universität wien

MASTERARBEIT / MASTER'S THESIS

Titel der Masterarbeit / Title of the Master's Thesis

„DIoT: Blockchain-based framework for trustworthy communication in IoT networks“

verfasst von / submitted by

Lukas Walisch BSc

angestrebter akademischer Grad / in partial fulfilment of the requirements for the degree of

Master of Science (MSc)

Wien, 2021 / Vienna, 2021

Studienkennzahl lt. Studienblatt / degree programme code as it appears on the student record sheet:	UA 066 921
Studienrichtung lt. Studienblatt / degree programme as it appears on the student record sheet:	Master Computer Science
Betreut von / Supervisor:	Univ.-Prof. Dipl.-Math. Dr.Peter Reichl, Privatdoz.

Acknowledgements

I am very thankful for having received a scholarship from the authority for federal aid for students in Austria (Österreichische Studienbeihilfenbehörde), which allowed me to focus entirely on the research, implementation and writing of this thesis.

In this regard, I also want to express my gratitude to my supervisor, Professor Peter Reichl, who greatly supported my request to prolong my scholarship. Further, I want to thank Professor Reichl for the valuable advice and guidance throughout the course of research and writing of this master thesis.

My thanks and appreciation are also directed to Nemanja Ignjatov, research and teaching staff member at the University of Vienna, for the time and expertise he offered to me in numerous meetings to discuss critical aspects of my studies and for his suggestions and proofreading during the writing process. I believe that his advice greatly improved the end result of this thesis.

Finally, I want to thank my student companions Jenny and Richard – I'm looking forward to see you finish your master theses very soon – and my family, who accompanied me all the way through my bachelor and master studies.

Abstract

Centralized entities, such as cloud services, message brokers or Public Key Infrastructure (PKI), pose a problem for the trustworthiness of Internet of Things (IoT) services, as they take up a central role in establishment of communication as well as data storage and therefore hide the internal data processing from the users. At the same time they are an attractive target for attackers to steal, manipulate or expose data captured by IoT devices. This thesis approaches the stated problem with the development of a messaging framework based on the decentralized Blockchain technology. Blockchains distribute the task of data storage among the nodes of a network and ensure that state transitions are only executed when the majority of network nodes endorse them, i.e., reaching consensus. As a result of this decentralization, full transparency of the stored data is given and there is no longer a single target for attackers. The proposed *Decentralized IoT Framework (DIoT)* differs from existing general-purpose Blockchain solutions as it is designed to incorporate existing devices of an IoT network, such as single-board computers or network gateways, as full Blockchain nodes and can therefore operate without the need of external mining infrastructure and transaction fees and still provide transparent and manipulation-resistant data storage and communication to IoT networks. Apart from the design and implementation of the DIoT framework, a series of experiments are conducted to evaluate functional and non-functional aspects of a DIoT Blockchain in execution.

Kurzfassung

Netzwerkstrukturen mit zentralen Anlaufstellen, wie Cloud Services, Message Broker oder Public Key Infrastructure (PKI), stellen ein Problem für die Vertrauenswürdigkeit von IoT-basierten Anwendungen dar, da diese zentralen Entitäten sowohl beim Kommunikationsaufbau zwischen IoT Geräten und zu Endverbrauchern, als auch bei der Verarbeitung und Speicherung von Daten eine unerlässliche Rolle einnehmen und dabei die internen Vorgänge vor der Außenwelt verbergen. Zusätzlich bieten zentrale Netzwerkknoten eine attraktive Angriffsfläche für Angreifer, Zugriff auf gespeicherte Daten zu erhalten um diese zu manipulieren oder zu verbreiten. Die vorliegende Arbeit widmet sich dieser Problematik mit der Entwicklung eines Messaging-Frameworks, welches auf der dezentralen Blockchain-Technologie basiert. Eine Blockchain repliziert die gespeicherten Daten auf zahlreiche Netzwerkknoten und stellt sicher, dass Änderungen an den gespeicherten Daten nur nach einer Mehrheitsentscheidung der Netzwerkknoten, dem sogenannten *Consensus*, durchgeführt werden. Die dezentrale Struktur einer Blockchain ermöglicht hohe Transparenz der gespeicherten Daten, da alle Änderungen auf vielen Netzwerkknoten repliziert werden und darüber hinaus zentrale Angriffspunkte auf die gespeicherten Daten im Netzwerk eliminiert werden. Das im Rahmen dieser Arbeit vorgestellte Decentralized IoT Framework (DIoT) unterscheidet sich von den vorhandenen allgemeinen Blockchain Lösungen durch die explizite Konzipierung für IoT-Netzwerke und die Einbindung von vorhandener IoT Infrastruktur, wie zum Beispiel Einplatinenrechner oder Netzwerk-Router, als eigenständige Blockchain-Knoten und kommt dadurch ohne externer Mining-Infrastruktur und ohne Transaktionskosten aus. Zusätzlich zu Design und Implementierung des DIoT Frameworks wird in der Arbeit auch eine Reihe an Experimenten zum Zweck der Evaluierung von funktionalen und nichtfunktionalen Anforderungen des DIoT Frameworks dokumentiert.

Contents

List of Tables

List of Figures

Listings

1. Introduction

Blockchain technology is again in the media spotlight due to the latest bull-run of crypto currencies which started to accelerate at the beginning of 2021. Beyond the use in crypto currencies, there is potential to use Blockchain technology in other information systems as well, as it is the key for the transition of traditional central state management in distributed systems to a completely decentralized way of managing state [Mat16]. Centralized state management is very efficient but has the disadvantage that the stored data can be manipulated arbitrarily by the entity that stores the data or by an attacker that manages to compromise this entity. The *Rebooting the Web of Trust (RWOT)* group describes this problem of centralized systems with the example of the centralized PKI for resolving domain names in the internet [ABB+15]. Decentralized state management with a Blockchain brings the major advantage of transparency and trustworthiness that the stored data cannot be manipulated by a single entity, since the data is stored redundantly on different independent nodes and only the consensus of a majority of the Blockchain nodes can lead to altering the Blockchain state [Mat16].

In recent years, Blockchain technology found many applications, for instance smart contracts, i.e., digital contracts between multiple parties, which are enforced by the Blockchain, decentralized file storage, or decentralized ledgers of asset ownership (e.g. notary services, real estate, etc.) [CPV16].

The IoT can profit from the decentralized way of storing data as well, since IoT devices produce high amounts of data, with a significant part being sensitive data that must be protected against manipulation. However, the integration of Blockchain technology in IoT networks seems to pose a particularly difficult challenge due to the limitation of resource constrained IoT devices and on the other side the high demand of computing power for mining new blocks in current Blockchains [WZN+19].

1.1. Motivation

Many survey papers, such as [FCFL18, WZN+19, KS18], review the use of Blockchain technology in IoT beyond crypto-currencies to address ongoing IoT challenges such as security, privacy, transparency, and trustworthiness. These survey papers claim, that the integration of Blockchain technology in IoT for communication is valuable, especially for improving trustworthiness and transparency of services provided in IoT networks, as the data stored on the Blockchain remains immutable and available for the entire Blockchain lifetime.

After conducting a literature research on the integration of Blockchain technology into IoT networks and which solutions are already available, it became apparent that several

experiments, such as [BM16, CBWF17, BBG+17], have been conducted with general purpose Blockchain platforms to realise particular IoT applications where Blockchain technology replaces traditional centralized communication systems. However, the literature research did not reveal any effort to develop a Blockchain-based communication framework, which is specifically designed to the requirements of IoT.

The most common way in these experiments to integrate a Blockchain in an IoT network is by installing a light Blockchain client on IoT devices or fog nodes so they can access the Blockchain for trustworthy and secure information exchange with other network nodes. However, IoT devices cannot actively participate in validating the Blockchain transactions and securing the Blockchain through mining because mining requires high computational effort and IoT devices have limited processing capabilities [WZN+19]. Therefore, additional mining nodes are introduced to the network which are responsible for mining new blocks and securing the Blockchain. The IoT devices running Blockchain light-clients have to trust mining nodes to properly execute the consensus mechanism and block validation.

The Blockchain can only be kept live and secure against data manipulation as long as a majority of mining nodes (or more accurately, the mining nodes with the majority of mining power) execute the consensus mechanism and the block validation correctly. If a majority of malicious mining nodes forms in the mining network, the liveness and manipulation resistance of the Blockchain-based communication is at risk.

In a small network of Blockchain miners, for instance a mining network that is specifically set up to facilitate the communication of IoT devices such as [BBG+17], the risk of a an attacker taking over the Blockchain consensus is rather high, as the attacker just has to have more mining power than the rest of the network.

Other attempts to realise Blockchain-based IoT applications, for instance [BM16, CBWF17], utilize the already existing mining infrastructure of large Blockchain networks such as the Ethereum mining network. In large public mining networks, such as the mining networks of Bitcoin and Ethereum, it is virtually impossible for an individual malicious entity to compete with the rest of the mining network, which attests these Blockchains a high trustworthiness through a high decentralization of the Blockchain mining network. However, large scale mining networks usually have downsides as well. They provide lower throughput and latency for transactions of a particular service, as the Blockchain resources have to be shared with other services. Additionally, they require significant amount of transaction fees that must be paid by the transaction sender.

With a new Blockchain solution that is specifically designed for IoT communication, a more suitable consensus mechanism can be chosen that allows the operation of a Blockchain node with existing devices in IoT networks, such as single-board computers, network gateways, mobile devices and small servers, to eliminate the need for additional mining nodes. It is believed by the author, that such a specialized Blockchain solution, used in IoT communication scenarios, can outperform existing general-purpose Blockchains regarding some of the just mentioned aspects, for instance the degree of decentralization, transaction fees, latency or throughput. These aspects are manifested as the research questions in the next section.

1.2. Research questions

The following research questions frame the research of this thesis.

Q1: How can trustworthiness of IoT services be improved with Blockchain technology? Centralized services, such as cloud services and PKI, are ubiquitous in IoT networks for processing, storage and transfer of data produced from IoT devices but also raise concerns about trustworthiness as the internal processing is hidden from the users. Blockchain technology is a potential alternative to centralized services for trustworthy data storage and transfer but also introduces integration challenges, especially for networks of resource-limited IoT devices. The question if the arising challenges make it even possible and beneficial to integrate Blockchain technology in IoT applications is a major concern of this thesis.

Q2: Can existing IoT infrastructure be used to deploy a Blockchain-based communication system in IoT networks? This research question builds upon Q1 and puts the current prevalent approach into question, which is to integrate Blockchain technology into IoT networks by introducing additional mining nodes to the existing network. This approach can cause some inconvenient consequences such as transaction fees for Blockchain transactions or operation costs for the dedicated mining nodes. It would be more economic to run the Blockchain only with hardware that already exists in IoT networks, such as single-board computers, network gateways, mobile devices and small servers. This may require a Blockchain solution that is specifically designed for IoT applications.

Q3: Can a Blockchain-based communication system achieve a transaction latency of below ten seconds in a mid-size network of IoT nodes? Blockchain-based communication systems are by design high-latency systems with a transaction latency of up to several minutes [Vuk15], which is identified as challenge for the integration in IoT systems by several survey papers (e.g. [FCFL18, WZN+19, GHY18]). A reasonable goal for a Blockchain-based communication system for IoT networks must be to perform comparable to the currently existing general-purpose Blockchains with rather low transaction latency, such as Cosmos (approx. 8 seconds[1]) or Ethereum (approx. 14 seconds[2]) and scale up to a network size of several hundred nodes. This would qualify a Blockchain-based communication system as a practical alternative to centralized communication in many IoT use cases that require high amount of trustworthiness including also use cases with user interaction, as the aimed latency aligns with the human acceptance of computer-system response time of approx. 10 seconds suggested by Nielsen et al. [Nie93].

[1] `https://atom.tokenview.com/en/blocklist`, accessed: 2021-07-07
[2] `https://eth.tokenview.com/en/blocklist`, accessed: 2021-07-07

1.3. Contribution

In this thesis, the DIoT (Decentralized IoT) framework is proposed and implemented to a proof-of-concept state. This framework allows to develop communication systems for IoT networks which use a Blockchain to transfer and store messages.

DIoT uses lightweight Tendermint consensus [Buc16] that does not involve resource-intensive mining, which allows to run Blockchain nodes on devices that already exist in IoT networks, such as single-board computers, network gateways and mobile devices, without the need of an additional mining network. Additionally, DIoT introduces a novel Validator Management process, that periodically swaps out validator nodes to maintain a dynamic validator set for a high degree of decentralization in the Blockchain network.

Apart from the design and implementation of the DIoT framework, the thesis also covers the development of simulation tools and the deployment of a DIoT Blockchain in a mid-size network of virtual machines to test the DIoT framework extensively and evaluate functional and non-functional requirements.

1.4. Thesis outline

The rest of the thesis is structured as follows. Chapter 2 covers the state-of-the-art of Blockchain and IoT research as well as existing solutions with relevance to the thesis' topic. In Chapter 3 the two most important sub-problems of this thesis' solution are elaborated and solved, that are the choice of a suitable underlying Blockchain technology for the DIoT framework and the design of the novel Validator Management process. Chapter 4 addresses additional design decisions and implementation effort to build a complete and usable Blockchain-based messaging framework around the core concepts from Chapter 3. The deployment of a test and evaluation scenario of a DIoT Blockchain, as well as several evaluation experiments are documented in Chapter 5. Eventually, in Chapter 6 the insights of the research effort are summarized, limitations of the presented solution are identified and further steps to improve and extend the DIoT framework are suggested.

2. Related work

This chapter covers general concepts and the state-of-the-art of Blockchain technology and IoT to introduce the reader to the research field of this thesis. The last section of this chapter focuses specifically on Proof of Work (PoW) algorithms as they provide an essential building block for the solution design.

2.1. Blockchain fundamentals

In this section, the fundamentals of Blockchain technology are presented, that were gathered from several survey papers of the area [CPV16, ZXD+18, SZE+18, WZN+19, Vuk15, DAAB+18]. The focus is directed towards the Blockchain data structure and the high-level message flow in a Blockchain network, as well as the main types of existing consensus mechanisms.

2.1.1. Concepts and terminology

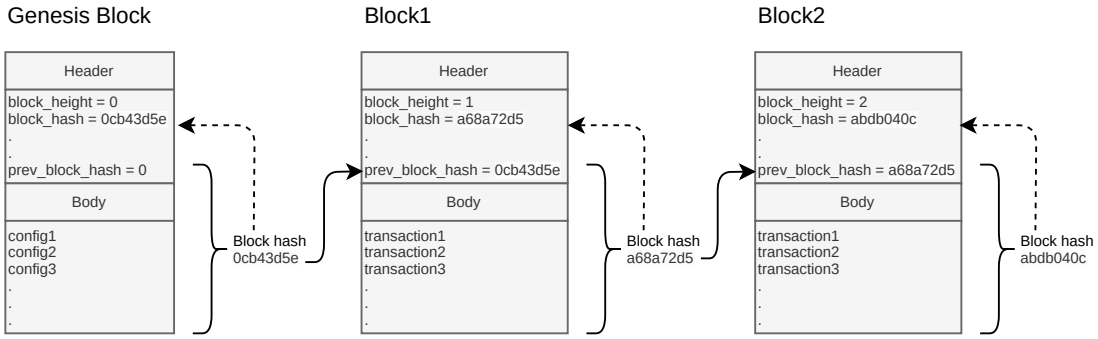


Figure 2.1.: Blockchain data structure

Blockchain data structure A Blockchain is a data structure that consists of blocks of data, that are linked together sequentially. As shown in Figure 2.1, every block contains a *block hash*, generated from the data payload of the block, as well as the block hash of its predecessor, which ensures that the order of the blocks is not changed at a later point and also that the content of the blocks cannot be manipulated once the next block is appended. Besides that, the block header also carries a *block height* value, that starts from zero and increments with every new block added to the chain. The very first block of

a Blockchain is called *genesis block* and may contain configuration data of the Blockchain [ZXD+18].

The Blockchain data structure is replicated among multiple nodes (e.g. personal computers (PC), servers, IoT devices), forming a Blockchain network [SZE+18], and kept synchronized through a distributed consensus mechanism (see Figure 2.2). The Blockchain specifies certain rules how the state can be manipulated. Users can send transactions to a Blockchain node to change the Blockchain state. If a transaction doesn't violate any of the rules enforced by the Blockchain, the transaction will eventually be propagated to all other Blockchain nodes and stored as part of a block in their local Blockchain [CPV16].

Blockchain state replication (consensus) A key aspect of every Blockchain is the replication of all data blocks, i.e., the Blockchain state, on every active Blockchain node, which results in the decentralized structure (see Figure 2.2). The transactions, that users issue to a Blockchain client are broadcasted to all other Blockchain nodes. Blockchain clients store all incoming transactions in a transaction pool [CPV16]. To keep the Blockchain state of all Blockchain nodes consistent, it is necessary that the nodes agree upon in which order the incoming transactions are applied to the Blockchain state. This agreement is referred to as *consensus* in the Blockchain terminology and it is an essential part of every Blockchain. A consensus mechanism, or consensus protocol, defines the communication between the Blockchain nodes that is needed to reach consensus [WZN+19]. Different Blockchains use different types of consensus mechanisms such as PoW, Proof of Stake (PoS), Proof of Authority (PoA) and consensus mechanisms based on Byzantine Fault Tolerance (BFT) [Vuk15, DAAB+18].

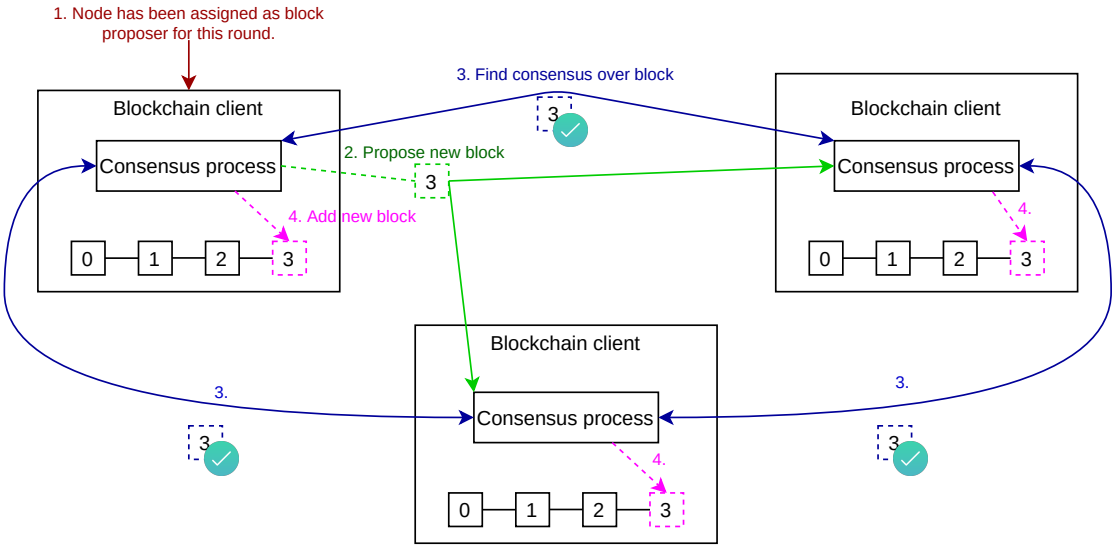


Figure 2.2.: A complete round of consensus to add a block

By executing one round of a consensus mechanism, the goal is to append a new block to the Blockchain. As illustrated in Figure 2.2, each round of consensus can be seen as a

four step process. The process is loosely based on the Tendermint consensus mechanism described by Buchman [Buc16], but generalized to show the essential steps of a consensus round that apply to all types of consensus mechanisms. The four steps are as follows:

1. In the first step of consensus, a node is assigned to propose the next block. This can either be a competition for example where the node that solves a computational expensive task first is assigned as block proposer (e.g. PoW [Nak08]) or simple round-robin selection of a network node (e.g. BFT, PoA [DAAB+18]).

2. In the next step, the block proposer of this consensus round broadcasts a block proposal to the other Blockchain nodes via a peer-to-peer network.

3. In step 3, the Blockchain nodes verify the correctness of the block proposal. In BFT-based consensus mechanisms it is required to send further messages whether the Blockchain nodes accept the block proposal or not. If Blockchain nodes receive a certain number of acceptance messages for a block proposal, they move on to step 4 [DAAB+18]. Other consensus mechanisms (e.g. PoW, PoA, PoS) do not require this additional message transfer and move on to step four right after block validation [DAAB+18].

4. In the final step, given that the previous steps were successful, the proposed block is added to the Blockchain by each individual Blockchain node.

The rules and procedure of different consensus mechanisms are explained in more detail in Subsection 2.1.4.

Transactions Transactions are the smallest unit of information in the Blockchain terminology. The Blockchain enforces a set of rules, the transactions must adhere to. Only valid transactions will be visible as part of the Blockchain, others are rejected. The rules for a transaction can be different depending on the purpose of the Blockchain. A general transaction rule is that they must contain a valid signature of the sender [WZN+19]. In crypto-currency Blockchains, an important rule is that Blockchain tokens, that have already been spent on an earlier transaction may not be spent again, i.e., double-spending (see Appendix C.1). A custom Blockchain rule can, for instance, enforce that only certain users may send certain types of transactions, as it is done in the DIoT framework for identity management. The DIoT framework enforces the rule that only the coordinator node may send transactions to add further nodes to the Blockchain network (see Subsection 4.4).

Blocks A block is an important unit for the synchronization of state in a Blockchain. Because the consensus mechanism is very time-consuming, it is not executed for a single transaction, but rather for a batch of valid, non-contradictory transactions, which is called a block [Buc16]. In every consensus round, Blockchain nodes make a collective decision whether a proposed block should be added to the Blockchain. Once a block has been added to the Blockchain, the transactions of the block can be considered finalized, i.e.,

they are visible to all Blockchain nodes and cannot be manipulated in the future [Vuk15]. However, in some types of consensus mechanisms (e.g. PoW, PoS), it is possible that a newly added block is abandoned in favor of a conflicting block of another block proposer shortly after, which is called a fork [ZXD+18]. In this case, the transactions contained in the abandoned block are reverted into a pending-state. Therefore, waiting for additional blocks to be appended is recommended before relying on transactions of a block in some Blockchains [Vuk15].

Mining/Minting Mining is a widely used term in the Blockchain realm that was forged by the crypto-currency community. It results in new blocks being created and new amount of the currency (also called coins or tokens) is distributed. Mining actually occurs only in PoW Blockchains and refers to Blockchain nodes solving a cryptographic puzzle of hashing random byte sequences to produce a desired hash. The node that solves the puzzle first gets to create the next block of the Blockchain and gain a reward in the form of newly minted amount of the currency [But13]. In PoS, creating new blocks is called minting, as this process does not incorporate Central Processing Unit (CPU) work but still results in creating new amount of currency alongside with the new block [KN12].

Fault tolerance The fault tolerance of a Blockchain describes the voting power of malicious nodes (also called *byzantine nodes*) in the Blockchain network, which is required to manipulate the outcome of the consensus mechanism or prevent the Blockchain from creating new blocks. If the byzantine nodes possess a voting power of less than the fault tolerance value, the Blockchain guarantees that so called *correct nodes* of the network, i.e., nodes with no malicious intent, will find consensus. Fault tolerance varies for different consensus mechanisms. For instance, PoW has a fault tolerance of 25% byzantine voting power, while BFT has 33% [Vuk15].

2.1.2. Attacks on Blockchains

Blockchains are known for being tamper-proof and for their resilience against attacks which originates from their decentralized nature. Even though it is made difficult, it is not impossible to attack Blockchains. Attacks that aim at getting hands on user credentials and private keys such as Phishing attacks [DTH06] or general network attacks such as Distributed Denial of Service (DDoS) [LRST00] are not covered, as these attacks do apply to centralized systems as well. Instead, the focus lies on attacks that are unique to Blockchain technology and are directed at the liveness and security of Blockchain systems. Because there is no single point of failure in a Blockchain network, it doesn't make sense to attack a few nodes of the network, therefore attacks focus on the consensus process. Attacking a Blockchain requires the attacker to control as significant amount of voting power to manipulate the consensus process [WZN+19]. The goal of an attack could either be to stop the Blockchain from creating further blocks, delaying or censoring certain transactions or to exploit the eventual consistency of a Blockchain, i.e., Blockchain forks, with a double-spending attack. All three types are briefly described.

The first type of attack can be executed in BFT-based Blockchains, where the attacker needs control over more than 1/3 of the voting power to launch the attack. If byzantine nodes do not send any messages during the consensus process, then the consensus process does not terminate and no further blocks are created [Tena].

The second attack can be carried out with any fraction of voting power in the consensus and in both BFT-based and PoW-based consensus mechanisms. It is more effective, the higher the voting power of the attacker. Based on the voting power, the attacker can propose a block more or less frequently. Whenever the attacker has the chance to propose a block, he can decide to not include certain transactions on purpose to cause delaying of these transactions, for instance based on who sends or receives the transaction. If the voting power of the attacker is high enough, i.e., >50% in PoW or >66% in BFT, he can censor the Blockchain arbitrarily [FCFL18].

Double-spending is another infamous Blockchain attack, especially for crypto-currency Blockchains. Since it has no direct relevance for the solution of this thesis, it is further elaborated in Appendix C.1.

2.1.3. Blockchain taxonomy

Since Blockchain technology started to adopt BFT consensus from distributed computing in recent years (e.g. Tendermint [Buc16], Hyperledger Fabric [ABB+18]), which follows a completely different approach than the original PoW consensus, new terms for the classification of Blockchains emerged. Permissionless, permissioned, public and private are the types of Blockchains that can be encountered in the literature. The scientific community does not seem to have a clear and unambiguous definition of these terms yet, therefore, for the context of this thesis, a Blockchain taxonomy based on [ZXD+18, Vuk15, But15] is defined. The taxonomy distinguishes Blockchains with permissionless or permissioned consensus and into public or private Blockchains.

- *Permissionless/Permissioned consensus*: In permissionless Blockchains, nodes have unrestricted access in participating in consensus, such as PoW and PoS Blockchains. In permissioned Blockchains, only specific nodes may participate in consensus which can be specified in the genesis block of the Blockchain or at runtime through voting of the active consensus nodes. BFT and PoA are common Permissioned Blockchain consensus mechanisms [DAAB+18].

- *Public/Private*: This property determines if the Blockchain allows everyone to read and send transactions to the Blockchain or if it permits read/write access for only a specified set of users. Public Blockchains are publicly known, allowing everyone to read and create transactions, such as crypto-currency Blockchains. Private Blockchains are closed to the public either for write access or both read and write [But15], for example a Blockchain that is used for a supply chain that involves multiple companies. In this case, only the companies involved are allowed to create transactions but external read access may be granted for auditing purposes. Many Blockchains can either be deployed privately or publicly, for example Ethereum is a

public Blockchain by default but Smart Contracts allow to create a private space on top of the public Blockchain where the smart contract implements access control.

2.1.4. Blockchain consensus

The different consensus mechanisms, which are relevant to this thesis, are briefly discussed in this subsection. This includes the well-known permissionless consensus mechanisms PoW [Nak08] and PoS [KN12] as well as important permissioned consensus mechanisms based on BFT [CLO99] and PoA [DAAB+18].

PoW consensus For the Bitcoin Blockchain, Nakamoto [Nak08] utilized a well-known PoW algorithm from Back [Bac02] as a competition between miners to decide for a block proposal. The consensus mechanism, that is used in the Bitcoin Blockchain is since then referred to as *Proof of Work consensus*. A key characteristic that makes PoW consensus suitable as a permissionless consensus mechanism is its protection against sybil attacks. A sybil attack is referred to as an attack in a distributed system where the attacker creates multiple fake identities to to influence voting-based decision in a network [Dou02]. In the case of Blockchains, sybil attacks are used by attackers to reach a majority of votes to dominate the consensus. With a majority of voting power, an attacker can vote for or against the creation of particular blocks, i.e., censorship, or prevent the system from reaching consensus.

PoW protects against sybil attacks by enforcing a one-CPU-one-vote policy instead of one-identity-one-vote [Nak08], which is essentially binding the voting power of an entity to its CPU power. CPU power is costly to increase, therefore gaining a majority of CPU power in the network is very expensive and almost impossible for larger Blockchain networks with many miners such as the Bitcoin network.

In PoW consensus, miners compete to create a block of transactions that is added to the Blockchain. They bundle valid transactions that are received over the Peer-to-Peer (P2P) Blockchain network in a block and generate a hash value from the block data. The bit-representation of the hash must have a certain number of leading zeros to be accepted by the network as a valid PoW. To achieve such a hash, miners put a so called *nonce* value, i.e., an arbitrary byte sequence to modify the resulting hash, into the block header. If the hash of the block including the nonce does not have the required leading zeros, they increment the nonce and apply the hash function again to receive a different hash value and repeat the process until a hash value with the desired number of leading zeros is obtained. Eventually, when one miner manages to get the required hash, it publishes the block with the nonce to the network and claims a reward in the form of newly minted crypto currency tokens and transaction fees of the transactions contained in the block. Other miners receiving the block can easily verify the correctness of the PoW by calculating a single hash value from the proposed block to see, if the resulting hash contains the required leading zeros. When a miner receives a valid block over the P2P network, it appends it to its Blockchain and starts mining the next block. Due to network latency, it is possible that multiple nodes solve the PoW task within a short amount of

time and publish different valid blocks. Depending on the distance to the other miners, some will accept the first block, and others will accept the other block. Eventually, all Blockchain versions propagate to all nodes and when a node receives a version that has more blocks than its current one, it will continue to mine the longer chain. This is called the longest-chain rule and it defines that the longest chain is the publicly accepted one, as it is the one with the most CPU power spent on [Nak08]. All correct nodes will adhere to that rule and as long as the correct nodes together have the majority of CPU power, they will agree upon which version of the Blockchain is the accepted one and find a public consent.

PoW as it is used by Bitcoin suffers from a high energy footprint caused by mining [OM14]. Furthermore, the computing task occupies the node's CPU and prevents it from being used for other work.

Latency and throughput are other problems that make it impractical for many use cases. Bitcoin creates a block every 10 minutes and has a throughput of approximately 7 transactions per second [Vuk15]. Further, it is recommended to wait 3 to 6 additional blocks to be sure that the block is not orphaned at a later point in time, i.e., it is included in a block that is not in the longest chain and therefore the transactions in it are rolled back to an unconfirmed state. Appendix C.2 states, how latency and throughput have been improved in newer PoW consensus mechanisms.

There are several variants of PoW consensus that generally follow the same steps as PoW, such as PoS, Proof of Capacity, Proof of Burn and Proof of Elapsed Time [Mat16]. They only differ in the way, nodes are assigned for creating a new block to improve some of the properties of PoW. The essential requirement for the *proof* in permissionless Proof-of-* consensus mechanisms is the protection against sybil attacks, therefore the *proof* must make it very expensive and unattractive for an attacker to gather a high number of votes in the consensus.

PoS consensus PoS was the first alternative approach to PoW for Blockchain consensus proposed by King and Nadal [KN12]. To participate in consensus, a node has to lock a part of its crypto tokens, i.e., *stake*, for a period of time. The *coinAge* is a value calculated as $lockedStake \times time$ [KN12]. All nodes are able to see the locked stake of every single node. For every new block, every consensus node calculates the coin age of all other nodes and the node with the highest coin age value is assigned as block proposer. Therefore, the longer stake is locked, the higher the chance of creating a block. The reward of creating a block are the transaction fees of the block transactions and newly minted crypto tokens. After a block was created, the locked stake of the block creator is unlocked and the gathered coin age is destroyed to prevent the most wealthy nodes from dominating the consensus [KN12]. In some Blockchains the concept of coin age is combined with a PoW task with reduced complexity based on the coin age value to make the proposer assignment less predictable. The benefit over PoW is in the reduced energy consumption, as there is much less effort spent on the PoW hashing task, if any [KN12].

Given this positive characteristic of energy-efficient consensus, PoS implementations also tend to be more complex due to additional logic to lock stake and to address the Nothing-at-Stake problem. Nothing-at-Stake describes the motivation of a consensus node to propose blocks on multiple conflicting Blockchain branches after a fork. This way, the consensus node is rewarded regardless of which branch the Blockchain converges to. This is not encouraged because it prevents resolving forks [Sal20]. PoS chains punish such behavior of consensus nodes through so called slashing, which is destroying the locked stake of a consensus node [BG19].

BFT consensus The Byzantine Generals Problem and the idea of BFT consensus was first framed by Lamport et al. [LSP82] already in 1982. The Byzantine Generals Problem defines the problem of correct nodes reaching consensus over a common state with unreliable communication in the presence of one or more so called byzantine nodes. Byzantine nodes in a distributed system can act maliciously trying to prevent the system from reaching consensus by sending inconsistent responses or by not responding at all. Byzantine Nodes can even coordinate over another communication channel and attack the system with a collective strategy. BFT systems can tolerate a certain amount of byzantine nodes in a system and still guarantee that correct nodes find consensus over the system state. The Practical Byzantine Fault Tolerance (PBFT) consensus mechanism by Castro et al. from 1999 [CLO99], which was originally developed for State Machine Replication in distributed databases, has recently been adopted for Blockchain consensus. BFT consensus implementations for permissioned Blockchains include Tendermint [BKT18] (used e.g. in Cosmos [KB16], Hyperledger Burrow [Dav19]), PBFT [CLO99] (used e.g. in Hyperledger Fabric [ABB+18]), and BFT-SMaRt [BSA14] (used e.g. in R3 Corda [BCGH16]).

BFT consensus mechanisms are based on the partially synchronous network model of distributed systems defined by Dwork et al. [DLS88]. The distinction between asynchronous networks and synchronous networks is based on an upper bound of message delivery. In synchronous systems, there is a fixed upper bound for the time in which a message can be delivered, which imposes a strong constrained model. If a message does not arrive in the given time span, the receiver can be certain that the sender did not send the message. In the asynchronous model, no upper bound for message delivery exists, therefore this model is weaker constrained [DLS88]. The asynchronous model does better describe information exchange in networks encountered in the real world such as the internet but it is much more difficult to define a consensus mechanisms that holds under the assumptions of this model. The partially synchronous model lies between an asynchronous and a synchronous model in terms of constraints. The model assumes a network that, when considered from any point in time, may start with an asynchronous phase but after the so called Global Stabilization Time (GST) transitions into a synchronous phase, where an upper bound for message delivery exists [DLS88].

Based on this model, BFT consensus assumes that there will always be long enough phases of synchronicity to allow to complete a round of consensus to update the system state [CV17, BKT18]. During phases where the network behaves asynchronous, i.e.,

before GST, BFT consensus does not terminate to preserve data consistency, respecting the Fischer-Lynch-Paterson impossibility result [FLP85], which states that deterministic, fault-tolerant, and deterministic consensus is impossible in fully asynchronous networks. In the boundaries of the partially synchronous model and under the additional assumption that more than two thirds of the consensus nodes behave correctly, i.e., $n \geq 3f + 1$, where n denotes the overall number of nodes and f the number of byzantine nodes [LSP82], BFT consensus guarantees termination of consensus in synchronous phases.

Because BFT consensus adheres to the model of partial synchrony by not achieving consensus in asynchronous phases, it can provide a very important Blockchain characteristic, which is known as consensus finality [Vuk15]. Consensus finality guarantees that the Blockchain will not fork as long as all consensus preconditions are fulfilled, i.e., more than two thirds being correct nodes and a partially synchronous system. In contrast, consensus mechanisms that derive from Bitcoin's PoW assume a fully asynchronous system and therefore only provide eventual consistency which manifests in so called forks of the Blockchain, where more than one branches of the Blockchain coexist until the chain converges towards one branch through a resolution rule, i.e., eventual consistency [DAAB+18]. A consequence from this is the recommendation when issuing a Bitcoin transaction, to wait three to six blocks until a transaction can be considered confirmed [Vuk15].

In BFT Consensus, timeouts are used to ensure that the consensus mechanism does not get stuck during asynchronous network phases and always restarts after some time. If a timeout is reached at some point, the consensus mechanism restarts without adding a new block. The following steps are executed in a BFT Blockchain consensus round, in this case from the Tendermint consensus mechanism [Buc16, Tenc]:

1. All nodes execute a deterministic function that determines the proposer of the round. The function should be fair, not discriminating any of the nodes. It is important that all consensus nodes, also called *validators*, know each others identity for deterministic behavior of the proposer function, which is only the case in a permissioned Blockchain.

2. The proposer node broadcasts a proposed block to all other validators. Validators wait for a proposed block or until a timeout is reached. If the timeout is reached, the consensus round is restarted from step 1.

3. If the proposed block is valid, correct validators broadcast a prevote message, indicating their acceptance of the block.

4. The validators wait until they receive a prevote message from 2/3 of all other validators or until a timeout is reached.

5. Correct validators broadcast a precommit message if the block was accepted by 2/3 majority of validators.

6. The validators wait until they receive a precommit message from 2/3 of the validators or until a timeout is reached.

7. If more than 2/3 precommit messages are received, a validator adds the block to its local chain along with the received precommit messages of at least 2/3 of validators.

BFT Consensus is only feasible in permissioned scenarios [Vuk15], i.e., where only a predefined set of nodes participate at consensus, because it cannot protect against sybil attacks. In a permissionless scenario, where every new node is automatically permitted to join the consensus, an adversary could easily spawn a high number of fake nodes (e.g. in the form of virtual machines) to gain a 1/3 fraction or even a 2/3 fraction of consensus nodes to take over consensus and perform attacks as described in Subsection 2.1.2.

PoA consensus PoA is a variant of PoW for permissioned consensus, which is implemented in Geth, the Go implementation of Ethereum, as the so called Clique protocol [Szi17]. Just like the other permissioned consensus family of BFT-based consensus mechanisms, PoA does not protect against sybil attacks for the same reason; it relies on identity-based voting. The first motivation behind PoA was to serve as an alternative for PoW in Ethereum testnets. As there is no serious mining going on in testnets, they were easy to attack. With PoA, testnets can operate with predefined mining nodes that are under the control of the Ethereum developers [Szi17]. In this consensus mechanism, a predefined set of nodes, so called authorities or sealers, take turns to propose blocks that are accepted or rejected by the network based on the Blockchain rules. Authorities must be known in the network and have a fixed order in which they propose a new block. Every authority knows based on its identifier, at which block height it is supposed to propose a new block. The authority, which is supposed to propose a block at the current block height is called the leader [DAAB$^+$18]. Because of network asynchrony, authorities cannot know how long they have to wait for the leader's block and if it will eventually arrive. Therefore, authorities are allowed to propose blocks at block heights before it is their actual turn but they need to delay these blocks to make it more likely that the leader's block arrives first at the other nodes to reduce the likelihood of a fork. If the leader does not propose a block for some reason (e.g. node is offline or the block is lost during transmission), the block of the authority that is ranked behind the leader is accepted instead. Similar as in PoW and in very contrast to BFT, nodes do not wait for a block of a specific authority to arrive, but use the first valid block that arrives to extend the Blockchain. If a conflicting block, i.e., a block with the same block height, arrives afterwards, a resolution strategy such as the longest-chain rule or Greedy Heaviest Observed Subtree (GHOST) [SZ15] (see Appendix C.2) is applied. The Ethereum version of GHOST used in Clique PoA prefers blocks of the leader before other blocks, which ensures that if the leader's block is delayed by a small amount due to unstable connection, it is still accepted eventually. But if it arrives so late that a conflicting branch is already far ahead, then it is not accepted by the GHOST resolution strategy [DAAB$^+$18]. In stable networks PoA manages to create a block with only one message broadcast, i.e. when the leader's or another authority's broadcasted block is accepted right away without any fork resolution. This is considerably faster than BFT-based consensus mechanisms, which always require at least three sequential rounds of message broadcasting [DAAB$^+$18]. On the other hand, PoA does not provide consensus finality.

2.2. Relevance of Blockchain technology in IoT

The following subsections cover important advances in IoT and bring Blockchain technology into the perspective as a potential solution for IoT use cases that require trustworthy and transparent communication and data storage.

2.2.1. Internet of Things

The idea of IoT is to move away from the paradigm of one device fulfilling one specific function towards networks of distributed things working together by sharing information and providing services to other things and human actors to solve real world problems [Bor14]. Human-machine interaction is aimed to be at a necessary minimum for configuration and setup purposes. Every-day objects that originally required active user input are equipped with microcontrollers, sensors, actuators and the ability to connect to the internet. They gather information about their surrounding through sensors and through Machine-to-Machine (M2M) communication with other devices and services to either assist human decision making or replace it to some extent [AFGM+15, Bor14]. IoT is enabled through a stack of lightweight and efficient communication protocols with compatibility to the traditional internet protocol stack. This allows embedded low-powered and resource constrained devices to communicate with every other device in the internet [PAV+12]. Applications of IoT are manifold, classified in the areas such as smart home, smart grid, smart city, agriculture, logistics, health, industry and transportation [AFGM+15, Bor14].

2.2.2. Trustworthy IoT services

From the very beginning, IoT solutions relied heavily on centralized services to relay processing and storage of collected data to servers that offer more resources than the small embedded chips in IoT devices [Bor14]. Especially cloud technology is an important enabler for the IoT paradigm, as cloud services are able to scale up well with rising number of IoT devices [AFGM+15]. However, the heavy dependence on centralized services in IoT raises concerns about the transparency and trustworthiness, that these services can offer to users [RMC+18].

In order to find a solution regarding these concerns, first the term *trust* must be defined which is not an easy task for its own. Elofson [Elo01] generally defines trust as the following:

> *"Trust is the outcome of observations leading to the belief that the actions of another may be relied upon, without explicit guarantee, to achieve a goal in a risky situation."*

If this definition of trust is applied to the situation in an IoT network, service consumers take the risk to rely on service providers to act as they promise to. Traditionally, service providers build up trust by showing competence, credibility, expertise, experience, and professionalism [Elo01]. Establishing a service in this way can be time and resource consuming. IoT, as a paradigm of interconnection between users and devices, could

profit from being able to build up trust in a faster and more automated way. Fortunately, Blockchain technology can facilitate the creation of trustworthy services through decentralization.

The distinction between centralized and decentralized services is based on who has control over the service [Hoo18]:

- Centralized services are under the control of a single entity. They can either run on a single node or be distributed over multiple nodes. Even if the service is distributed, changes of the service's state are controlled by a single service provider. Typical examples for centralized services are cloud services or Certificate Authorities (CA) that store X.509 [BSP+08] public key certificates.

- Decentralized services are distributed among different nodes of independent entities. State changes can only be accomplished if the majority of entities agrees upon a change. The key aspect which leverages trustworthiness of decentralized services is that users only have to rely on that the majority of network nodes behaves correctly, instead of a specific entity. Decentralized services can be realized with a Blockchain with customized application logic, for example the Blockchain-based IoT software update system by Boudguiga et al. [BBG+17], or by deploying a smart contract on a general purpose Blockchain platform, as it is done in the odometer fraud prevention system by Chanson et al. [CBWF17], which uses the public Ethereum Blockchain.

In most cases centralized services are preferred as they can be operated efficiently without redundant processing and storage or additional message transfer. In decentralised Blockchain-based services, on the other hand, the service state is replicated on every Blockchain node and also the nodes periodically need to reach consensus over state transitions which requires several extra messages sent between the nodes. However for use cases, that require a high amount of trustworthiness and transparency in a network of untrusted entities, it can be worth accepting the overhead that comes with decentralized services.

2.2.3. Existing IoT communication protocols and Blockchain-based communication

IoT devices use wireless technologies for communication with each other and with services. For the network layer and the data link layer of the Open Systems Interconnection (OSI) model [KR17], IoT specific protocols were developed, such as Routing Protocol for Low power and Lossy Networks (RPL) and IPv6 over Low power Wireless Personal Area Network (6LoWPAN) [AFGM+15]. Theses protocols are adapted to the power and processing limitations of IoT while remaining compatible to the protocols of the information layer and above allowing constrained devices to be easily integrated into existing network infrastructure [PAV+12].

IoT devices, for example smart home, wearables or eHealth devices, generate sensitive personal data which is why security and privacy are important challenges to address in

IoT. Thanks to the compatibility to the traditional internet protocol stack, IoT devices can use Transmission Control Protocol (TCP) [Pos81] and User Datagram Protocol (UDP) [Pos80] for communication at the transport layer allowing to leverage the existing encryption methods Transport Layer Security (TLS) [Res18] for encrypting TCP segments, respectively Datagram TLS (DTLS) [RM12] for UDP datagrams for secure communication [AFGM+15].

On the application layer, HTTPS [Res00], in combination with protocols like REST [Fie00] or JSON-RPC [JSO13], could be utilized for secure communication for example to securely transmit sensor data from an IoT device to a server. However, HTTPS is far from ideal for most IoT use cases, as HTTP message header are rather big and it is designed for client-server type communication only [Nai17]. IoT also demands M2M communication to enable information exchange between devices [Bor14]. Message Queuing Telemetry Transport (MQTT) and Constrained Application Protocol (CoAP) are application layer messaging protocols that were designed especially for IoT with support for M2M communication and small message header for efficient and power saving communication [Nai17]. CoAP is similar to HTTP as it is also designed for client-server communication but provides the additional methods OBSERVE and DISCOVER, to support M2M communication as well [SHB14]. MQTT is a messaging protocol with a central message broker that follows the publish-subscribe messaging pattern to support temporal decoupling of sending and retrieving messages for use cases that incorporate power constrained devices that switch into a power saving mode and only wake up in intervals to publish new sensor values or retrieve the latest information from other devices [BBBR19].

The mentioned IoT communication protocols of the application layer have in common that the communication always relies on central components. MQTT communication relies on the central message broker to route messages to recipients. Communication built with HTTP and CoAP requires a trusted CA for the validation of X.509 certificates as soon as they need to communicate with unknown or untrusted devices (e.g. devices controlled by other organizations). Besides that, HTTP and CoAP enforce a client-server architecture where a service provider serves multiple requests from clients and therefore act as central components as well. Central components in a network, such as message brokers, PKI or service providers, are prone to be attacked and compromised by malicious actors which can cause the whole network communication to drop out or sensitive data being deleted, manipulated or exposed [ABB+15, FBVI17]. Blockchain technology can be used as an alternative for centralized communication in computer networks. For example, Suzuki et al. [SM17] use a Blockchain communication channel to leverage auditability and transparency of distributed systems such as eHealth systems. As of now, resource constrained IoT devices might not yet be capable of running Blockchain clients on their own which prevents the usage of Blockchain technology in many IoT scenarios. Fog computing is a possible solution to this integration problem as described in Subsection 2.2.4.

2.2.4. Fog computing as opportunity for Blockchain integration

IoT faces a scalability challenge considering the ever rising number of deployed IoT devices. The IoT market research company *IoT Analytics*[1] reported a number of 12 billion connected IoT devices deployed worldwide at the end of 2020 and predicts further growth of 20% per year which would lead to 30 billion connected IoT devices by 2025. Centralized cloud services would very fast reach their resource limits in terms of connectivity infrastructure if they would have to serve millions of IoT devices at the same time. Fog computing is an important tool to cope with the massive upscaling of deployed IoT devices [YFN+19]. Fog computing is a computing paradigm that incorporates nodes such as servers and routers in the local network to provide computation, storage, networking, decision making, and data management services for nearby resource constrained devices [YFN+19]. As opposed to cloud resources, which are bound to data centers that are costly to expand once the resources are exhausted, fog nodes can be scaled up in logarithmic order to the deployed devices with relatively low additional costs to keep up with the data processing demands and relieve cloud instances [AFGM+15]. Fog nodes form an intermediate layer between IoT devices and cloud services and can perform aggregation and preprocessing of data produced by IoT devices [AFGM+15]. The transition of IoT networks away from being dependent on centralized cloud-based services towards fog computing offers a good opportunity for the integration of Blockchain technology in IoT, as fog nodes should be powerful enough to run at least lightweight Blockchain client software, and therefore can act as proxy for Blockchain-based communication for IoT devices in the same subnet.

2.2.5. Transparency in IoT applications

It is important to note that in use cases, where trustworthiness and transparency are crucial, centralized communication protocols are only sufficient for communication in a trusted network. Hence, this introduces a problem when an IoT application spans across multiple non-trusted networks of different organizations such as in smart cities or supply chains. For these use cases, an additional party, trusted by all entities (e.g. a trusted CA) is required for bootstrapping the trusted communication.

Blockchain technology can facilitate the trustworthy information exchange beyond the trusted network without a trusted third party as it enables the communication of untrusted parties [CPV16]. An example for an IoT use case with high demand for transparency would be supply chain management, which is examined in detail by Abeyratne and Monfared [AM16]. Supply chains often involve multiple stakeholder and trust domains but still demand transparency at each processing step. Capturing information about supply chain assets is done using IoT devices (e.g. asset tracking via RFID, monitor proper conditions via temperature sensors) [AM16]. If those devices would communicate the information for example to a MQTT broker under the control of a single company, this company would have the possibility to alter the information to its advantage, or in

[1] `https://iot-analytics.com/state-of-the-iot-2020-12-billion-iot-connections-surpassing-non-iot-for-the-first-time/`, accessed: 2021-09-07

case cryptographic signatures are used, the party owning the MQTT broker could at least censor the network by discarding certain messages.

With a Blockchain employed for communication instead, the supply chain information of an asset can be stored decentralized as Blockchain transactions and no single actor has the possibility to manipulate, censor or delete the stored transactions.

2.3. Proof of Work algorithms

PoW algorithms play an important role for the solution of this thesis and shall therefore be examined in this section. Note that PoW algorithms are clearly distinguished from the PoW consensus mechanism described in Subsection 2.1.4, as they only describe ways to prove and verify that computational expensive work was spent, which is only a small part of the PoW consensus mechanism. PoW algorithms were researched many years before Blockchain technology was first introduced.

The first PoW algorithm found application as a mechanism to discourage the generation of high quantities of spam mails as proposed by Dwork and Naor [DN92] in 1992. The essential idea was to employ a mandatory computational expensive but easily verifiable task with the creation of every individual email, that poses a negligible additional effort for regular users that send emails to a small number of recipients, but becomes expensive with high quantities of mails, as it is practice with spam mails. The problem that has to be solved in order to accomplish the computational expensive task is to extract a square root of the hash of some data x modulo a prime p, i.e. $f(x) = \sqrt{x} \mod p$. The difficulty can be adjusted by choosing a larger p. The verification can be done with one calculation step by reversing the operation [DN92].

Later, this family of algorithms was generalized and the term *Proof of Work* was defined by Jakobsson and Juels [JJ99] as

> *"a protocol in which a prover demonstrates to a verifier that she has expended a certain level of computational effort in a specified interval of time".*

With *Hashcash* [Bac02], a new sub-type of PoW algorithms was introduced, that relies on finding partial collisions of hashes instead of the prior square root operation. The problem that must be solved in order to complete this PoW is to find a nonce that, hashed together with a piece of data, results in a bitstring that starts with a certain number of zeros. The fastest way to solve this problem is brute force, and its difficulty is adjustable through the required number of leading zeros [Bac02]. These aspects qualify the partial hash collision problem for a PoW. This type of PoW algorithm gained popularity as essential component of the consensus mechanism of the Bitcoin Blockchain [Nak08] followed by many other crypto-currency Blockchains.

The weakness of the Hashcash PoW is that the PoW problem can solved significantly faster through massive parallelization. Therefore, parallel hardware such as Graphics Processing Units GPU and Application Specific Integrated Circuits acrshortasic can outperform conventional hardware by far [Lar13], which results in a requirement of special hardware for competitive mining in Blockchain networks that utilize PoW consensus.

Momentum PoW [Lar13] picks up the approach of using partial hash collisions as a PoW problem and additionally introduces the requirement of memory to remedy the advantage of massive parallelization. Instead of finding a hash with leading zeros, Momentum suggests to use a birthday-collision problem instead, where it is required to store the calculated hashes in memory and check if a current partial hash collides with a previous one. Due to memory locking, parallelization does not bring an advantage in solving this problem. These types of PoW algorithms are called memory-hard [Per09]

Further PoW algorithms worth mentioning are Scrypt [Per09] and Equihash [BK17], that also elaborate the idea of of requiring memory to make parallelization of the PoW task impractical.

2.4. Chapter summary

To summarize the insights from related work, existing IoT communication protocols, such as MQTT and CoAP, are lightweight and fast, but the dependence on central entities limits the trustworthiness of these communication methods. Blockchain-based communication, on the other hand, can be trusted even in a network of non-trusted entities, since the users only need to trust that the majority of Blockchain nodes is correct. Blockchain-based communication, however, comes at the cost of significant additional communication overhead, which hinders the integration in IoT networks. In the next chapter, the integration challenges, that come with Blockchain technology, are approached to make Blockchain-based communication a viable alternative to existing communication methods.

3. Solution design

The goal of this thesis is to find out if Blockchain-based communication can be facilitated in IoT networks as an alternative to centralized communication protocols to leverage trustworthiness and transparency. The main contribution thereby is to develop a Blockchain-based communication system, named *DIoT*, which stands for *Decentralized IoT*. This chapter explains the most important research and design steps that have been conducted to realize the DIoT framework.

Solution requirements The requirements for the Blockchain-based communication system are derived from the research questions in Section 1.2:

- **R1: Decentralization**: Decentralization is the key attribute of Blockchain technology to achieve trustworthy and transparent information exchange as explained in Subsection 2.2.2. It is crucial that this advantage over traditional, centralized IoT communication protocols is preserved in a Blockchain-based communication system, otherwise there would be no benefit in using a Blockchain over a centralized communication protocol.
- **R2: No additional infrastructure for Blockchain consensus**: Many Blockchain-based applications are built upon public Blockchains to utilize their robust mining infrastructure or deploy dedicated mining nodes to secure the Blockchain consensus from attackers. Both strategies incorporate additional operating costs, i.e., transaction fees or operating costs of mining nodes. An improvement to the existing solutions would therefore be to propose a Blockchain-based communication system that does not depend on additional infrastructure to reduce additional operating costs.
- **R3: Moderate transaction latency**: Blockchains are rather slow communication mechanisms. The time that passes from the creation of a transaction until it is part of the Blockchain, i.e., the transaction latency, varies dependent on the chosen Blockchain solution, from a few seconds to several minutes. To provide a relevant communication system for many IoT-related use cases, a transaction latency under 10 seconds is the chosen target for a Blockchain-based communication system as defined in research question Q3 (Section 1.2).

Solution design process The process that led to the proposed system of this thesis was twofold. In first instance, a research of available Blockchain solutions was conducted to find a suitable Blockchain technology for a Blockchain-based communication system for IoT

networks. The results of the research yielded a promising candidate for the implementation of the desired system; the Tendermint consensus engine. Tendermint allows to create private permissioned Blockchains that already fulfill the solution requirements R2 and R3. However, private permissioned Blockchains use only a static set of validators that are allowed to take part at the consensus process to create new blocks, which limits the achieved decentralization of such a Blockchain system, i.e., R1 is not satisfied.

Therefore, the second part of the solution design process and the main contribution of this thesis was to build a process that maintains a dynamic validator set on top of a private permissioned Blockchain to achieve a solution that does also satisfy the solution requirement R1.

The following sections of this chapter examine the steps of the solution design process in more detail. Section 3.1 focuses on the research of existing Blockchain solutions and Sections 3.2-3.5 elaborate the Validator Management process that maintains a dynamic validator set on top of a permissioned Tendermint Blockchain.

3.1. Blockchain selection for IoT communication systems

This section focuses on choosing a Blockchain among the vast amount of available options, that is suitable for establishing communication between devices of IoT networks. The primary goal is to find a Blockchain that satisfies the solution requirements for a Blockchain-based communication system defined at the beginning of the Solution design chapter.

The methodology used for finding a suitable Blockchain is a top-down approach. First, the fundamental decision between public or private Blockchain is made. This narrows down the available consensus mechanisms. After committing to a consensus mechanism, the actual Blockchain solutions that use this specific consensus mechanism are examined and eventually a decision for a Blockchain solution is made.

3.1.1. Private or public Blockchain

The first step of the selection process was to choose between private and public Blockchain types. The distinction between these types is defined in Subsection 2.1.3.

Public Blockchains allow every node to join the network and use it to exchange transactions with other nodes without a prior registration. This is essential when a Blockchain is used as transaction storage for a crypto currency as everyone should have the chance to use it [Vuk15].

In private Blockchains, on the other hand, only certain nodes are granted with permissions to send or read Blockchain transactions. It is required that the identity of all permitted nodes of the network is known. Nodes must register at a central identity management entity in order to use a private Blockchain [Vuk15].

This is not necessarily a disadvantage to public Blockchains. For the purpose of communication it is often sufficient to reach only a set of nodes that are interested in the

exchanged data. For instance, in a network of smart homes that exchange data about new occurring security vulnerabilities, it is sufficient to reach out to other registered smart homes and the smart home provider. This example is further elaborated in Section 5.1. A private Blockchain can be used to limit the connections only to the necessary nodes and therefore optimize the communication efficiency. Vukolic [Vuk15] points out further fields of application that are compatible with private Blockchains, such as land and real-estate ownership ledgers, as node identification is obligatory in these applications.

Private Blockchains do not require the consensus mechanism to concern for sybil attacks because they are implicitly prevented by the Blockchain's identity management. This clears the way for permissioned consensus mechanisms such as BFT and PoA, that cannot protect against sybil attacks, but provide several other advantages over permissionless consensus. Permissioned consensus mechanisms generally provide better throughput and lower transaction latency than permissionless consensus mechanisms and are not demanding to computing resources [Vuk15].

Choosing a private Blockchain with permissioned consensus mechanism is a promising approach to satisfy the solution requirements R2 and R3. With the low demand in computing resources of permissioned consensus, it seems feasible to operate a Blockchain network only with the available devices of IoT networks. The fast consensus time of permissioned consensus can lead to a decent transaction latency for a Blockchain-based solution (e.g. under 10 seconds).

At the same time it is clear that using a permissionless consensus mechanism in either a public or a private Blockchain would deliver a greater degree of decentralization and would therefore be a better fit to satisfy the solution requirement R1. But these options perform worse regarding the solution requirements R2 and R3.

The eventual decision was to use a private Blockchain with permissioned consensus and combine it with the idea of Validator Management to remedy the disadvantages of permissioned consensus regarding solution requirement R1. The Sections 3.2, 3.3, 3.4 and 3.5 address the idea and implementation of Validator Management in detail. The next subsection deals with the decision between available permissioned consensus mechanisms.

3.1.2. Selection of a consensus mechanism

Having the Blockchain type defined, there are two fundamentally different families of permissioned consensus mechanisms available, BFT and PoA (see Section 2.1.4). The work of De Angelis et al. [DAAB+18] compares these two types of consensus mechanisms, which provides a solid foundation to find the most suitable type of consensus mechanism for the proposed Blockchain-based communication system of this thesis. They use Brewer's CAP theorem [Bre00], which states that distributed systems can only satisfy two of the three properties *consistency*, *availability*, and *partition tolerance*, to categorize PoA and BFT consensus.

The CAP theorem applied to both consensus mechanisms reveals that PoA gives up consistency for availability, which manifests in possible forks of the Blockchain, similar to PoW and PoS [DAAB+18].

BFT, on the other hand, cannot guarantee availability at any time, as the Blockchain may stall if the Blockchain network cannot reach a synchronous phase [DAAB+18] but guarantees consistency of the blocks in the Blockchain at any time. The consistency guarantee is also referred to as block finality [Vuk15].

In practice, the lack of guaranteed availability in BFT consensus mechanisms is compensated by restarting the consensus round if nodes do not respond after a timeout and this timeout is increased after every unsuccessful consensus round until a sufficient number of nodes respond within the timeout and consensus is reached [Buc16]. This can lead to a marginally slower block creation time than PoA [DAAB+18].

However, with regards to the Validator Management process that has to be developed in order to satisfy the solution requirement R1, the consistency guarantee of BFT is much more important, as it allows the Validator Management process to rely on data from the previous blocks of the Blockchain and still preserve consistency over all Blockchain nodes of the system. As it turned out, the final version of the Validator Management process, which is elaborated in the following sections of this chapter, does in fact rely on the strong consistency provided by BFT consensus, which is why it was selected over PoA.

3.1.3. Selection of a Blockchain framework

Implementing a secure and optimized Blockchain from scratch requires a lot of know-how in terms of cryptography, encoding and P2P networks and would go beyond the scope of this thesis. Fortunately, there are several open source Blockchain development frameworks with BFT consensus available to implement the underlying Blockchain for the DIoT framework. These frameworks essentially facilitate the P2P transfer of Blockchain transactions in a Blockchain network and the inclusion of transactions in blocks of the Blockchain via a consensus mechanism.

The well-established Blockchain frameworks Fabric [ABB+18] and Burrow [Dav19] from the Hyperledger project, as well as R3 Corda [BCGH16] and Tendermint[Buc16] are considered. Since all of these frameworks use BFT consensus, they all basically satisfy the requirement of the DIoT framework. The decision for one of these frameworks was only a matter of details. The most important selection criteria was modifiability, since a new Validator Management component (see 3.1) has to be integrated as additional functionality into the source code of the used Blockchain framework.

Under these circumstances, Tendermint was chosen for its minimalistic approach of only providing functionality concerning the BFT consensus mechanism and a simple Application Programming Interface (API) that has to be implemented by the Blockchain application for the correct interaction with the Tendermint process. Moreover, the developers of the Tendermint project show lots of effort to design their BFT-based consensus mechanism with regards to transaction performance, i.e., transaction latency and throughput, and network scalability [BKT18]. The Tendermint project was also perceived by the author as a well-structured and developer-friendly code base with clear instructions on how to properly modify certain parts of the source code.

The other options, on the other hand, provide a more complete and sophisticated framework with additional capabilities such as smart-contract support, private channel

communication, interoperability of Blockchains and different consensus mechanisms to choose from. It was anticipated that this additional functionalities would make these frameworks more difficult to modify in the desired way to accommodate the Validator Management component. Therefore the Tendermint project seemed to be the best option.

3.2. Validator Management

BFT-based Blockchains are, in general, not designed to run with more than a few hundred validator nodes because the number of messages sent rises exponentially with the number of nodes participating in consensus [Vuk15]. If the number of validator nodes scales beyond that magnitude, negative impact on block creation time and transaction throughput has to be expected [KB16]. Tendermint consensus was tested thoroughly in terms of the number of validators it can handle by the Cosmos team in the *Game of Stakes*[1] test scenario before launching the public Cosmos Blockchain, which uses Tendermint consensus. They claim that a Tendermint network with 64 validators can still maintain a throughput of thousands of transactions per second and a latency in the order of one to two seconds [KB16]. The public Cosmos Blockchain has currently 125 validators at the time of writing and can maintain an average block creation time of around 7 seconds[2]. Cosmos is going to increase its validator set size to eventually reach 300 validators 10 years after Blockchain launch, in 2029 [KB16]. Given these references, it is apparent that it would not be feasible to run Tendermint networks with more than a few hundred validator nodes. However, given that IoT networks often span over thousands of nodes, the DIoT communication system must be able to scale beyond a few hundred nodes. In order to scale up such a network, only a subset of nodes can be validators who participate at the consensus. The other nodes cannot vote for the inclusion of new blocks and have to trust the decision of the validator nodes.

Most permissioned Blockchain solutions use a static validator set by default that contains only a subset of all Blockchain nodes (e.g. Tendermint, Ethereum PoA, MultiChain). There are several problems with a static validator set, especially in a network of IoT devices:

- A static validator set can lead to centralization when multiple validators form a union to control consensus. A Blockchain with centralized consensus cannot provide the benefits in terms of trustworthiness over traditional centralized communication systems such as MQTT, therefore it is important to preserve decentralization.
- Validators are endangered to be target of attackers who want to gain control over the Blockchain consensus (see Section 2.1.2).
- The dependency of a network with a few nodes that act as validators would make the Blockchain as communication system less general and limits the use cases. It

[1] `https://medium.com/coinmonks/breaking-down-the-cosmos-game-of-stakes-5cbc538bcedb`, accessed: 2021-10-06

[2] `https://atom.tokenview.com/en/blocklist`, accessed: 2021-10-06

can be difficult to decide which nodes are validators, as they carry the responsibility and effort to keep the Blockchain running. It would be easier if all nodes that use the Blockchain for communication would have equal responsibility.

A solution to solve these problems could be to use a dynamic validator set instead, where nodes from the network are randomly assigned to become validator for a fixed time period as shown in Figure 3.1. Let this idea be evaluated against the problems listed above:

- A dynamic validator set increases decentralization because all nodes are participating at the consensus from time to time. A higher degree of decentralization leverages the trustworthiness of the Blockchain, as described in Subsection 2.2.2.
- Likewise, coordination between nodes for malicious intentions is made more difficult in a dynamic validator set because byzantine nodes that build a union will very rarely be in the same validator set and also only for a limited amount of time.
- Finally, there is no inequality between the network nodes in this scenario, as every node will have to perform the tasks of a validator from time to time.

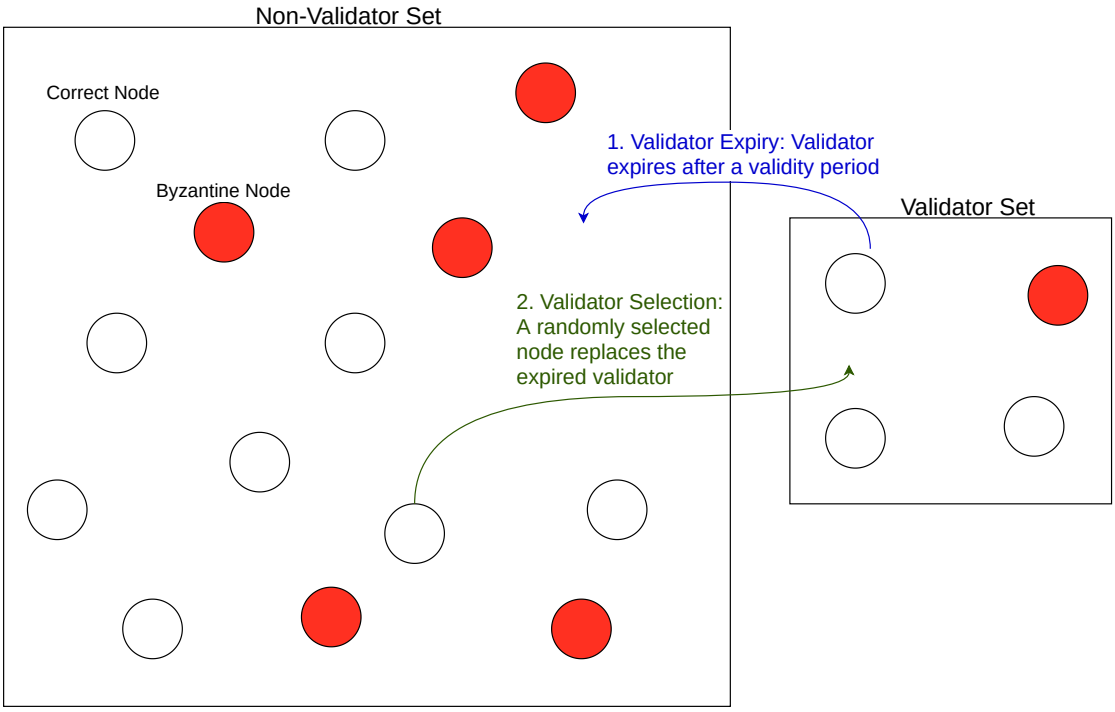


Figure 3.1.: Dynamic validator set maintained by Validator Management process.

This is a simple idea at first glance, but a system that implements a dynamic validator set on top of an existing permissioned Blockchain has to ensure that no vulnerability

is exposed through the dynamic validator set, that could be exploited by attackers to compromise the Blockchain security. Concrete requirements for the security considerations have been formulated in Subsection 3.4.1 that are subsequently used to assert proposed solutions to implement a dynamic validator set.

In the DIoT framework, the process of maintaining the changes in the validator set is called *Validator Management*. This process runs locally on every Blockchain node where it manages the local replication of the *validator set* and the *non-validator set*. The existing nodes that form the Blockchain network belong either to the former or the latter and can be transferred from one set to the other by Validator Management as visualized in Figure 3.1. Validator Management is divided into two sub-tasks, *Validator Expiry* and *Validator Selection*, which are executed sequentially every time a new block was added to the Blockchain. The former takes care of removing nodes from the validator set after some time, while the latter has the responsibility to transfer nodes from the non-validator set to the validator set to maintain a target number of validators. When designing a mechanism for Validator Management, it is crucial to consider that byzantine nodes can exist in the network, denoted as red spheres in Figure 3.1, as they are tolerated to a certain extent by the Tendermint consensus. Sections 3.3 - 3.5 explain the mentioned sub-tasks of Validator Management in greater detail.

3.3. Validator Expiry

```
1  func (app *Application) ValidatorExpiry() {
2    // Check if Validator Expiry is executed for this block
3    if app.blockHeight % app.config.ValidatorExpiryFreq != 0
4        || len(app.validators) < app.config.TargetNumValidators {
5      return
6    }
7    // Sort validators by age (oldest validators first)
8    sort.Slice(app.validators, func(i, j Node) bool {
9      return app.validators[i].validatorStartHeight <
10       app.validators[j].validatorStartHeight
11   })
12   // Transfer oldest validators to non-validator set
13   numExpired := app.config.numValidatorsExpire
14   app.nonValidators = append(
15     app.nonValidators,
16     app.validators[:numExpired-1]...
17   )
18   app.validators = app.validators[numExpired:]
19 }
```

Listing 3.1: Validator Expiry concept

The first task, which is visualized as a blue arrow in Figure 3.1, is called Validator Expiry. Listing 3.1 shows a simplified draft of Validator Expiry in Go syntax[3], similar to how it

[3] `https://golang.org/`, accessed: 2021-09-10

is implemented in the DIoT framework.

Validator Expiry is executed after every block or every few blocks, depending on the *ValidatorExpiryFreq* configuration parameter (see Listing 3.1, Line 3). Validator Expiry follows the simple decision rule of removing validators that resided in the validator set for the longest time, i.e., first in, first out, or in other words, based on the the validator age. The data structure, that represents a node in the DIoT framework, contains the field *validatorStartHeight*, that stores the block height at which the node was added to the validator set. Validator Expiry uses the *validatorStartHeight* to find and transfer the oldest validators to the non-validator set (Lines 8-18). How many validators are removed in each execution of Validator Expiry is determined by the *numValidatorsExpire* configuration parameter. When a node is selected for a further period as validator at a later point in time, *validatorStartHeight* is overwritten with the new block height so the calculated validator age is reset to zero. This is shown in the listing about Validator Selection below (Listing 3.2, Lines 14)

3.4. Validator Selection

Validator Selection is the second part of Validator Management, visualized as the green arrow in Figure 3.1. After Validator Expiry is executed, Validator Selection verifies if the target number of validators is still satisfied. If this is not the case, new validators are selected randomly from the non-validator set and transferred to the validator set to satisfy the target number of validators. Validator Selection is not as easy to design as Validator Expiry for its requirement of random selection. It is particularly difficult to design a decentralized mechanism, that combines determinism and randomness. Determinism is a mandatory property for the entire Validator Management process. If deterministic behavior is not guaranteed, it would lead to inconsistencies in the validator sets of the different nodes in the Blockchain network, i.e., Blockchain forks, and eventually to consensus failure.

3.4.1. Validator Selection requirements

The mechanism for Validator Selection, especially the solution to the sub-problem of *Seed Generation* (see Section 3.5), is the most significant contribution of this thesis. It needs to be designed very carefully and must fulfill several requirements in order to not jeopardize the security of the Blockchain. To start off the design process for Validator Selection, first the requirements are defined to be able to verify several approaches against these requirements in Section 3.5.

Determinism Validator Selection, just as Validator Expiry, must execute deterministically. At a certain block height, every correct node must select the same nodes from the non-validator set to transfer them into the validator set. If the selection of some correct nodes deviate from other correct nodes, the Tendermint consensus will fail eventually [Tenb].

Tamper-resistance Blockchains generally tolerate a certain percentage of byzantine nodes, i.e., fault tolerance. When designing a Validator Selection mechanism, byzantine nodes have to be considered and attempts of byzantine nodes manipulating the Validator Selection of correct nodes must be prevented. If byzantine nodes get the opportunity to manipulate the Validator Selection, they could specifically try to transfer other byzantine nodes into the validator set to increase the number of byzantine nodes in the validator set and eventually control the consensus or halt the Blockchain.

Unpredictability Besides tamper-resistance, unpredictability is another important requirement to increase the difficulty for attackers to exploit Validator Selection and undermine the Blockchain security. If the result of the Validator Selection mechanism of future blocks would be predictable, an attacker could try to position byzantine nodes in the non-validator set where they are selected by the Validator Selection mechanism.

3.4.2. Validator Selection process

The general process of *Validator Selection* is straightforward. It is roughly composed of the following steps:

- Generation of a seed value used to seed a Pseudo Random Number Generator (PRNG). Every correct validator node must use the same seed value at a particular block height.
- Generation of indexes using the PRNG to select nodes from the non-validator set.
- The user entries in the non-validator set are selected and removed by index and added to the validator set.

The remainder of this section explains the above steps in more detail. Listing 3.2 shows, a simplified version of Validator Selection, similar to as it is implemented in the DIoT framework. The most significant challenge for the distributed running Validator Selection process is to share a seed value between the nodes in a way that the requirements from Subsection 3.4.1 are satisfied. At this point it is assumed, that a Seed Generation function, called *ValidatorSelectionSeed*, exists, which generates an identical seed value on all correct Blockchain nodes at the same block height.

Therefore, Listing 3.2 only shows the call to *ValidatorSelectionSeed* in Line 8 without further defining it. It is further assumed that the *ValidatorSelectionSeed* function satisfies the above stated requirements. In Section 3.5, the *ValidatorSelectionSeed* function, that is used by DIoT for Validator Selection, is introduced. The seed value is an important root of determinism in the Validator Selection process. It is the only input to the Validator Selection that must be shared among all correct nodes. All subsequent steps of Validator Selection depend on the seed value.

```
1  func (app *Application) ValidatorSelection() {
2    // Check if new validators needed
3    numNewValidators = app.config.TargetNumValidators - len(app.Validators)
4    if numNewValidators <= 0 {
5      return
6    }
7    // Deterministic seed value generation
8    seedVal := app.validatorSelectionSeed()
9    // Validator Selection with PRNG
10   rand.Seed(seedVal)
11   app.nonValidators.SortByID()
12   for i := 0; i < numNewValidators; i++ {
13     r := rand.Intn(len(app.nonValidators)) // PRNG call
14     app.nonValidators[r].validatorStartHeight = app.blockHeight
15     // Add new validator to validator set
16     app.validators = append(
17       app.validators,
18       app.nonValidators[r],
19     )
20     // Remove validator from non-validator set
21     if r == len(app.nonValidators)-1 {
22       app.nonValidators = app.nonValidators[:r]
23     } else {
24       app.nonValidators = append(
25         app.nonValidators[:r],
26         app.nonValidators[r+1:]...,
27       )
28     }
29   }
30 }
```

Listing 3.2: Validator Selection concept

In the next step after the seed value generation, a PRNG is used as a tool for deterministic conversion of a single input value, which is the seed value, into an arbitrary long sequence of indexes in the range of indexes of the non-validator set (Lines 12-13). The seed value is used to initialize the PRNG and determines the outcome of all subsequent operations of the PRNG (Line 10). It is assumed that PRNGs of different correct processes A and B, initialized with the same seed value, will produce an identical sequence of output, i.e., being deterministic.

The last step of Validator Selection is to apply changes to the non-validator set and the validator set. In order to transfer the selected new validators to the validator set, they are simply accessed and removed from the non-validator set by the given indexes and added to the validator set (Lines 16-28).

With Listings 3.1 and 3.2 showing Validator Expiry and Validator Selection , Validator Management is almost entirely defined. The only exception is the *ValidatorSelectionSeed* function, which is elaborated in the following section.

3.5. Validator Selection Seed Generation

Validator Selection Seed Generation, for the rest of the section simply referred to as *Seed Generation*, is a mechanism that generates a seed value, which is required for Validator Selection to initialize the PRNG. The Seed Generation mechanism is implemented in the DIoT framework by the *ValidatorSelectionSeed* function (see Listing 3.2, Line 8). It must achieve the distribution or the decentralized generation of a seed value to allow all correct nodes at the same block height to access an identical seed value for deterministic Validator Selection. Furthermore, it must satisfy the requirements stated in Subsection 3.4.1 to not expose a weakness for the Blockchain security. Seed Generation is a key aspect of Validator Management which required the most time in the solution design process. As it was unclear at the beginning, how to achieve this particular aspect of the DIoT framework, first the requirements in Subsection 3.4.1 were defined, followed by an applied research phase, where four approaches were elaborated, prototyped, improved and evaluated against the requirements. The process of finding a suitable mechanism for Seed Generation is extensively documented in this section.

3.5.1. Seed Generation approaches overview

Approach	Centralized/Decentralized	Impl. complexity	Issues
Central SG with TPM	Centralized	Complex	Not tamper-resistant, unnecessary complex
SG with previous block hash	Decentralized	Simple	Not tamper-resistant
SG with master seed value	Decentralized	Simple	Predictable
SG with block hash + PoW	Decentralized	Moderate	None

Table 3.1.: Seed Generation (SG) mechanisms overview.

Before going into more detail about the four approaches on Seed Generation, Table 3.1 provides an overview of the results of each Seed Generation approach. The reason that so many different approaches were considered for the Seed Generation mechanism is that during the elaboration of every of the first three approaches, a weakness was discovered that violated one of the requirements stated in Subsection 3.4.1 (see Table 3.1, Column 4). Eventually, the fourth approach led to a Seed Generation mechanism that satisfies all requirements, at least in a probabilistic way. The unsuccessful approaches are still covered in this chapter, as they contributed valuable insights for the final Seed Generation mechanism and may be useful for future research on this topic for the awareness that these approaches have already been attempted.

3.5.2. First approach: Central seed value generation and distribution over the Blockchain

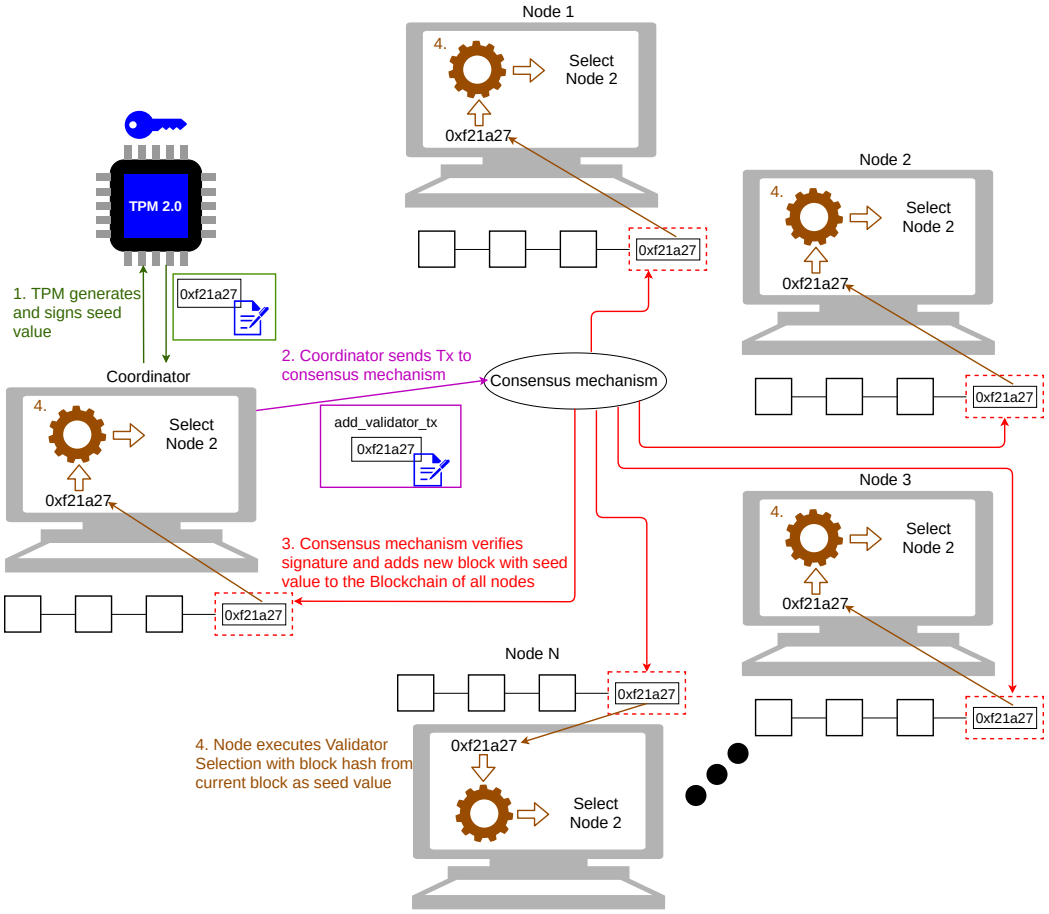


Figure 3.2.: Validator Selection with centralized Seed Generation.

The first intuitive approach for a Seed Generation mechanism pursued in the solution design process was to use a single node to generate a random byte sequence as seed value and broadcast it to the other nodes as a Blockchain transaction as shown in Figure 3.2. There is already one node with a special role available in a DIoT Blockchain, which is the *coordinator* node, that can be utilized for the purpose of generating and broadcasting the seed value. The coordinator node exists in a DIoT Blockchain to add new and remove existing Blockchain nodes from the Blockchain network (see Section 4.4).

In the first step, denoted as green arrows in Figure 3.2, the coordinator node internally generates a random byte sequence as the seed value for the next Validator Selection. The blue symbols in Figure 3.2 are ignored for now and explained later in this section. In step two (purple arrows) the coordinator sends an *addValidator* transaction, which carries the seed value as payload, to the Blockchain. In step three (red arrows), the decentralized

consensus mechanism, which is executed collectively by the Blockchain nodes, adds the *addValidator* transaction to a newly created block. The consensus mechanism further takes care of adding the new block to the local Blockchain replications of all Blockchain nodes. At this point, all nodes are able to perform deterministic Validator Selection, as shown in Listing 3.2 using the seed value that is contained in the most recent block (step 4; brown-colored arrows and symbols). In the example from Figure 3.2, all nodes conclude that Node 2 is going to be selected to be added to the validator set.

Design issues This simple idea turned out to be a rather complex solution. The network nodes cannot use the seed value generated by the coordinator node immediately at the same block height, but have to wait until an *addValidator* transaction from the coordinator is added to the Blockchain by the decentralized consensus mechanism in a future block (step 2 and 3 in Figure 3.2). This increases the implementation complexity, as Validator Management rounds must be introduced to preserve Validator Management state over multiple blocks.

Despite the complex solution, the mechanism does generally fulfill the functional requirements from Subsection 3.4.1 when byzantine nodes are not considered, as in every successful Validator Management round it distributes a seed value to all nodes, to allow a deterministic Validator Selection. Furthermore, when byzantine nodes are considered in this approach, but under the assumption that the coordinator is a correct node, it can be expected to produce a genuine random number that leads to a Validator Selection that fulfills all requirements from Subsection 3.4.1. If, on the other hand, the coordinator is a byzantine node (e.g. compromised by an attacker), it can easily decide not to send a random byte sequence, but instead a specific byte sequence to add other byzantine nodes to the validator set. This essentially gives the coordinator full control over the Validator Management process and, in further consequence, control over the Blockchain consensus mechanism. Also note, that Blockchain nodes are only motivated to remain correct by the intrinsic value the DIoT Blockchain adds as communication system to an application and not by monetary incentives, as the system is designed without transaction fees. Therefore, the assumption that one specific node, namely the coordinator, remains correct could not be made without degrading the trustworthiness of the DIoT framework.

Another weakness of this approach is that the liveness of the Blockchain depends on the coordinator. If the coordinator stops sending *addValidator* transactions to add new validators (e.g. because of a coordinator process crash or malicious behavior), due to Validator Expiry, eventually there will be no validators left to execute the consensus mechanism and produce new blocks.

The latter was approached by introducing a timeout to exit a Validator Management round when an expected *addValidator* transaction does not arrive from the coordinator after some number of blocks to ensure that the Blockchain can bypass Validator Management completely and resort to a static validator set without Validator Expiry for the time period where the coordinator does not respond.

Trusted Execution Environment (TEE) as potential solution The former problem, the coordinator being able to manipulate Validator Selection with arbitrary seed values, was approached by only allowing the generation of random seed values inside of a TEE. TEEs consist of software and hardware systems, separated from a host system, that can trustworthy provide cryptographic functionality to applications running on the host system, even if the host system is compromised. A TEE is able to remain trustworthy, as it holds a key pair, that is created during manufacturing, whose private key is never revealed to the host system. TEEs are essential in alternative Blockchain consensus mechanisms such as *Proof of Luck* [MHWK16] or *Proof of Elapsed Time* [CXS+17], which is why it is also considered as a possible solution for Seed Generation in the DIoT framework.

For the Seed Generation mechanism, Trusted Platform Module (TPM) 2.0 [CL13] was intended to come to use as TEE, as it is already widely deployed in existing notebooks, servers and available as extension for IoT hardware such as the Raspberry Pi 3 (RPi3). In the intended solution, the coordinator node must be equipped with a TPM 2.0, as shown in Figure 3.2. In step 1 of Figure 3.2 the coordinator node requests a random byte sequence from the TPM module, signed with the TPM private key, as seed value. The associated public key is known by all Blockchain nodes. With the signature sent along with the *addValidator* transaction in step 2, validator nodes can verify while executing the consensus mechanism in step 3 (red arrows in Figure 3.2), that the seed value was not created by the coordinator host system, that possibly could have manipulated the seed value, but by a trustworthy TPM 2.0, that is expected to send a genuine random seed value.

With the effort of using a TPM in the Seed Generation process, the coordinator node does not have any longer the possibility to send an arbitrary seed value and therefore Centralized Seed Generation seems to satisfy all requirements defined in Subsection 3.4.1. However, after closer examination it is apparent, that there is still a weakness in the approach that allows the coordinator node to manipulate the seed value in order to select a byzantine node for the validator set which violates the tamper-resistance requirement.

Weakness of the TEE-based solution Although, a signed random byte sequence from the TPM can be verified by other nodes which makes it impossible for a byzantine coordinator to forge a seed value (under the assumption that the TPM is tamper-proof), there is no way other nodes can tell how often the coordinator host system invoked the TPM to produce the seed value. The remaining process of Validator Selection depends entirely on the seed value, therefore the host process could determine the outcome of Validator Selection with a seed value received from its TPM before even publishing the seed value to the other nodes via an *addValidator* transaction, i.e., between steps 1 and 2 of Figure 3.2. This allows a byzantine coordinator host system to invoke the TPM to generate a seed value and then pre-calculate the Validator Selection, and repeat those steps until a seed value leads to another byzantine node being selected.

Experiment: Feasibility of a coordinator brute-force attack The extra time, that this brute-force approach takes, is not significant enough to detect malicious behavior of the coordinator node, as a delay can also be caused by network latency or occupied computing resources. Let there be the following experiment to visualize the feasibility of performing such an attack:

For the experiment, a network with a a fraction of 5% byzantine nodes is assumed. The goal is to find out how many times a byzantine coordinator has to invoke a TPM in order to find one or more seed values that leads to a byzantine node being selected as validator with a high probability of 95% and the delay, that this brute-force attack requires. A detection threshold of 1 second is assumed, thus, if the coordinator delays its response longer than 1 second, other nodes can detect malicious behavior. The assumed detection threshold is justified below in the discussion of this experiment. One communication round trip time between the coordinator host system and the TPM 2.0 is defined with $rtt = 0.04$ seconds based on observing a test communication between a notebook (Intel Core i7-8565U CPU, 16 GB RAM) and its integrated TPM 2.0. The additional steps of Validator Selection that are performed on the host system after an invocation of the TPM to calculate the outcome of Validator Selection based on the seed value are negligible as they have minimal impact on execution time.

As TPM generates a random byte sequence for the seed value, every node has equal chance to be selected by the coordinator, thus, a binomial distribution can be applied to solve this problem.

$$P(X = k) = \binom{n}{k} p^k (1-p)^{n-k}$$

Let $p = 0.05$, corresponding to the fraction of byzantine nodes in the network (5%).
Let $n \in \mathbb{N}$ be the number of times the coordinator has to invoke the TPM.
Let $P(X > 0) = 0.95$, the probability that a byzantine node is selected at least once.

Complementary probability is applied to get $P(X > 0)$ into the form of $P(X = k)$ to insert it into the binomial distribution equation:

$$P(X = 0) = 1 - P(X > 0)$$
$$P(X = 0) = 1 - 0.95$$
$$P(X = 0) = 0.05$$

The binomial distribution for the case $P(X = 0)$ can be simplified:

$$P(X = 0) = \binom{n}{0} p^0 (1-p)^{n-0}$$
$$P(X = 0) = (1-p)^n$$

Insert all parameters and solve for n:

$$0.05 = (1 - 0.05)^n$$
$$n = \frac{\ln(0.95)}{\ln(0.05)}$$
$$n \approx 58.4$$

The delay of the attack is calculated as follows.

$$d = n \times rtt$$
$$d \approx 58 \times 0.04$$
$$d \approx 2.32$$

As the detection threshold is assumed with 1 second, it can be argued that in this case the delay of 2.32 seconds is significant enough to be detected as malicious behavior of the coordinator node, however, this represents a very unfortunate case for the byzantine coordinator, as it could also have received a desired seed value in an earlier attempt (e.g. $P(X > 0) = 0.5$).

	Many byz. nodes p = 0.1	Moderate num. of byz. nodes p = 0.05	Few byz. nodes p = 0.01
Good case P(X>0) = 0.1	TPM calls: 1 Delay: 0.04 s.	TPM calls: 2 Delay: 0.08 s.	TPM calls: 10 Delay: 0.42 s.
Average case P(X>0) = 0.5	TPM calls: 7 Delay: 0.26 s.	TPM calls: 13 Delay: 0.54 s.	TPM calls: 69 Delay: 2.76 s.
Bad case P(X>0) = 0.9	TPM calls: 22 Delay: 0.87 s.	TPM calls: 45 Delay: 1.80 s.	TPM calls: 229 Delay: 9.16 s.

Table 3.2.: Chances of successfully circumventing central Seed Generation from the view of a byzantine coordinator with detection threshold of 1 second.

Table 3.2 lists results of this experiment using different variations of the input parameters p and $P(X > 0)$, which shows that this attack can be successful under different circumstances. Variations in $P(X > 0)$ (vertical axis) describe the luck of a byzantine validator receiving a desired seed value after few tries (good case) or many tries (bad case). Variations in p (horizontal axis) consider different scenarios with few byzantine nodes in the network (1%) or many byzantine nodes (10%). Yellow colored cells indicate a delay below the detection threshold, which means that under these circumstances, the byzantine coordinator could successfully perform the attack without being detected by other nodes. Note that the assumed threshold of 1 second is only a rough estimate and

could be lower in reality, but lowering the threshold would also increase the chance of false positives. For this problem, delay-based detection does not seem to be a suitable measure to detect malicious behavior anyway, as it would most likely cause the Seed Generation mechanism to become non-deterministic. There are several more factors besides malicious behavior that influence the response time of the coordinator node, such as network load and distance between network nodes, that could lead to some correct nodes detecting malicious behavior while other correct nodes do not.

The conclusion of this experiment is that a byzantine coordinator can very likely receive a desired seed value from a TPM after a moderate amount of attempts (e.g. within 100 attempts). A delay-based detection threshold as countermeasure for this attack is difficult to implement due to the requirement of deterministic execution of the Seed Generation mechanism. Even if there is a solution to design and implement a delay-based detection mechanism that does not violate the determinism requirement, a byzantine coordinator could still succeed in a large amount of attempts performing this attack, as shown in Table 3.2. Due to the discovered weakness, the idea of centralized Seed Generation was not longer pursued in the solution design process.

3.5.3. Decentralized Seed Generation approaches

After the first approach of centralized generation and distribution of a seed value did not turn out to be a sufficient solution for Seed Generation, a fundamentally different direction was considered, that is, using a completely decentralized Seed Generation function that every node executes independently to receive the same seed value, using only information as input that already exists from previous blocks of the Blockchain.

Figure 3.3 shows a visual representation of the decentralized Seed Generation and how it integrates into the Validator Selection process. For comparison, Centralized Seed Generation, shown in steps 1-3 of Figure 3.2, involves the coordinator to send a transaction to the Blockchain, which is then used by other nodes when the transaction is added to a block. As opposed to that, in decentralized Seed Generation (Figure 3.3), every node executes a local Seed Generation mechanism by itself by deriving the seed value from the block hash of an already existing block. A major conceptual difference is that decentralized Seed Generation does not need to differentiate between normal nodes and a coordinator. It is further apparent that a decentralized Seed Generation approach is less complex to implement as the entire Validator Management process does not require waiting for an *addValidator* transaction to be added in a future block.

However, a decentralized approach was also perceived as less intuitive than a centralized one, because it seemed difficult to design such a mechanism that satisfies the requirement from Subsection 3.4.1 without an input from a central entity that all nodes can rely on. This is why it was not considered in the first place. In line with the difficulty of the task of designing a decentralized Seed Generation mechanism, the first two attempts did not succeed, eventually followed by an approach that fulfills all requirements from Subsection 3.4.1. The first two unsuccessful attempts and their problems are covered briefly, before the final Seed Generation mechanism is explained in detail.

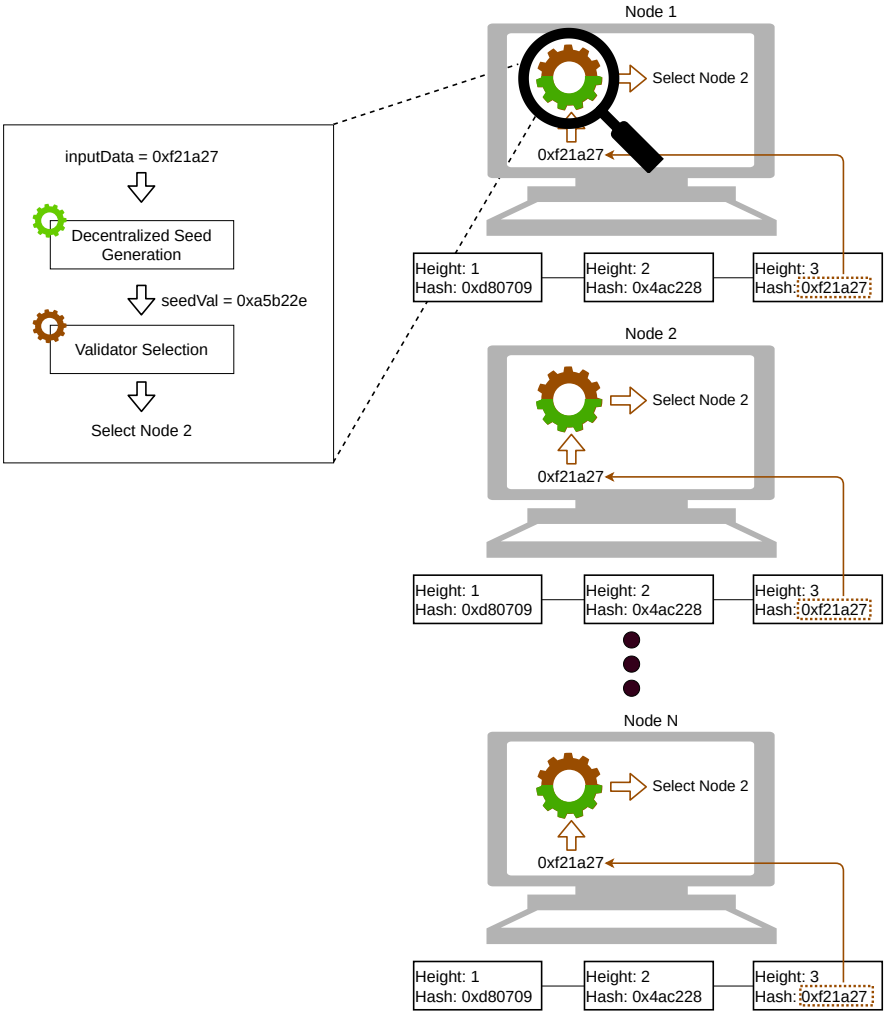


Figure 3.3.: Validator Selection with decentralized Seed Generation.

3.5.4. Second approach: Seed value from previous block hash

The first idea to achieve Seed Generation without dedicated input from a central entity was to use already existing information as input, that all nodes are able to access at the time Seed Generation is executed. To be more specific, the block hash of the previous block should serve as seed value. BFT-based consensus, such as Tendermint, allows to use information from the previous block for this purpose as it guarantees that no forks occur as long as the fault tolerance property is not violated, i.e., $n \geq 3f + 1$ (see Subsection 2.1.4). Note that this approach would not work with other consensus mechanisms such as PoA or PoS, as these mechanisms allow occasional Blockchain forks, i.e., not providing block finality. Listing 3.3 gives a simplified example in Go syntax, how Seed Generation would be implemented in this case.

```
1 // Method of diot.Application
2 func (app *Application) validatorSelectionSeed() int64 {
3     var seedBytes []byte
4     seedBytes = app.blockchainState.LastBlockHash
5
6     // PRNG requires int64 seed
7     // therefore conversion from []byte to int64
8     return int64(binary.BigEndian.Uint64(seedBytes))
9 }
```

Listing 3.3: ValidatorSelectionSeed method with block hash from previous block

Let this approach be evaluated against the Validator Selection requirements from Subsection 3.4.1:

- Determinism is fulfilled, as every node has access to the previous block hash. The block hash that is obtained by all correct nodes at the same block height is identical because BFT consensus does not produce forks.
- Unpredictability is also fulfilled, as nodes cannot know the block hash of a block that has not been mined yet. The seed value, that is valid for Validator Management at a certain block height is only available once the block of the previous block height has been mined successfully.
- Tamper-resistance seems to be fulfilled at first sight, as the last block hash appears to be a random hash that is not manipulable. But actually, a byzantine block proposer of the previous block has the opportunity to alter the block hash in a similar way as the byzantine coordinator in the centralized Seed Generation mechanism of Subsection 3.5.2, which is explained in the following paragraph.

Brute-force attack to manipulate Validator Selection A block proposer can decide, which valid transactions from the pool of pending transactions to add to a proposed block, and also the transaction order. By adding more or less transactions to a block or altering the transaction order, a byzantine block proposer can obtain a large number of different block hashes that are all considered valid by other nodes. He can try out different combinations of transactions and pre-calculate the result of the Validator Selection with a given block hash to see if another byzantine node is selected for the validator set. If necessary, the process can be repeated until the byzantine block proposer achieves its goal.

Basically, this approach on Seed Generation achieves a very similar result as the centralized approach before, but with less implementation complexity. The big difference to the centralized approach is that in this scenario, not only the coordinator, but every validator has the chance to manipulate the result of Validator Selection when he becomes the block proposer. The experiment, that was conducted in Section 3.5.2 revealed that

approximately 100 iterations of variating the input arguments and calculating Seed Generation most likely yields a desired seed value for the attacker. This does also apply for this approach. In this approach, 100 variations are even faster to achieve for a byzantine block proposer, as there is no time-consuming interaction with a TPM involved.

3.5.5. Third approach: Master seed value

The second approach for decentralized Seed Generation follows the idea of a *master seed value*, which is an arbitrary byte sequence that is added to the Blockchain in the genesis block. A seed value can then be obtained by generating a hash value (e.g. sha256) from of the master seed value combined with the current block height. Listing 3.4 shows the simplified implementation of Seed Generation with a master seed value in Go syntax.

```
1  // Method of diot.Application
2  func (app *Application) validatorSelectionSeed() int64 {
3
4      // create hash of master seed value and block height
5      h := sha256.New()
6      h.Write(app.blockchainState.GenesisMasterSeedVal)
7      h.Write(app.blockchainState.LastBlockHeight)
8      seedBytes := h.Sum(nil)
9
10     // PRNG requires int64 seed
11     // therefore conversion from []byte to int64
12     return int64(binary.BigEndian.Uint64(seedBytes))
13 }
```

Listing 3.4: ValidatorSelectionSeed method with master seed value

Let this approach be evaluated against the Validator Selection requirements from Subsection 3.4.1:

- Every correct node can execute this function at every block height and receive a seed value, that is equal to other correct node's seed value at the same block height, as all nodes have access to the same master seed value in the genesis block and are aware of the current block height. Therefore, the determinism requirement is fulfilled.

- This approach is also tamper-resistant because, other than in the previous approach, byzantine nodes cannot manipulate the input parameters for Seed Generation that are used by correct nodes as all input parameters are predetermined from the start of the Blockchain.

- However, this approach does not satisfy the unpredictability requirement, as the result of Seed Generation can be calculated at any time for every block height, as the input parameters for Seed Generation are defined in the genesis block.

Since this approach violates the unpredictability requirement of Validator Selection stated in Subsection 3.4.1, it is not an option for the Seed Generation mechanism.

3.5.6. Fourth approach: Seed Generation with block hash and PoW

After the recent failures in designing a Seed Generation mechanism, it was observed that the first and second approach are vulnerable to a similar type of brute-force attack, where the attacker pre-calculates Seed Generation results with different valid variations of the input arguments, before eventually publishing input arguments to other nodes, that result in an advantageous Seed Generation result for the attacker. The circumstances that allow this kind of brute-force attack in these approaches are the attacker's ability to sufficiently variate the input arguments for Seed Generation and the fact that there is enough time for the attacker to try out different variations before publishing the input arguments for Seed Generation to the other nodes. If a way is found to limit either the possible valid variations of input arguments to Seed Generation or the numbers of pre-calculations a byzantine node can perform, the first and second approach would be acceptable for Seed Generation. By re-phrasing the remaining problems of Seed Generation approaches in this way, PoW appears to be a suitable cryptographic tool to prevent this type of brute-force attack by constraining the ability of an attacker to variate the input multiple times before publishing it to other nodes.

PoW is in this context not used as challenge between nodes for the privilege to mine a block, as it is often used in Blockchains such as Bitcoin, but rather as a cryptographic verifiable proof that a node has spent a significant amount of time to perform a computational expensive task. The PoW is employed in the Seed Generation mechanism in a way, that every iteration of a brute-force attack also involves the PoW. Therefore, the time consumption of the PoW becomes more significant with every additional iteration of the brute-force attack. The aim is to configure a PoW to require just enough computing time to be able to detect if the Seed Generation was performed a single time as intended, or multiple times which indicates a brute-force attack. It is expected that a PoW task, that causes additional execution time of up to a few seconds, should be sufficient for this purpose.

Reuse the second Seed Generation approach In order to show how PoW is employed in one of the above approaches to solve the brute-force vulnerability, the second approach is reused, as it is the simpler out of the two approaches that share the brute-force vulnerability, i.e., first approach and second approach (Subsections 3.5.2, 3.5.4). To recapitulate, the second approach uses the block hash of the previously added block as input for Seed Generation. A byzantine block proposer can perform a brute-force attack to manipulate the output of the next execution of Seed Generation by assembling blocks of different sets of valid transactions and verifying if the resulting block hash as input to Seed Generation would lead to another byzantine node being selected as validator. If the result is not satisfactory, the process is repeated with a different set of transactions to get a different block hash, until the result of Seed Generation and in further consequence the result of Validator Selection is as needed. One of the insights from the experiment in

Subsection 3.5.2 was that if a byzantine node can generate approximately one hundred variations of the input arguments for Seed Generation, i.e., in this case the block hash from a proposal, before publishing the input arguments to the other validators, he has already a good chance of successfully manipulating the outcome of Seed Generation. Although Tendermint uses a timeout, before which a block proposal has to arrive at other validator nodes (by default three seconds), it is feasible for a byzantine block proposer to perform one hundred or more block proposals with different block hashes and still send a proposal timely, as the generation of a block out of available transactions poses only a negligible performance impact.

In order to increase the difficulty for attackers to create multiple block proposals, a PoW task was considered to be embedded into the block proposal process. If the creation of a block proposal involves solving a PoW, one iteration of the brute-force attack consumes significantly more time, which restricts the amount of variations a byzantine block proposer can perform before the proposal timeout.

PoW algorithms have the feature of adjustable difficulty to steer the expected time, that a node requires to accomplish the PoW task, towards a certain average value (see Section 2.3). A PoW difficulty must be chosen, that allows correct nodes to calculate the PoW once and publish the block proposal timely, while at the same time make it unlikely, that a byzantine node manages to calculate the PoW twice or more times before the timeout. As the time to solve a PoW has a certain variance to it, it is both possible, that a correct node cannot finish a single PoW task and propose a block before the proposal timeout, as well as a successful execution of the brute-force attack by a byzantine node. Both scenarios are manageable to a certain extent. If a brute-force attack is executed successful in rare cases and a byzantine node is added to the validator set, it does not have a strong impact on Validator Management, because the same could happen through the random selection of new validator nodes by the Validator Management process. The Blockchain can compensate such cases through its byzantine fault tolerance property and the Validator Expiry, which ensures, that byzantine nodes do not remain in the validator set for longer than a few blocks. On the other hand, if a correct block proposer cannot solve a single PoW task in time, the consequences would be that Tendermint chooses another validator to propose the block instead, which results in a slightly increased block creation time of the current block. How often one of the scenarios occur depends on the chosen difficulty.

Integration of PoW into the existing system As justified above, a PoW as addition to the second approach that was proposed in Subsection 3.5.4 fulfills the tamper-resistance requirement from Subsection 3.4.1 in a probabilistic way. The other two requirements have already been fulfilled in the original approach.

The characteristics of a PoW function are defined by Dwork and Naor [DN92], who refer to it as a *pricing function*. They define $f(x)$, as the pricing function, which must be of moderate difficulty to compute and yield a result y. Given the input argument x and the result y, it must be easy to determine the correctness of $y = f(x)$.
In the context of this approach, the two functions *ProofOfWork* and *ValidatePoW* are used, which represent the pricing function as follows:

$ProofOfWork(inputData) \longrightarrow nonce$
$ValidatePoW(inputData, nonce) \longrightarrow boolean$

The variables *inputData* and *nonce* are equivalent to x and y in the pricing function. Note that the term *nonce* complies to widely accepted terminology of recent PoW algorithms such as [Lar13], [BK17]. *ProofOfWork* is moderate to compute, i.e., few seconds, and yields *nonce* as evidence for the successful executed PoW. *ValidatePoW* is easy to compute, i.e., few milliseconds, and returns a truth value about correctness of the PoW execution with the given *nonce*. Further information about the implementation of *ProofOfWork* and *ValidatePoW* will be provided in Section 4.7 of the Implementation chapter.

The realization of this approach is slightly more complicated than the approach of Section 3.5.4. It is necessary to embed the *ProofOfWork* function into the block proposal step of Tendermint consensus and the *ValidatePoW* function into the block validation step. Block proposal and block validation are placed at the beginning of a Tendermint consensus round (see Paragraph *"BFT consensus"* in Section 2.1.4 and Tendermint documentation [Tenc]). Listings 3.5 and 3.6 contain the changes made to the Tendermint consensus process to realize this approach.

Listing 3.5 shows very simplified in Go syntax, how the execution of a PoW is embedded into the Tendermint source code. Note that certain parts from the original source code, that are not important in this context, are not displayed in the listing for brevity. The *MakeBlock* method is executed when a node proposes a new block to the network. The PoW difficulty is defined in a configuration file and passed to the function as argument (Line 8). Lines 15-17 show that the input for PoW consists of random bytes and the block hash, which is called $state.AppHash$ in Listing 3.5. The nonce value, that was used during the PoW to solve the PoW problem, is returned by the $ProofOfWork$ function. All important information to verify that the PoW was solved correctly, i.e., random bytes, block hash, and nonce, must be sent as part of the block proposal to other nodes (see Lines 23-30), to allow them to perform the PoW validation.

```
1  // Method of tendermint.state,
2  // the module that holds the state of the current block.
3  func (state State) MakeBlock(
4    height int64,
5    txs []types.Tx,
6    commit *types.Commit,
7    proposerAddress []byte,
8    powDifficulty *diotpow.PoWDifficulty,
9  ) *types.Block {
10   // Build base block with block data.
11   block := types.MakeBlock(height, txs, commit)
12   // Random bytes are mandatory so the next proposer
13   // will try to solve the PoW with a different value
14   // in case the current proposer fails.
15   randomBytes := []byte(diotpow.RandomString(32))
16   inputData := append(state.AppHash, randomBytes...)
17   powNonce := diotpow.ProofOfWork(inputData,powDifficulty)
18   // If the PoW timeout is triggered, return nil to abort block proposal
19   if len(powNonce) == 0 {
20     return nil
21   }
22   // Fill the block header
23   block.Header.Populate(
24     state.AppHash,
25     state.LastResultsHash,
26     proposerAddress,
27     powNonce,
28     randomBytes,
29   )
30   return block
31 }
```

Listing 3.5: Simplified excerpt from modified Tendermint source code showing block proposal with PoW.

The source code in Listing 3.6 again has been shorted to the relevant changes that were made in the *validateBlock* function. The *validateBlock* function is executed, when a new block proposal arrives at a node to validate the correctness of the proposed block. At this point, among other validation steps, the PoW validation takes place. $ValidateBlock$ receives the same $PoWDifficulty$ parameter as the $ProofOfWork$ method from Listing 3.5, to verify if the resulting hash satisfies the required difficulty (Listing 3.6, Line 6). Line 10 shows, that the PoW input data is assembled identically as in $ProofOfWork$ in Listing 3.5. In contrast to the $ProofOfWork$ function, $ValidatePoW$ takes an additional *nonce* argument from the received block, which is the result of the previous $ProofOfWork$ execution of the block proposer. Internally, $ValidatePoW$ calculates the hash of inputData and nonce and checks if the required difficulty is achieved by the resulting hash. If this is not the case, a validation error is returned.

```go
// Function of of tendermint.validation module,
// which is executed once a block proposal arrives
func validateBlock(
  state State,
  block *types.Block,
  powDifficulty *diotpow.PoWDifficulty,
) error {
//\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/
  // Validate Proof of Work
  inputData := append(block.AppHash, block.PoWRand...)
  if !diotpow.ValidatePoW(
      block.PoWNonce,
      inputData,
      powDifficulty,
  ) {
    return powValidationError()
  }
//\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/
  return nil
}
```

Listing 3.6: Excerpt from modified Tendermint block validation function showing PoW validation.

Listing 3.7 shows how the DIoT framework uses the additional PoW fields from the block header in the Seed Generation process, that are available through the changes made to the Tendermint source code. The changes in the DIoT source code compared to the original approach from Listing 3.3 are minimal. The only important difference can be observed by comparing Line 4 of Listing 3.3 with Lines 7-9 of Listing 3.7. The new approach does not directly apply the block hash as the seed value, but takes a hash of the PoW nonce together with the random byte string that was also part of the PoW. Note that the input data for the PoW is essentially the previous block hash, which is called *AppHash* in Tendermint (see Listing 3.5, Line 16), therefore the nonce and subsequently the seed value indirectly depends on the block hash as well, similar to the second Seed Generation approach. The important conceptual difference between the second and the fourth approach is that through the PoW in the fourth approach, the chance of a byzantine block proposer to manipulate the Seed Generation outcome is reduced drastically to receive a probabilistic tamper-resistance.

```
1  // Method  of  diot.Application
2  func (app *Application) validatorSelectionSeed() int64 {
3    block := app.blockStore.LoadBlock(
4      app.blockchainState.LastBlockHeight
5    )
6    h := sha256.New()
7    h.Write(block.Header.PoWNonce)
8    h.Write(block.Header.PoWRand)
9    seedBytes := h.Sum(nil)
10
11   // convert hash to int64 as input for the PRNG seed
12   seedInt := binary.BigEndian.Uint64(seedBytes)
13   return int64(seedInt), nil
14 }
```

Listing 3.7: ValidatorSelectionSeed method using the PoW from previous block proposal

3.6. Chapter summary

This chapter collects all fundamental building blocks necessary for a Blockchain-based communication system for IoT networks. A Validator Management process was designed to run on top of Tendermint, a private permissioned Blockchain solution, to achieve enhanced decentralization by continuously swapping out validators. For a deterministic and fair selection of new validators, the Validator Management process uses a PRNG initialized with a special Seed Generation function, that creates a seed value based on a PoW, which has already been calculated during a previous Block proposal. The PoW is a crucial aspect of this design, as it prevents a brute-force attack that could otherwise lead to manipulation of the validator selection in the Validator Management process and in further consequence to a majority of byzantine nodes in the consensus voting. The next chapter documents how the elaborated building blocks are used to implement the DIoT framework.

4. Implementation

The *go-diot* project[1] emerged as the practical aspect out of this master thesis with the purpose to experiment and evaluate, whether the expectations, framed as the research questions, can be satisfied by the concept of a decentralized Validator Management process, that was elaborated in the Solution design chapter. The DIoT framework is implemented as a software library and additional tools, that can be used by application developers to realize P2P publish-subscribe communication in a network of IoT devices and other nodes such as servers, gateways and mobile devices. At the current stage of the project, it is intended that application developers learn the usage of the DIoT library by inspecting the example application provided within the go-diot project. This chapter addresses the system architecture and important design decisions regarding the implementation of the DIoT framework.

4.1. System requirements

As a communication solution developed for IoT networks, DIoT aims to have very few system requirements to allow the installation on several different devices. During the research of existing Blockchain solutions that can enable such a communication system, which resulted in the selection of Tendermint as underlying Blockchain solution (see Section 3.1), it became also apparent that there is no Blockchain client available that is compatible to micro controller platforms such as Arduino[2]. Probably, the existing micro controllers are also not yet powerful enough to run a Blockchain client software. The DIoT framework inherits this limitation, as it uses internally Tendermint Blockchain client software. The smallest type of hardware in a network of IoT devices, that is supported by the DIoT framework, are single-board computers like the RPi3[3] or similar devices that can run Linux operating system. The RPi3 is used to represent this class of hardware throughout this thesis, as it is a popular and highly available option among the existing devices of this class. The DIoT framework has been tested frequently during development on a RPi3 device with Raspberry Pi OS 64-bit[4] and a AMD64-PC with Linux Mint 19.2[5] to ensure that these target systems are supported.

[1] `https://bit.ly/3rHFxVh`, accessed: 2021-10-06
[2] `https://www.arduino.cc/`, accessed: 2021-09-11
[3] `https://datasheets.raspberrypi.com/rpi3/raspberry-pi-3-b-plus-product-brief.pdf`, accessed: 2021-10-06
[4] `https://downloads.raspberrypi.org/raspios_arm64/images/`, accessed: 2021-10-18
[5] `https://linuxmint.com/edition.php?id=267`, accessed: 2021-10-18

4.2. System architecture

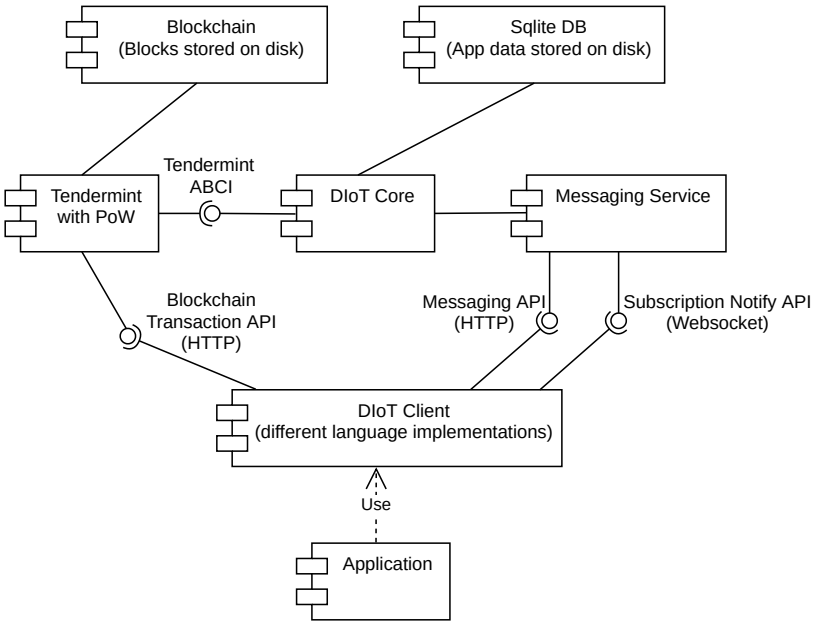


Figure 4.1.: DIoT UML component diagram

The architecture of the DIoT framework is presented as a Unified Modeling Language (UML) component diagram in Figure 4.1. The functionality of the most important components and interfaces of the go-diot project are addressed in this section.

Persistent storage components (Blockchain and Sqlite database) The DIoT framework uses two persistent storage components; a Blockchain, which sequentially stores the blocks of transactions that have passed the consensus process and a SQLite database, which stores relevant data from the Blockchain transactions in a relational data structure to allow the implementation of all messaging-related functionalities of the DIoT framework (e.g. topic subscription, message notification).

Tendermint component Tendermint is a central component of the system that facilitates the interaction of a DIoT node with the decentralized Blockchain. In particular, Tendermint allows sending new transaction to the Blockchain and receiving Blockchain transactions from other DIoT nodes as well as executing the Blockchain consensus process. Tendermint exposes a Blockchain API via HTTP, which clients can use to send transactions to the Blockchain that should be included in future blocks. The Tendermint component used in the DIoT framework has been extended by the author to solve a PoW during block proposal, which is used by the DIoT core component for Validator Management (see Section 4.6).

DIoT core component DIoT core implements the *Application Blockchain Interface* (ABCI)[6], an interface specified in the Tendermint system to allow a Blockchain application, in this case the DIoT core component, to process incoming Blockchain transactions before they are stored in the local Blockchain by Tendermint. Upon receiving new Blockchain transactions from Tendermint, DIoT core extracts messages relevant to the DIoT node from the transaction and stores them in the relational data scheme of the SQLite database. Additionally, the DIoT core component executes the Validator Management process, which was explained in Section 3.2, every time it receives a new block from Tendermint to maintain a dynamic validator set.

Messaging service component The messaging service runs as separate process with access to the DIoT core instance. It exposes two interfaces to client processes via HTTP, one Messaging API to provide clients with access to messages that are stored in the SQLite database, and a websocket API (Subscription Notify API) to allow the messaging service to push notifications about newly arrived messages on a topic, the client subscribed to.

DIoT client component The DIoT client component is very loosely coupled to the other components as it only interacts with the other components via HTTP-based interfaces. This design brings two important benefits. First, the DIoT client component can be implemented in different programming languages which allows applications of several programming languages to use the core components of the DIoT framework. Second, the loose coupling allows also multiple processes, even on other devices in a local network, to use a single DIoT node simultaneously. This can be useful for instance in a network of constrained IoT devices that may not be able to run a DIoT node (e.g. ESP32 NodeMCU[7]) to facilitate communication to devices of a different subnet via a proxy in the same local network that runs a DIoT node (e.g. RPi3). In the go-diot project, an implementation of the DIoT client component in Go can be found in the *client* directory. Implementations in other languages may follow in the future.

Application component The Application component is a placeholder for any application that uses a DIoT node for Blockchain-based communication with other nodes. An application simply has to import the DIoT client component as library to connect to a DIoT node. An example application in Go for reference can be found in the *example* directory of the go-diot project. The example application shows how to use the DIoT Client library and all currently available functionalities of the DIoT framework.

4.3. Message data structure

The DIoT framework uses a simple data structure to implement publish-subscribe messaging which is displayed as Entity-Relationship (ER) diagram in Figure 4.2. This data

[6] `https://docs.tendermint.com/master/spec/abci/`, accessed: 2021-05-14

[7] `https://docs.ai-thinker.com/_media/esp32/docs/nodemcu-32s_product_specification.pdf`, accessed: 2021-10-15

structure is used in the SQLite database to store the application data of the running DIoT node.

Topic is an entity, which can be seen as a channel, where messages of type *entry* can be added to. A topic has a title and is owned by a *user* entity, which represents a DIoT node in the network. Users can create topics and thereby become the topic owner. The DIoT framework enforces that only the owner of a topic can add entries to the topic which is essentially a write restriction. In order to restrict read access, entry content can be encrypted with the public key of the desired message recipient. The DIoT node offers an API to its clients to query for topics, entries and users stored in the database and also provides the option to subscribe to any topic to notify its connected clients when an entry is appended to one of the subscribed topics.

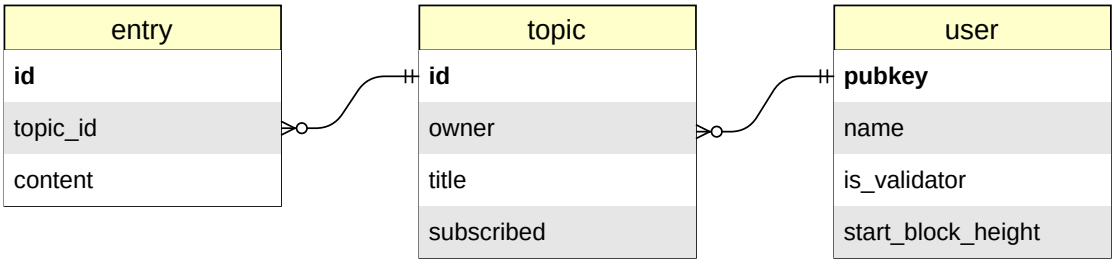


Figure 4.2.: ER-diagram of the DIoT message data structure

The publish-subscribe messaging pattern was chosen for its versatility as it allows both, sending messages to a single or multiple recipients. In the following, two examples for communication between nodes are listed to show how the publish-subscribe messaging works with the provided entities, i.e., entries, topics, and users, in the DIoT framework:

- **One-to-One communication:** To establish a directed communication channel between two specific nodes (e.g. an IoT control unit sends configuration data to a device), the sending node can create a dedicated topic and add entries to the topic that contain the messages to be sent to the recipient. The recipient can either subscribe to the sender's topic to receive a notification when a new entry arrives or query the database for entries with the specific topic and topic owner. Bidirectional communication can be established similarly with the simple addition that both nodes set up a dedicated topic for the communication, i.e., a second communication channel for the other direction.

- **Many-to-Many communication:** If multiple nodes should publish the same type of information (e.g. multiple IoT devices that produce the same type of sensor readings), each of the nodes can create a topic with the same name and add entries to the topic containing the information to publish. Similar to above, interested recipients can query the entries in the database with the specific topic name to receive all entries referring to the same topic or subscribe to the specific topic of every node they are interested in.

4.4. Identity management

Since DIoT Blockchains are private, only nodes that are known to the other Blockchain nodes may send messages, which makes identity management necessary in the DIoT framework. For this purpose, the *coordinator* node is introduced, which is usually the DIoT node that initializes the Blockchain by creating the genesis block. The public key of the coordinator node is specified at launch of a DIoT Blockchain in the genesis file. The coordinator has the additional responsibility to send proposal transactions to the Blockchain network to either add new or remove existing Blockchain nodes from the Blockchain, i.e., *addUser* and *removeUser* messages, containing the public key and potentially additional information to prove the legitimacy of the node. When one of the mentioned transactions are added to the Blockchain, every Blockchain node decides, whether to accept the proposal and modify its local set of Blockchain nodes accordingly, or to reject the proposal by ignoring the transaction.

Proof of node legitimacy to avoid sybil attacks It is important to protect the DIoT Blockchains from sybil attacks, which could lead to a majority of byzantine nodes in the validator set and the consequences mentioned in Subsection 2.1.2. The validation of a proof of legitimacy of new Blockchain nodes can be embedded into the process of adding a node to the Blockchain to hinder attackers from creating large numbers of fake nodes. It is important that the additional information for a proof of legitimacy can be validated within a deterministic function, which is called *ValidateIDProof* in the DIoT framework, to allow every correct Blockchain node to execute this function and receive the same result to either accept or reject a proposal to add a node. The implementation of the deterministic *ValidateIDProof* function may be specified by developers, who use the DIoT framework to develop a DIoT Blockchain. This way, the Blockchain developers have the flexibility to decide for a type of proof that is suitable for the use case, which is served with the Blockchain. To be specific, *ValidateIDProof* is a function interface rather than an actual function, as it only defines the function arguments and return values. A function that adheres to the *ValidateIDProof* interface can be registered in a DIoT Blockchain as shown in Listing 4.1.

The *main* function (Line 25) is the executable entry point to initiate a DIoT node. In Line 30 the *validateID* function is registered as the *ValidateIDProof* mechanism of the DIoT node. As shown in *validateID* (Lines 5-23), a function that implements the *ValidateIDProof* interface, the proof information is received as byte array, i.e., *signedPDF* in this case, and a boolean *validity* value, which indicates the validity of the proof, is returned together with a hash of the proof information (*idProofHash)* to allow the DIoT node to verify if a valid proof was used sometime before. A complete example of how a Blockchain with a proof of legitimacy is set up is provided under the *example/idproof* directory of the go-diot project.

```go
// validateID implements the ValidateIDProof interface. It receives
// the proof information as bytes, i.e. signedPDF in this case, and
// has to return a boolean validity value and a hash, which is used
// in the DIoT BC to ensure that every ID proof is used only once.
func validateID(signedPDF []byte) (validity bool, idProofHash []byte) {

  // Check if the PDF contains a valid signature
  err := registration.VerifyPDFSignatureBytes(
    signedPDF,
    "cert_chain.pem"
  )
  if err != nil {
    fmt.Println(err.Error())
    return false, nil
  }
  fmt.Println("signature valid!")

  // Since the signed pdf is valid, generate id proof hash.
  h := sha1.New()
  h.Write(signedPDF)
  idProofHash = h.Sum(nil)
  return true, idProofHash
}

func main() {
  app := diot.NewBCNode(
    "diot_config.yaml",
    "blockchain.sqlite"
  )
  app.RegisterValidateIDProofFunc(validateID) {
  app.Start(tendermintConfig)
}
```

Listing 4.1: Register a custom ValidateIDProof function in a DIoT Blockchain

The following list gives suggestions for the information that could be used as proof of legitimacy in a *ValidateIDProof* function:

- **Proposal is sent from the coordinator:** It might be the case that the operator of the coordinator node has built up enough reputation for nodes to trust that he handles the task of adding and removing nodes reliable and correctly, which means that no further proof of legitimacy, other than the proposal being sent from the coordinator node, is required. This option has been used throughout the evaluation phase as it is convenient for testing.
- **Certificate issued by trusted CA:** Common practice to mitigate the risk of sybil attacks is to demand a certificate from new network nodes, that is issued by a trusted third party such as a CA or identity verification service provider [BS12]. A

certificate together with a digitally signed document to demonstrate the possession of the private key can be a suitable proof of legitimacy. It can be easily verified by the Blockchain nodes in a deterministic function without the need for additional validation requests to other parties.

- **TPM Manufacturer certificate:** Similar to a CA certificate attesting that a node is associated with an identity, it is also possible to bind a Blockchain node to a certain piece of hardware, which also restricts the possibility to create an arbitrary number of virtual sybil nodes. TPM 2.0 hardware, which is commonly used in notebooks and also available as extension for single-board computers, includes a manufacturer certificate [CL13], which could be used as proof of legitimacy in a similar way as the previous option.

- **Certificate from decentralized Blockchain-based PKI:** In recent years, a lot of research and development was invested into decentralization of PKI, notable projects being for instance CertCoin [FVY14] and Authcoin [LCMR16]. Both projects encourage proper handling of identity verification requests by rewarding verifiers with built-in crypto-currency tokens. However, there is no production-ready solution available just yet [BKUE20]. The availability of decentralized PKI would reduce the dependency on centralized components of the DIoT framework even further and might be worth considering in the future, when more mature solutions come up.

Figure 4.3 illustrates the message flow of adding a new node in a DIoT Blockchain on a high level, using the second of the above mentioned options for a proof of node legitimacy, i.e., by incorporating certificates from a trusted CA. Every Blockchain node keeps a list of the currently active Blockchain nodes. Only transactions received from one of these nodes are accepted and added to the Blockchain. At start, the Blockchain counts three nodes; Coordinator, Node1 and Node2. Node3 would like to join the Blockchain and sends a request containing a CA certificate and signature to the register service, that is operated by the coordinator (step 1 and 2). The coordinator creates a proposal to add the new node and transmits it via a Blockchain transaction to the other nodes (step 3). Every node validates the proof of legitimacy provided by Node3 (step 4). This step involves the custom implementation of the *ValidateIDProof* interface. On successful validation, the Blockchain nodes add Node3 to their local list (step 5), which concludes the process of adding a new node to a DIoT Blockchain.

4.5. Technology stack

The DIoT framework builds upon a set of enabling technologies apart from its main dependency Tendermint. These technologies are mentioned in this Section and additionally, the reason to use the selected technologies in particular is clarified.

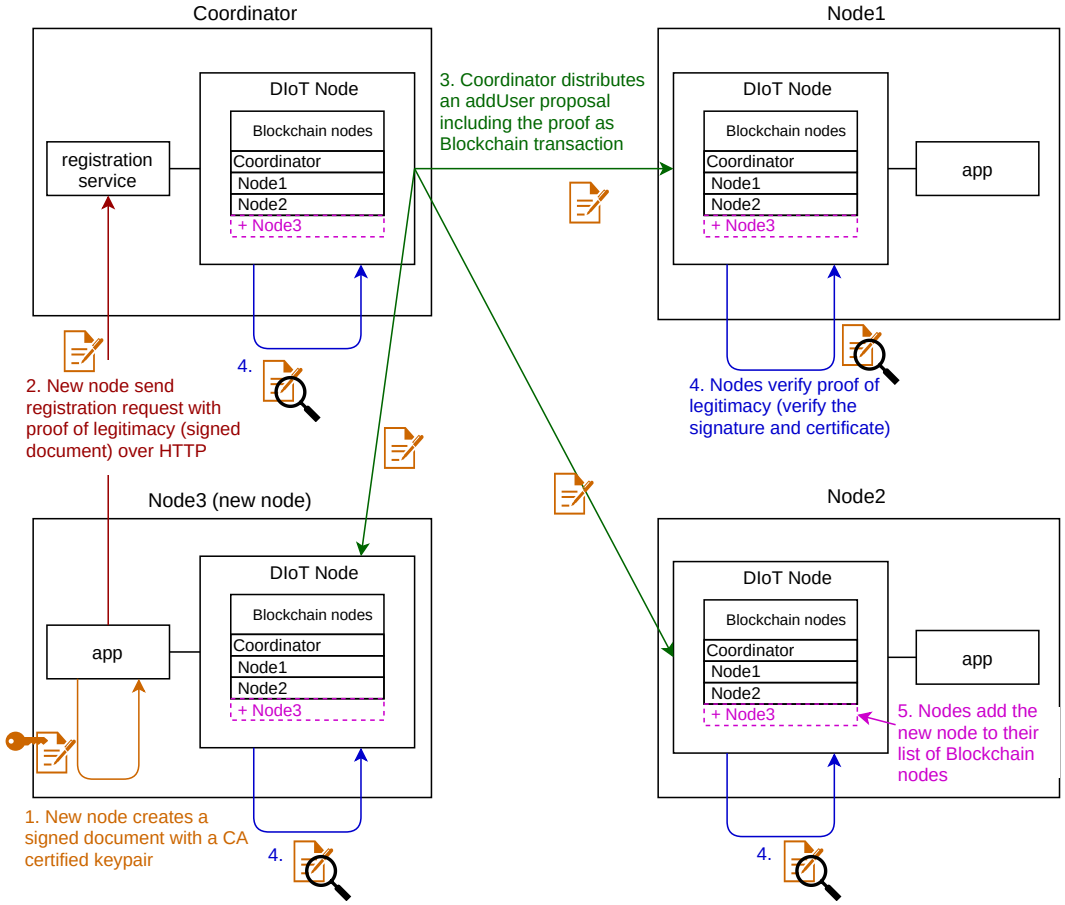


Figure 4.3.: Message flow of adding a new node to a DIoT Blockchain

4.5.1. Serialization

Storage is a big concern in Blockchain technology as the transactions added to a block remain in the Blockchain for the entire life time. DIoT uses *Protocol Buffers*, also called *protobuf*[8], to serialize Blockchain transactions for sending transactions from a client application to the DIoT node and for on-disk storage on a DIoT node. Protobuf is a language-independent binary serialization protocol which allows to develop DIoT client implementations in many different programming languages. Further, it achieves better results in terms of serialization speed and storage consumption compared to other language-independent serialization protocols such as JSON or XML, as investigated by Sumaray and Makki [SM12].

[8] `https://developers.google.com/protocol-buffers`, accessed: 2021-05-14

4.5.2. Database

Additionally to the transactions that are stored sequentially in the Blockchain, relevant application data is also stored in a database to allow querying for messages that have been added to the Blockchain in the past. For the used database, a few important requirements have to be considered:

For the realization of publish-subscribe messaging, which DIoT aims to provide, a simple key-value store is not sufficient to model the required data structure. Database systems that can incorporate complex data structures, such as document-databases or Structured Query Language (SQL) databases, are required.

A further important requirement results from the way Tendermint organizes the creation of blocks in a proposal-commit scheme [Tenc], i.e., first a block proposal is made and if it is accepted by other validators, the block is committed to the chain. In order to ensure consistency between the application data stored in the database and the Blockchain state, the database must provide a similar way to perform updates to the stored data, for instance as an atomic batch of transactions, i.e., collect pending transactions in a session and commit them all at once.

SQLite[9] is a very lightweight SQL database that does not need a dedicated process to operate and it fulfills all of the above mentioned requirements, which makes it ideal for the use in the DIoT framework.

4.5.3. Programming languages

The major part of the go-diot project is written in Go, which includes the DIoT node and the modifications of the Tendermint source code. Since Tendermint as the main dependency of the DIoT framework is also implemented in Go, this is the easiest way to integrate Tendermint into the DIoT framework. It is however technically possible to develop language bindings to the DIoT framework for many different programming languages. Therefore, application developers are not limited to use Go for an application that uses the DIoT framework.

Python also plays an important role in the go-diot project for implementing the simulation tools that are introduced in Chapter 5 as well as scripts to help deploying the evaluation testnet. Python has been chosen for these smaller tasks, as it provides a developer-friendly syntax and tools for development and debugging, which greatly accelerates development.

4.5.4. Cryptography algorithms

The DIoT framework requires cryptographic algorithms for hashing (e.g. to generate ids and block hashes, and to perform PoW) and digital signing (e.g. sign Blockchain transactions). For hashing, SHA256 comes to use as it is commonly used in cryptographic applications such as TLS and supported in most programming language's standard library. For digital signatures and key-pair generation, the ED25519 elliptic-curve algorithm is

[9] `https://www.sqlite.org/index.html`, accessed: 2021-05-17

used, as Tendermint uses this specification [Buc16] and the DIoT framework utilizes the public keys generated by Tendermint as well.

4.6. Modification of the Tendermint implementation

The Tendermint source code of version 0.34.0[10] has been edited to include a PoW in the block proposal process and the validation of the PoW of a proposed block into the block validation process in a similar way as the code shown in Listing 3.5 and 3.6 of the Solution design chapter.

```
1  {
2    "jsonrpc": "2.0",
3    "id": -1,
4    "result": {
5      "block_id": {...},
6      "block": {
7        "header": {
8          "version": {...},
9          "chain_id": "test-chain-ElOG5e",
10         "height": "50",
11         "time": "2021-06-22T11:55:09.053726048Z",
12         "last_block_id": {...},
13         "last_commit_hash": "D7109596C4BD1B30F8D7967B2F99BFB3F5B04728EE6901F898D84254BF4181A6",
14         "data_hash": "C740C38C41CE33C5F9E354ED48A7ABFF02BED170AB6D63EC4173143C9BD5BF3F",
15         "validators_hash": "369A363CCD7B932FA18139932AAEC23740871B84BAF43E96615C4AA68FFCD05F",
16         "next_validators_hash": "9CA5D2C9F2BEB7A3905A9CFB4CAF2FB4CEBB36F04AF2D56912A18B14C078481C",
17         "consensus_hash": "048091BC7DDC283F77BFBF91D73C44DA58C3DF8A9CBC867405D8B7F3DAADA22F",
18         "app_hash": "C21A47FD2503E094AEFA759A09A82C1E705C60E6D6539AE2B0508CFCC3A18FB7",
19         "last_results_hash": "7162ED848F19740E53766CE01AC099523B099D593E0782DDBC5296EECE50EC50",
20         "evidence_hash": "E3B0C44298FC1C149AFBF4C8996FB92427AE41E4649B934CA495991B7852B855",
21         "proposer_address": "074944E8684855F3CA574CD22CE40EF85BD1A2E8",
22         "pow_nonce":  "000000000000048900000000001395A5000000000021B8D0",
23         "pow_rand":  "58747A76454252386E68686378694B38664A686A3336343966457442696442 30"
24       },
25       "data": {...},
26       "evidence": {...},
27       "last_commit": {...}
28     }
29   }
30 }
```

Listing 4.2: Modified Tendermint block header with additional PoW header fields.

The conducted modifications result in a change of the Tendermint block header. Two additional header fields have been added to the existing fields; *pow_nonce* and *pow_rand*. Listing 4.2 shows an example block of the modified Tendermint system with the additional header fields (Line 22+23). Both values are necessary for validator nodes to validate the PoW of the block proposal and need therefore be included in the block header. The block proposer uses the PoW header fields together with the *app_hash* field as follows:

$$pow\_nonce = ProofOfWork(hash(app\_hash, pow\_rand), diff, timeout)$$

[10] `https://github.com/tendermint/tendermint/tree/v0.34.0`, accessed: 2021-05-12

If the block proposer manages to solve the PoW before *timeout*, *pow_nonce* and *pow_rand* are added to the proposed block's header. Validator nodes validate the PoW upon receiving the block proposal using the same arguments:

$$validated = ValidatePoW(hash(app\_hash, pow\_rand), pow\_nonce, diff)$$

The *app_hash* field in a Tendermint Blockchain is equivalent to the block hash in other Blockchains as it is the hash of all contained transactions of a block and the *app_hash* of the previous block, similar to the block hash visualized in Figure 2.1. It is used in the PoW as input argument to ensure that the PoW is not pre-calculated at an earlier point in time than during the block proposal.

The *diff* value describes the PoW difficulty. It is a static value that is set in a config file for all Blockchain nodes and therefore does not have to be included in the block header.

PoW_rand is a fixed-length byte string with arbitrary value and the only part of the PoW, that can be chosen freely by the block proposer. The purpose of *pow_rand* is to change the input arguments of subsequent PoW attempts for the same block. In case a PoW happens to be very difficult and the block proposer fails to solve it before the timeout, without *pow_rand*, the next block proposer would face the same PoW problem and is likely to fail as well. But with the *pow_rand* argument hashed together with the *app_hash*, a subsequent PoW attempt is always different from the previous one with a potentially less difficult PoW that is more likely to be solved.

PoW_nonce (also referred to as *proof*) is the evidence that the PoW has been completed by the block proposer. How exactly the proof value is composed, as well as the *ProofOfWork* and *ValidatePoW* functions, is described in Section 4.7.

4.7. Proof of Work implementation

Conveniently, an open source implementation of a birthday-collision PoW (see Section 2.3) in Go does already exist[11], which is used as reference for the PoW algorithm in the DIoT framework. In fact, only minor changes have been made to integrate the existing algorithm into the DIoT framework and comments have been added for better understanding. The algorithm consists of two functions; *ProofOfWork* and *ValidatePoW*, which are both examined in this section. The goal of this implementation of a birthday-collision PoW algorithm is to find three hashes of the same input data and different nonce values, that all start with the identical bit sequence, i.e., a partial hash collision. The PoW difficulty is given by the number of bits required for the partial hash collision.

Listing 4.3 shows the *ProofOfWork* function, which is executed by the block proposer to accomplish the PoW. The main part of the function is executed in a loop until the PoW is completed or the PoW timeout, which is passed as function parameter, runs out (Lines 12-16). In every iteration, an integer nonce value starting from 1, is incremented to generate a new hash value.

[11] `https://github.com/bwesterb/go-pow`, accessed: 2021-03-18

```go
// Code adopted from https://github.com/bwesterb/go-pow
func ProofOfWork(data []byte, diff uint32, timeout time.Duration) []byte {
  type Pair struct {
    First, Second uint64
  }
  startTime := time.Now()
  var i uint64 = 1
  nonce := make([]byte, 8)
  resBuf := make([]byte, 32)
  partialCollisionsMap := make(map[uint64]Pair)
  h := sha256.New()
  for {
    // check if timeout is triggered
    if elapsed := time.Since(startTime); elapsed > difficulty.Timeout {
      return nil
    }
    // calculate a hash of the nonce and the input data
    binary.BigEndian.PutUint64(nonce, i)
    h.Write(nonce)
    h.Write(data)
    h.Sum(resBuf[:0])
    // Extract the least significant bits from the bitstring
    // up to the position according to diff.
    // E.g.: diff = 5; 101101011011 & 11111 = 11011
    res := binary.BigEndian.Uint64(resBuf) & ((1 << diff) - 1)
    // If there is already an entry for the partial hash
    // in the map, a partial collision with a previous hash was found
    if pair, ok := partialCollisionsMap[res]; ok {
      // If the first and the second value of the pair are
      // already set, the PoW is completed
      if pair.Second != 0 {
        ret := make([]byte, 24)
        binary.BigEndian.PutUint64(ret, pair.First)
        binary.BigEndian.PutUint64(ret[8:], pair.Second)
        copy(ret[16:], nonce)
        return ret
      }
      // If the second value is empty, the partial collision occurred the first time.
      // Store the nonce as the second value of the pair
      partialCollisionsMap[res] = Pair{First: pair.First, Second: i}
    } else {
      // The partial hash does not exist in the map yet.
      // Store the partial hash and nonce in the map.
      partialCollisionsMap[res] = Pair{First: i}
    }
    // Clear the hash and increment nonce for next iteration
    h.Reset()
    i++
  }
}
```

Listing 4.3: The ProofOfWork function used in the go-diot project

The hash value is calculated from the input data, which is passed as argument to the function, and the nonce (Lines 18-21). In Line 25, the $(1 << diff) - 1$ part generates a bitmask using bit shifting with a number of *1*-bits according to the *difficulty* (e.g. $difficulty = 5 \rightarrow bitmask = 11111$). The bitmask is further used in the same line to extract a partial bitstring from the beginning of the hash, i.e., right-hand side, using the bitmask in combination with a *bitwise and* operation. (E.g. $hash = 101101011011; 101101011011\&11111 = 11011$). The partial bitstring is looked up in the *partialCollisionsMap* to see if previous iterations received the same partial bitstring, and if so, a collision is found. Once three identical partial bitstrings are found, i.e., two collisions with an already existing partial bitstring in the map, the PoW is completed and the proof, which consists of the nonce values of all three hashes that are used to produce the same partial bitstring, is returned (Lines 28-36). Otherwise the algorithm proceeds with another iteration (Lines 40-49).

```
1  // Code adopted from https://github.com/bwesterb/go-pow
2  func ValidatePoW(proof []byte, data []byte, diff uint32) bool {
3    resBuf := make([]byte, 32)
4    if len(proof) != 24 {
5      return false
6    }
7    // Extract the nonce values from the proof argument
8    nonce1 := proof[:8]
9    nonce2 := proof[8:16]
10   nonce3 := proof[16:]
11   // Recurring nonce values in the proof are invalid
12   if bytes.Equal(nonce1, nonce2) || bytes.Equal(nonce2, nonce3)
13       || bytes.Equal(nonce1, nonce3) {
14     return false
15   }
16   // Calculate the hash value of the nonce and the input data
17   // and extract the partial bitstring from the hash for validation.
18   h := sha256.New()
19   h.Write(nonce1)
20   h.Write(data)
21   h.Sum(resBuf[:0])
22   res1 := binary.BigEndian.Uint64(resBuf) & ((1 << diff) - 1)
23   h.Reset()
24   h.Write(nonce2)
25   h.Write(data)
26   h.Sum(resBuf[:0])
27   res2 := binary.BigEndian.Uint64(resBuf) & ((1 << diff) - 1)
28   h.Reset()
29   h.Write(nonce3)
30   h.Write(data)
31   h.Sum(resBuf[:0])
32   res3 := binary.BigEndian.Uint64(resBuf) & ((1 << diff) - 1)
33   // Validate that the bitstrings are identical, i.e., partial hash collision
34   return res1 == res2 && res2 == res3
35 }
```

Listing 4.4: The ValidatePoW function used in the go-diot project

Listing 4.4 shows the *ValidatePoW* function, which is executed by the block validators to verify the correctness of the PoW. First of all, the three nonce values that should lead to the identical partial bitstring are extracted from the proof argument, which is the return value from the *ProofOfWork* function (Lines 8-10). During the rest of the function, the hash generation and extraction of partial substrings are repeated exactly as in the *ProofOfWork* function with the given nonce values (Lines 16-32). At the end, all three partial bitstrings must be identical to ensure that the PoW is valid (Line 34).

The chosen PoW implementation has the benefit of being a well tested and established solution that it is easy to understand and therefore easy to integrate into the go-diot project. The main concern regarding PoW algorithms of this thesis is to elaborate the use of PoW algorithms in general to countermeasure brute-force attacks on the Seed Generation mechanism. Investigating further PoW algorithms, such as Equihash [BK17] or Scrypt [Per09], for additional optimization of the DIoT framework is considered an important topic, but goes beyond the scope of this thesis.

4.8. Chapter summary

The DIoT framework implements publish-subscribe messaging using an underlying Blockchain with Tendermint consensus. The Blockchain transactions are interpreted as messages sent between the Blockchain nodes. Every Blockchain node executes a DIoT node process which handles the interaction with other DIoT nodes and can connect to multiple client applications. The Tendermint implementation has been modified to incorporate the PoW in the block proposal process, which is used during the Validator Management process in the Seed Generation function as described in Chapter 3. Having all important implementation-related aspects of the DIoT framework covered, in the next chapter the DIoT framework is used to set up an actual Blockchain network to evaluate if the requirements of a Blockchain-based communication method for IoT networks are met.

5. Evaluation

Chapter 3 proposed Validator Management to achieve a highly decentralized private Blockchain that is suitable for trustworthy information exchange in IoT networks and Chapter 4 documented the steps to create the publish-subscribe messaging framework DIoT on top of this novel Blockchain technology as a solution to the research questions of this thesis. In the evaluation phase, which is described in this chapter, the quality of the proposed solution is evaluated regarding several functional and non-functional aspects of the proposed system. The go-diot implementation[1] contains all evaluation related programs and scripts as well as instructions how to reproduce the conducted experiments.

5.1. Evaluation scenario

The second experiment (Subsection 5.3.2), which is the most important experiment of the evaluation, is carried out in a medium-sized network of approximately 100 network nodes, that are all interconnected via a DIoT Blockchain. The devices are expected to have at least moderate computing power, comparable to a single-board computer such as the RPi3.

The network is not actually physically assembled in this evaluation, but rather simulated with the help of Docker[2] virtualization on a single PC. One Docker container simulates one node of the network, therefore the evaluation hardware must be able to simultaneously run 100 Docker containers. The network, that was specifically developed for the evaluation, is called *DIoT testnet* or just *testnet*, for short. The testnet is further elaborated in Section 5.2.

Example application: IoT vulnerability detection system In order to demonstrate the relevance of the evaluation experiment setting, an example scenario from the application field of smart home is presented that would be deployed in a similar way as the testnet:

Security is an ongoing challenge in IoT. Smart home devices are hijacked by attackers to form botnets to carry out DDoS attacks (e.g. Mirai [KKSV17]). This example application shows the idea of deploying an intrusion detection system in a smart home control unit together with a DIoT Blockchain to detect attempts to hijack smart appliances and report them to the smart home service provider and other smart homes in a transparent and reliable way. Figure 5.1 shows the deployment and the message flow of this application.

As the DIoT framework uses a private Blockchain, the smart home units must first register at the smart home service provider, which is the coordinator node of this

[1] `https://bit.ly/3rHFxVh`, accessed: 2021-10-15
[2] `https://www.Docker.com`, accessed: 2021-09-14

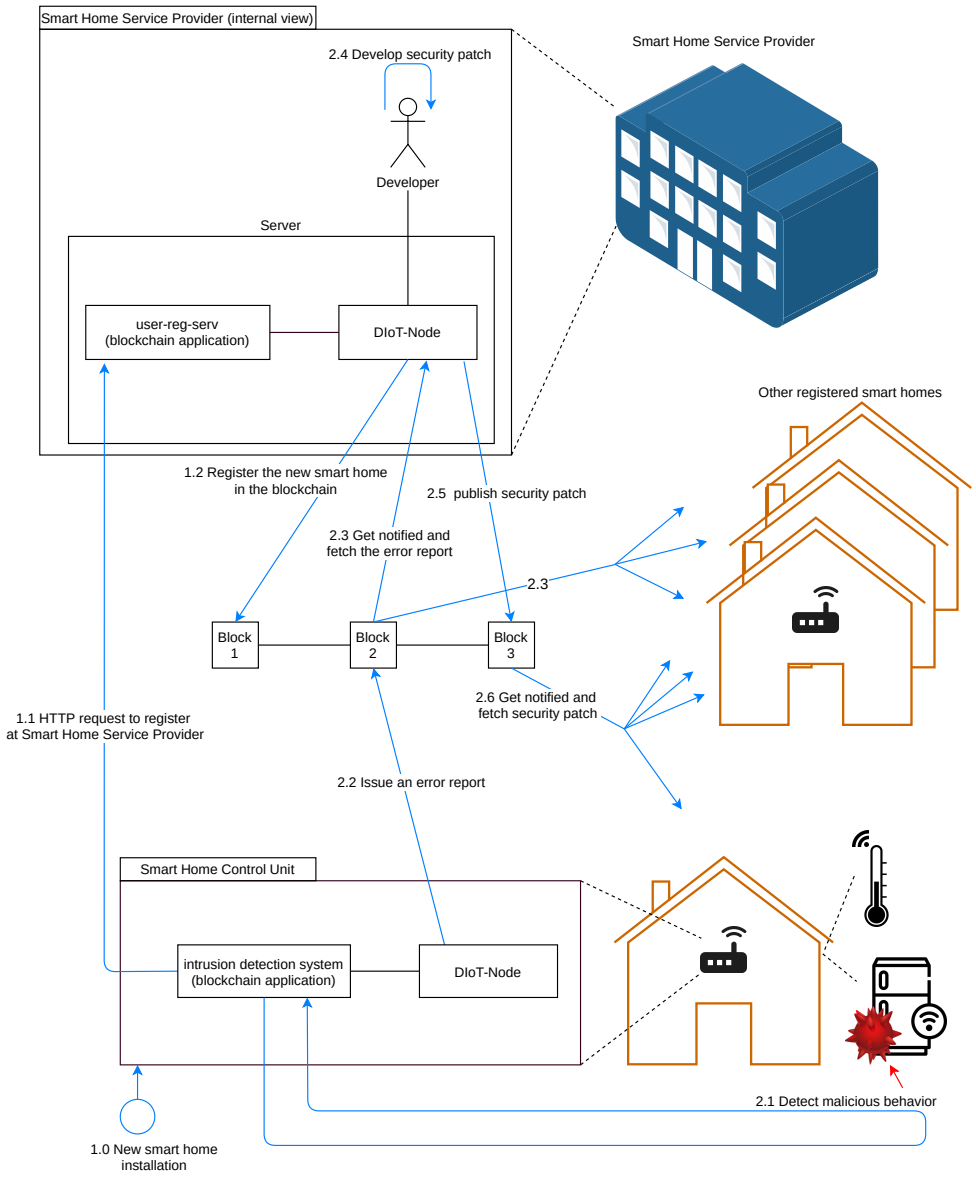


Figure 5.1.: Evaluation scenario: IoT vulnerability detection with decentralized communication over a DIoT Blockchain

Blockchain, in order to communicate using the Blockchain (Actions 1.0 - 1.2 in Figure 5.1). Once the Blockchain transaction containing the registration data is added to the Blockchain in block 1, all other smart home units and the smart home service provider are aware of the new member of the network.

After the successful registration, the intrusion detection system of a smart home unit monitors the devices in the local network. Action 2.1 is triggered by suspicious behavior of one of the IoT devices. An error report is created and sent as a Blockchain transaction

to the DIoT Blockchain (Action 2.2) and is added to the Blockchain as part of block 2. On receiving the error report contained in block 2 (Action 2.3), the smart home units can react to the discovered vulnerability (e.g. by limiting functionalities). The smart home service provider develops a security patch and either publishes it directly on the DIoT Blockchain or, in case the file size is too big, stores it on a decentralized storage such as the Filecoin Blockchain[3] and publishes a reference to the DIoT Blockchain (Actions 2.4 and 2.5). The security patch is added as part of block 3 to the DIoT Blockchain where it is accessible for all smart home units to apply it to the affected IoT device of the local network (Action 2.6).

As mentioned above, the testnet for the evaluation of the DIoT framework is set up with similar deployment and message flow as the just presented example, but it is also simplified to allow for easy reproduction of the experiments to validate the experiment results. A Docker container in the testnet corresponds to a smart home control unit in the given example. The role of the smart home service provider, that handles adding and removing nodes from the Blockchain, is performed by a DIoT node on the physical machine in the testnet. The testnet uses only a very simplified Blockchain application that only exchanges transactions with dummy payload instead of meaningful messages for intrusion detection or any other use case. The next section elaborates further on the composition and implementation of the testnet.

5.2. DIoT testnet

Distributed system tests, in particular for Blockchains, can be time-consuming and expensive, as it can be necessary to set up a large network of nodes to test certain aspects of the system, such as transaction time or fault tolerance.

The test environment of the DIoT framework, called *testnet*, strives to be highly automated and configurable to efficiently allow the execution of tests with several different configurations of the DIoT framework. In order to allow simple reproducibility of the conducted tests with minimal hardware requirements, Docker containers are utilized to simultaneously run several network nodes on a single physical machine. Python scripts are used for the automation of the configuration and the build process to set up the Docker containers and to collect and analyze the logs from the running Docker containers. The testnet is located in the go-diot project under the *testnet* sub-directory. The included *README.md* file in the testnet directory shows how to set it up.

The testnet mimics a network of nodes that uses DIoT for communication, similar to the scenario shown in Figure 5.1. Each Docker node runs identical DIoT node and client programs. The behavior of the different network nodes is determined by environment variables, that are injected into the Docker containers via the Docker Compose tool[4]. Every node receives a different ID via an environment variable, which influences the timing of when nodes send transactions to the network. Additionally, it is possible to configure a certain number of the nodes to act malicious, i.e.. byzantine nodes, via a

[3] `https://filecoin.io`, accessed: 2021-04-22

[4] `https://docs.docker.com/compose/`, accessed: 2021-10-15

configuration file. This information is also passed to the containers via an environment variable.

5.3. Evaluation experiments

With the help of the DIoT testnet (see Subsection 5.2) and the *vssim* and *powsim* simulation tools, that are part of the go-diot project (see Appendix A), three experiments are conducted in the evaluation phase to evaluate relevant functional and non-functional aspects of the DIoT framework.

The first experiment focuses on the evaluation of the effectiveness of PoW as a measure against the brute-force attack described in Paragraph *"Brute-force attack to manipulate Validator Selection"* of Subsection 3.5.4 and on the adjustment of PoW specific system parameters.

The second experiment verifies the correctness of the Validator Management process by comparing the validator set changes in the testnet to the Validator Management simulation tool. The testnet is further used to observe transaction times which gives insight about the expectable message latency in a production system.

The last experiment is conducted to find out if the DIoT framework also runs on ARM CPU architecture, which is often used in single-board computers such as the RPi3. Compatibility to ARM hardware is particularly interesting for several IoT use cases. Furthermore, the PoW execution time of a RPi3 is compared to more powerful hardware systems, to find out if an advantage can be generated by an attacker using more powerful hardware than the correct nodes of the network.

5.3.1. Experiment 1: Effectiveness of the embedded PoW against brute-force attacks targeting a DIoT Blockchain

PoW can only be used as an effective measure against the brute-force attack that targets Validator Management (described in Paragraph *"Brute-force attack to manipulate Validator Selection"* of Subsection 3.5.4), if it delays a single generation of a seed value enough to avoid multiple seed value generations within the predefined timeout. Timeout and PoW difficulty have to be configured in a way that the expected PoW time is slightly below the timeout, to allow a single PoW calculation within the timeout, but not multiple calculations of the PoW.

Since it is not possible to determine, how many attempts will be necessary to find a hash that solves a PoW, using a PoW as measure against the mentioned vulnerability is a probabilistic approach. An attacker could be lucky and face several relatively easy PoW problems in a row which could all be solved within a single timeout. Every additional PoW calculation an attacker can solve within the timeout improves his chances to manipulate the Validator Selection. The challenge, an application developer has to face, is to provide enough margin between the expected PoW execution time and the timeout to allow PoW executions that are slightly above the expected PoW time to be within the timeout, and at the same time make it unlikely that an attacker manages to calculate the PoW multiple

times before the timeout. This experiment shows a practical approach for an application developer to choose proper values for all PoW related parameters of a DIoT Blockchain.

An important quality attribute for this application of the PoW is its variance. With a PoW algorithm that has low variance, a small margin can be chosen between the expected PoW time and the timeout to allow a single PoW execution and avoid multiple PoW executions with a high probability. As mentioned in Section 4.7, the choice between different PoW algorithms is beyond the scope of this thesis and only a single PoW algorithm is at disposal for now. Given this limitation, it would be interesting to know if, besides the PoW algorithm, the PoW difficulty can impact the variance as well. When comparing variances of different PoW difficulties, the variance relative to the corresponding average value is the determining factor.

PoW difficulty	20	25	30	31	32	33	34
Average PoW time [ms]	7.7	54.42	681.96	1145.32	1718.4	2743.39	4574.91
PoW standard deviation [ms]	3.24	24.97	278.18	431.12	755.21	975.53	1898.46
Relative standard deviation [%]	42.07%	45.89%	40.79%	37.64%	43.95%	35.56%	41.50%

Table 5.1.: Average PoW times and correlation between PoW difficulty and variance.

A quick preliminary experiment to verify the assumption of a correlation between PoW difficulty and variance is conducted by comparing series of PoW executions with different difficulty values. The result is presented in Table 5.1. Each series of the experiment, represented as a column in Table 5.1, consists of 100 PoW executions with a particular PoW difficulty. An average PoW execution time is calculated from the PoW executions. The standard deviation is used to express the variance of the PoW executions from the average time. The relative standard deviation is obtained by $\frac{standardDeviation}{averageTime}$.

As the result of this experiment shows, there is no noticeable trend of increasing or decreasing relative standard deviation visible with increasing PoW difficulty, which leads to rejecting the assumption that the chosen difficulty can influence the variance of PoW execution time.

Although the choice of the PoW difficulty does not seem to affect the variance, the PoW difficulty should still be chosen to aim for an expected PoW time at least in the range between one and a few seconds to make other factors that influence the block proposal time, such as the data transfer over the network, negligible. This way, no additional factors other than the PoW time have to be taken into account when defining a timeout for the block proposal. In other words, if a very easy PoW difficulty is chosen that allows to solve PoW tasks in a few milliseconds (e.g. difficulty 20 or 25, see Table 5.1), it is hard to determine for other nodes if a delay comes from an attacker calculating multiple PoWs for a brute-force attack, or, for instance, just from a bad network connection of the block proposer. On the other hand, if a PoW with higher difficulty is chosen, and a delay of a couple seconds occurs, it is barely possible that this rather long delay is caused by a bad network connection or any other additional factor. Considering the average times in Table 5.1, a PoW difficulty of 31 seems to provide a reasonable delay for this purpose, which is at average 1145 milliseconds.

The other important PoW parameter left to determine is the PoW timeout, that ensures that the block proposal is aborted when a node takes longer time than expected for the calculation of the PoW. The timeout is not expressed through an absolute value, but rather through an additional percentage margin from the expected PoW time. Therefore, the insights regarding the PoW timeout margin from this experiment should also be applicable to other DIoT Blockchains with different *expectedPoWTime*. The actual timeout value is calculated as:

$$timeout = expectedPoWTime * (1 + margin) \tag{5.1}$$

To find out a suitable margin for the PoW timeout, the following experiment is conducted:

First of all, the *expectedPoWTime* value, which is required to calculate a PoW timeout, is simply obtained by calculating the average execution time of a series of PoW executions, as performed in the preliminary experiment.

After that, a series of margin values from 0% to 120% in 20% steps, i.e., $[0.0, 0.2, ..., 1.2]$, is used to observe the changes of block proposal time and the effectiveness of the PoW against brute-force attacks at different timeout margins. For every margin value, 100 block proposals are simulated and logged, using a PoW difficulty of 31. If the mandatory PoW for a block proposal exceeds the timeout, the block proposal is restarted and an additional penalty of one second is added to the execution time, which is roughly the time it takes in Tendermint consensus, until the next validator would begin with the block proposal attempt. To measure the effectiveness of the PoW against the brute-force attack with different timeout margins, the sequential PoW executions, that can be accomplished within a single timeout, are traced from the PoW execution logs. It is expected that with a greater timeout, more and longer sequences of PoW executions within a timeout will occur. The aim of the experiment is to find a PoW timeout, that allows only a low amount of PoW sequences and at the same time does not slow down the time for block proposals too much, since the block proposal time determines the communication latency of the Blockchain-based communication.

The entire experiment, as it is explained above, is implemented in the *powsim/main.go* file in the go-diot project. The program executes the experiment and writes the results to the *experiment1_result.json* file. The *generate_chart.py* script uses the experiment results to render a diagram as presented in Figure 5.2. The diagram shows an overlay of two charts that share the same x-axis to emphasize the trade-off between block proposal time and the number of occurring sequential PoWs within a timeout.

The left y-axis scale corresponds to the blue line, showing the change in block proposal time with increasing timeout margin. With a timeout that is close to the expected PoW execution time, i.e., a low timeout margin, correct nodes will occasionally trigger a timeout while calculating the PoW, which slows down the average block proposal time. With increased timeout margin, correct nodes have a better chance to finish the PoW before the timeout. This leads to rare timeouts and a decreased average block proposal time, which is preferable from an application perspective as this means also lower message latency.

The right y-axis scale corresponds to the three stacked area plots (yellow, orange and red) that show the probability of the occurrence of two, three, and four or more PoWs

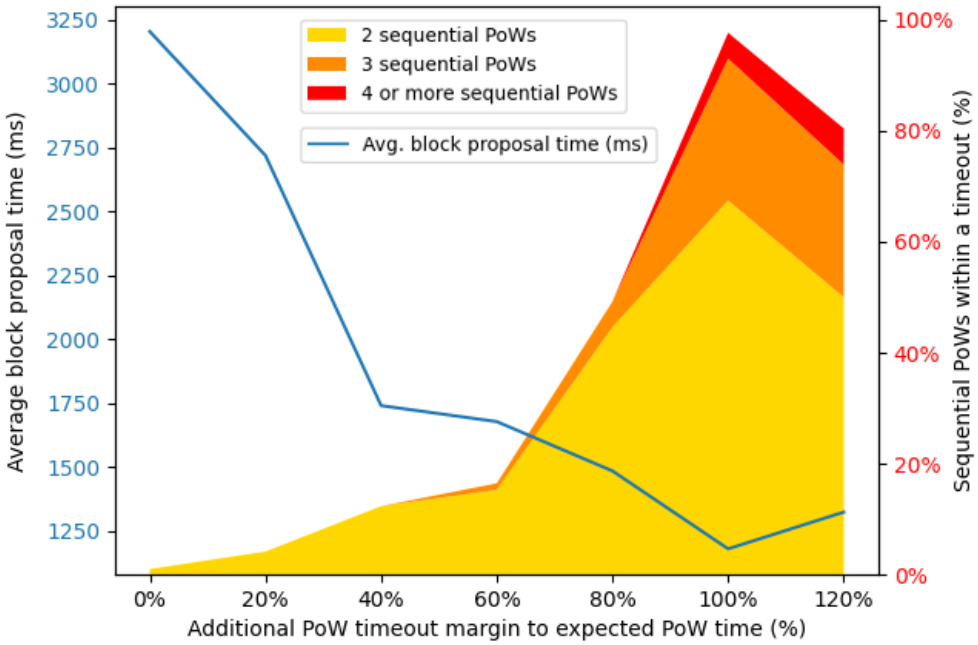


Figure 5.2.: Block proposal time and occurrence of sequential PoWs with increasing timeout.

within a single timeout. With increasing timeout, the chance of sequential PoWs within a single timeout increases as expected. The color change from yellow, to orange and red represents the different threat-levels of the occurring sequential PoWs. The occurrence of two sequential PoWs within a timeout (yellow) only poses a moderate threat of an attacker performing a brute-force attack, as with only two chances, a brute-force attack is very unlikely to be successful. The class of 4 or more sequential PoWs on the other hand (red) poses a higher threat and should be avoided by choosing a smaller timeout margin on the x-axis.

As the diagram in Figure 5.2 shows, 40% timeout margin appears to be a good trade-off to allow both, acceptable block proposal time and effective protection against the brute-force attack. The PoW timeout corresponding to this margin value is 1724 milliseconds in this experiment, as obtained from the generated *experiment1_result.json* file.

In conclusion of this experiment, PoW with a moderate difficulty in combination with a well balanced timeout is an effective measure to greatly reduce the risk of the brute-force attack that was described in Chapter 3 and still allows a reasonable block proposal time of a few seconds. Readers are invited to reproduce this experiment by following the instructions of the *README.md* file in the go-diot project under the *powsim* sub-directory.

5.3.2. Experiment 2: Functional evaluation of DIoT Validator Management in the presence of byzantine nodes

This experiment serves as the first test run of a DIoT Blockchain in a network of multiple nodes. The DIoT testnet is used for the purpose of setting up a test network of DIoT nodes on a single physical machine with multiple Docker containers. The aim of this experiment is to verify if the core component of the system, the decentralized Validator Management process, behaves as intended and to gather first reference values regarding message latency and fault tolerance for the use of a DIoT Blockchain as Blockchain-based communication system in actual networks.

The testnet is configured with the largest possible number of Docker containers that can be managed by the available hardware (PC with AMD Ryzen 5 3600 CPU, 16 GB RAM) to achieve an environment for the experiment that is as close as possible to a Blockchain network running in production. The limit of Docker containers running the testnet software, that could be managed with the available hardware, was found to be around 100 containers.

For this experiment, the testnet was configured with 10% byzantine nodes, as the vssim tool (see Appendix A.1) showed in preliminary executions, that a correct behaving Validator Management should be able to withstand this amount of byzantine nodes in the network. Byzantine nodes in the testnet follow a simple strategy of terminating the program after successful registration in the Blockchain network. This is essentially only a crash-fault strategy and not a byzantine-fault strategy but as the Tendermint consensus protects against crash faults and byzantine faults equally well, it is not necessary to use a more sophisticated strategy.

The plan for this experiment was to use the PoW difficulty and PoW timeout margin that were found out in experiment 1 (difficulty=31; timeout=1724 ms, see Subsection 5.3.1), but it was observed in preliminary test runs, that the execution time for PoW tasks is significantly longer than the times measured in experiment 1, presumably due to the additional load of managing an extensive amount of Docker containers.

To compensate for the additional workload, the PoW difficulty was reduced to 30, which resulted in an average measured PoW time of 1098 milliseconds under the load of the Docker containers. To maintain a timeout margin of 40% as it was used in experiment 1, the PoW timeout was adjusted to a value of 1537 milliseconds according to Equation 5.1 of experiment 1:

$$1098 * (1 + 0.4) \approx 1537$$

Table 5.2 shows the exact parameters that were used in the testnet during the experiment. It also lists the path to the config files where the parameters are set, relative to the go-diot root directory.

.tendermint/config/config.toml	
pow_timeout	1537ms
pow_num_places	30
timeout_propose	2537ms

diot_config.yaml	
targetNumValidators	80

testnet/py-gen-Docker-compose/gen_Docker-compose_config.yaml	
num_nodes	100
num_entries	500
num_byzantine_nodes	10

Table 5.2.: Experiment parameters

After the execution of the testnet, the vssim tool is executed with the same parameters to provide a baseline for verifying the correct behavior of the Validator Management process. Similar results of the simulated and the real validator set indicate a correct behaving Validator Management process. The validator set changes of the simulation and the testnet are logged and processed to retrieve the charts in Figure 5.3 and Figure 5.4.

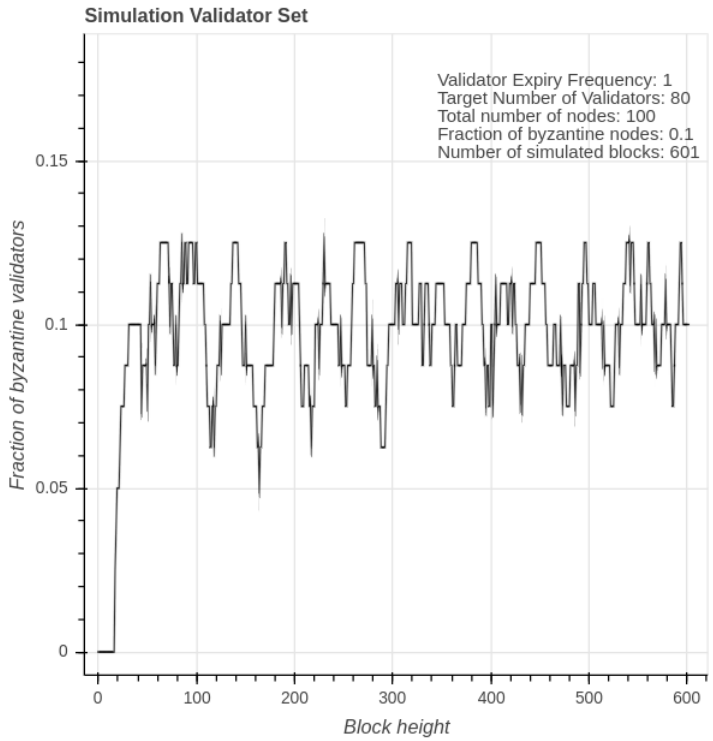


Figure 5.3.: Changes in the validator set (simulation)

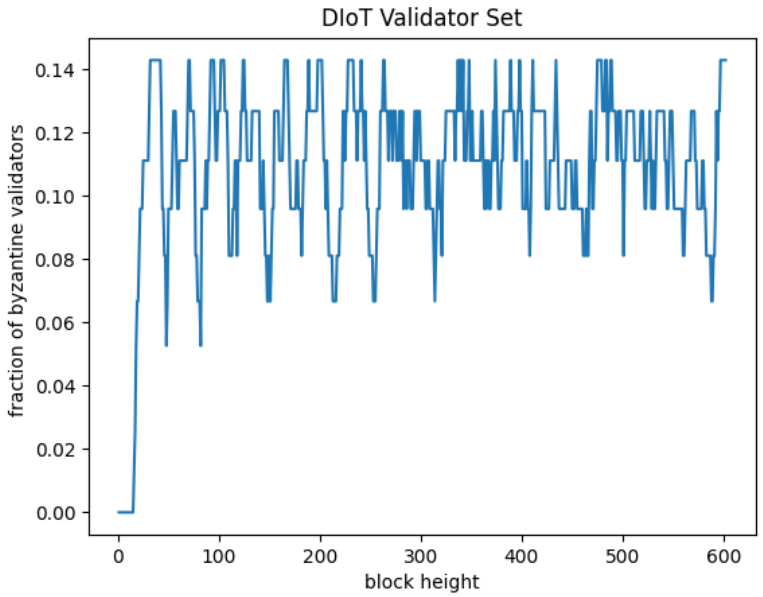


Figure 5.4.: Changes in the validator set (testnet)

It is apparent that both results share similar characteristics, such as similar extreme and mean values, but are not identical. The reason for the divergence is that the random selection of the simulation is not determined by a PoW as it is the case in DIoT Blockchains, but by the CPU clock. Nevertheless, the similarity of the results suggests that the decentralized Validator Management behaves correctly.

A further important insight from this experiment is that in both of the charts, the fraction of byzantine nodes in the validator set stays far beyond the critical 33% mark, which is the fault tolerance limitation of BFT-based Blockchains (see Paragraph *"BFT consensus"* in Subsection 2.1.4). DIoT Blockchains should therefore be able to tolerate a fraction of at least 10% byzantine nodes in the network over a long period. This aspect of the DIoT framework is further investigated in Section 5.4.

Besides the tracking of the validator set, further important aspects of the DIoT framework were captured during the experiment. Table 5.3 yields a summary of the results that are gathered from the log files of the executed testnet. The occurrences of the sequential PoWs adhere to the results of experiment 1. The class of two sequential PoWs occurred in 8.48% and the class of three sequential PoWs in only 0.14% of the possible cases. In the other cases there was only time to perform at most one PoW execution. This grants only a minimal chance of manipulating the Validator Selection through a brute-force attack. The average transaction times, as well as the extreme values show the message latency, that can be expected from the system. Arguably, the PoW timeout value in this experiment could be raised a couple hundred milliseconds (e.g. to 1800 milliseconds) to improve the transaction time of the Blockchain, since the sequential PoW occurrences are still very low. It is expected by the author, based on the

results of this experiment, that a well tuned system can reach an average transaction time of 5000 milliseconds or less. The PoW failure rate and the average PoW time per block are an important metric for the comparison of different PoW algorithms which may be implemented into the system in the future to improve the message latency and the protection against brute-force attacks. The reliable message transmission of the DIoT framework can also be confirmed in this experiment, as the expected number of transactions is delivered by the testnet. The expected and actual number of transaction are also stated in Table 5.3.

Average transaction time [ms]	6838.42
Minimum transaction time [ms]	2109
Maximum transaction time [ms]	47042
Average PoW time per block [ms]	1481.83
PoW failure rate [%]	32.28%
Sequences of 2 PoWs within a timeout [%]	8.48%
Sequences of 3 PoWs within a timeout [%]	0.14%
Sequences of 4 or more PoWs within a timeout [%]	0.00%
Expected number of transactions	45191
Actual number of transactions	45191

Table 5.3.: Results of experiment 2

In conclusion of this experiment, which conducted a test run of a DIoT Blockchain, it is shown that the Validator Management process behaves as expected and has proven to withstand a fraction of 10% byzantine network nodes. The experiment results show further, that the system has the potential to deliver transactions of network nodes within few seconds. While the DIoT Blockchain can certainly not be considered a low-latency communication system, still for several applications in IoT and other fields it provides sufficiently low latency. The results of experiment 1 regarding effective protection against brute-force attacks are supported by the results in this experiment, as the percentages of sequential PoWs are similar, and in fact, even lower than the results received in experiment 1. Similar to experiment 1, this experiment can also be reproduced to validate the presented results by setting up the testnet included in the go-diot project and editing the configuration files according to Table 5.2. The instructions for setting up the testnet are provided in the *README.md* file in the *testnet* directory of the go-diot project.

5.3.3. Experiment 3: Compatibility to IoT devices

In the final experiment, it is evaluated if the DIoT framework can run on an IoT device to verify if it is suitable for a variety of IoT use cases.

In first instance, a DIoT Blockchain was set up on three different devices; a desktop PC, notebook, and one RPi3, with the DIoT Blockchain node and client applications from the *example/basic* directory of the go-diot project installed on all devices. All features of the DIoT framework worked on the RPi3 as expected and it accomplished to solve

PoWs with difficulty 28 for block proposals within the default PoW timeout of 2000 milliseconds. However, the RPi3 performed noticeable slower than the other devices solving the PoW, which led to a further investigation of the performance of the RPi3 regarding PoW execution. The average PoW time with difficulty 30, out of 100 PoW executions, was captured on all three devices, and compared leading to the result in Table 5.4.

Device	Average time [ms]
Notebook (Intel Core i7-8565U CPU, 16GB RAM)	1258.76
PC (AMD Ryzen 5 3600 CPU, 16GB RAM)	681.96
RPi3 (ARM Cortex-A53 CPU, 1GB RAM)	9072.46

Table 5.4.: Comparison of PoW execution on different devices

The result of this experiment shows that the problem, which the approach with PoW is facing, is the performance difference of different hardware. Even though the PoW does not demand massive amount of computing power and can be accomplished with the RPi3, the difference in execution time to more capable systems, such as a decent desktop PC, is significant. An attacker who uses hardware with more computing power could increase its chance of performing a brute-force attack if the PoW timeout is designed for a network of single-board computers such as the RPi3.

This problem may be remedied in the future by the combination of increase of computing power and memory in single-board computers such as the Raspberry Pi in its fourth generation[5], and advancements in PoW algorithms. In PoW algorithm research, a trend is developing towards algorithms that are less dependent on CPU speed and more on memory access speed and a certain amount of memory, which aims to reduce the discrepancy of different hardware. This topic is further discussed in Section 2.3.

An intermediate solution could be to enforce a network of nodes with a relatively homogeneous computing power by accepting only nodes in the network, that can solve a set of PoWs in a predefined time to assure that all nodes in the network are capable of performing the PoW before the timeout. In this case only more capable devices, such as network gateways, small servers, or decent single-board computers may be able to join the Blockchain and act as proxies for more lightweight IoT nodes.

In conclusion of this last experiment, it is shown that a DIoT Blockchain node does run on single-board computers with ARM architecture and Linux operating system such as the RPi3. However, the compatibility to these devices is currently limited since they cannot keep up with more powerful nodes when calculating the PoW as shown in Table 5.4. They would require a rather low PoW difficulty or an extended timeout to calculate the PoW within the available time. More capable nodes could exploit the low PoW difficulty to increase their chance of successfully performing a brute-force attack to manipulate the Validator Management. It is, however, assumed that future advancements in PoW algorithms and improvements of mobile processors will remedy this limitation.

[5] `https://datasheets.raspberrypi.com/rpi4/raspberry-pi-4-datasheet.pdf`, accessed: 2021-10-18

5.4. DIoT fault tolerance

To conclude the evaluation, the fault tolerance of the DIoT framework is examined. Blockchains are byzantine fault tolerant systems which is one advantage that sets them apart from centralized alternatives such as client-server communication. Consensus mechanisms tolerate byzantine nodes with an accumulated voting power of up to one third or one half of the total voting power [Vuk15]. The DIoT framework makes no guarantees about the fault tolerance property, because it depends on Validator Management configuration and probability introduced by the random selection of validators in the Validator Management process. Generally, the DIoT fault tolerance is weaker compared to other Blockchains using BFT consensus with a static validator set (e.g. Cosmos, Hyperledger Fabric), because DIoT sacrifices some amount of fault tolerance in favor of a dynamic validator set and therefore a higher degree of decentralization.

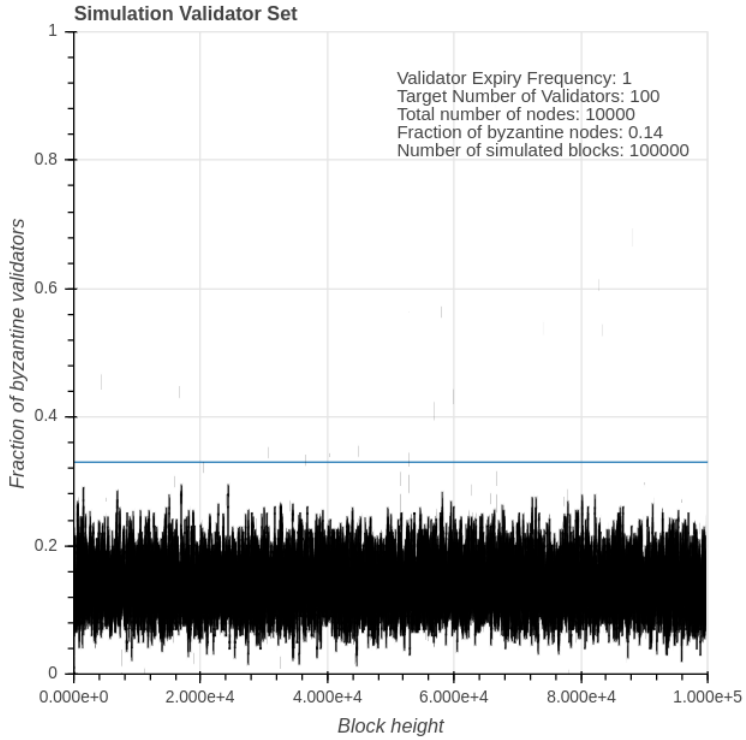


Figure 5.5.: vssim execution with 14% byzantine nodes in the Blockchain network.

In a DIoT Blockchain, byzantine nodes can reside in the validator set and the non-validator set (see Figure 3.1). In the non-validator set, byzantine nodes cannot harm the Blockchain consensus, as they cannot participate in it. Therefore, there is no primary impact on fault tolerance from byzantine nodes in the non-validator set. However, the number of byzantine nodes in the non-validator set surely has a secondary impact on fault tolerance. With an increasing number of byzantine nodes in the non-validator set, the probability of

drawing a byzantine node into the validator set is increased.

The fault tolerance in the validator set is predetermined by the Byzantine Fault Tolerance property of Tendermint consensus which is defined as $n \geq 3f + 1$ or rewritten $f \leq \frac{n-1}{3}$, with n being the number of nodes in the validator set and f being the number of byzantine nodes in the validator set [BKT18]. In other words, more than two third of the nodes in the validator set must be correct for Tendermint to reach consensus.

Byzantine nodes will try to move to the validator set in order to reach $f > \frac{n-1}{3}$ to control or stop the consensus process (see Subsection 2.1.2). Validator Management ensures that byzantine nodes cannot manipulate the selection of validators, instead the Validator Selection happens randomly. Random selection of nodes from the non-validator set implies that at some point, $f > \frac{n-1}{3}$ can be true even if $F \leq \frac{N-1}{3}$ is satisfied, N being the overall number of nodes and F being the overall number of byzantine nodes in the network. Therefore, it can be expected that the ratio $\frac{F}{N}$ must be significantly less than $\frac{1}{3}$ to sustainably maintain a dynamic validator set that satisfies $f \leq \frac{n-1}{3}$.

The *vssim* tool (see Appendix A.1) is very useful to explore the fault tolerance of DIoT Blockchains with various different parameters. Several simulation iterations with different parameters led to the conclusion that the fault tolerance mainly depends on the validator set size. If a reasonable size of the validator set is chosen, e.g. n = 100 nodes , a fault tolerance of 14% byzantine nodes can be maintained over a long term as shown in Figure 5.5 without the distribution of byzantine validators in the validator set climbing above the critical 33% mark (denoted as the horizontal line in Figure 5.5). With larger validator set size, the fault tolerance of a DIoT Blockchain will converge towards the fault tolerance value of BFT-based consensus (33%), but it is not feasible to choose a validator set size greater than couple hundred nodes as explained in Section 3.2. More results of the vssim simulation, that support this conclusion, have been added to the Appendix B.

5.5. Chapter summary

To summarize, the evaluation of the DIoT framework shows that the embedded PoW increases the difficulty to manipulate the Validator Selection. Although, if attackers have access to devices with significantly more computing power than the average nodes in the Blockchain network, they might still be able to exploit the Validator Management process for a majority of byzantine nodes in the Blockchain consensus voting. From a functional perspective, the *testnet* Blockchain shows that all messages, that were sent between the nodes, were successfully added to the Blockchain despite the presence of 10% byzantine nodes. The limit for the fault tolerance, the tolerance of byzantine nodes in the Blockchain network, was examined to be around 14% for most DIoT Blockchain configurations.

6. Discussion and conclusion

In this chapter, the results of the evaluation experiments are discussed with respect to the research questions, that were defined in Section 1.2. Further, potential future work and open issues are identified and examples for possible applications of the DIoT framework are named.

6.1. Discussion of the research questions

Results regarding Q1: How can trustworthiness of IoT services be improved with Blockchain technology? Previous works, such as [Mat16, AM16], already concluded that Blockchain technology in general can contribute to a better trustworthiness for network communication compared to centralized communication protocols. The main concern of this thesis was to find out if Blockchain technology can be integrated specifically in IoT networks for communication and if so, identify remaining challenges and potentially improve the existing solutions.

As briefly discussed in Chapter 1, there is one prevalent approach to achieve the integration of Blockchain technology in IoT networks, which is essentially the outsourcing of the Blockchain consensus to a powerful mining network and equipping IoT devices only with lightweight Blockchain clients to access the Blockchain data. The used mining network can either be a well established public Blockchain platform such as Ethereum (e.g. [BM16, CBWF17]) or a dedicated mining network operated by the stakeholders of the system ([BBG+17]).

The DIoT framework does now provide a viable alternative to the prevalent approach as a Blockchain framework that was designed with the requirements of IoT networks in mind. The lightweight Tendermint consensus mechanism is used to allow existing devices in IoT networks to run full Blockchain nodes and participate at the consensus. A lot of effort when designing the DIoT framework was spent to preserve a high degree of decentralization in the underlying Blockchain by introducing the Validator Management process, as high decentralization can leverage the trustworthiness of a Blockchain.

The results of experiment 2 in Subsection 5.3.2, and Figure 5.4 in particular, show the dynamicity in the validator set of a deployed DIoT Blockchain. Every couple blocks the Validator Management process swaps out validators based on random selection which gives every Blockchain node a fair chance to participate at the Blockchain consensus from time to time. It can be argued that the dynamic validator set gives the permissioned Tendermint consensus of the underlying Blockchain the decentralization characteristic of a permissionless consensus mechanism.

The Validator Management process was very carefully designed to not allow attackers

to manipulate the selection of new validators, as this would lead to a serious vulnerability of the underlying Blockchain. Experiment 1 in Subsection 5.3.1 shows that the chance of attackers to manipulate the outcome of the Validator Management process is reduced drastically when a PoW of moderate difficulty, i.e., a PoW that keeps a device occupied for one to two seconds, is embedded into the block proposal step of the Tendermint consensus. The Validator Management process relies on this solution to make manipulations of the validator set unlikely.

To summarize the research results regarding Q1, the proposed DIoT framework and the test deployment of a DIoT Blockchain as part of the evaluation show that the integration of Blockchain technology in IoT networks for the purpose of communication between network nodes is feasible.

Results regarding Q2: Can existing IoT infrastructure be used to deploy a Blockchain-based communication system in IoT networks? It is assumed that IoT networks include different classes of devices, from low energy devices operated by micro controllers, i.e., constrained nodes as defined in RFC 7228 [BEK14], to more capable devices running for instance ARMv7, ARMv8 or AMD64 CPUs with Linux OS.

As far as the research of existing solution in this thesis goes, no Blockchain solutions have been found that are supported in micro controller platforms such as Arduino, so micro controller devices are excluded from running a Blockchain node for now. The DIoT framework cannot improve the current situation in this regard as it is written in Go and therefore needs at least a system for which a Go compiler exists.

The class of more capable devices, such as single-board computers (e.g. RPi3) and comparable devices are supported by the DIoT framework although they perform considerably slower in calculating the required PoW for block proposals compared to more powerful devices such as PCs or servers as shown in experiment 3 in Subsection 5.3.3. As a consequence, a reduced PoW difficulty has to be used to give the Blockchain nodes enough time for the PoW calculation. An attacker using faster hardware could exploit the reduced PoW difficulty to calculate multiple PoWs before the timeout and thereby increase his chance of successfully manipulating a Validator Selection of the the Validator Management process.

At the same time it also has to be considered that there are already single-board computers available with faster CPUs that could calculate the PoW faster than the RPi3, that was used in experiment 3. To find out if a network of state-of-the-art single-board computers such as the Raspberry Pi 4 can sufficiently secure a DIoT Blockchain from attempts to manipulate the validator set of rather powerful PCs or servers, a more thorough test with a DIoT Blockchain deployed on multiple single-board computers is required. It is expected by the author that further research effort to find a more suitable PoW algorithm, that is even less dependent on the CPU power than the currently used PoW algorithm in the DIoT framework, can reduce the observed discrepancy between single-board computers and more powerful devices which could eventually lead to production-ready Blockchain-based communication solutions that can be safely operated by compact single-board computers and network gateways of an IoT network.

Results regarding Q3: Can a Blockchain-based communication system achieve a transaction latency of below ten seconds in a mid-size network of IoT nodes? Tendermint, the used consensus mechanism of the DIoT framework should be well capable of achieving this goal as experiments of the Tendermint developers show [KB16], both, in terms of network scalability, and transaction latency. However, the DIoT framework uses a modified version of Tendermint consensus which includes a PoW with every block proposal. This additional effort has negative impact on the transaction latency but the test run of experiment 2 in Subsection 5.3.2 still shows an average transaction time of 6.8 seconds. Based on this experiment result, it seems feasible also for real-world applications that the goal for transaction latency of ten seconds can be reached.

6.2. Future work and open issues

PoW algorithm The most important factors to choose the currently used PoW algorithm is its simplicity and the fact that there is already an established and tested open source implementation in Go available. The existing source code could be easily adapted with only few changes to be integrated into the DIoT framework. This allowed the development effort of the DIoT framework, which was already exceeding the expectations, to stay within reasonable boundaries. In a future research effort, a survey of existing PoW algorithms may be conducted, which can potentially reveal PoW algorithms that are more suitable than the current one, to improve some of the characteristics of the DIoT framework, such as transaction latency and PoW calculation time on slower hardware.

Dependence on central entities for identity management As discussed in Section 4.4, the protection against sybil attacks when adding new Blockchain nodes is an important aspect in the DIoT framework to keep DIoT Blockchains secure and trustworthy. The DIoT framework allows to plug in different mechanisms to verify the legitimacy of new Blockchain nodes. However the most feasible approaches still depend on central entities, such as the reliance on CA certificates in the example shown in Figure 4.3. Decentralized Blockchain-based alternatives would be desirable to resolve the last dependency on central entities in the DIoT framework but there seem to be no stable and mature options available just yet [BKUE20]. Future research may be dedicated to discover and implement a suitable decentralized mechanism for identity management of the DIoT framework.

Limitation of data transfer A further desirable feature of the DIoT framework, which has not been implemented yet, would be to allow application developers to set a limit for the number of bytes per block that a specific Blockchain node can send as transaction payload to the Blockchain. This feature could prevent attempts of byzantine nodes to flood the Blockchain with spam transactions to make the Blockchain quickly outgrow the storage capabilities of Blockchain nodes. The enforcement of the data transfer limit could be implemented for instance with a punishment of nodes that violate the limit (e.g. adding the node's public key to a black-list of nodes that may not send transactions).

Further development of the go-diot project Since the design and development effort to achieve the core aspects of the DIoT framework exceeded the expectations, the time and resources to develop the application logic was limited. There is still room for improvement regarding the following areas of the go-diot project:

- The messaging related protocol and database model needs to be revisited to improve efficiency.
- Code quality, including standardized error handling, comments and documentation, can be improved.
- The provided client API is not yet fully implemented. This involves mainly API calls to query the existing messages stored in the Sqlite database. It is assumed at the current state, that application developers write the code to query the required messages directly from the database via SQL queries.

Deployment of a DIoT Blockchain in an actual IoT network Until now, the experiments with the DIoT framework mainly took place in a simulated environment using virtualization on few physical nodes. Some important aspects, such as the network stability and network latency, can only be taken into account in a network of physical nodes with a genuine geographical distance from each other. A larger scale deployment of a DIoT Blockchain can also reveal problems, that have not been detected in simulations before. Once the DIoT framework is in a more mature state of development, testing in real-world scenario is a next important step to consider.

6.3. Possible applications

DIoT is a messaging framework based on Blockchain technology for secure, transparent and trustworthy communication. This can be especially useful for applications, where many nodes share data and all other nodes should have access to all published data without the possibility of manipulation or censorship by a single entity. DIoT supports a publish-subscribe messaging pattern which allows application developers to develop client applications that use DIoT transactions for communication with other DIoT nodes. Transactions can be packed with custom payload to enable a variety of different applications. In this section, some possible applications are briefly mentioned, where a DIoT Blockchain can improve existing solutions in terms of security, transparency and immutability and therefore leverage trustworthiness in these applications. One possible application, a decentralized IoT vulnerability detection system, was already mentioned in Section 5.1 and presented visually in Figure 5.1.

Quality of Service Error reports and user complaints regarding a service can be published over a DIoT Blockchain to to allow other users to be informed about occurring problems as well and not only the vendor. This increases transparency compared to the traditional

client-server systems, where nodes send error reports only to the vendor, who can then decide which information to reveal to the public.

Supply Chain Many supply chains are already using IoT devices to record important information along the way. Abeyratne and Monfared [AM16] suggest to store this information in a Blockchain, so different stakeholders and external auditors could have direct access to the data stored on the Blockchain instead of having to trust a specific actor in the system to store and deliver the information properly. The DIoT framework could be very useful for this type of application, as it is specifically designed to develop Blockchain-based communication systems for IoT networks.

Smart City traffic data sharing Capturing the current traffic situation around a vehicle and sharing the information among other traffic participants can greatly improve the overall load of the traffic infrastructure of a city. A P2P network is well suited for this application as the users of such a system are motivated to contribute their information if they get information from other traffic users in return. Sharing traffic data via a DIoT Blockchain makes it completely transparent and reasonable for the user, which data is shared with the network. In such a system, the DIoT Blockchain can be coordinated for instance by the smart city administration, thereby enabling sharing traffic information between the vehicles in the area. When vehicles enter the area of a smart city, they can register at a registration node operated by the smart city, which establishes the connection of a vehicle to the Blockchain. When registered successfully, a car can receive and send traffic information using the Blockchain-based system.

Manipulation prevention for sensitive data records Using a centralized system with a single entity taking care of storing sensitive data records, such as patient medical records, vehicle inspection records or public key certificates, bears the risk that the entity is compromised by an attacker and the data is manipulated, revealed or deleted. A DIoT Blockchain can present a viable alternative to centralized systems as it offers moderate transaction latency and no transaction fees, similar to a centralized solution but additionally provides benefits of Blockchain technology such as immutability, transparency and resilience against attacks.

6.4. Conclusion

This thesis attempts to contribute to the integration of Blockchain technology in IoT. For this matter a Blockchain-based communication framework, named DIoT, was developed, which uses a lightweight private and permissioned Blockchain to allow existing IoT devices to run full Blockchain nodes. The DIoT framework provides developers with the possibility to develop a publish-subscribe messaging infrastructure with a decentralized Blockchain to facilitate the communication between nodes of an IoT network as alternative to centralized communication protocols for trustworthy and transparent end-to-end communication. A unique and important feature of the DIoT framework is its Validator Management process, which maintains a continuously changing set of validator nodes to achieve a high decentralization in the private permissioned Blockchain. Not only the working solution for the Validator Management process was explained and implemented in this thesis, but also several misleading approaches that have been encountered, were documented to ease future research on this topic.

I would be pleased if other researchers find some of the insights of this thesis useful and are encouraged to elaborate further on the integration of Blockchain technology in IoT or apply the principles to other Blockchain-related research fields.

Bibliography

[ABB+15] Christopher Allen, Arthur Brock, Vitalik Buterin, Jon Callas, Duke Dorje, Christian Lundkvist, Pavel Kravchenko, Jude Nelson, Drummond Reed, Markus Sabadello, Greg Slepak, Noah Thorp, and Harlan T Wood. Decentralized Public Key Infrastructure. `https://github.com/WebOfTrustInfo/rwot1-sf/blob/master/final-documents/dpki.pdf`, 2015. Accessed: 2021-11-25.

[ABB+18] Elli Androulaki, Artem Barger, Vita Bortnikov, Christian Cachin, Konstantinos Christidis, Angelo De Caro, David Enyeart, Christopher Ferris, Gennady Laventman, and Others Manevich, Yacov. Hyperledger fabric: A Distributed Operating System for Permissioned Blockchains. In *Proceedings of the thirteenth EuroSys conference*, pages 1–15, 2018.

[AFGM+15] Ala Al-Fuqaha, Mohsen Guizani, Mehdi Mohammadi, Mohammed Aledhari, and Moussa Ayyash. Internet of things: A survey on enabling technologies, protocols, and applications. *IEEE Communications Surveys & Tutorials*, 17(4):2347–2376, 2015.

[AM16] Saveen A. Abeyratne and Radmehr P Monfared. Blockchain ready manufacturing supply chain using distributed ledger. *International Journal of Research in Engineering and Technology*, 5(9):1–10, 2016.

[Bac02] Adam Back. Hashcash - A Denial of Service Counter-Measure. `http://www.hashcash.org/papers/hashcash.pdf`, August 2002. Accessed: 2021-11-25.

[BBBR19] Andrew Banks, Ed Briggs, Ken Borgendale, and Gupta Rahul. *MQTT Version 5.0*. Organization for the Advancement of Structured Information Standards (OASIS), `https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.pdf`, March 2019. Accessed: 2021-11-25.

[BBG+17] Aymen Boudguiga, Nabil Bouzerna, Louis Granboulan, Alexis Olivereau, Flavien Quesnel, Anthony Roger, and Renaud Sirdey. Towards better availability and accountability for iot updates by means of a blockchain. In *2017 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, pages 50–58. IEEE, 2017.

[BCGH16] Richard Gendal Brown, James Carlyle, Ian Grigg, and Mike Hearn. Corda: An Introduction. `https://blockchainlab.com/pdf/corda-introductory-whitepaper-final.pdf`, August 2016. Accessed: 2021-11-25.

[BEK14] Carsten Bormann, Mehmet Ersue, and Ari Keränen. Terminology for Constrained-Node Networks. RFC 7228, May 2014.

[BG19] Vitalik Buterin and Virgil Griffith. Casper the Friendly Finality Gadget. *arXiv preprint arXiv:1710.09437v4*, 2019.

[BK17] Alex Biryukov and Dmitry Khovratovich. Equihash: Asymmetric Proof-of-Work Based on the Generalized Birthday Problem. *Ledger*, 2:1–30, 2017.

[BKT18] Ethan Buchman, Jae Kwon, and Zarko Milosevic Tendermint. The latest gossip on BFT consensus. *arXiv preprint arXiv:1807.04938v3*, September 2018.

[BKUE20] Clemens Brunner, Fabian Knirsch, Andreas Unterweger, and Dominik Engel. A Comparison of Blockchain-based PKI Implementations. In *Proceedings of the 6th International Conference on Information Systems Security and Privacy (ICISSP)*, pages 333–340, 2020.

[BM16] Arshdeep Bahga and Vijay K Madisetti. Blockchain platform for industrial internet of things. *Journal of Software Engineering and Applications*, 9(10):533–546, 2016.

[Bor14] Eleonora Borgia. The internet of things vision: Key features, applications and open issues. *Computer Communications*, 54:1–31, 2014.

[Bre00] Eric A Brewer. Towards robust distributed systems. In *Proceedings of the nineteenth annual ACM symposium on Principles of distributed computing (PODC)*, 2000.

[BS12] Nitish Balachandran and Sugata Sanyal. A review of techniques to mitigate sybil attacks. *arXiv preprint arXiv:1207.2617*, 2012.

[BSA14] Alysson Bessani, João Sousa, and Eduardo E P Alchieri. State machine replication for the masses with BFT-SMART. In *44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 355–362, 2014.

[BSP+08] Sharon Boeyen, Stefan Santesson, Tim Polk, Russ Housley, Stephen Farrell, and David Cooper. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. RFC 5280, May 2008.

[Buc16] Ethan Buchman. Tendermint: Byzantine Fault Tolerance in the Age of Blockchains. Master's thesis, University of Guelph, 2016.

[But13] Vitalik Buterin. A Next Generation Smart Contract & Decentralized Application Platform. `https://ethereum.org/en/whitepaper/`, 2013. Accessed: 2021-11-25.

[But15] Vitalik Buterin. On Public and Private Blockchains. `https://blog.ethereum.org/2015/08/07/on-public-and-private-blockchains/`, 2015. Accessed: 2021-11-25.

[CBWF17] Mathieu Chanson, Andreas Bogner, Felix Wortmann, and Elgar Fleisch. Blockchain as a privacy enabler: An odometer fraud prevention system. In *Proceedings of the 2017 ACM International Joint Conference on Pervasive and Ubiquitous Computing (UbiComp '17)*, pages 13–16, 2017.

[CL13] Liqun Chen and Jiangtao Li. Flexible and scalable digital signatures in TPM 2.0. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 37–48, 2013.

[CLO99] Miguel Castro, Barbara Liskov, and Others. Practical Byzantine Fault Tolerance. In *Proceedings of the Third Symposium on Operating Systems Design and Implementation*, pages 173–186, 1999.

[CPV16] Michael Crosby, Pradan Pattanayak, and Sanjeev Verma. BlockChain Technology: Beyond Bitcoin. *Applied Innovation Review*, 2/2016, June 2016.

[CV17] Christian Cachin and Marko Vukolić. Blockchain consensus protocols in the wild. *arXiv preprint arXiv:1707.01873*, 2017.

[CXS+17] Lin Chen, Lei Xu, Nolan Shah, Zhimin Gao, Yang Lu, and Weidong Shi. On Security Analysis of Proof-of-Elapsed-Time (PoET). In *19th International Symposium on Stabilization, Safety, and Security of Distributed Systems*, pages 282–297, 2017.

[DAAB+18] Stefano De Angelis, Leonardo Aniello, Roberto Baldoni, Federico Lombardi, Andrea Margheri, and Vladimiro Sassone. PBFT vs Proof-of-Authority: Applying the CAP Theorem to Permissioned Blockchain. In *2nd Italian Conference on Cybersecurity (ITASEC)*, 2018.

[Dav19] Silas Davis. Burrow - The Boring Blockchain. `https://www.hyperledger.org/blog/2019/10/08/burrow-the-boring-blockchain`, October 2019. Accessed: 2021-08-29.

[DLS88] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Consensus in the presence of partial synchrony. *Journal of the ACM (JACM)*, 35(2):288–323, 1988.

[DN92] Cynthia Dwork and Moni Naor. Pricing via Processing or Combatting Junk Mail. In *12th Annual International Cryptology Conference*, pages 139–147, 1992.

[Dou02] John R. Douceur. The Sybil Attack. In *First International Workshop on Peer-to-Peer Systems (IPTPS)*, pages 251–260, 2002.

[DTH06] Rachna Dhamija, J Doug Tygar, and Marti Hearst. Why phishing works. In *Proceedings of the SIGCHI conference on Human Factors in computing systems*, pages 581–590, 2006.

[Elo01] Greg Elofson. Developing Trust with Intelligent Agents: An Exploratory Study. In *Trust and deception in virtual societies*, chapter 6, pages 125–138. Springer, Dordrecht, 2001.

[FBVI17] Syed Naeem Firdous, Zubair Baig, Craig Valli, and Ahmed Ibrahim. Modelling and Evaluation of Malicious Attacks against the IoT MQTT Protocol. In *2017 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, pages 748–755, 2017.

[FCFL18] Tiago M Fernández-Caramés and Paula Fraga-Lamas. A review on the use of blockchain for the internet of things. *IEEE Access*, 6:32979–33001, 2018.

[Fie00] Roy Thomas Fielding. *Architectural styles and the design of network-based software architectures*. PhD thesis, University of California, 2000.

[FLP85] Michael J Fischer, Nancy A Lynch, and Michael S Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM (JACM)*, 32(2):374–382, 1985.

[FVY14] Conner Fromknecht, Dragos Velicanu, and Sophia Yakoubov. Certcoin: A namecoin based decentralized authentication system. Technical report, Massachusetts Institute of Technology, 2014. `https://courses.csail.mit.edu/6.857/2014/files/19-fromknecht-velicann-yakoubov-certcoin.pdf`.

[GHY18] Weichao Gao, William G Hatcher, and Wei Yu. A survey of blockchain: Techniques, applications, and challenges. In *2018 27th international conference on computer communication and networks (ICCCN)*, pages 1–11. IEEE, 2018.

[Hoo18] Parikshit Hooda. Comparison – Centralized, Decentralized and Distributed Systems. `https://www.geeksforgeeks.org/comparison-centralized-decentralized-and-distributed-systems/`, 2018. Accessed: 2020-08-03.

[JJ99] Markus Jakobsson and Ari Juels. Proofs of work and bread pudding protocols. In *Proceedings of the IFIP TC6/TC11 Joint Working Conference on Secure Information Networks: Communications and Multimedia Security (CMS '99)*, pages 258–272, 1999.

[JSO13] JSON-RPC Working Group. JSON-RPC 2.0 Specification. `https://www.jsonrpc.org/specification`, January 2013.

[KB16] Jae Kwon and Ethan Buchman. Cosmos: A network of distributed ledgers. `https://cosmos.network/whitepaper`, 2016. Accessed: 2021-11-25.

[KKSV17] Constantinos Kolias, Georgios Kambourakis, Angelos Stavrou, and Jeffrey Voas. DDoS in the IoT: Mirai and other botnets. *IEEE Computer*, 50(7):80–84, 2017.

[KN12] Sunny King and Scott Nadal. PPCoin: Peer-to-Peer Crypto-Currency with Proof-of-Stake. `https://www.peercoin.net/whitepapers/peercoin-paper.pdf`, 2012. Accessed: 2021-11-25.

[KR17] James F. Kurose and Keith W. Ross. *Computer Networking: A Top-Down Approach, 7th edition.* Paerson, 2017.

[KS18] Minhaj Ahmad Khan and Khaled Salah. IoT security: Review, blockchain solutions, and open challenges. *Future Generation Computer Systems*, 82:395–411, 2018.

[Lar13] Daniel Larimer. Momentum–A Memory-Hard Proof-of-Work via Finding Birthday Collisions. `http://www.hashcash.org/papers/momentum.pdf`, October 2013. Accessed: 2021-11-25.

[LCMR16] Benjamin Leiding, Clemens H Cap, Thomas Mundt, and Samaneh Rashidibajgan. Authcoin: validation and authentication in decentralized networks. *arXiv preprint arXiv:1609.04955*, 2016.

[LRST00] Felix Lau, Stuart H Rubin, Michael H Smith, and Ljiljana Trajkovic. Distributed denial of service attacks. In *SMC 2000 conference proceedings. 2000 IEEE International Conference on Systems, Man Cyberbetics - "Cyberbetics Evolving to Systems, Humans, Organizations, and their Complex Interactions"*, volume 3, pages 2275–2280, 2000.

[LSP82] Leslie Lamport, Robert Shostak, and Marshall Pease. The Byzantine Generals Problem. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 4(3), 1982.

[Mat16] Juri Mattila. The blockchain phenomenon–the disruptive potential of distributed consensus architectures. Technical report, The Berkeley Roundtable on the International Economy (BRIE), 2016.

[MHWK16] Mitar Milutinovic, Warren He, Howard Wu, and Maxinder Kanwal. Proof of luck: An efficient blockchain consensus protocol. In *Proceedings of the 1st Workshop on System Software for Trusted Execution (SysTEX '16)*, 2016.

[Nai17] Nitin Naik. Choice of effective messaging protocols for IoT systems: MQTT, CoAP, AMQP and HTTP. In *2017 IEEE international systems engineering symposium (ISSE)*, pages 1–7. IEEE, 2017.

[Nak08] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. `https://bitcoin.org/bitcoin.pdf`, October 2008. Accessed: 2021-11-25.

[Nie93] Jakob Nielsen. Response Times: The 3 Important Limits. `https://www.nngroup.com/articles/response-times-3-important-limits/`, January 1993. Accessed: 2021-08-26.

[OM14] Karl J. O'Dwyer and David Malone. Bitcoin mining and its energy footprint. In *25th IET Irish Signals Systems Conference 2014 and 2014 China-Ireland International Conference on Information and Communications Technologies (ISSC 2014/CIICT 2014)*, pages 280–285, 2014.

[PAV+12] Maria Rita Palattella, Nicola Accettura, Xavier Vilajosana, Thomas Watteyne, Luigi Alfredo Grieco, Gennaro Boggia, and Mischa Dohler. Standardized protocol stack for the internet of (important) things. *IEEE communications surveys & tutorials*, 15(3):1389–1406, 2012.

[Per09] Colin Percival. Stronger key derivation via sequential memory-hard functions. In *BSDCan*, 2009.

[Pos80] Jon Postel. User Datagram Protocol. RFC 768, August 1980.

[Pos81] Jon Postel. Transmission Control Protocol. RFC 793, September 1981.

[Res00] Eric Rescorla. HTTP Over TLS. RFC 2818, May 2000.

[Res18] Eric Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446, August 2018.

[RM12] Eric Rescorla and Nagendra Modadugu. Datagram Transport Layer Security Version 1.2. RFC 6347, January 2012.

[RMC+18] Ana Reyna, Cristian Martín, Jaime Chen, Enrique Soler, and Manuel Díaz. On blockchain and its integration with IoT. Challenges and opportunities. *Future generation computer systems*, 88:173–190, 2018.

[Sal20] Fahad Saleh. Blockchain Without Waste: Proof-of-Stake. *The Review of Financial Studies*, 34(3):1156–1190, 2020.

[SHB14] Zach Shelby, Klaus Hartke, and Carsten Bormann. The Constrained Application Protocol (CoAP). RFC 7252, June 2014.

[SM12] Audie Sumaray and S Kami Makki. A comparison of data serialization formats for optimal efficiency on a mobile platform. In *Proceedings of the 6th international conference on ubiquitous information management and communication(ICUIMC)*, pages 1–6, 2012.

[SM17] Shigeya Suzuki and Jun Murai. Blockchain as an audit-able communication channel. In *Proceedings of the 41st Annual Computer Software and Applications Conference (COMPSAC)*, volume 2, pages 516–522, 2017.

[SZ15] Yonatan Sompolinsky and Aviv Zohar. Secure High-Rate Transaction Processing in Bitcoin. In *Financial Cryptography and Data Security*, pages 507–527, 2015.

[SZE+18] Tara Salman, Maede Zolanvari, Aiman Erbad, Raj Jain, and Mohammed Samaka. Security services using blockchains: A state of the art survey. *IEEE Communications Surveys & Tutorials*, 21(1):858–880, 2018.

[Szi17] Peter Szilágyi. Clique proof-of-authority consensus. EIP 225, Ethereum Improvement Proposals, March 2017.

[Tena] Tendermint Documentation: Attack Scenarios. `https://docs.tendermint.com/master/spec/light-client/accountability/#problem-statement`. Accessed: 2021-10-15.

[Tenb] Tendermint Documentation: Determinism. `https://docs.tendermint.com/master/spec/abci/abci.html#determinism`. Accessed: 2021-10-15.

[Tenc] Tendermint Documentation: Consensus Overview. `https://docs.tendermint.com/master/introduction/what-is-tendermint.html#consensus-overview`. Accessed: 2021-10-15.

[Vuk15] Marko Vukolic. The Quest for Scalable Blockchain Fabric: Proof-of-Work vs. BFT Replication. In *Proceedings of the International Workshop on Open Problems in Network Security (iNetSec)*, pages 112–125, 2015.

[WZN+19] Xu Wang, Xuan Zha, Wei Ni, Ren Ping Liu, Y Jay Guo, Xinxin Niu, and Kangfeng Zheng. Survey on blockchain for internet of things. *Computer Communications*, 136:10–29, 2019.

[YFN+19] Ashkan Yousefpour, Caleb Fung, Tam Nguyen, Krishna Kadiyala, Fatemeh Jalali, Amirreza Niakanlahiji, Jian Kong, and Jason P Jue. All one needs to know about fog computing and related edge computing paradigms: A complete survey. *Journal of Systems Architecture*, 2019.

[ZXD+18] Zibin Zheng, Shaoan Xie, Hong-Ning Dai, Xiangping Chen, and Huaimin Wang. Blockchain challenges and opportunities: A survey. *International Journal of Web and Grid Services*, 14(4):352–375, 2018.

Acronyms

6LoWPAN IPv6 over Low power Wireless Personal Area Network. 16

API Application Programming Interface. 24, 48–50, 78

BFT Byzantine Fault Tolerance. 6–10, 12–14, 23–25, 38, 39, 70, 73, 74

CA Certificate Authority. 16–18, 52, 53

CoAP Constrained Application Protocol. 17, 20

CPU Central Processing Unit. 8, 10, 11, 70, 76

DDoS Distributed Denial of Service. 8, 61

DIoT Decentralized IoT Framework. iii, v

DTLS Datagram TLS. 17

ER Entity-Relationship. xiii, 49, 50

GHOST Greedy Heaviest Observed Subtree. 14, 96

GPU Graphics Processing Unit. 19

GST Global Stabilization Time. 12, 13

HTTP Hypertext Transfer Protocol. 17, 48, 49

HTTPS Hypertext Transfer Protocol Secure. 17

IoT Internet of Things. iii, v, 1–6, 15–18, 20–23, 25, 34, 47, 49, 50, 60–64, 71, 72, 75–80

M2M Machine-to-Machine. 15, 17

MQTT Message Queuing Telemetry Transport. 17–20, 25

OSI Open Systems Interconnection. 16

P2P Peer-to-Peer. 10, 24, 47, 79

PBFT Practical Byzantine Fault Tolerance. 12

PC Personal Computer. 6, 47, 61, 68, 71, 72, 76

PKI Public Key Infrastructure. iii, v, 1, 3, 17, 53

PoA Proof of Authority. 6, 7, 9, 10, 14, 23–25, 38

PoS Proof of Stake. 6–12, 23, 38

PoW Proof of Work. xv, 5–11, 13, 14, 19, 20, 23, 31, 41–46, 48, 55–57, 59, 60, 64–68, 70–72, 74, 76, 77, 91, 92, 95, 96

PRNG Pseudo Random Number Generator. 29–31, 46

RFID Radio Frequency Identification. 18

RPi3 Raspberry Pi 3. 34, 47, 49, 61, 64, 71, 72, 76

RPL Routing Protocol for Low power and Lossy Networks. 16

RWOT Rebooting the Web of Trust. 1

SQL Structured Query Language. 55, 78

TCP Transmission Control Protocol. 17

TEE Trusted Execution Environment. 34

TLS Transport Layer Security. 17, 55

TPM Trusted Platform Module. 31, 34–37, 40, 53

UDP User Datagram Protocol. 17

UML Unified Modeling Language. xiii, 48

Appendices

A. DIoT simulation tools

The go-diot project includes, besides the DIoT system implementation and the testnet, two simulation tools, that are important for the evaluation process of the DIoT system as well as for application developers using the framework.

A.1. Validator set simulation tool (vssim)

The validator set simulation tool, or *vssim*, was developed as a python script at the beginning of solution design, where only the idea of a private permissioned Blockchain with a dynamic validator set existed. It mocks the Validator Management of DIoT very simplified in a centralized way to see how the dynamic validator set would impact on the fault tolerance of the Blockchain. The vssim tool is implemented as interactive web-application where users can modify parameters via a GUI and run several simulations. It was developed in this way to help application developers to fine tune the parameters of a DIoT Blockchain according to the application requirements, before the Blockchain is launched.

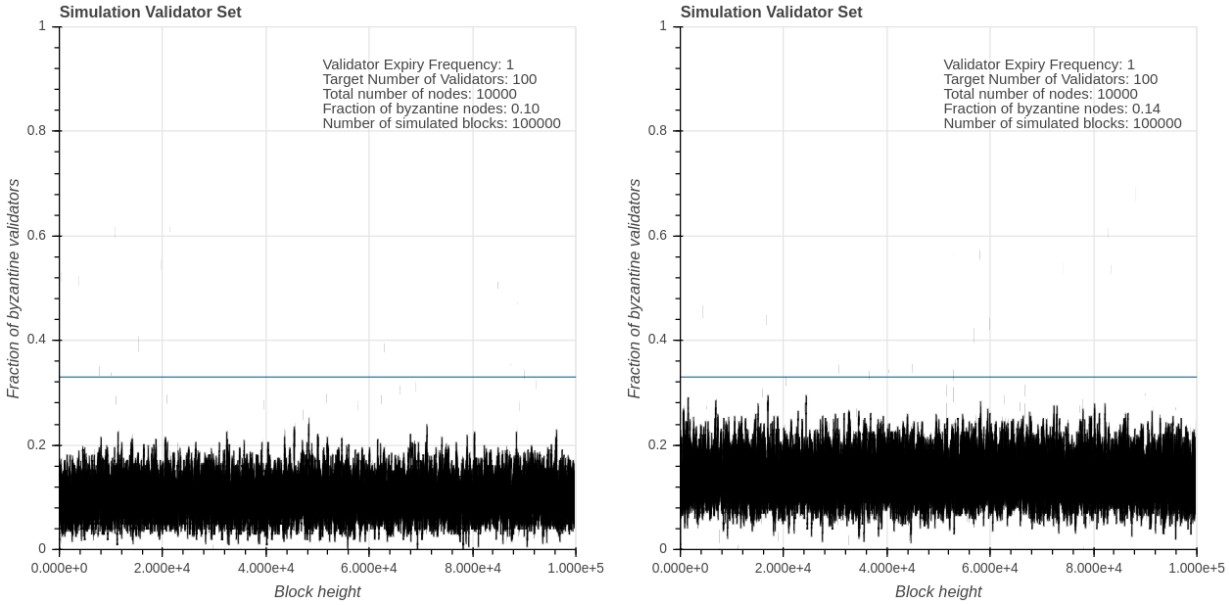
In the evaluation phase, vssim is used as baseline to verify the functional correctness of the decentralized Validator Management. It is examined in Subsection 5.3.2, if the final version of the decentralized Validator Management performs comparable to the vssim tool in terms of selecting validators randomly. Identical output of simulation tool and the DIoT Validator Management may not be expected, as the simulation tool receives its source of randomness from the CPU clock while the Validator Management uses a PoW for this purpose. Nevertheless, similarities between the output of simulation tool and Validator Management should be observable. The vssim tool can be found in the go-diot project under the *simulation* directory. The included *README.md* file in the *simulation* directory shows how to set it up.

A.2. PoW simulation tool (powsim)

The PoW simulation tool, or *powsim*, was developed specifically for the experiment conducted in Subsection 5.3.1 to adjust the PoW task, in particular the PoW timeout, to find a good trade-off between transaction latency and protection against the brute-force attack that was explained in paragraph *"Brute-force attack to manipulate Validator Selection"* of Subsection 3.5.4. The powsim tool is written in Go to directly access the PoW implementation of the DIoT system. The main functionality of the tool is to simulate the scenario of a byzantine node trying to complete as many PoW executions as possible before the PoW timeout, which is necessary to perform a brute-force attack to manipulate Validator Selection (see Subsection 3.5.6). This scenario is repeated several times with different PoW timeout values to see how many PoW tasks an attacker can complete at a particular PoW timeout and for an estimated transaction latency. An additional python script is included to render a diagram of the simulation result as shown in Figure 5.2. Powsim does not have an interactive GUI, as it was primarily developed for this master thesis, but it is still valuable for application developers, to find an appropriate PoW timeout at a given difficulty as it is demonstrated in the experiment of Subsection 5.3.1. However, it is important that powsim is executed on a hardware that is representative in terms of computing power for the devices that will later form the network to find a well suited PoW timeout value for a particular DIoT Blockchain. The powsim tool can be found in the go-diot project under the *powsim* directory. The included *README.md* file in the *powsim* directory shows how to set it up.

B. DIoT fault tolerance experiment

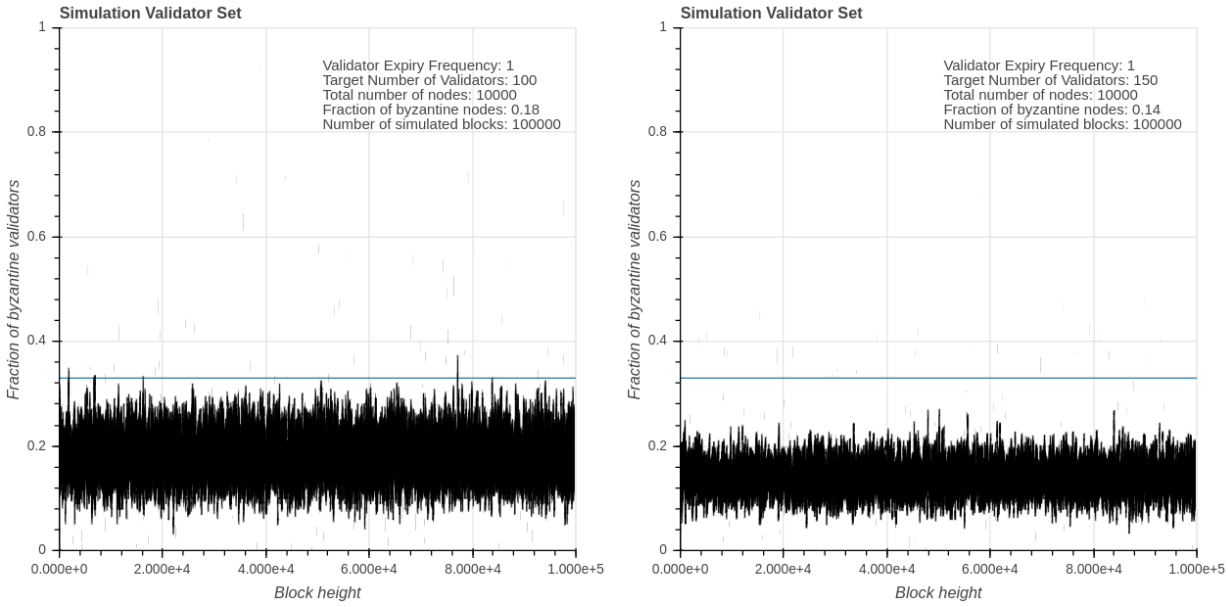
Below, several executions of the vssim tool are listed to show the distribution of byzantine nodes in the validator set over a period of one million blocks (roughly corresponds to >100 days of continuous operation of a Blockchain) with different fractions of byzantine nodes in the Blockchain network. If the fraction of byzantine nodes in the validator set exceeds 33%, termination of the consensus process can no longer be guaranteed, which means that the byzantine nodes can prevent the Blockchain from creating further blocks. Therefore, the fraction of byzantine nodes in the validator set must stay beyond that critical value. The results of the vssim executions show that with a validator set of 100 nodes, a fault tolerance of 14% can be confidently maintained, meaning that with this fraction of byzantine nodes in the Blockchain network, the fraction of byzantine nodes in the validator set does not reach the critical 33% value. With a larger validator set of 150 validators, an even higher fault tolerance of up to 18% byzantine nodes can be managed by the Blockchain.



(a) Byzantine nodes: 10%; Validator set: 100 nodes (b) Byzantine nodes: 14%; Validator set: 100 nodes

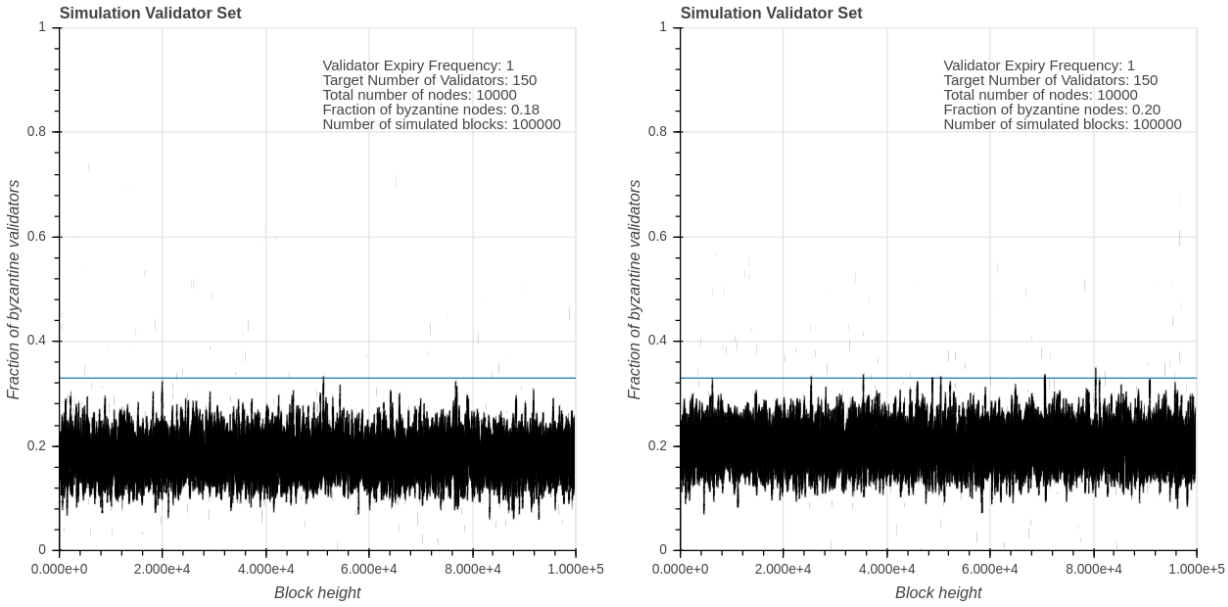
Figure B.1.: vssim executions 1/3

B. DIoT fault tolerance experiment



(a) Byzantine nodes: 18%; Validator set: 100 nodes (b) Byzantine nodes: 14%; Validator set: 150 nodes

Figure B.2.: vssim executions 2/3



(a) Byzantine nodes: 18%; Validator set: 150 nodes (b) Byzantine nodes: 20%; Validator set: 150 nodes

Figure B.3.: vssim executions 3/3

C. Blockchain consensus addendum

C.1. Double-spending attack

The double-spending attack is infamous in crypto-currency Blockchains. It exploits the longest-chain rule, which is used in PoW consensus and its derivatives (e.g. PoS, PoA). Essentially, the longest-chain rule expresses that a honest node will decide for the longest version of the Blockchain when multiple valid alternative variants of the Blockchain are propagated through the network [Nak08]. An attacker can perform a double-spending attack only if he is able to create valid blocks faster than the rest of the network, which is the case if he possesses more than 50% of the mining power, hence, it is often referred to as 51% attack.

The attack is launched by transferring crypto tokens to the victim to pay for a service and the same amount to a separate crypto wallet in a secret fork of the Blockchain leaving the balance of the attacker's wallet at zero. Once the service is consumed, the attacker mines several Blocks to the secret Blockchain fork until this version of the Blockchain contains the most blocks, and then reveal it to the public. Due to the longest-chain rule, other correct Blockchain nodes will accept this manipulated version of the Blockchain that does not contain the transaction from the attacker to the victim. Even though the transaction to the victim is still known by other miners, it is considered invalid as the attacker's wallet does not have sufficient balance to execute the transaction in the manipulated version of the Blockchain.

Nakamoto [Nak08] states that although, this attack is theoretically possible, it is very expensive for an attacker to gain control over so many resources to mine several continuous blocks faster than all the correct nodes of the network. In many cases it would cost more to launch the attack than the possible reward for the attacker. This holds for many crypto-currencies as they are backed by a large amount of mining power but if a Blockchain is used in a smaller network with less mining power, attackers would also have to use less computing resources to successfully launch a double-spending attack.

C.2. Improvement of early PoW consensus

Sompolinsky et al.[SZ15] proposed GHOST to improve latency and throughput of early PoW Blockchain implementations such as Bitcoin. GHOST is an improvement to the longest-chain rule of Bitcoin. In Bitcoin, the PoW task has a particular difficulty to achieve a block creation time of about 10 minutes. This is done to have forks occur very rarely. The highest value of orphaned blocks per year in Bitcoin was in 2015, counting 7 orphaned blocks[1]. If forks would occur more often, the honest nodes would effectively loose hashpower because many miners would mine a branch that is later orphaned. This would make it easier for an attacker to mine the longest chain, as it would require less hashpower to do so. GHOST allows for more frequent forks because it also takes orphaned blocks into account when calculating the longest chain. If a block references a known orphan block in its header, its weight increases. This reduces the efficiency loss when honest nodes mine blocks that are later orphaned. Therefore the block creation time can be drastically reduced to a few seconds. Ethereum uses a simplified version of GHOST [But13] and has a block currently creation time of about 14 seconds[2].

[1] https://www.blockchain.com/en/charts/n-orphaned-blocks, accessed: 2020-08-18
[2] https://etherscan.io/chart/blocktime, accessed: 2020-08-18