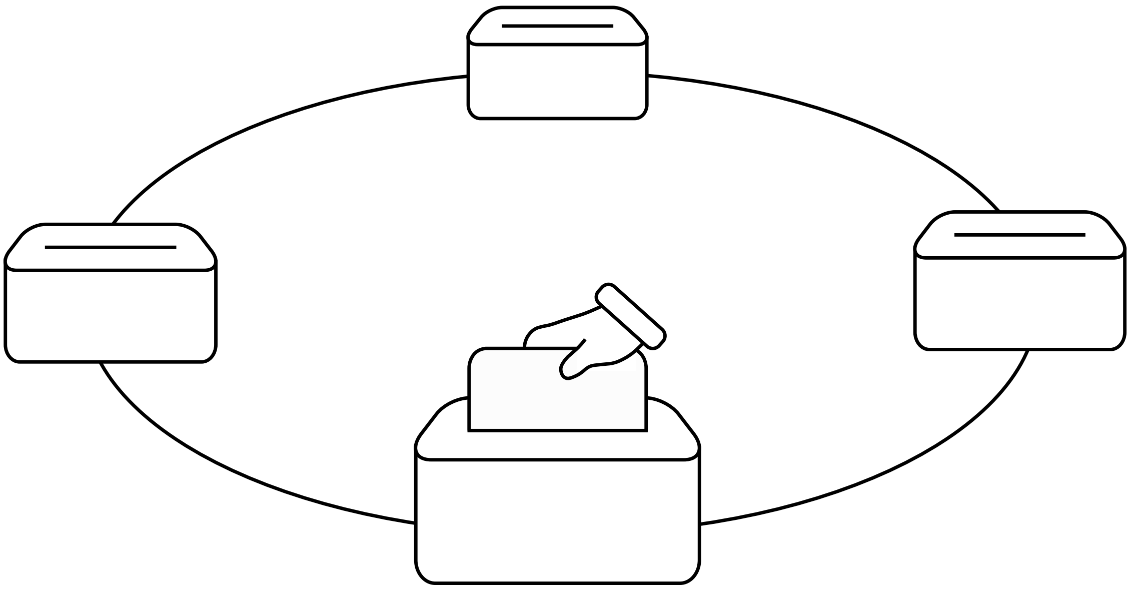
CHALMERS
UNIVERSITY OF TECHNOLOGY



A Decentralized Voting System

Bachelor Thesis
Computer Science and Engineering

Fahmi Abdulqadir Ahmed
Gustaf Engström
Omar Hindawi
Tom Tobiasson
Melker Veltman
Oskar Wallgren

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2020

Bachelor thesis 2020

A Decentralized Voting System

Fahmi Abdulqadir Ahmed
Gustaf Engström
Omar Hindawi
Tom Tobiasson
Melker Veltman
Oskar Wallgren

Department of Computer Science and Engineering
Chalmers University of Technology
Gothenburg, Sweden 2020

A Decentralized Voting System
Fahmi Abdulqadir Ahmed
Gustaf Engström
Omar Hindawi
Tom Tobiasson
Melker Veltman
Oskar Wallgren

Supervisor: Carlo Brunetta, Department of Computer Science and Engineering

Bachelor Thesis 2020
Department of Computer Science and Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Gothenburg, Sweden 2020

A Decentralized Voting System
Fahmi Abdulqadir Ahmed
Gustaf Engström
Omar Hindawi
Tom Tobiasson
Melker Veltman
Oskar Wallgren
Department of Computer Science and Engineering
Chalmers University of Technology

Abstract

The target of the project *"A Decentralized Voting System"* is to investigate whether recent developments in distributed systems can make a decentralized voting system capable of replacing the current systems in place. The current voting systems that are taken into account are primarily democratic electoral voting systems, such as the Swedish national electoral voting system. To remove the single central authority conducting the vote, distributing the responsibility is the primary goal of the project. The voting system developed relies on several trusted entities where the system tolerates conflict of interest to a specified degree. It is built using a software framework for building blockchain applications. The system is capable of managing all aspects of voting except the distribution of a digital voter card. A frontend presents the voting stages of the application but does not offer an interface for election administrators. The system is verified by several criteria that apply to any voting system, but it is not capable of replacing a large scale election in its current state.

Ethical aspects of digitizing and decentralizing are discussed, with particular consideration to societal and environmental impact. Voter turnout change with respect to age, privacy concerns and logistical solutions for the transition to a digitized voting system are also discussed.

Finally, a theoretical replacement of the current Swedish electoral system with a decentralized voting system could initially exclude a social group but would reduce cost and environmental impact.

Keywords: Decentralized, Voting, Election, Blockchain, Hyperledger, Sawtooth, Consensus, Transaction Processor, Distributed Systems

Sammandrag

Syftet med projektet *"A Decentralized Voting System"* är att undersöka om det gångna decenniets utveckling av distribuerade system kan möjliggöra ersättandet av nuvarande röstsystem med ett decentraliserat sådant. De nuvarande röstsystem projektet kommer jämföras med är huvudsakligen demokratiska, allmänna val, som till exempel det svenska nationella röstsystemet. Projektets främsta mål är att bygga ett system som kan förflytta makten över röstningsprocessen från en central auktoritet ut till flera betrodda aktörer från vilka systemet tolererar intressekonflikter till en viss grad. Systemet är byggt med ett mjukvaruramverk som används till konstruktion av blockkedjebaserade applikationer. Systemet kan hantera alla aspekter av röstningsprocessen, med undantag för distribuering av digitala röstkort. Ett grafiskt gränssnitt hanterar alla steg i röstningsprocessen, men systemet saknar ett gränssnitt för valadministration. Systemet har verifierats enligt generella kriterier som krävs för alla röstsystem, för vilka systemet uppfyller samtliga. I dess nuvarande tillstånd är det däremot inte kapabelt att ersätta befintliga röstsystem.

Vidare diskuteras vilka etiska frågeställningar digitalisering och decentralisering medför med avseende på samhälleliga och miljömässiga aspekter. Valdeltagandet med hänsyn till ålder och demografi, integritetsfrågor och logistiska lösningar för övergången till ett digitalt röstsystem diskuteras också.

Slutligen dras slutsatsen att ett teoretiskt ersättande av Sveriges nationella valsystem med ett decentraliserat röstsystem inledningsvis skulle kunna exkludera vissa samhällsgrupper, men också reducera kostnader och miljöpåverkan.

Acknowledgements

The completion of this thesis would not have been possible without the guidance, assistance and active support of our supervisor Carlo Brunetta and examiner Arne Linde. Their contributions are sincerely acknowledged and highly appreciated.

Contents

List of Figures

1 Introduction

General elections, referendums and annual general meetings are three examples of election settings where a given set of votes is summarized into a decision or advice. Today, it is common that voting during these meetings is arranged by a central authority, such as a company or a government. These types of elections can at times seem dated. To use pen and paper to cast votes at geographically bound polling stations gives elections shortcomings such as exposure to human errors. Another aspect of physical polling stations is the danger of attacks on locations in countries with heavily polarized elections [1]. Furthermore, the Swedish election authority calculated that the total cost of administration and ballot printing surpassed 348 million SEK for the 2018 election [2]. The Swedish government has previously recommended that future elections might be done over the internet, to make voting easier and lower administrative costs [3].

Electronic voting systems have been tried as a means to thwart these shortcomings in other countries (most notably in Estonia), and successfully so, but have also introduced shortcomings of their own such as an administrative centralization wherein one organizing authority with the proper access can change the outcome of an entire election [4].

1.1 Objective

The main purpose of this project is to develop a decentralized voting system using distributed ledger technologies, *i.e* frameworks that use geographically spread out records to store information. The system will be constructed with a focus on security and robustness to ensure reliability. Additionally, the system should maintain transparency and anonymity. Lastly, the system should, ideally, offer an alternative to conventional voting systems.

The focus of the research is on how to use an already existing blockchain platform and adapt it to the purpose of democratic voting. Upon this blockchain platform, a voting application will be built featuring enforced logic by leveraging transaction processors. Furthermore, the applicability of the system to elections on a national level will be evaluated, and future improvements will be proposed. Finally, the user experience in the aspect of usability across the complete spectrum of end-users will be examined.

1.2 Scope

The project will mainly be limited by two sets of factors: those sprung from the limitations of blockchain technology, and those that follow the nature of voting and trust.

Blockchain Limitations

This project will not delve into the realm of blockchain construction or, in any way, implement a blockchain. It will instead build upon already existing technologies, and with that, adopt its strengths and liabilities. The project does not aim towards building a fully operational, ready-to-use system, but will still implement a functional user interface that serves the purpose of demonstrating the system as a proof of concept.

On the subject of trust, the main delimitation will be the fact that the chosen blockchain technology will not allow the voting process to be fully decentralized. A system such as bitcoin in which any party is free to join the mining process is more decentralized, but also more susceptible to attacks as described in Section 4.2. By leveraging a blockchain platform and implementing a consensus algorithm, the system will allow the trust to be decentralized to a predetermined set of validators such as political parties or shareholders. Thus, rendering the system *decentralized*, although not at the scale of traditional blockchains such as Bitcoin.

Administrative Limitations

The voting system will be specifically tailored to meet the requirements of the voting scenarios in which each entity eligible to vote has a single vote that can only be used *once* and has the same *value* as all the other votes. Much similar to a political electoral voting system or any other democratic process. The developed product will not target a specific country or organization.

Even though the system itself will be implemented in such a way that it is both *anonymous* and *transparent*, no consideration will be taken towards the potential real-world administration complications that, for example, an election or a shareholder meeting might entail. This might include tasks such as distributing private keys in an anonymous manner, whilst still guaranteeing that each eligible voter gets only one. The system in place for key management will also have to ensure that no two voters get the same key and that no key can be connected to the identity of a voter.

2

Technical Concepts

This chapter introduces several cryptographic concepts that serve as background for the rest of the thesis. At the end of the chapter, a simple overview of the blockchain concept is explained.

2.1 Cryptographic Hash Functions

Since one can represent all types of data as an array of bytes, it is possible to define a mathematical function H operating on an any kind of data in the form an arbitrarily sized list of bytes, returning a fixed-size list of bytes. This result is called the *hash* or *message digest* of the input data. Ideally, the hash function should make it computationally difficult to derive the data from the message digest. It should also be computationally difficult to derive another value that relates to the same hash [5]. These requirements make hash functions a viable option to verify the integrity of data. These integrity verifications are of great importance to blockchain technology to confirm that data has not been changed.

2.2 Asymmetric Cryptography

Asymmetric cryptography is based around two keys, *i.e* a secret key sk_i, and a public key pk_i for each person i, *e.g* sk_{Alice}, pk_{Alice}. These keys are generated using a generating function $G \rightarrow (sk, pk)$. With these keys one can define the functions $Enc(pk_i, m) \rightarrow c$ and $Dec(sk_i, c) \rightarrow m$, where m is a plain-text message and c is that same message encrypted as ciphertext. Given the assumption that the functions conform to $m = Dec(sk_i, Enc(pk_i, m))$, the algorithm is deemed to be *correct*. These conditions give the algorithm the possibility to send encrypted messages to one specific person without another person being able to eavesdrop the information, as can be seen in Figure 2.1. Algorithms that implement these functions should make it computationally infeasible to derive m from c without the secret key sk_i and to derive the secret key sk_i from the public key pk_i to be deemed *secure*.

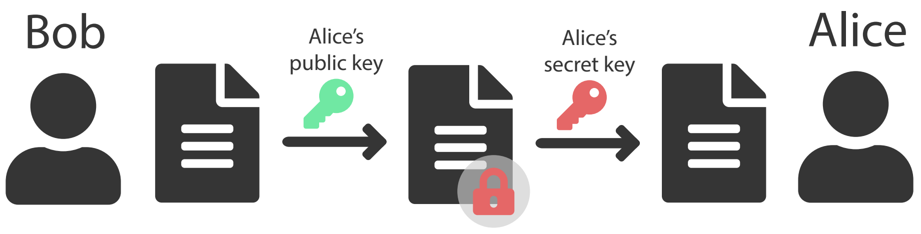


Figure 2.1: Bob uses Alice's public key to encrypt a message. Alice uses her secret key to decrypt the message.

Now assume that the algorithm conforms to a condition that any of the keys may be used as secret and the other as public, the functions may be defined as $Enc(k_i, m) \rightarrow c, Dec(\hat{k}_i, c) \rightarrow m$, where k_i is one of the keys and $\hat{k}_i$ is the other. This gives the algorithm the possibility to digitally prove that a specific person has produced or verified the data, providing a *digital signature* together with the data. This works by signing or encrypting the data with the secret key, and giving everybody the possibility to decrypt the signature using the public key, to verify the equality of the decrypted message and the original [5]. In blockchain systems, digital signatures are used to provide a way of identification.

2.3 Blockchain

A blockchain is a technical implementation of a data structure that contains one or more unchangeable records of information about transactions, votes, or other information that one deems to be important. A different name for such an unchangeable record log is an immutable ledger. Central in the implementation is the usage of cryptographic hash functions to verify the integrity of the data.

Some of the key aspects that make blockchain technology important are that it is decentralized, transparent, and immutable. Data is recorded in a network of distributed nodes, thereby not having a single point of failure nor a central authority taking control over the network. As a consequence of the distributed record log across all nodes, the data is accessible to all the nodes involved in the network, and each log is easily verifiable.

As different concrete blockchain implementations refer to internal constructs using different names, this thesis will use the following names for clarity:

Account: A public key within the addressing scheme of the blockchain.

Block: A single atomic unit of change on the blockchain. A block contains one or more transactions, a header and a signature for data integrity.

Block Header: Metadata for a block containing the signature of the previous block and arbitrary information regarding the distributed agreement of the block. It may also contain other information depending on the implementation.

Transaction: A transaction from one account to another. Transactions contain metadata signed by the sender for authentication, a recipient, and an optional payload representing the message data.

Transaction Processor: A program that processes transactions. Transaction processors build up the business logic of a blockchain application.

Figure 2.2 describes the blockchain data structure. The *signature* field describes a message digest of the transactions and the previous signature. Due to the dependence on the previous signature in the current signature, the data structure enforces immutability [6].

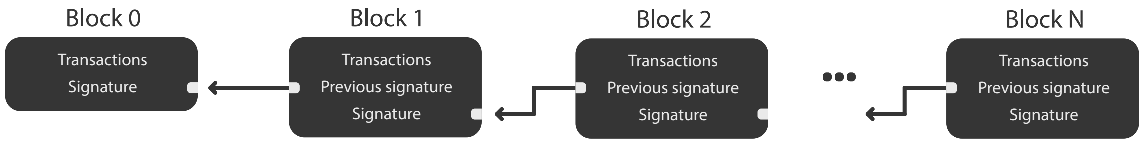


Figure 2.2: Blockchain data structure visualization.

3

Consensus

In any distributed system or database, there has to exist a set of rules by which each participating node in the distributed network can determine the veracity of each other nodes' claims. This especially applies to systems managing sensitive information such as monetary claims or electoral votes. For example, if one node claims that a vote has been cast for the red party, and another claims the same vote was cast for the blue party, how do the other nodes in the network know or choose whom to trust?

3.1 The Byzantine Generals Problem

The **Byzantine Generals Problem** describes a situation in which several armies are surrounding a city, who all need to attack at the same time to defeat it. The *only* way to communicate between the armies is via messengers, who may be captured and replaced with an impostor by the city. There is also a risk that an army may be treacherous towards the others. When an army is misbehaving, it is called *Byzantine* and creates a *Byzantine fault* in the situation.

For instance, army A is going to invade a castle together with army B. For this to succeed, both armies need to perform their attack simultaneously. To synchronize their attacks, they use messengers. Army A sends a messenger to army B proposing an attack on Friday. However, when the messenger arrives at army B, army B declines the request and proposes an attack on Saturday. The messenger returns to army A to give the new proposal, but during the travel, the messenger gets replaced with an impostor. This makes army A receive a false agreement of attacking on Friday as can be seen in Figure 3.1. The outcome being that both armies attack on different days and fail with their task.

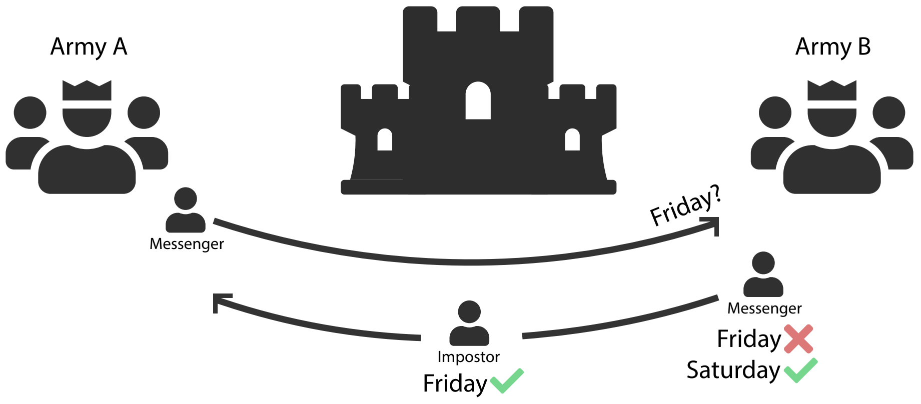


Figure 3.1: A Byzantine situation

The Byzantine Generals Problem also represents a blockchain network in which any one node may be fraudulent and, for example, claim to have more money than it has. Solutions to the problem have been implemented in numerous different ways, and the common denominator for all implementations is that they can handle fraudulent nodes in the network and still reach consensus among the non-fraudulent ones [7].

Consensus is the agreement of an arbitrary value over several parties, one can abstract the problem and use the Byzantine Generals Problem to inspect the properties of a specific consensus process [7]. Drawn in parallel, the nodes of a decentralized system could be seen as the armies, and the task of attacking the castle is the nodes trying to reach consensus. To prevent that, a faulty node (the traitor) affects the outcome of the consensus, the properties of the generalized Byzantine Generals Problem are used by consensus algorithms to produce a fault-proof consensus outcome.

3.2 FLP Impossibility Proof

Another important discovery regarding consensus algorithms is that of the **FLP** impossibility result. Fischer, Lynch and Paterson (FLP) [8] proved that no asynchronous distributed system could guarantee that consensus would be reached with one or more crashed processes. Since a public blockchain per definition is an asynchronous distributed system, the proof applies in that context too. The proof in the case of consensus algorithms for blockchains may be interpreted as a choice between three properties: **safety**, **liveness** and **fault-tolerance**, where you may choose at most two of these properties as visualized in Figure 3.2 [8] [9].

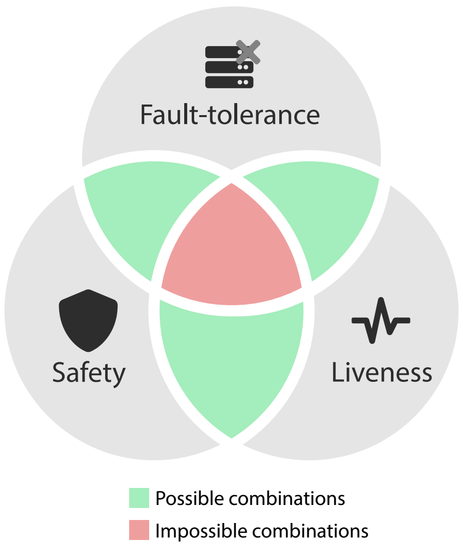


Figure 3.2: Venn diagram over the possible combinations in accordance to FLP

The property safety implies that all results are valid and identical for all nodes. A node should, therefore, not be able to send a malicious result to the client. Liveness is the act of guaranteeing that nodes that do not fail always produce a result. Lastly, fault-tolerance guarantees that a system can tolerate the failure of a node at any point [10].

3.3 Double-spending Problem

The **double-spending problem** is a core problem that decentralized consensus algorithms try to oppose in different ways. Double spending in blockchain terms is the act of using the same digital currency twice. For instance, in Figure 3.3, a user *Bob* wants to buy a fish from an online merchant. Bob places one transaction X of *50 USD* to himself, and then one transaction Y of *50 USD* to the merchant. The merchant does not wait for confirmation of transaction Y before releasing the product to Bob. The system will later reject transaction Y due to insufficient balance on Bob's account, and the merchant receives nothing [11].

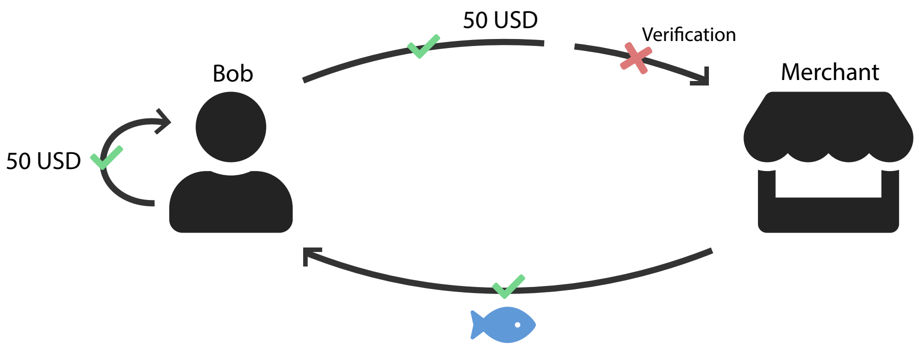


Figure 3.3: Bob sending two transactions and only having funds for one. Merchant sends the fish before verification of transaction.

For a centralized system, it is easier to detect and prevent double-spending since a third party is available for verification. For a decentralized system, additional algorithms are required to maintain a double-spending proof system. Most consensus algorithms are devoted to solving this problem, as this is not an easy task for a decentralized system to solve. There are different ways for consensus algorithms to solve this problem. However, one key concept is that each transaction must be verified and accepted by a set of validators before it is added to the ledger. This prevents the confirmation of the latter transaction and instead marks it as invalid [11]. For instance, the practical Byzantine Fault Tolerance algorithm (see 3.6) requires a fixed number of validators to verify a transaction before it is added to the ledger [12].

3.4 Proof of Work

Proof of Work (PoW) is a consensus mechanism used by blockchains to confirm transactions and produce blocks. PoW-based networks are mainly used for users to transmit cryptocurrency in the form of transactions [13] [6]. Transactions are collected in blocks and saved in a decentralized ledger, which every participating node has access to. PoW uses **miners**, generally nodes run by users, for the creation of blocks and confirmation of the validity of a block [14]. Miners achieve this by solving and verifying mathematical problems related to the block. Generally, to solve the mathematical problem should be difficult *enough*, in relation to the size of the network, but easy to verify [13]. **Hashcash** is an example of a PoW algorithm, which is used on the Bitcoin network. The algorithm used by the Bitcoin network operates by guessing a value that, when fed through a hash function together with the block header, should result in a value following specific properties. These are the properties that determine the complexity of the problem [6].

In practice, PoW works as illustrated in Figure 3.4. In the illustrated example, *Alice* is transmiting *five* cryptocurrency to *Bob*. For this transaction to be valid, it has to be confirmed by a miner. *Carol* is running a mining node and happens to be the one to solve the hash for the block. Carol is rewarded with a certain amount of cryptocurrency for solving the hash.

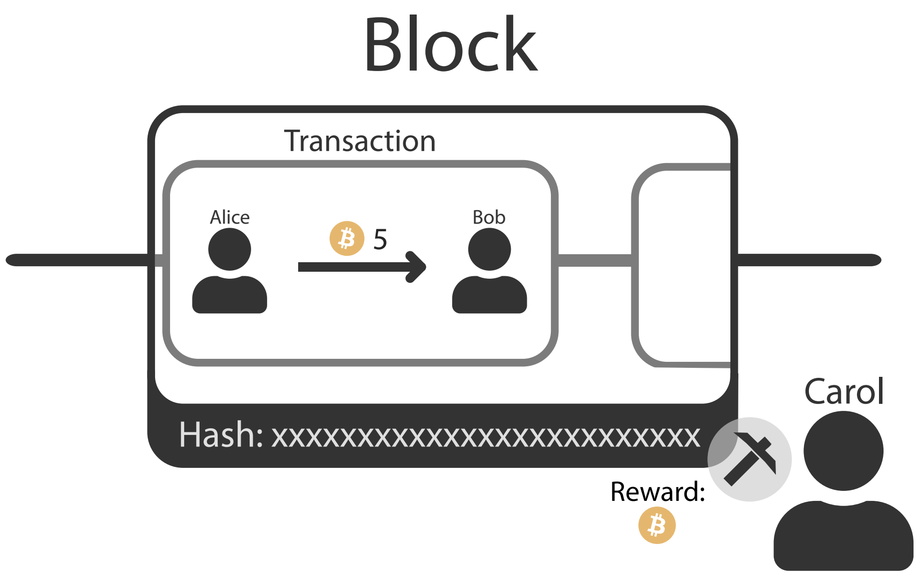


Figure 3.4: A block mined by Carol in a PoW-based system.

As of today, there are both significant benefits and disadvantages of PoW. PoW is beneficial in terms of security, as it limits many actions that could be performed in the network. Most attacks on the network require a considerable amount of computational power and time to do the necessary calculations. Another benefit of PoW is that it does not matter how wealthy a miner is; instead it is about how much computational power the miner can provide. Therefore, the miners with the most amount of funds are not in charge of the decisions made in the network. However, using computational power as the primary workforce in PoW, makes the amount of electricity consumed for computation increase significantly. Not to mention, the calculations done in the PoW networks are only applicable in guaranteeing the security of the network, whilst it could be used somewhere else. For instance, the computational power could be used to do calculations in the fields of science or biochemistry [13].

PoW is today applied in many of the largest blockchain projects, the most distinguished being *Bitcoin* and *Ethereum*. Bitcoin was the first blockchain project to use this type of consensus algorithm in their blockchain [13].

3.5 Proof of Stake

Proof of stake (PoS) is a consensus mechanism for confirming the validity of new transactions on the chain. In the proof of stake system, nodes are chosen at random to commit new blocks to attach to the end of the blockchain. Nodes are said to be given the task of *forging* the block when committing to this task. The aforementioned *stake* refers to what percentage of the chains total monetary capacity any one node *invests* into a specific stake-wallet. That percentage directly reflects the probability of that node being next in line to validate the next block on the chain [15].

When a node is given the task of forging the next block, it validates all transactions in the block and then signs it. To incentivize this, the node is given all transaction fees from the block it just signed. If the node is found to have included fraudulent transactions or otherwise tampered with the block, it is penalized by not getting a part of its stake back and will no longer be allowed to continue as a committer [16].

For instance, if Bob has a stake of 30 bitcoins whilst Alice has a stake of 5 bitcoins, Bob's node has a higher chance of becoming the next validator in comparison to Alice's node. In Figure 3.5, Bob is chosen as the next validator. He is now tasked with forging the next block. Bob's node then validates all the transactions in the block and signs them, gaining all the transactions fees from the block. If, however, Bob fails to forge the next block due to tampering or including fraudulent transactions, he will lose a percentage of his stake and gets a lower possibility to becoming a validator in the future.

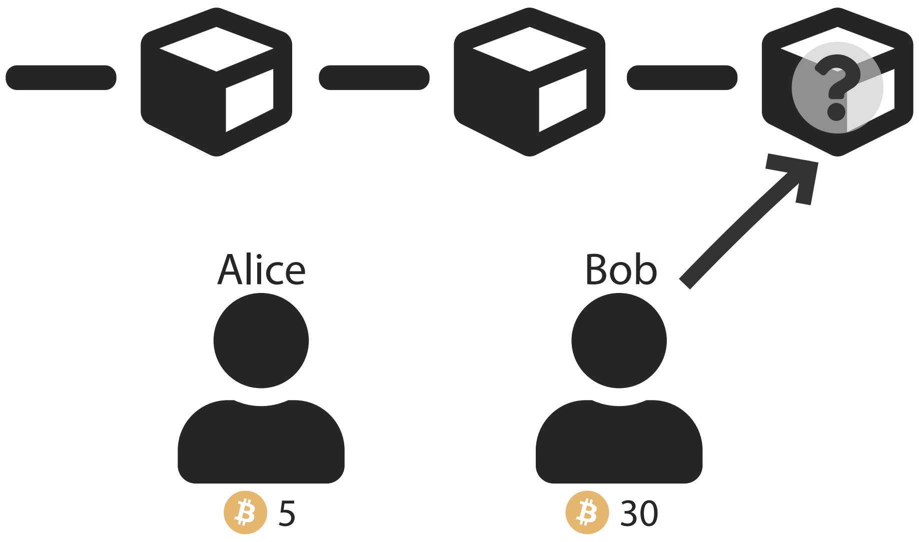


Figure 3.5: Bob is chosen to be the validator and forges the next block.

Compared to PoW-based systems that rely on computers to do work that take the form of solving equations, it has a fairer distribution of funds and rewards on the blockchain. Proof of stake also consumes orders of magnitude less computational power, *i.e* electricity.

As of today, PoS is not used in any major decentralized system as a standalone consensus algorithm [17]. There are variations of PoS, most predominant being *Delegated Proof of Stake*, which is an improved alternative to both PoS and PoW [18]. However, many PoW-based systems are planning to switch over to PoS mainly due to the great benefits PoS has in terms of electricity efficiency. The cryptocurrency *Ethereum* is planning to adopt PoS in the near future, and *Bitcoin* could most likely do likewise [19].

3.6 Practical Byzantine Fault Tolerance

The **practical Byzantine Fault Tolerance** (pBFT) algorithm is a way for a distributed computer system network to tackle *byzantine faults*. pBFT executes in different rounds called *views*, which further are divided into four phases, demonstrated in Figure 3.6. For each view, there is a primary node, also seen as the leader node. The primary node is changed in a *round-robin* matter but could externally be changed if the primary node times out or another node proposes a view change due to the primary node being malicious. For the proposed view to occur, a supermajority is required. The next elected leader in a view change conventionally follows the round-robin method of electing the leader node [20]. Once the leader has been elected, the view performs the four phases.

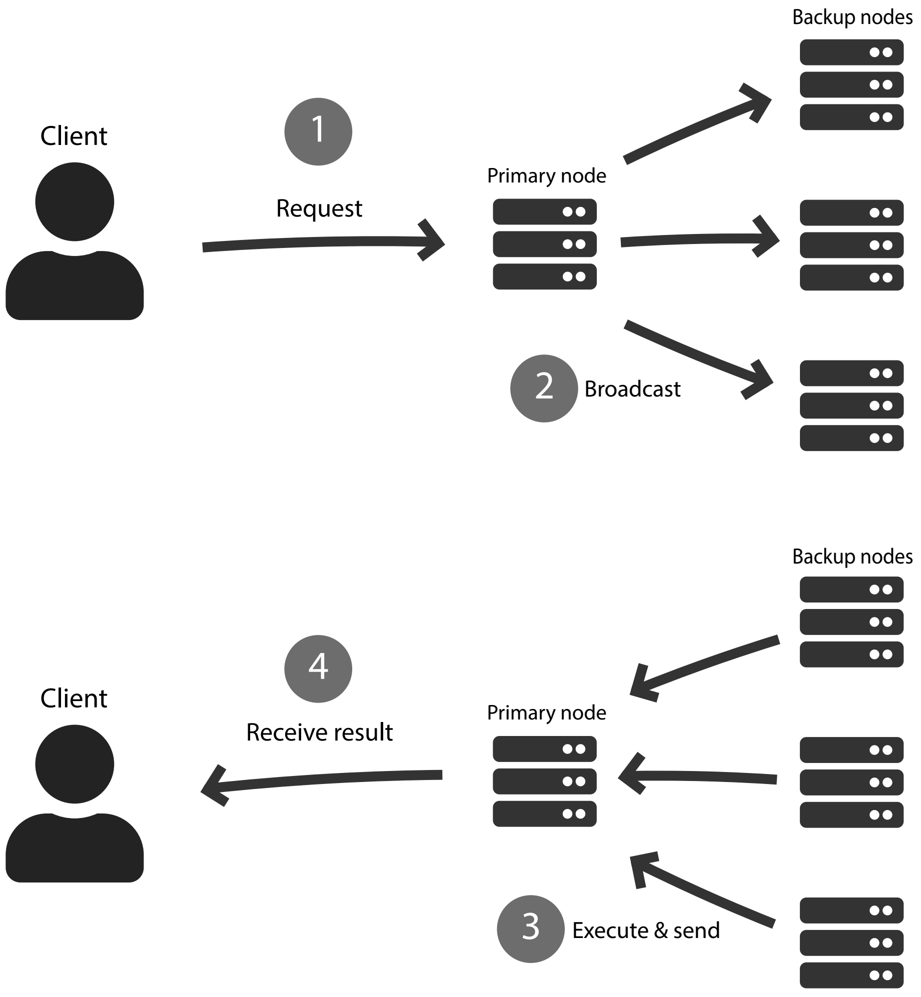


Figure 3.6: A pBFT round executing the 4 different phases.

1. The primary node receives requests from a client.
2. The primary node broadcasts the request to all other nodes, also called *backup nodes*.
3. All nodes execute the request and send a reply to the client.

4. the client receives the replies and waits for *maximum faulty nodes + 1* replies from different nodes with the same results

The final result is the majority of identical results received by the client [12].

pBFT assumes that there are independent node failures and therefore has a constraint that at most one third of the nodes are allowed to be faulty. In terms of *scalability*, the more nodes the system has, the more unlikely it is that one third of the nodes are faulty at the same time [20]. However, the algorithm is not scalable in practice due to the number of messages sent between nodes. Besides tackling byzantine faults, the system also provides *liveness* and *safety* if at most $\frac{n-1}{3}$ are faulty at the same time.

The algorithm is currently used in the blockchains *Zilliqa*, *Hyperledger Fabric* and *Tendermint* in combination with other consensus algorithms. Zilliqa uses pBFT in combination with PoW, Tendermint uses pBFT in combination with *Delegated Proof-of-Stake* (DPoS), whilst Hyperledger has adopted pBFT as a *permissioned* version [21]. A permissioned version of a blockchain implicates that contributors to the blockchain must be given access to it before being able to contribute to it [22].

3.7 Federated Byzantine Agreement

Federated Byzantine Agreement (FBA) operates much like the *practical Byzantine Fault Tolerance.* It mainly focuses on solving the *Byzantine Generals problem* but has a slightly different approach. FBA does not have a general leader node that acts as the receiving endpoint for consensus network. For an FBA system to reach consensus, the system uses *quorum slices* [23]. Quorum slices are a subset of nodes that each node has predefined, as seen in Figure 3.7. Quorum slices can be seen as unique lists of trusted nodes for each node. All participants in the quorum slice work together to reach consensus regarding the transaction sent to the primary node in the quorum slice. To synchronize and keep the nodes in the quorum slice up to date, each node has a candidate set that holds the transactions they have received. The candidate set is broadcasted to other participants in the quorum slice to keep them updated. The result of having quorum slices is that FBA systems do not need a predefined list of nodes; it is instead public for any node to join the system at any point in time [23].

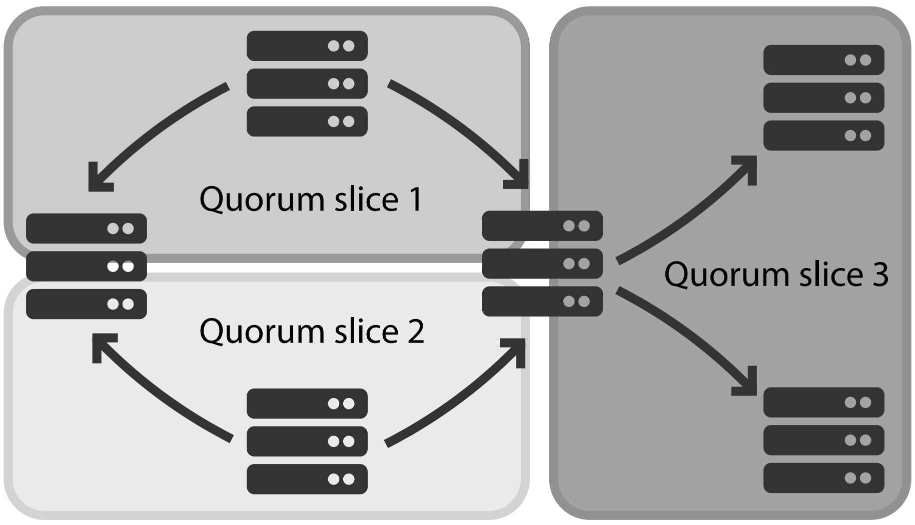


Figure 3.7: Three quorum slices with different nodes

There are two predominant solutions of the FBA system, *Ripple* and *Stellar*. The latter solution is the most recent and up to date solution in terms of safety. Ripple is, however, easier to implement. Ripple is broken down into four major phases and, additionally, four stages for the voting phase [24].

1. **Initializing phase**, takes all new transactions and all failed transactions from the previous round and adds them to the candidate set.

2. **Merging phase**, each node in the quorum slice merge their candidate sets and make them ready for the voting phase.

3. **Voting phase**, each node in the quorum slice votes for the transactions in their candidate set. The voting phase has four stages.

   I. **50% voting stage**, each stage holds a voting percentage threshold for a transaction to proceed to the next stage.

   II. **60% voting stage** (Fig. 3.8)

   III. **70% voting stage**

   IV. **80% voting stage**, successfully voted transactions are moved to the final phase. Each transaction that fails a voting stage is discarded or inserted to the candidate set on the initializing phase.

4. **Final phase**, all nodes broadcast their successfully voted transactions to their peers and commit it to the ledger. Once the ledger has been established, the nodes return to the initializing phase.

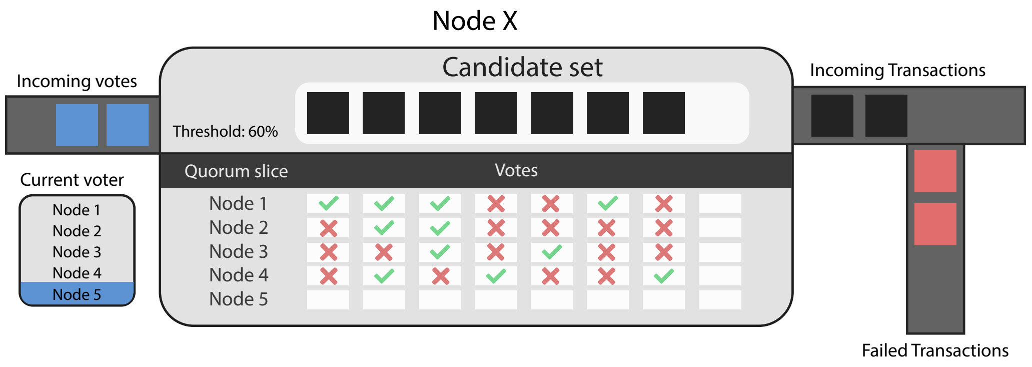


Figure 3.8: The 60% voting stage for an FBA system. Node 5 is committing its votes to Node X.

The most significant advantage that FBA has over pBFT is that it is more decentralized, less central authority is taking part in the selection of nodes. FBA also gives freedom to both users and nodes, as both can choose which nodes they trust. This is especially important for nodes that belong to a specific company, for instance, an automobile finance company. The node could require confirmation from a trusted banking node, a trusted credit agency node, and a department of motor vehicles node. The quorum slice for the automobile finance node would then be built up by these nodes and allow for a transaction to be verified by all authorities [23]. This means that the automobile finance company can be entirely sure that their transactions are getting verified by trusted authorities, and therefore do not need a central authority to provide this degree of trust.

FBA is currently used in the form of Ripple in the payment network *RippleNet* [25]. RippleNet is in turned used by financial institutions around the world. The most significant application of RippleNet being *XRP*, a fast and scalable digital asset for payments around the world [26]. The other major adoption of FBA, Stellar, is an open network for storing money and performing payments. The network's services are, for instance, used by the well known IT-company *IBM*. IBM's usage of Stellar provides the ability for financial institutions to internationally send payments of any currency [27].

4

Hyperledger Sawtooth

Back in the year 2015, a blockchain framework called **Ethereum** was released. The goal was to create a blockchain that supported a scripting language that would allow people to build applications on top of it. Since Ethereum was released, several blockchain frameworks can be found on the market; one of them is **Hyperledger Sawtooth**. *Security*, *scalability*, and *modularity* are mentioned as Sawtooth's main benefits in its white paper [28]. Modularity is most present in the choice of the consensus protocol. Instead of a consensus, Sawtooth provides an interface allowing several different protocols to be used. Aside from many other distributed ledgers, cryptocurrency is not one of Sawtooth's aims. Instead, it is an open-source ledger aimed towards enterprise applications [29].

Processing transactions in the right order is ensured through strict ordering by Sawtooth. Transactions are placed in a block by a client, which then signs the block and sends it to a validator. When the consensus engine verifies it, the block is committed to the blockchain [29].

Sawtooth comes with a set of transaction processors which represent core functionality. This includes a settings transaction processor that, as the name suggests, stores and handles on-chain settings. Another one is the identity transaction processor, which is responsible for the permissioning and a validator registry for registering validators. Aside from these, Sawtooth provides some transaction processors for development and testing. To control state-changes on Sawtooth, one can make a set of transactions with specified operations, called a *transaction family*. Sawtooth allows a developer to write additional transaction families in several different languages. It also supports the use of Solidity together with Hyperledger Burrow EVM [28].

4.1 Peer-to-Peer Communication

Nodes on the Sawtooth network use TCP to send messages about peers and blocks. To communicate these messages, Sawtooth uses the *ZeroMQ (ØMQ) library*, containing objects serialized using Google's protocol buffer library. Sawtooth itself does not ensure that each message is shared with each node. Instead, Sawtooth relies on a protocol that relies on each neighboring node, forwarding the message to its neighbor, a *gossip* protocol. The Sawtooth network can control who gets to join it, send consensus messages, and who can submit transactions by assigning *permissions* to the nodes it wants to control.

4.2 Security & Privacy

When it comes to security with regards to Sawtooth, there are multiple ways the level of security can be tweaked to fit the needs of the end-user, such as the internal settings and the type of consensus algorithm that is used. Privacy is an essential factor to consider when developing an application on top of Sawtooth, which makes it crucial to employ encryption wherever suitable.

Threats to Security

The Sawtooth nodes are separated by physical distance, and they use networks to communicate with each other, which in turn introduces vulnerabilities. Mitigating vulnerabilities that are based on the structure of the communication between the nodes is challenging. Sawtooth's use of the *ØMQ protocol*, which supports encryption and is backed by a large open-source community, lays the groundwork for security. Even so, more action is required to guard against DDoS attacks since the main objective is to disrupt rather than to gain unauthorized access. Furthermore, if the attackers manage to compromise the registry that holds the IP-Addresses to other nodes, which is called *Domain Name System (DNS) Hijacking*, then they would be able to route traffic through the attacker's system. From there, they can choose to either re-route the traffic to an attacker's node or listen in on the sent information.

Distributed Denial of Service Attack (DDoS)

Any service that uses a network connection for communication is vulnerable to **DDoS attacks**. They are primarily targeting the availability of a system and does not directly threaten the stored information or the user's privacy and security in any way. Often they are performed by an attacker that has access to a large number of devices with an internet connection. If the attacker knows the IP-Address of any nodes or node in the system, then they can send requests of various types to one or several nodes, which can be seen in Figure 4.1. Doing this will cause congestion in the network traffic, and as a result, stalling or even halting the voting process [30].

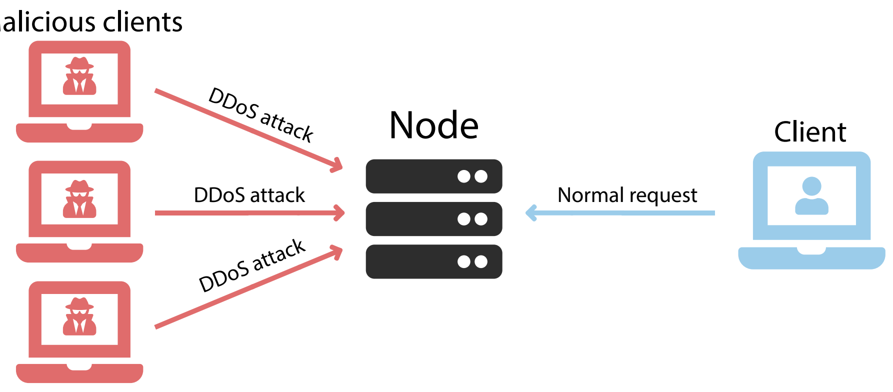


Figure 4.1: Malicious users overfloding the node with a DDoS attack. The normal user's request gets interrupted and delayed.

Domain Name System (DNS) Hijacking

When a new node joins the network, it will need to know the IP-addresses to the other nodes. This allows an attacker the opportunity to *tamper* with the registry that holds these addresses so that the new node will communicate with the attacker's system [30]. They can then proceed to feed the node with a false state of the network or keep that node from taking part in the voting process. Another, more dangerous activity the attacker could do is to *steal* the information the node is sending. To achieve this, the attackers can route the network traffic through the attacker's system and analyze the transmitted data.

Client Application Security

The application developer has the responsibility of making sure the client takes the necessary safety precautions and appropriately handles data. The private key needs to be stored in a secure place so that any vulnerabilities that may give attackers access to it are minimized.

Network Security

The Sawtooth network is vulnerable to DDoS attacks when it is not running on properly configured hosts [31]. There are several ways to mitigate this vulnerability. Either by configuring both the machine Sawtooth runs on or by configuring the Sawtooth network itself, to make sure that no unauthorized entities can access the system.

Transaction Processor Security

Different public keys in a Sawtooth application can have individually set permissions called *roles* [32]. Those can dictate if a particular public key has permission to post transactions, which is checked by the transaction processor when it receives a transaction. It can then choose to either *Challenge* or *Trust* depending on the roles of the signing public key [32].

5

Implementation Strategy

The development of a decentralized voting system *without* the use of a previously existing public blockchain is a task that may be split into several subprojects. To ease the development of the application, the Hyperledger Sawtooth framework is used due to its modular nature.

The different subprojects are the consensus engine, the transaction processor, the application programming interface, and the frontend. These subprojects together make up the voting application, from end-user to the blockchain and distributed consensus as shown in Figure 5.1. This chapter will explain, in further detail, the methods and technologies used within each subproject, which are then used together to create the voting application.

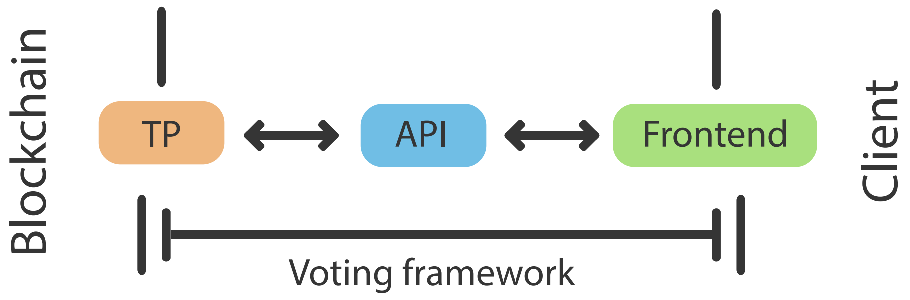


Figure 5.1: Application flow

5.1 Consensus Engine

The consensus engine is responsible for the *distributed agreement* in the system, and is therefore of high importance in the application. Since the Hyperledger Sawtooth framework is modular and supports custom consensus algorithms by making use of *software abstractions*, the implementation of a consensus engine is an isolated task. The Hyperledger Sawtooth framework provides supporting services for implementing a consensus engine in several languages; however none exists for Java [33], which is the language of choice for this subproject. Therefore these supporting services are developed first, followed by an iterative approach when developing the consensus engine.

Supporting Services

The development of supporting services is inspired by the implementation in other languages to mimic the expected behavior. The primary responsibility of these supporting services will be to abstract over the communication layer with the validator. Moreover, this will make the implementation of the concrete consensus engine more focused on the task of achieving *distributed consensus* instead of communication.

Iterative Approach

Building the consensus engine using an iterative approach makes the *learning curve* of distributed consensus less steep. Another positive result of doing so is that the supporting services will be verified in an early phase, mitigating complexity when debugging later phases. The implementation begins by implementing what the Hyperledger Sawtooth calls a **devmode** consensus. This type of consensus is simple and features an algorithm selecting a random leader and committing blocks as fast as possible. Further on, a consensus engine implementing the pBFT algorithm will be developed to ensure the distributed aspect of the system works correctly. Moving forward from the pBFT algorithm, the implementation of a consensus engine using the FBA algorithm will be examined.

Technologies Used

The library JeroMQ is used be able to communicate with the Hyperledger Sawtooth validator. JeroMQ is a Java implementation of the ZeroMQ communication library [34]. To be able to test the solution, a combination of **JUnit**, **Mockito**, and **Docker** is used. JUnit is a testing library for Java, which in conjunction with the mocking library Mockito, brings a way to perform unit tests of small components [35].

Docker is a thin virtualization layer, giving possibilities to create *integration tests* for the consensus engines and verify the functionality in multi-node clusters on a single computer [36]. Within Docker, you can spin up several environments and specify a network that they are able to communicate over, giving a simulation of a real-world environment.

5.2 Transaction Processor

To give the voting application the ability to handle and enforce business logic, a transaction processor is implemented. The transaction processor is implemented using the Hyperledger Sawtooth Java SDK. Using the Sawtooth SDK simplifies the development of a transaction processor by making use of *abstractions* to *minimize* complexity. A transaction processor contains two top-level components; a processor and a handler. The Sawtooth Java SDK provides a general-purpose implementation of the processor class [33].

The voting transaction processor is implemented by first instantiating the general-purpose transaction processor. Upon instantiation, the address of the validator is provided to connect the voting transaction processor with the validator. The second step is to register the voting handler class to the voting transaction processor.

On the other hand, the voting handler class is implemented to define the business logic of the system in the *apply* method and its helper methods and classes. The voting transaction processor relies on a data model to ensure deterministic behavior. The data model is developed to separate the details of state encoding and decoding, transaction payload processing, validation logic, and addressing from updating the state on the blockchain.

Technologies Used

The transaction processor makes use of the aforementioned Hyperledger Sawtooth Java SDK and the Jackson library for serialization and deserialization purposes.

The Jackson library provides data-binding assistance to serialize a Java object to Javascript Object Notation (JSON). In the words of the developers, Jackson is the *best JSON parser for Java* [37].

5.3 Application Programming Interface

To support the end-user clients, an Application Programming Interface (API) is needed to communicate between the client and the validator node. The purpose of the API is to form an abstraction layer over the API interface supplied by Hyperledger Sawtooth. With the use of the same data model as the transaction processor, one gets consistency of terminology and data format across the system.

Communicating with the API from the client is performed over the Hypertext Transfer Protocol (HTTP) using data in the format of JSON.

Technologies Used

The library Rapidoid in combination with Jackson is used to create the HTTP server for the API. Since the API is a small and quite simple component of the system, the choice of tech stack is tailored to minimize size, complexity and development velocity.

Rapidoid is branded as a *fast, simple and powerful* Java web framework which supports the goals of minimized complexity within the application perfectly [38].

The Jackson library is used to support the serialization and deserialization needs of the API. For more information regarding Jackson, see section 5.2.

5.4 Frontend

As the application should be presented by an interface, a frontend is developed. As most of the project scope and focus is the development of the consensus algorithm and transaction processor, the frontend only represents the most necessary features for finding an election and placing a vote.

Requirements

A set of requirements and use cases is set up to define the end goal of the frontend. The use cases defined for the frontend are:

- Sign in with a private key to view elections one has access to
- View list of elections connected to a key
- Show details of one election
- Vote on a candidate
- View feedback from placing a vote
- Show election state

The initial plan is to meet these requirements following a mock-up developed in Adobe XD. Adobe XD is a tool for building interactive interfaces without any code. This is used primarily for simplifying the decision process in what has to be included in the interface.

Technologies Used

To quickly accomplish a working frontend, the web application is developed in *React*. React is a JavaScript library for building user interfaces but is often compared to frameworks like VueJS and Angular. This application is developed in React as it is the most popular JavaScript framework among developers [39]. One advantage of using a popular framework is that documentation is easy to find, which eases troubleshooting. Another reason is that many other libraries are built to be compatible with it. One example of this is *React Bootstrap*. React Bootstrap is a component and CSS library developed from the already existing library *Bootstrap*, but adapted to React [40]. React Bootstrap is used as the main component library in the application to achieve a coherent design throughout the application. Furthermore, the communication between the frontend and API is achieved using *Axios*. Axios is a promise-based HTTP client that makes it easy to use with JavaScript async and await [41].

5.5 Testing the System

For the solution to be complete and usable, a few properties of the system have to be tested. The four properties that the system must pass are the following:

1. **Correctness** - The test for the correctness aspect of the voting system will verify that every cast vote in the blockchain follows all specified rules for the voting system. It will also check that the blockchain does not allow multiple votes to be cast from the same user. The blockchain should also allow every user with a key to cast their vote and have their cast vote correctly inserted to the blockchain.

2. **Security** - In regards to the security aspect of the voting system, the test will primarily check that an already cast vote block cannot be tampered with. The possibility of a third party casting one or more unwanted votes will also be analyzed.

3. **Transparency** - The transparency of the system will be examined as the possibility of a voter or third party to inspect and verify the result and that the predetermined rules of a vote are followed.

4. **Anonymity** - The anonymity aspect of the system will be inspected by examining how the system ensures anonymity and what properties could be used to determine the identity of a cast vote. Such a test includes not being able to trace back the voter by using their key. The test case will check if a vote can be deanonymized and be paired with a voter in the case of a private key being leaked.

Tests for the properties will be conducted based on logical reasoning and proofs deducted from the properties of a public blockchain and the consensus algorithm.

6

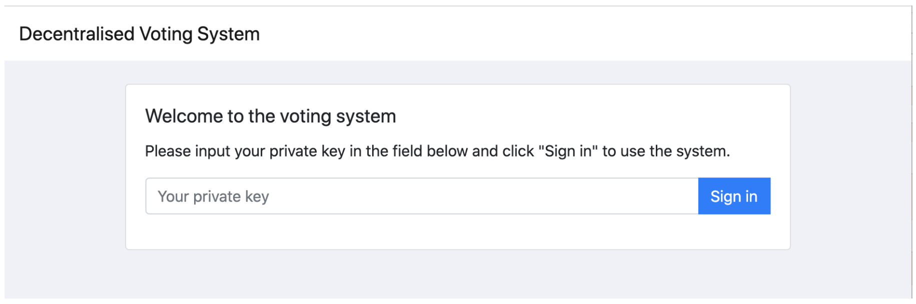
Voting Over Sawtooth

This chapter describes the system and the different modules that make up the system. It is organized as follows: Section 6.1 presents the web application, Section 6.2 gives an overview of the voting process, and Section 6.3 describes the different network components. The source code of the system is available at `https://github.com/DATX02-87/DecentralizedVoting` under the MIT license.

6.1 Frontend

Showing the underlying logic and functionality that has been built is an important part of the goal application. That said, only previously mentioned requirements have been fulfilled in the interface through three main pages:

1. The purpose of the *Sign in page* is for the voter to enter a private key, and then get access to the elections for which that voter is eligible. This is achieved through a simple input text area with a *sign-in* button, shown in 6.1. When entered, the key is stored in a state which is supported by a functional state management system in React [42]. Other than storing the key, there is a validation process making sure the key entered is valid. Upon validation, the user is routed to the votations page.


Decentralised Voting System

Welcome to the voting system

Please input your private key in the field below and click "Sign in" to use the system.

Figure 6.1: Sign in page

2. The *Votations page* renders all elections for the key stored in the state. The elections are fetched from the API and shown in a list component, where each item is designed as a card with simple election information shown in Figure

6.2. The button redirects to the detailed view of that votation.

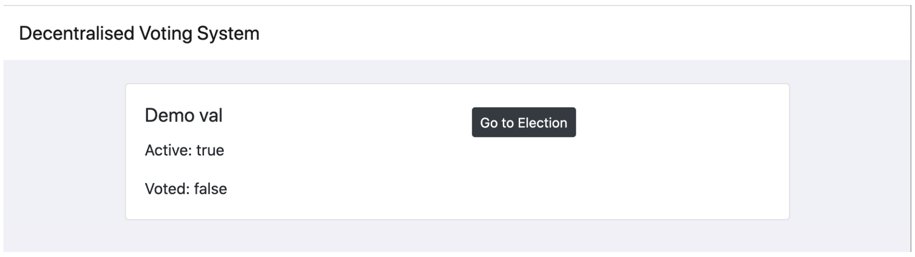


Figure 6.2: Votations page

3. The *Votation Detail page* consists of a card showing detailed information of one election, visualized in Figure 6.3. Upon choosing a candidate and casting a vote, the vote is converted into a transaction, including the candidate and election. The transaction is then sent together with the signature of the payload onto the blockchain. After casting a vote, the view is updated to a verification showing that the vote has been counted before routing back the votations page.

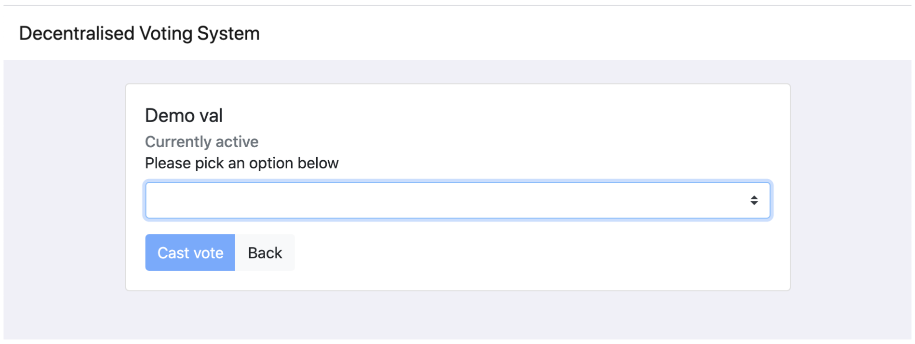


Figure 6.3: Votation detail page

6.2 Overview of the Voting Protocol

There are two types of clients interacting with the system, voters and administrators. As mentioned in Section 1.2, the network of validators maintaining the blockchain has no role in the registration of voters or the distribution of public and private keys. As a result, each election is *initialized* with a file containing the keys of eligible voters and the administrators. Given that different people may be eligible for different elections, initializing each election with a list of the voters that can participate in a particular election ensures that no voter can take part in an election they are not eligible for, under the assumption that the distribution of keys is accurate and free from errors.

Voters interact with the voting system through the web-based application described in the previous section (Section 6.1). When a voter signs in with his or her private key, information about currently available elections is fetched from the blockchain through the API. The voter can then choose one of the active elections they are eligible for, followed by casting the ballot. A voter casts the ballot by selecting one of the available options from the list of candidates for the election in question. This ballot is signed by the voter with their private key. See Figure 6.4.

The signed ballot is submitted to a validator node in the network for processing through the API. A valid ballot must correspond to an active election, include the signature of an eligible voter and include one of the listed options. If the validators reach consensus and the transaction is found to be valid, the vote is recorded in the global state and stored in the blockchain. This ensures consistent data across the distributed nodes in the network. At the end of the election, the voter can use the same private key to view the results.

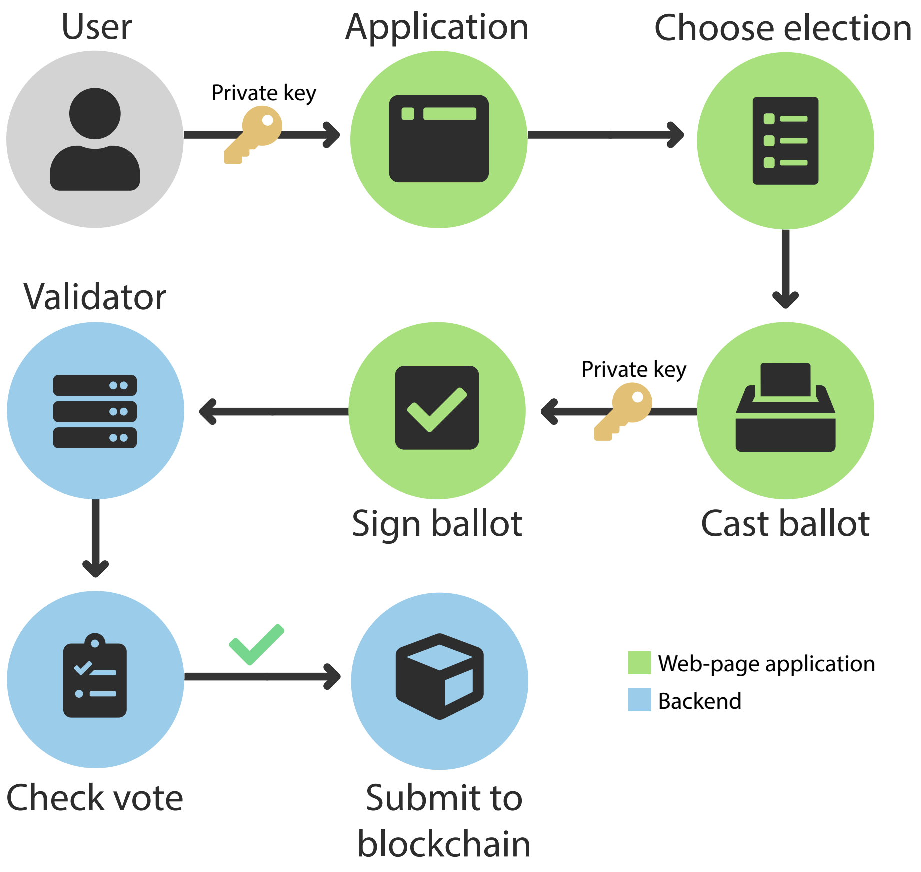


Figure 6.4: A flow chart over the voting process.

6.3 Node Components

The network is built on the Hyperledger Sawtooth platform. As Figure 6.5 shows, each node in the network includes a consensus engine, an API, a set of transaction processors, and runs a validator.

Validator

All validators in the network verify the validity of a ballot that has been cast using the same set of *transaction processors*. This ensures the *consistency*, *accuracy* and *reliability* of the system across multiple and independent nodes. In addition, validators verify that each voter can cast a vote *exactly* once in any given election by checking that the voter public key does not match a ballot already included in the blockchain.

The validators on the same network establish initial connectivity and peer discovery using gossip ØMQ protocol over TCP on port 8800. External clients communicate with validators using the API through HTTP port 8008. The components connected to the validator, such as the transaction processors and API, are installed in Docker containers and connected to the validator through TCP on port 4004.

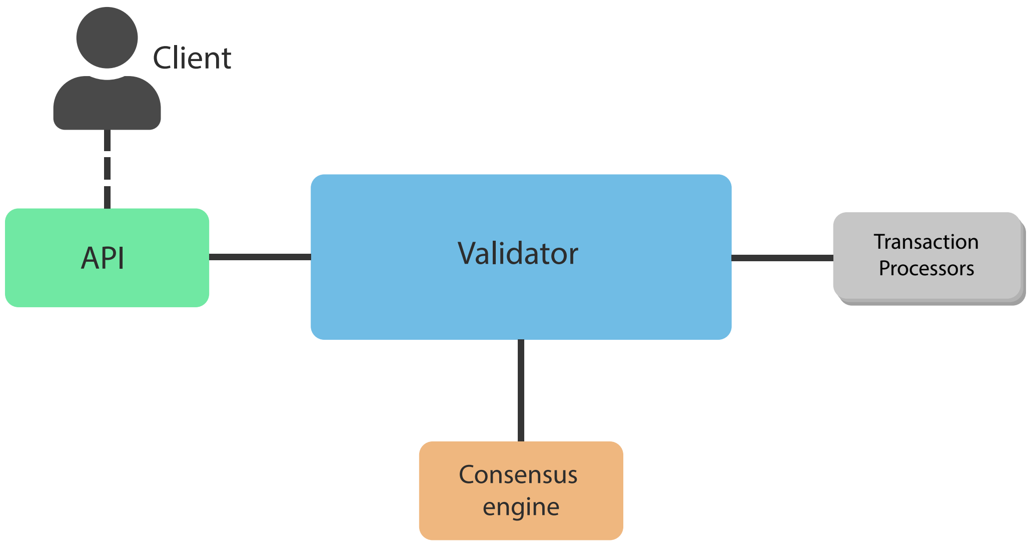


Figure 6.5: The network components for a node in the Sawtooth platform.

Transaction Processor

The main transaction processors are the *Voting Transaction Processor* and the *Settings Transaction Processor*. The settings transaction processor handles the network settings and stores on-chain configuration settings. The Voting Transaction Processor handles the submission of ballots to existing elections and relies on the data model to validate a transaction.

The *data model* is responsible for the addressing scheme, serialization and deserialization of data, and handling the enforced logic. The client and voting transaction processor must share the same data model. This is important because the client encodes the data in a payload, which is decoded by the transaction processor. Additionally, the transaction processor encodes data to be stored in the state, which the client needs to know how to decode.

Furthermore, the data model defines the methods that validate the transaction, enforce permissions of different requests by different clients, and *apply* the transaction. Applying a transaction includes the verification of the eligibility of the voters, adding candidates to an existing election, and making sure that a voter is eligible and can cast the ballot *only once* for each available, and active election.

The data model includes various classes that facilitate transaction processing. Some of these classes are `Action`, `TransactionPayload`, `Transaction`, and the `Reducer` class. The `Action` class represents the different actions that a transaction payload may contain. There are five actions defined by the system: `ADD_ELECTION`, `ADD_CANDIDATE`, `CAST_VOTE`, `END_ELECTION`, and `INIT`. The `ApplyTransaction` method defined in the `Reducer` class takes the following steps to apply the transaction:

- Get the `Action` from the transaction payload
- Deserialize the transaction data from the payload by instantiating a Java Object from the transaction data encoded in a JSON string for each action mentioned above.
- Call the appropriate handler method for each action.
- Return an object of type `GlobalState`, which represents the new state.

The handler methods are action-specific and perform the necessary tests to validate the transaction. For example, the `handleCastVoteAction` method performs five tests by checking:

1. that the supplied election exists
2. that the election is active
3. that the candidate exists
4. if the voter is eligible for this election
5. whether or not the transaction submitter has voted before.

The other handler methods perform similar tests, however, the transaction submitter must be an administrator.

The voting transaction processor decodes the transaction payload, gets the current state from the *context*, calls the `ApplyTransaction` method, which returns an updated state from the data model, as seen in Figure 6.6. Finally, it stores updated state back to the context. Both updating and getting the current state through the context requires making the address at which the state information is stored. The length of the address is 70 characters. The first six characters are obtained from the SHA-512 hash of the UTF-8 encoding of the `FAMILY_NAME`. The remaining 64 characters are the hash of the public key that is passed as a parameter to the constructor of the voting handler class. The transaction processor turns the `GlobalState` object into bytes and stores it into the Radix-Merkle Tree of the validator based on this address.

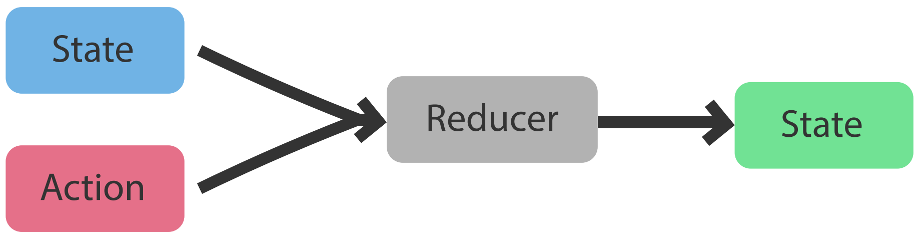


Figure 6.6: A state and an action reduced to a state.

Consensus Engine

All nodes in the network include a process running the consensus engine, which is responsible for the progression of the blockchain. This requires the consensus engine to be able to handle messages received by peers and the validator. The backbone for handling received operations is the consensus algorithm that the engine implements [43]. The algorithm implemented in the system is the practical Byzantine Fault Tolerance algorithm and uses pBFT specific protobuf messages to communicate with the engine's peers. The following pBFT messages are used for communication [44]:

PbftMessageInfo holds information about the message type, the current view number, the block sequence number, and the id of the node that signed the message.

PbftMessage is a generic pBFT message that is used to communicate with other peers regarding the current phase the node is in.

PbftNewView is a message sent by the new primary node to signify that a new view should be started.

PbftSignedVote is a committed vote sent from a node to the leader node.

PbftSeal is a seal to verify that a specific block should be committed to the ledger. It holds both a list of PbftSignedVotes and a PbftMessageInfo.

The consensus engine uses Sawtooth specific protobuf messages to communicate with the validator. Sawtooth messages are primarily used to signal the engine about blocks and peer connectivity. When the consensus engine receives a block-related Sawtooth message, the algorithm tries to reach consensus. This is done by going through the different pBFT phases and communicating with peers through pBFT messages until the final result is sent to the validator through a Sawtooth message [44]. The blockchain has then progressed.

7

Discussion

This chapter discusses how well the final system meets the requirements, as well as the social, economic and environmental impact such a system might have if implemented in a real-life situation.

7.1 Conformance to Requirements

The system conforms to the requirements stated in 5.5, which were conditions required for any general democratic voting system.

After a vote has been cast, one can get the state of the blockchain and confirm that the transaction indeed has taken place, fulfilling the requirement for *transparency*. As for *anonymity*, there is no way to connect a cast vote or a private key to a participating voter from within the system. However, as mentioned in section 1.2, no concern is taken for the possibility to track private keys back to a voter outside of the system.

After a vote has been added through a block that has been committed to the chain (given that the block was not tampered with and rejected), that vote is *unchangeable*. This, of course, only applies to the extent to which one accepts that the blockchain is immutable. Together with the highly secure nature of the blockchain, the risk that anyone without the proper private key could commit a fraudulent transaction as if made with that key is non-existent.

Regarding *correctness*, the system has firstly met all previous specifications and further meets all specifications of a voting system as specified in 1.2. It allows everyone with a key to vote, but does not allow multiple votes to be cast from a single private key. All nodes in the network have a copy of the number of votes on each candidate, facilitating the process of verifying election results.

The voting application built upon the blockchain fulfills all requirements for interactions as a voter. However, it does lack an interface for administrating an election. The administrative part includes setting up candidates, distributing keys, creating– and closing the election. Administrating the elections can, of course, be done, but not in an easy to use interface. This currently limits the application from being used without interaction from the creators, which makes it unsuitable for distribution.

On the other hand, the pBFT consensus algorithm allows high transaction throughput and negligible power consumption. However, it scales *poorly* when the network becomes too large due to the communication overhead introduced by the nodes communicating with every other node. It also limits the decentralization aspect since only a predetermined list of nodes can act as validators. Some of these problems can be mitigated by a successful implementation of the FBA consensus algorithm. For example, FBA allows more decentralization since there is no recommended list of validators chosen by a central authority.

7.2 Ethics & Sustainability

The possible implications of implementing a system such as the one proposed in this report in a national election with regards to ethics and sustainability manifest themselves most clearly in three categories: social, economic, and environmental.

Social Implications

There is a vast potential for considerable democratization of the election process through the usage of *decentralized e-voting*. Partially because of the trust distribution through blockchain technology, and partially because of the potential for a higher voter turnout due to the logistical simplicity of digital voting [45]. In countries that struggle with corruption and dishonest governments, a voting system controlled by all participating candidates or parties would be an unprecedented improvement to the democratic process. Other problems such as lost vote-ballots, uncertain ballot counts, or invalid votes due to misspelled party names would as well be eliminated. However, such a system also introduces potential vulnerabilities of its own to the process.

As with any effort to digitize and streamline an old system, the transition ought to be gradual in order not to leave subscribers to the old system behind. Even though electronic voting has not been tried in enough countries to conduct a quantitative study, there are studies on the Estonian electronic voting system. Implemented in 2005, it shows, among other things, that even though the e-voting alternative initially was used mainly by Estonian-speaking people of narrow age groups that were computer literate. The usage group after four elections was diversified enough to weaken, or invalidate, the previous conclusions [46]. Extrapolating from this, it would be reasonable to assume that parties with young, computer-literate, native voters would initially be favored through a higher voter turnout but that this advantage would eventually die off.

Furthermore, there are other, more direct, threats to democracy posed to such a system if not implemented correctly, such as the potential to track private keys back to voters, which might enable the selling of votes or even threats to vote for a specific party.

Economic & Environmental Implications

Arranging an election is a costly matter in terms of both money and resources. According to the Swedish election authority, *Valmyndigheten*, the Swedish general election of 2010 cost 258 million SEK, of which 148 million SEK was dedicated to printing ballots and other peripheral costs [47]. In 2018 the total cost even surpassed 348 million SEK [2]. A voting system like the one proposed in this paper would eliminate those costs, but would, of course, introduce new costs for servers and management.

With our proposed system used in parallel with- or as a substitute for the current election system, the environmental impact of the millions of ballots that are printed for each election, the transportation of said cards and transportation of voters themselves could be mitigated or even completely removed. Due to the nature of our proposed consensus algorithm, the energy consumption of processing the blockchain is nearly negligible.

8
Conclusion

The process of digitizing conventional paper and pen election schemes offers the potential to make a voting process more accessible, faster, and cheaper. This project developed a blockchain-based decentralized voting system that is built on the Hyperledger Sawtooth framework. The primary purpose of developing a decentralized voting system upon an already existing distributed ledger technology has therefore been achieved.

Its core attributes stated in the objective in Section 1.1, *security* and, *robustness* have been realized through the development of a consensus algorithm and transaction processor. These ensure that trusted parties validate each vote and, when cast, cannot be tampered with. As the system is aimed to be transparent and anonymous, it is made sure that a vote is counted upon placement while it is not possible to connect a vote or key to a voter. The confirmation of a vote is by showing the state of an election after casting a vote.

On the other hand, as a result of the administrative limitations described in Section 7.1, there is no way of making sure that the distribution of keys is accurate as there is a central authority involved in that state. Although the blockchain technology does not allow any committed ledger to be tampered with, there is no guarantee that a key cannot be traced back to a voter. The system could be developed further to include the registration of voters using some form of government-issued identification, such as a driver's license. In addition, different cryptographic techniques can be used to ensure anonymity. Some techniques that might be considered are zero-knowledge proof, blind signature, homomorphic encryption, or ring signature.

The current state of the system cannot be used in a large-scale or a national election, particularly in countries with a large population due to the limitations presented above. However, it could be used in smaller voting schemes.

Bibliography

[1] S. M. Atuobi, ”Election-related violence in Africa”, *Conflict Trends*, pp. 10–15, 2008, ISSN: 1561-9818. [Online]. Available: `https://journals.co.za/content/accordc/2008/1/EJC16002`.

[2] Valmyndigheten, ”Årsredovisning för Valmyndigheten”, Report, 2014. [Online]. Available: `https://www.val.se/om-oss/vart-uppdrag/arsrapporter.html`.

[3] Vallagskommittéen, *E-röstning och andra valfrågor*. Statens Offentliga Utredningar, 2013, ISBN: 978-91-38-23922-3. [Online]. Available: `https://www.riksdagen.se/sv/dokument-lagar/dokument/statens-offentliga-utredningar/e-rostning-och-andra-valfragor_H1B324`.

[4] M. Kitsing, ”An Evaluation of E-Government in Estonia”, in *conference “Internet, Politics and Policy*, 2010, pp. 16–17.

[5] W. Stallings, *Cryptography and Network Security: Principles and Practice*, 5th. Prentice Hall Press, 2010, ISBN: 0136097049.

[6] S. Nakamoto, ”Bitcoin: A Peer-to-Peer Electronic Cash System”, *Cryptography Mailing list at https://metzdowd.com*, 2009.

[7] G. Coulouris, J. Dollimore, T. Kindberg, and G. Blair, *Distributed Systems: Concepts and Design*. Addison-Wesley Publishing Company, 2011, ISBN: 0132143011.

[8] M. J. Fischer, N. A. Lynch, and M. S. Paterson, ”Impossibility of distributed consensus with one faulty process”, *Journal of the ACM*, vol. 32, no. 2, pp. 374–382, 1985, ISSN: 0004-5411. DOI: `10.1145/3149.214121`.

[9] D. Sorensen. (2019). On FLP Impossibility. Accessed: 2020-04-23, [Online]. Available: `https://medium.com/pyrofex/on-flp-impossibility-c0280bb965da`.

[10] Stellar, ”Safety, liveness and fault tolerance - the consensus choices”, 2014. [Online]. Available: `https://www.stellar.org/blog/safety-liveness-and-fault-tolerance-consensus-choice` (visited on 05/04/2020).

[11] B. Asolo, ”Double-Spending Explained”, 2018. [Online]. Available: `https://www.mycryptopedia.com/double-spending-explained/` (visited on 04/24/2020).

[12] M. Castro and B. Liskov, ”Practical byzantine fault tolerance and proactive recovery”, *ACM Transactions on Computer Systems (TOCS)*, vol. 20, no. 4, pp. 398–461, 2002, ISSN: 0734-2071. DOI: `10.1145/571637.571640`.

[13] A. Tar, ”Proof-of-Work, Explained”, 2018. [Online]. Available: https://cointelegraph.com/explained/proof-of-work-explained.

[14] J. Frankenfield, ”Proof of Work”, 2018. [Online]. Available: https://www.investopedia.com/terms/p/proof-work.asp.

[15] F. Saleh, ”Blockchain Without Waste: Proof-of-Stake”, *SSRN Electronic Journal*, 2018, ISSN: 1556-5068. DOI: 10.2139/ssrn.3183935.

[16] O. Vashchuk and R. Shuwar, ”Pros and cons of consensus algorithm proof of stake. Difference in the network safety in proof of work and proof of stake”, *Electronics and Information Technologies*, vol. 9, no. 9, pp. 106–112, 2018, ISSN: 2224-087X. DOI: 10.30970/eli.9.106.

[17] A. Rosic, ”Proof of Work vs Proof of Stake: Basic Mining Guide”, *Blockgeeks*, 2017. [Online]. Available: https://blockgeeks.com/guides/proof-of-work-vs-proof-of-stake/ (visited on 02/19/2020).

[18] G. Konstantopoulos, ”Understanding Blockchain Fundamentals, Part 3: Delegated Proof of Stake”, 2018. [Online]. Available: https://medium.com/loom-network/understanding-blockchain-fundamentals-part-3-delegated-proof-of-stake-b385a6b92ef (visited on 05/07/2020).

[19] M. Huillet, ”Bitcoin Will Follow Ethereum And Move to Proof-of-Stake, Says Bitcoin Suisse Founder”, 2020. [Online]. Available: https://cointelegraph.com/news/bitcoin-will-follow-ethereum-and-move-to-proof-of-stake-says-bitcoin-suisse-founder.

[20] B. Curban, *What is Practical Byzantine Fault Tolerence?*, Accessed: 2020-04-23, 2020. [Online]. Available: https://blockonomi.com/practical-byzantine-fault-tolerance/.

[21] P. Hooda, ”practical Byzantine Fault Tolerance(pBFT)”, 2018. [Online]. Available: https://www.geeksforgeeks.org/practical-byzantine-fault-tolerancepbft/ (visited on 05/07/2020).

[22] N. Singh, ”Introduction to Permissioned Blockchains”, *101 Blockchains*, 2019. [Online]. Available: https://101blockchains.com/permissioned-blockchain/ (visited on 03/15/2020).

[23] S. Ray, ”Federated Byzantine Agreement”, *towards data science*, 2018. [Online]. Available: https://towardsdatascience.com/federated-byzantine-agreement-24ec57bf36e0 (visited on 04/23/2020).

[24] D. Schwartz, N. Youngs, and A. Britto, ”The Ripple Consensus Algorithm”, 2018. [Online]. Available: https://xrpl.org/consensus-research.html (visited on 04/23/2020).

[25] Ripple, *RippleNet*, Accessed: 2020-05-07, 2015. [Online]. Available: https://ripple.com/ripplenet.

[26] XRP Ledger, *Introduction to Consensus*, Accessed: 2020-03-25, 2020. [Online]. Available: https://xrpl.org/intro-to-consensus.html.

[27] Stellar, *Projects and Partners - Stellar*, Accessed: 2020-05-07, 2020. [Online]. Available: https://www.stellar.org/ecosystem/projects.

[28] K. Olson, M. Bowman, J. Mitchell, S. Amundson, D. Middleton, and C. Montgomery, ”Sawtooth: An Introduction”, 2018. [Online]. Available: https://www.hyperledger.org/resources/white-papers.

[29] B. Ampel, M. Patton, and H. Chen, ”Performance Modeling of Hyperledger Sawtooth Blockchain”, IEEE. DOI: 10.1109/isi.2019.8823238.

[30] M. Saad, J. Spaulding, L. Njilla, C. Kamhoua, S. Shetty, D. Nyang, and A. Mohaisen, ”Exploring the Attack Surface of Blockchain: A Systematic Overview”, 2019. [Online]. Available: https://arxiv.org/abs/1904.03487 (visited on 03/23/2020).

[31] D. Anderson, *Hyperledger Sawtooth Blockchain Security (Part Three)*, 2018. [Online]. Available: https://www.hyperledger.org/blog/2018/12/07/hyperledger-sawtooth-blockchain-security-part-three.

[32] D. Anderson, *Hyperledger Sawtooth Blockchain Security (Part Two)*, 2018. [Online]. Available: https://www.hyperledger.org/blog/2018/11/23/hyperledger-sawtooth-blockchain-security-part-two.

[33] Hyperledger, *Summary of Available SDKs*, Accessed: 2020-04-27, 2019. [Online]. Available: https://sawtooth.hyperledger.org/docs/core/releases/latest/app_developers_guide/sdk_table.html.

[34] The ZeroMQ project, *Pure Java ZeroMQ*, Accessed: 2020-04-27, 2020. [Online]. Available: https://github.com/zeromq/jeromq.

[35] S. Acharya, *Mastering Unit Testing Using Mockito and JUnit.* Packt Publishing, 2014, ISBN: 1783982500.

[36] J. Willis, ”Docker and the three ways of DevOps”, *Docker Blog*, 2015. [Online]. Available: https://goto.docker.com/rs/929-FJL-178/images/20150731-wp_docker-3-ways-devops.pdf (visited on 05/11/2020).

[37] FasterXML LLC, *Jackson*, Accessed: 2020-04-27, 2020. [Online]. Available: https://github.com/FasterXML/jackson.

[38] N. Mihajlovski, *Rapidoid - Extremely Fast, Simple and Powerful Java Web Framework!*, Accessed: 2020-04-27, 2018. [Online]. Available: https://www.rapidoid.org/.

[39] P. Vorbach, *Download statistics for packages react, vue, @angular/core, svelte, jquery*, Accessed: 2020-04-18, 2019. [Online]. Available: https://npm-stat.com/charts.html?package=react&package=vue&package=%5C%40angular%5C%2Fcore&package=svelte&package=jquery&from=2015-12-30&to=2019-12-30.

[40] P. V. Stephen J. Collings Matthew Honnibal, *React Bootstrap*, Accessed: 2020-04-21, 2020. [Online]. Available: https://react-bootstrap.github.io/.

[41] Axios, *Axios*, Accessed: 2020-04-29, 2020. [Online]. Available: https://github.com/axios/axios.

[42] ReactJS, *Introducing Hooks – React*, Accessed: 2020-04-14, 2020. [Online]. Available: https://reactjs.org/docs/hooks-intro.html.

[43] Hyperledger, *Sawtooth pBFT Consensus RFC*, Accessed: 2020-03-23, 2018. [Online]. Available: `https://github.com/hyperledger/sawtooth-rfcs/blob/master/text/0019-pbft-consensus.md`.

[44] Hyperledger, ”Sawtooth Consensus Engine RFC”, Report, 2018. [Online]. Available: `https://github.com/hyperledger/sawtooth-rfcs/blob/master/text/0004-consensus-api.md`.

[45] K. Vassil and T. Weber, ”A bottleneck model of e-voting: Why technology fails to boost turnout”, *New Media & Society*, vol. 13, no. 8, pp. 1336–1354, 2011. DOI: `10.1177/1461444811405807`.

[46] K. Vassil, M. Solvak, P. Vinkel, A. H. Trechsel, and R. M. Alvarez, ”The diffusion of internet voting. Usage patterns of internet voting in Estonia between 2005 and 2015”, *Government Information Quarterly*, vol. 33, no. 3, pp. 453–459, 2016. DOI: `10.1016/j.giq.2016.06.007`.

[47] S. L. Bränström, ”Valet kostar halv miljard”, *SvD Näringsliv*, 2010. [Online]. Available: `https://www.svd.se/valet-kostar-halv-miljard` (visited on 04/22/2020).