to the way Algorand nodes are implemented and optimized, not all issued proposals and votes are necessarily observed. For example, the proposal broadcast optimization causes relay nodes to drop low priority proposals. Other committee votes are collected by a node only until a threshold is reached. More votes could then still exist in the network, but the node is already able to advance its protocol state and no longer collects older votes. The votes evaluated are therefore the ones observed and collected by the node with the given active optimizations.

6.5.1 Committee Sizes

In each step of the consensus protocol, certain participants are chosen as committee members to propose or vote for new blocks. With enough votes for one proposal in one step, the protocol reaches a threshold and advances to the subsequent step and might certify a new block successfully. The number of participants in these committees is not a fixed value, but a probabilistic one. It depends on the economic distribution of stake, the amount of stake participating in the protocol and also on randomness.

An evaluation of the number of observed voters in each committee provides a first impression of the decentralization of the consensus committees. On average in Mainnet, 3 block proposals are observed per protocol period. More proposals might actually be issued but filtered out due to the proposal broadcast optimization mechanism. *Soft* committees consist of around 80 voters. In case of *cert* committees, 85 voters are

responsible for the certification decision of new blocks on average. In rare cases where more than one period is necessary to find consensus and finalize a block, between around 67 and 85 voters transition the protocol to a subsequent round period. The evaluation details can be found in Table 6.13.

Protocol step	**Avg voter count**
0 (propose)	3.2
1 (soft)	79.7
2 (cert)	85.2
3 (next)	85.2
4 (next)	67.0

Table 6.13: Mainnet observed voting committee sizes

For the Testnet, 3 block proposals are observed per period, similar to the Mainnet. The observed committee participant counts for the *soft* and *cert* step are lower in comparison. Respectively, 27 and 21 voters were involved in these voting steps. *Next* committees were not observed in the Testnet. Table 6.14 summarizes these results.

Protocol step	**Avg voter count**
0 (propose)	3.0
1 (soft)	27.0
2 (cert)	20.7

Table 6.14: Testnet observed voting committee sizes

With 473 individual Mainnet and 33 Testnet voters observed in total, a sizeable subset of participants is chosen in voting committees, which is important for the security and decentralization of the consensus protocol.

6.5.2 Committee Unanimity

A consensus voting step is successful if one voting option receives a supermajority of total vote weight. The more unanimous the vote results are, the more stable the consensus progress. Should multiple voting options frequently compete with each other, liveness and performance of the protocol decreases, as thresholds are reached slower or maybe not at all. One possible reason for a suboptimal unanimity ratio is a disturbance on relay network level. Should proposals or votes not be propagated properly, nodes may not receive them in time and vote for a competing option instead. Attackers could also try to negatively influence the unanimity level by intentionally voting for a competing option.

An evaluation of the observed voting committees shows that the Algorand network maintains a near perfect unanimity level. Table 6.15 and Table 6.16 list the average voting step unanimity ratios for Mainnet and Testnet respectively. *Soft* and *cert* step committees complete fully unanimous. The presence of *next* votes in the Mainnet signals that in rare cases more than one protocol period was needed to find consensus about a new block. In these next vote committees, some voting uncertainty could be discovered, even though the unanimity ratio was still very high.

Protocol step	**Avg unanimity ratio**
1 (soft)	1
2 (cert)	1
3 (next)	0.968
4 (next)	1

Table 6.15: Mainnet voting committee unanimity

Protocol step	**Avg unanimity ratio**
1 (soft)	1
2 (cert)	1

Table 6.16: Testnet voting committee unanimity

With the observed high unanimity ratios, a high liveness can be attributed to the consensus operation.

6.5.3 Voter Power Distribution

The expected voting power of a participant is proportional to their stake. In consensus committees, voting power is expressed as vote weight. A proposal counts as accepted by a committee if enough voters vote for the proposal, so that their combined vote weight surpasses a hard-coded threshold weight. Maintaining a large amount of stake increases not only the probability of being chosen as committee member but also the expected weight of the own vote.

In order to evaluate the distribution and decentralization of voting power, the observed consensus participants with most power are determined. First, an aggregated vote weight value is calculated for each observed voter. The resulting individual weights are then put into proportion with the total weight of all observed committees. This produces a fractional voting power value for each consensus participant. By ordering the list by power, the most powerful voters are identified. Table 6.17 shows the relative voting power of the 10 voters in Mainnet with highest aggregated weight. The most influential

voter maintains around 5 % of total observed voting power. The participants on rank 2 to 4 carry a power fraction of around 3 % each.

Voter weight	Total weight fraction
210 749 753	0.048 713
147 065 588	0.033 993
124 843 389	0.028 857
122 786 363	0.028 381
121 432 199	0.028 068
101 994 357	0.023 575
83 795 978	0.019 369
83 319 399	0.019 259
82 981 171	0.019 180
82 961 396	0.019 176

Table 6.17: Mainnet top voters by total weight

The results show that there is no single powerful active voter that would be able to consistently control voting committees. A group of powerful voters would have to collude to reach a significant level of influence. In order to evaluate this in more detail, cumulative power values of the top voters are calculated. Figure 6.7 visualizes the power curve for Mainnet. The top 20 voters maintain a cumulative voting power of around 45 %. Combining the power of the top 40 voters yields a cumulative power of over 70 %. In total, 99 % of power can be attributed to less than 100 consensus participants.

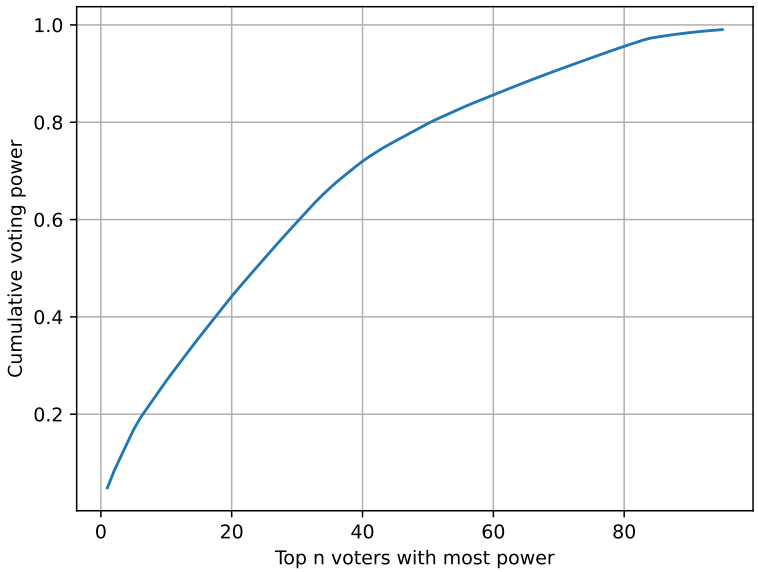

Figure 6.7: Mainnet cumulative voting power of top voters

The same evaluation is also performed on the measurement results from the Testnet. There, a more equal power distribution can be observed, but less participants in total. The top 10 voters maintain a power ratio of 4.5 % each. Table 6.18 shows the details.

Voter weight	Total weight fraction
201 069 510	0.045 014
201 065 050	0.045 013
201 064 352	0.045 013
201 060 618	0.045 012
201 059 420	0.045 012
201 048 982	0.045 009
201 047 177	0.045 009
201 035 697	0.045 006
200 740 471	0.044 940
200 697 206	0.044 931

Table 6.18: Testnet top voters by total weight

The cumulative power curve has a more linear shape compared to Mainnet. The Testnet power curve is visualized in Figure 6.8. The top 15 participants wield a combined voting power of around 66 %. In total, only 27 voters are observed in the Testnet.

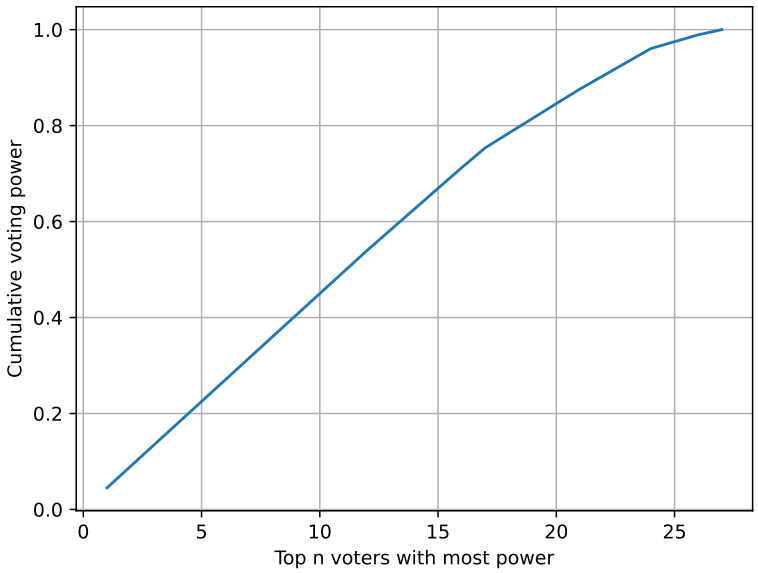

Figure 6.8: Testnet cumulative voting power of top voters

In summary, no single voter participates in the consensus protocol with significant power. However, a group of powerful voters could collude to disturb the liveness of the protocol.

As such a group would own a significant amount of stake, it would be unwise to risk devaluing their stake by negatively influencing the system.

6.6 Algorand Evaluation Summary

Based on the evaluation of Algorand in this thesis, the posed research questions can be answered. An evaluation summary is given regarding the extent the Algorand goals performance, scalability, decentralization and security have been put into practice. The identified main open issues and current goal progress of Algorand are described. Finally, the state of the "blockchain trilemma" is evaluated and the attempt of Algorand to solve it.

Performance: The Algorand network appears performant and stable. An operation of a client node is possible with modest hardware resources. Quite consistent performance of the relay core network is assured by Algorand with the help of contract clauses and matching financial rewards. Various message filtering mechanism assist at maintaining gossip network performance. On consensus level, a high amount of protocol liveness can be observed. Consensus committees find agreement about new blocks quickly and unanimously.

Scalability: Algorand achieves scalability with Proof-of-Stake committees at the core of the consensus protocol. Only a limited subset of users actively participate in a committee each round. In combination with the self-selection mechanism, only a low amount of communication is necessary. The broadcast-based message forwarding dominates the scalability complexity of Algorand. A linear scaling of network load can be identified when increasing the number of client nodes.

Decentralization: The Proof-of-Stake protocol enables Algorand to achieve full decentralization on consensus level. The absence of stake lock in and mandatory stake delegation keeps the barrier to actively participate in the protocol with a client low. Decentralization could be observed in practice, as the evaluation of voting committee sizes and power distributions shows. On the other hand, Algorand on network level was shown to be only partially decentralized. While relay nodes are found to be diversely distributed among countries, the majority of relays are hosted on the same two cloud providers. Additionally, Algorand maintains centralized control over the admission of new relays to the core network. A lack of a proper long-term reward model further limits the current decentralization of Algorand. The evaluation results of Algorand in practice contradict the claim of Algorand to be entirely decentralized without central authority.

Security: Signatures of proposals, votes and transactions are the main security feature of the Algorand blockchain. On network level, Algorand implements a variety of mechanisms to assure a high degree of availability. A consistent use of TLS could further improve the confidentiality and authenticity of gossip network communication. The non-public nature of client node identities provides client privacy on the network layer. Some network data is only available to relay operators and Algorand.

Main Open Issues: The reward model and the relay admission scheme were identified as main open issues of Algorand. Rewards for relay nodes are currently managed directly by Algorand. Consensus participation is not directly rewarded at all. Algorand wields too much power over the relay core network for it to be considered fully decentralized. A couple of technical issues have been identified as well during the research of Algorand, such as outdated dependencies or the lack of signatures for released binaries. In this thesis, some solution ideas for the outlined core issues were explored. The problem of fully decentralized relays, consensus participation rewards and a reward model for relays based on path proofs were discussed.

Goal Progress: Algorand does manage to fulfill most of its goals, but not all of them. It can be observed that there is an intention to come closer to these goals. With the introduction of the governance program and the phasing out of stake rewards, an evolution of the reward model can be seen. A long-term reward model for consensus participation and relay node operation however has yet to be presented. The relay runner pilot program shows effort of Algorand to slowly increase the level of decentralization in the relay core network. If or how Algorand fully decentralizes the relay admission is yet to be seen. For the time being, Algorand appears to desire to maintain some level of control to assure a stable and growing ecosystem.

"Blockchain Trilemma" State: The Algorand marketing material claims that the blockchain properties decentralization, scalability and security have been unified and the "blockchain trilemma" has been solved. The evaluation of Algorand however showed that the property of decentralization is achieved only in a limited way. While the Algorand Proof-of-Stake consensus protocol itself might be fully decentralized, the open issues found in the reward model and relay core network make the surrounding blockchain platform only partially decentralized. Until these points are addressed, the blockchain trilemma cannot be considered solved.

Chapter 7

Related Work

Some related work about Algorand exists that mainly addresses security aspects of Algorand. One paper investigated the participation reward model of Algorand.

Related work regarding Algorand:

- In 2019, Fooladgar, Manshaei, Jadliwala, and Rahman [28] created a game-theoretic model of Algorand, identified issues with current reward distribution model and proposed a better reward scheme. This scheme however still featured the Algorand Foundation as central rewards authority, which does not align with the blockchain trilemma goal of fully decentralizing the network.
- In 2019, Conti, Gangwal, and Todero [29] presented a DDoS-like flooding attack on Algorand nodes that takes advantage of proposal message validation complexity. As a sidenote, they also pointed out that relay nodes are prone to Sybil attacks as an attacker could create many relay nodes for free and nodes do not consider the relay's stake when connecting. Algorand's current implementation does not seem to be affected by these attacks. Proposal messages are split into proposal vote and actual block messages. The large blocks are only considered if an earlier matching proposal vote has been validated successfully. The relay Sybil attack is currently prevented by allowing only certain nodes to be relay nodes. When decentralizing relays more in the future, Algorand will have to address this attack possibility in their design.
- In 2020, Liu investigated the Algorand consensus algorithm in their Master's thesis [30]. They built an Algorand simulator in Python that runs the consensus protocol. With this, they simulated an Algorand network and evaluated scalability

and security properties of the protocol. When configuring delay parameters in their simulator, they used performance metrics from Bitcoin as reference. Their evaluation results suggest that the round completion time of the Algorand protocol scales well with an increasing number of participating nodes. When adding up to 20 % dishonest voters to the simulation, only a small increase in block time was observed.

- In 2020, Alturki, Chen, Luchangco, Moore, Palmskog, Peña, and Roşu [31] created an Algorand protocol model in the Coq proof assistant and proved asynchronous safety of the protocol which guarantees that two different blocks cannot be certified in the same round.
- In 2021, Bartoletti, Bracciali, Lepore, Scalas, and Zunino [32] created a formal model of Algorand's stateless smart contracts that allows reasoning on the security of smart contracts.

In more general related work about Proof-of-Stake, Algorand has been mentioned and compared to other Proof-of-Stake approaches. Some theoretical challenges to Proof-of-Stake, such as decentralization incentives and issues with forking, have been addressed in research.

Related work regarding Proof-of-Stake:

- In 2019, Nguyen, Hoang, Nguyen, Niyato, Nguyen, and Dutkiewicz [9] compared the Proof-of-Stake consensus of Ouroboros, Chains-of-Activity, Casper, Algorand and Tendermint regarding their protocol mechanism, incentives, security and performance. They then presented Proof-of-Stake applications in vehicle blockchain networks. Finally, they analyzed Proof-of-Stake stake pooling and its effect on decentralization with game theory.
- In 2019, Brown-Cohen, Narayanan, Psomas, and Weinberg [33] described security problems and design challenges of longest-chain Proof-of-Stake protocols that arise from the Nothing-at-Stake problem.
- In 2020, Baldimtsi, Madathil, Scafuro, and Zhou [34] attempted to decouple user identities from Proof-of-Stake lottery mechanisms to improve privacy. They described an ideal anonymous lottery function that applies to the Ouroboros Proof-of-Stake approach, but not Algorand.

Performance metrics of blockchains have been defined and evaluated in research, both for Proof-of-Work and Proof-of-Stake approaches. However, an in-depth evaluation of the Algorand blockchain is lacking.

Related work regarding blockchain performance evaluation:

- In 2018, Zheng, Zheng, Luo, Chen, and Liu [35] defined a series of blockchain performance metrics such as transactions per second and transaction confirmation latency. They also developed a blockchain monitoring framework that is able to analyze various metrics from log files and evaluated Ethereum with it.
- In 2018, The Linux Foundation [36] defined basic blockchain performance evaluation terms and metrics in context of Hyperledger. They also outlined test environment considerations, e.g. concerning the network model and the workload.
- In 2020, Lepore, Ceria, Visconti, Rao, Shah, and Zanolini [5] provided a general comparison of node scalability of Proof-of-Work and Proof-of-Stake blockchains and throughput in terms of transactions per second as well as transaction confirmation latency.

Chapter 8

Future Work

The analysis of Proof-of-Stake in practice can be continued in various ways.

Specifically to Algorand, the decentralization of stake could be evaluated. This is a relevant aspect because stake translates to power in Proof-of-Stake blockchains. The distribution of stake among wallet addresses is a publicly accessible information directly stored in the blockchain. The actors with most individual stake could be identified and their relative power in the network evaluated. Additionally it would be interesting to analyze how much power the creators of Algorand currently maintain, as they allocated a significant amount of stake to themselves in the genesis block that they use to influence the growth of the Algorand ecosystem. A large amount of stake gives actors a proportionally large participation power in the consensus committees that propose and vote on new blocks. The evaluation chapter of this thesis already analyzed the voter power distribution of actors actively participating in the consensus protocol with their stake. Not all stake in circulation is necessarily associated with a client node for consensus participation purposes. Therefore, an evaluation of distribution of circulating stake would complement the analysis of distribution of participating stake.

Another part of Algorand that could be subjected to further analysis is the relay core network. In this thesis, the relay network could only be evaluated from the outside with a client node. Relay domains, locations, hosting organizations and provided bandwidth have been analyzed. For a direct first-hand evaluation, operating an own relay node would be necessary. Connectivity behavior of incoming peers and relay performance metrics and requirements as well as relay operating cost could then be evaluated in more detail. Information about client nodes could be learnt such as their amount of associated stake, their geographical locations and their user agent header values. More insights

could be gained about the gossip messages exchanged in the core network. As relay nodes are more exposed in the network, possible unusual or malicious network activity could also be observed better.

On a larger scale, the Proof-of-Stake approach by Algorand can be compared in more detail with other Proof-of-Stake implementations, some of which have been outlined in the background chapter. Similar measurement setups can be created for these other blockchains. Resulting measurement data would then allow for an in-depth comparison between the different approaches in practice regarding aspects such as performance, decentralization and security. Interesting to analyze would be how these other Proof-of-Stake implementations attempt to solve the core issues identified in Algorand and in turn which points are solved by Algorand better than by the competitors. As extension, a broader comparison with Proof-of-Work is also possible, for example regarding the energy usage.

Chapter 9

Conclusion

This thesis analyzed permissionless Proof-of-Stake consensus in practice at the example of the Algorand blockchain. Based on a review of core aspects of Algorand, a measurement setup with an own Algorand node was designed and implemented in order to observe and evaluate the Algorand network and consensus operation in practice. Using the evaluation results, it was possible to analyze to what extent Algorand manages to fulfill their goals in practice. Additionally, some main open issues of Algorand could be identified and corresponding solution ideas explored.

For the Algorand Mainnet, the following main numeric results have been observed: Two thirds of relays are hosted by two organizations. 75 % of relays can be mapped to 5 countries. Regarding the consensus operation, 30 distinct votes were observed per second. Voting committees consisted of around 80 participants on average. In almost all cases, committees came to an unanimous agreement. The most influential consensus participant maintained a consensus power of 5 %.

Algorand appears to fulfill most of its goals in practice, specifically the core goals of performance, scalability and security. However, the goal of decentralization is only fulfilled partially. The open issues identified with the reward model and relay admission system are the reasons for this conclusion. While a long-term plan to address these points has not yet been presented, an intention to come closer to the goal of decentralization can be observed. An exploration of own solution ideas for these aspects shows that there is potential for such improvement. However, the development of a fair, secure and fully decentralized relay reward model poses a challenge. Until these points are addressed, the "blockchain trilemma" of unifying full decentralization, scalability and security cannot be considered solved by Algorand.

Nevertheless, the Algorand blockchain is a practical Proof-of-Stake consensus implementation with real monetary transactions that has a series of desirable properties. While it has not yet reached its full potential, Algorand can be considered a qualitative Proof-of-Stake realization.

List of Figures

List of Tables

Bibliography

[1] Satoshi Nakamoto, *Bitcoin: A Peer-to-Peer Electronic Cash System*, https://bitcoin.org/bitcoin.pdf, [Accessed: 2021-06-05], 2008.

[2] Ethereum Foundation, *ethereum.org*, https://ethereum.org/, [Accessed: 2021-10-26], 2021.

[3] Cardano, *cardano.org*, https://cardano.org/, [Accessed: 2021-10-26], 2021.

[4] Algorand, *algorand.com*, https://www.algorand.com/, [Accessed: 2021-06-05], 2021.

[5] C. Lepore, M. Ceria, A. Visconti, U. P. Rao, K. A. Shah, and L. Zanolini, "A Survey on Blockchain Consensus with a Performance Comparison of PoW, PoS and Pure PoS", *Mathematics*, vol. 8, no. 10, 2020, ISSN: 2227-7390. DOI: 10.3390/math8101782. [Online]. Available: https://www.mdpi.com/2227-7390/8/10/1782.

[6] R. C. Merkle, "A Digital Signature Based on a Conventional Encryption Function", in *Advances in Cryptology — CRYPTO '87*, C. Pomerance, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 1988, pp. 369–378, ISBN: 978-3-540-48184-3.

[7] M. Vukolić, "The Quest for Scalable Blockchain Fabric: Proof-of-Work vs. BFT Replication", in *Open Problems in Network Security*, J. Camenisch and D. Kesdoğan, Eds., Cham: Springer International Publishing, 2016, pp. 112–125, ISBN: 978-3-319-39028-4.

[8] A. de Vries, *Bitcoin Energy Consumption Index*, https://digiconomist.net/bitcoin-energy-consumption, [Accessed: 2021-11-04], 2021.

[9] C. T. Nguyen, D. T. Hoang, D. N. Nguyen, D. Niyato, H. T. Nguyen, and E. Dutkiewicz, "Proof-of-Stake Consensus Mechanisms for Future Blockchain Networks: Fundamentals, Applications and Opportunities", *IEEE Access*, vol. 7, pp. 85 727–85 745, 2019. DOI: 10.1109/ACCESS.2019.2925010.

[10] Sunny King and Scott Nadal, *PPCoin: Peer-to-Peer Crypto-Currency with Proof-of-Stake*, https://www.peercoin.net/whitepapers/peercoin-paper.pdf, [Accessed: 2021-06-05], 2012.

[11] Coinbase, *coinbase*, https://www.coinbase.com/, [Accessed: 2021-10-26], 2021.

[12] Ethereum, *Ethereum Proof-of-Stake Consensus Specifications*, https://github.com/ethereum/consensus-specs, [Accessed: 2021-10-26], 2021.

[13] Web3 Foundation, *Polkadot*, https://polkadot.network/, [Accessed: 2021-10-27], 2021.

[14] Avalanche, *avax.network*, https://www.avax.network/, [Accessed: 2021-10-28], 2021.

[15] Team Rocket, M. Yin, K. Sekniqi, R. van Renesse, and E. G. Sirer, *Scalable and Probabilistic Leaderless BFT Consensus through Metastability*, 2020. arXiv: 1906.08936 [cs.DC].

[16] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich, "Algorand: Scaling Byzantine Agreements for Cryptocurrencies", in *Proceedings of the 26th Symposium on Operating Systems Principles*, ser. SOSP '17, Shanghai, China: Association for Computing Machinery, 2017, 51–68, ISBN: 9781450350853. DOI: 10.1145/3132747.3132757. [Online]. Available: https://doi.org/10.1145/3132747.3132757.

[17] Rand Labs, *AlgoExplorer - Algorand Blockchain Explorer*, https://algoexplorer.io/, [Accessed: 2021-06-05], 2021.

[18] Algorand Inc., *Algorand Developer Docs*, https://developer.algorand.org/docs/, [Accessed: 2021-09-14], 2021.

[19] J. Chen, S. Gorbunov, S. Micali, and G. Vlachos, *Algorand Agreement: Super Fast and Partition Resilient Byzantine Agreement*, Cryptology ePrint Archive, Report 2018/377, https://eprint.iacr.org/2018/377, 2018.

[20] Algorand Inc., *go-algorand*, https://github.com/algorand/go-algorand, [Accessed: 2021-07-10], 2021.

[21] S. Micali, S. Vadhan, and M. Rabin, "Verifiable Random Functions", in *Proceedings of the 40th Annual Symposium on Foundations of Computer Science*, ser. FOCS '99, USA: IEEE Computer Society, 1999, p. 120, ISBN: 0769504094.

[22] Algorand Foundation Ltd., *Algorand Foundation*, https://algorand.foundation/, [Accessed: 2021-09-15], 2021.

[23] O. Ersoy, Z. Ren, Z. Erkin, and R. L. Lagendijk, "Transaction Propagation on Permissionless Blockchains: Incentive and Routing Mechanisms", in *2018 Crypto*

Valley Conference on Blockchain Technology (CVCBT), 2018, pp. 20–30. DOI: 10.1109/CVCBT.2018.00008.

[24] Y. He, H. Li, X. Cheng, Y. Liu, C. Yang, and L. Sun, "A Blockchain Based Truthful Incentive Mechanism for Distributed P2P Applications", *IEEE Access*, vol. 6, pp. 27 324–27 335, 2018. DOI: 10.1109/ACCESS.2018.2821705.

[25] R. von Seck, "Incentives for Participation in Anonymising Networks", Master's thesis, Technical University of Munich, 2018.

[26] ipinfo.io, *ipinfo.io*, https://ipinfo.io/, [Accessed: 2021-10-28], 2021.

[27] Amazon Web Services, Inc., *AWS Pricing Calculator*, https://calculator.aws/, [Accessed: 2021-11-09], 2021.

[28] M. Fooladgar, M. H. Manshaei, M. Jadliwala, and M. A. Rahman, "On Incentive Compatible Role-based Reward Distribution in Algorand", 2019. arXiv: 1911.03356 [cs.CR]. [Online]. Available: https://arxiv.org/abs/1911.03356.

[29] M. Conti, A. Gangwal, and M. Todero, *Blockchain Trilemma Solver Algorand has Dilemma over Undecidable Messages*, 2019. arXiv: 1901.10019 [cs.CR]. [Online]. Available: https://arxiv.org/abs/1901.10019.

[30] Y. Liu, "Investigating Byzantine Agreement Consensus Algorithm of Algorand", Master's thesis, University of Technology Sydney, 2020.

[31] M. A. Alturki, J. Chen, V. Luchangco, B. Moore, K. Palmskog, L. Peña, and G. Roşu, "Towards a Verified Model of the Algorand Consensus Protocol in Coq", *Formal Methods. FM 2019 International Workshops*, 362–367, 2020, ISSN: 1611-3349. DOI: 10.1007/978-3-030-54994-7_27. [Online]. Available: https://dx.doi.org/10.1007/978-3-030-54994-7_27.

[32] M. Bartoletti, A. Bracciali, C. Lepore, A. Scalas, and R. Zunino, *A formal model of Algorand smart contracts*, 2021. arXiv: 2009.12140 [cs.CR]. [Online]. Available: https://arxiv.org/abs/2009.12140.

[33] J. Brown-Cohen, A. Narayanan, A. Psomas, and S. M. Weinberg, "Formal Barriers to Longest-Chain Proof-of-Stake Protocols", in *Proceedings of the 2019 ACM Conference on Economics and Computation*, ser. EC '19, Phoenix, AZ, USA: Association for Computing Machinery, 2019, 459–473, ISBN: 9781450367929. DOI: 10.1145/3328526.3329567. [Online]. Available: https://doi.org/10.1145/3328526.3329567.

[34] F. Baldimtsi, V. Madathil, A. Scafuro, and L. Zhou, "Anonymous Lottery In The Proof-of-Stake Setting", in *2020 IEEE 33rd Computer Security Foundations Symposium (CSF)*, 2020, pp. 318–333. DOI: 10.1109/CSF49147.2020.00030.

[35] P. Zheng, Z. Zheng, X. Luo, X. Chen, and X. Liu, "A Detailed and Real-Time Performance Monitoring Framework for Blockchain Systems", in *2018*

IEEE/ACM 40th International Conference on Software Engineering: Software Engineering in Practice Track (ICSE-SEIP), 2018, pp. 134–143.

[36] The Linux Foundation, *Hyperledger Blockchain Performance Metrics*, `https://www.hyperledger.org/learn/publications/blockchain-performance-metrics`, [Accessed: 2021-06-12], 2018.