École Polytechnique Fédérale de Lausanne

DEDIS

D-voting Production Readiness

by Amine Benaziz, Albert Troussard

Bachelor Project Report

Supervised by:

Bryan Ford
Project Advisor

Noémien Kocher, Pierluca Borsò
Project Supervisors

January 5, 2023

Abstract

Electronic voting systems have the potential to improve the transparency and trustworthiness of elections by using blockchain technology to provide a secure record of all votes casted. DEDIS has been a pioneer in the e-voting field with its D-Voting project[3], an e-voting system based on the high-level blockchain components in the DELA project[4].

In our work, we thoroughly implemented a series of automatic tests to fully evaluate the correctness, performance, robustness of the system in realistic scenarios and we were able to successfully improve its usability. While our work allowed the system to be overall more reliable, secure, and easy to use, a problem when the system is receiving numerous transactions in a short time has been discovered and important leads have been found towards its resolution .

Overall, our work on the D-Voting system has helped to ensure that it is secure, usable, and reliable, and can soon be ready for use in high-stakes elections where trust is of the utmost importance.

Contents

Chapter 1

Introduction

1.1 Work

Decentralized voting systems have the potential to revolutionize the way we conduct elections. By leveraging the power of blockchain technology, decentralized voting systems can provide a transparent, secure, and accessible platform for voting. The D-Voting project has been initated in summer 2021 and since then, numerous master and bachelor students have worked on it in various fields such as front-end development or the implementation of the E-Voting. Our objective is to bring the product to production readiness – that is, making it ready for deployment in a real-world election –. We thus have to take a step back and consider all the previous student's work as one project and improve it if needed. However, bringing a decentralized voting system to production readiness is no small feat. It requires addressing a wide range of technical and logistical challenges, and considering a variety of factors that can impact the system's performance and reliability.

One of the key considerations when evaluating the production-readiness of a decentralized voting system is security. Ensuring the security of the voting process is crucial to maintaining the integrity of the election and the trust of the voters to avoid for instance leaks of some ballot's content or the corruption of other ones. This requires careful attention to the design of the system, including the use of secure cryptographic protocols and the implementation of robust security measures. It also requires extensive testing and verification to ensure that the system is resistant to various types of attacks and vulnerabilities.

Another important factor is scalability. A decentralized voting system needs to be able to handle the large volume of transactions that can occur during an election, especially if the system is being used by a large number of voters. This requires the system to be designed in a way that allows it to scale up as needed, without sacrificing security or performance.

Usability is also an important consideration. In order for a decentralized voting system to be successful, it must be easy for voters as well as operators to use . This means that the system should be intuitive and user-friendly, with clear instructions and guidance for voters.

In addition to these technical factors, there are also regulatory considerations to take into account. Different institutions have different laws governing the use of electronic voting systems, and it is important for a decentralized voting system to be compliant with these rules. This may require adapting the system to meet specific requirements, such as verifying the identity of voters or ensuring the privacy of the vote.

In our work on the production-readiness of a decentralized voting system, we focused on these factors and combined them with other factors that can impact the system's performance and reliability. Our findings showed that the system we studied has strong security and is generally user-friendly. However, there are also areas where the system could be improved, such as in terms of scalability or accessibility for people with disabilities.

Overall, the production-readiness of a decentralized voting system is a complex and multifaceted issue, requiring consideration of careful attention to a wide range of technical, logistical, and regulatory considerations. By addressing these challenges and continuously improving the system, we can bring decentralized voting systems closer to being a viable option for the future.

1.2 Pre-project observation

At the beginning of the semester, the system was functioning properly when it was run manually. However, we observed that there were not many tests conducted under realistic conditions, which raised concerns about the reliability and performance of the system under heavy load.

Without thorough testing, it was difficult to be confident that the system would function properly in a production environment, where it would be subjected to a high volume of transactions and potentially adversarial attacks. As such, we decided to focus our efforts on testing the correctness, performance, and robustness of the system under a variety of realistic scenarios.

1.3 Methodology

The process we followed in our journey, which was inspired by the "Scrum methodology", involved identifying problems or opportunities for improvement, developing solutions to address these issues, and then testing and refining these solutions in a series of short, focused efforts.

To begin, we would conduct tests or observations to gather data and identify areas in need of improvement. We would then brainstorm and prototype potential solutions, choosing the most promising option to implement and test. After testing, we would evaluate the results and determine whether the solution was successful in addressing the identified problem. If the solution was not successful, we would go back to the beginning of the process and repeat it with a new solution. If the solution was successful, we would move on to the next problem or opportunity for improvement. By following this iterative and incremental process, we were able to continually improve and refine our solution, resulting in a more robust and effective final product.

To provide a comprehensive overview of our project, we have organized our report into three main sections that are both chronological and reflect the pillars that structured our work : Usability, Robustness & Performance, Evaluation.

- The Usability section will describe how we started exploring the project in its surface by tackling issues related to the "cleanliness" of the code and how it is presented to the user. This section will describe how we started exploring the subject by implementing new metrics giving us insights about the quality of the code and some of its vulnerabilities (code smells, code coverage ...etc), discover issues related to usability and how we overcame these challenges with our data and observations.
- The Robustness & Performance section goes a bit on a higher level of the code since it involves the integrity of the entire system.The section details the steps we took to create our own means of judgement of the quality and the robustness i.e our own tests and to make an in-depth diagnosis of the code, how we discovered issues and vulnerabilities thanks to them and how we managed to solve these issues.
- The Evaluation section is used to give a retrospective on our work, how it improved the overall system on numerous aspects, the issues we encountered at each step and how we managed to solve them. It presents the outcome of our solution, including any data or analysis that demonstrates its effectiveness in addressing the identified problems.

By dividing the report this way, we aim to provide a clear and thorough understanding of the process we followed in our development and the results we achieved to gather because it also reflects our work on a chronological order. Indeed, we first started exploring the system on a lower-level by facing issues related to usability or code smells that do not instantly require us a full understanding of the project as a whole but can be used as a smooth introduction

towards it. Secondly after gaining a stronger knowledge on the system, we started tackling more technical issues like the code coverage, the authentication system or the optimization of the overall performance. Finally, we gave the Evaluation section to provide a good understanding of the impact our work had on the project.

Chapter 2

Background

2.1 Overall System

D-voting is an e-voting platform that is based on the DELA blockchain, which is also a project of the DEDIS LAB. It allows to the voting process to take place in a decentralized system guaranteeing the privacy of votes.

To launch an election we firstly need to create it, either manually by filling the type of question, the subject, the answers..etc or automatically with a JSON file. Then, we have to initialize the nodes and set them up which will also launch the encryption process and the generation of a key-pair. After that, the admin can create the election and close it at any time, which will launch the process of shuffling and decrypting the ballots.

To give an idea of how the D-Voting platform works, we first have to understand its components. The system is made of two back-ends, one that implements all the blockchain logic, and another one that corresponds to the web-back-end and the server's logic that is used, among other things, for the authentication process. Finally, the front-end part is used to "build" the system in a real-world, i.e mainly help with the interactions with real-world users such as authenticating users, managing elections and rights, voting ...

2.2 Smart Contract

The electronic voting process, as implemented through a smart contract on the DELA blockchain developed by DEDIS, involves storing data on each node of the blockchain and executing various steps in the voting process by adding transactions to the chain. To mitigate the strain on the network, a protocol allowing to include in each transaction a timestamp, sign it and authenticate it has been made. While this system has been designed to be fault-tolerant, it is important to

consider the potential challenges that may arise in a real-world setting.

Overall, the use of smart contracts and blockchain technology in the voting process offers numerous potential benefits. Smart contracts can help to increase transparency in the election process by providing a record of all votes casted and the results of the election. This can help to reduce the risk of fraud and ensure that the results of the election are accurate. They can help to improve the security of the election process by using cryptographic techniques to protect against tampering and unauthorized access. Furthermore, they can make it easier for people to participate in elections, even if they are unable to physically go to a polling place which can help to increase voter turnout.

2.3 Distributed Key Generation

The D-voting system implemented the Pedersen Distributed Key Generation (DKG) protocol as a way to ensure the secrecy of votes. This method generates a key pair in a decentralized manner, with the following properties:

Public key: available to all parties, used to encrypt the contents of the votes Private key: divided into shares and distributed among the system's actors, a threshold of private shares is required to decrypt the voting results after the shuffling step that separates the voters and their votes Our project also utilized DKG to create a secure and decentralized communication system, with the secret key being divided into shares and the nodes combining their shares to decrypt messages. Additionally, our system includes the ability for the nodes to produce a cryptographic proof of decryption to verify the accuracy of the decryption process and prevent tampering with the encrypted information.

Overall, the use of the Pedersen DKG protocol in decentralized voting systems helps to ensure the security and integrity of the voting process, allowing for transparent and verifiable elections[6].

2.4 Neff Shuffle

The Neff Shuffle protocol is a cryptographic technique that is used to securely shuffle a set of values in a decentralized manner. It is very important in electronic voting systems because it helps to ensure the anonymity and integrity of the voting process[5].

In a decentralized voting system that uses the Neff Shuffle protocol, voters are able to cast their votes anonymously, without revealing their identity or the content of their vote to any other party. This is achieved through the use of a series of cryptographic operations, including the generation of a unique "commitment" for each vote and the use of zero-knowledge proofs.

Zero-knowledge proofs are a type of cryptographic proof that allow one party (the prover) to prove to another party (the verifier) that they possess certain information without revealing the actual content of the information.

Once all of the votes have been cast, the Neff Shuffle protocol is used to shuffle the commitments in a random order. This helps to prevent any correlations between the commitments and the votes they correspond to, further protecting the privacy of the voting process. The shuffled commitments are then revealed to the network and used to determine the final vote tally. The Neff Shuffle protocol allows the network to verify the validity of the commitments and the accuracy of the final vote tally, helping to ensure the integrity of the voting process.

Overall, the Neff Shuffle protocol is an important component of decentralized voting systems, helping to ensure the privacy and integrity of the voting

Chapter 3

Usability

3.1 Bad metrics

3.1.1 Problem

One of the issues we faced during the development of the project was a lack of significant metrics due to improper configuration of our tools, especially SonarCloud . This meant that we were unable to accurately determine for example the code coverage or the number of code smells which is problematic because it makes it difficult to evaluate the completeness of the testing. Without sufficient testing, it is more likely that bugs and other issues will go unnoticed and may not be addressed until the system is deployed. This can lead to disruptions and failures in the system, which can be costly and frustrating for users.

3.1.2 Fix

To address the issue of bad metrics, we correctly configured SonarCloud so that tests are correctly labeled. By correctly setting up SonarCloud, we were able to accurately measure the performance and reliability of the system and identify any areas that needed improvements.

3.2 Difficulty of Usage

3.2.1 Problem

One of the first challenges we faced during our development project was the complexity of the setup process. In order to run the system, you had to follow these steps:

- Run the blockchain nodes: This involved opening a terminal window, navigating to the project root and run a script
- Setup the blockchain nodes: After the completion of first step you need to run another script to setup the nodes
- Run the backend: This required opening a new terminal window and navigating to the backend's folder and run a command
- Run the frontend: This involved opening another terminal window and navigating to the frontend's folder and run a command

The complexity of the setup process was a problem for several reasons.

First, it made it difficult for team members to get started with the project, which slowed down progress and increased frustration. Second, it increased the risk of errors and misconfigurations, which could lead to issues further down the line. Thirdly, even when we started to understand how to setup the system, it was really time consuming each time we made a modification and wanted to see the result. Finally, when a system is less understandable and complex to setup or use, it can be intimidating and prevent a user to adopt the system.

Overall, the complexity of the setup process was a significant challenge that we needed to address in order to make the system more user-friendly.

3.2.2 Fix

To solve the challenge of the complex and confusing setup process, we created a script that automated the steps needed to get the project up and running and with the possibility to avoid some of them by adding parameters.

By creating this script, we were able to significantly streamline the setup process. Instead of having to manually open multiple terminal windows and navigate to different folders, users could simply run the script and have all of the necessary components and services launched automatically in the same tmux session. This made it much easier for users to get started with the project, which improved efficiency and reduced frustration.

Chapter 4

Robustness and performance

4.1 Testing

One of the major discoveries introduced by the implementation of new code metrics was the relatively low code coverage on certain aspects of the system, especially the back-end. We had at some point a 20% overall coverage of some parts of the code base. Hence, we choose to focus mainly on testing the functionality of the system to make sure the system is working correctly, we have looked deeper into the testing part and worked on it gradually such that we started on unit tests, continued on integration tests and finally implemented load tests and benchmarks to have a thorough understanding on the integrity of the system.

4.1.1 Unit tests

Firstly, we started by writing unitary tests to ensure that each section of the D-voting system works and behaves as designed. The use of unit tests can help to improve the reliability and security of the D-voting system by identifying and fixing issues early in the development process. Furthermore, we wanted to put a strong emphasis on some parts of the back-end that were poorly tested but critical for the rest of the project, among them the code related to the Distributed Key Generation which is an essential part of the system but also the proxy.

4.1.2 Integration Tests

Having gained a stronger understanding on the robustness of the code with the unit tests, we needed to test these modules together. Initially the integration tests were not exposing any failures in the program but an investigation has to be done to make sure the tests were effectively

testing cases and naturally test even more cases since the older version was limited to only a single case.

Hence, we wanted to write new integration tests. However, we discovered that the initial test was a bit flaky and failed randomly (if it failed, we sometimes only had to re-run the test to make it pass). Hence, we ended up restructuring the whole part dedicated to integration testing. From the Fakes that were not using every type of possible forms to the tests themselves that were deeply updated and modified.

We implemented new test cases such as voting with Null ballots, modifying the nodes' structure (crashing multiple ones during an election). But also testing the integrity of the system when a vote is changed multiple times and checking that it has been effectively casted and taken into account during the counting of the votes.

4.1.3 Scenario Test

The scenario test is used to simulate a real election simulating the interactions of the user with a proxy. It emulates an election from the start with the creation of the nodes and the setting up of the DKG to the end with the decryption of the ballots and includes the proxy's interactions, the marshalling of the ballots ... etc.

However, the main disadvantage of this scenario is while it simulated realistically an election recreating the entire process with proxies instead of real users, the voting rate was quite low and we always had to wait for a transaction to be accepted before submitting a newer one to avoid flooding the system. Hence, we also needed a test to simulate the voting rate of a real world election.

4.1.4 Load Test

Therefore, we decided to test these new integration tests on larger samples with a load test. A load test is a type of performance testing that is used to determine the behavior of a system under a specific expected load or workload[1]. In the case of the new integration tests for the distributed voting system, it was decided to test these tests on larger samples to more closely mimic a real-world implementation of the system. It is very similar to the Scenario Test described above but the main difference is in how the sending of transactions is handled, the Scenario Test waits for the previous transaction to be accepted before submitting a newer one while the Load Test does not take this into consideration and submit them anyway to increase the realism of the test this way :

```
1
2  switch testType {
3    case SCENARIO:
4     votesfrontend = castVotesScenario(numVotes, BallotSize, ...)
5     t.Log("Cast votes scenario")
6    case LOAD:
7     votesfrontend = castVotesLoad(numVotes/numSec, numSec, BallotSize,...)
8     t.Log("Cast votes load")
9    }
```

The purpose of performing a load test in this situation is to ensure that the system can handle the expected level of traffic and usage without experiencing any performance issues or failures. This is an important step in the development and deployment process, as it helps to identify and address any potential bottlenecks or vulnerabilities that may impact the system's reliability and stability.

To conduct the load test, a specific workload or load scenario was likely defined and used to simulate the expected level of usage on the system. This includes a specific number of transactions being processed within a certain time frame (we tested numerous possibilities such as 2 votes per seconds during 60 seconds to mimic an extreme case close to the closing of an election where everybody wants to cast or change his vote). The system was then monitored and tested under these conditions to determine its behavior and performance. Furthermore, we added the capacity for a vote to be resubmitted after a certain time (30 seconds) if it has not been included yet.

Overall, the goal of the load test was to ensure that the distributed voting system is able to handle the expected level of usage and traffic, and to identify any potential issues or weaknesses that may need to be addressed before the system is deployed in a real-world setting.

4.2 Profiling

Another challenge we faced during our development was the performance of the system when conducting integration tests with a large number of nodes and ballots. To better understand this issue and identify potential solutions, we decided to run a profiling of the system with 10 nodes and 100 ballots using the pprof tool[2].
You can find below the header of this profiling, which includes the relevant information about the test environment and the results of the profiling:

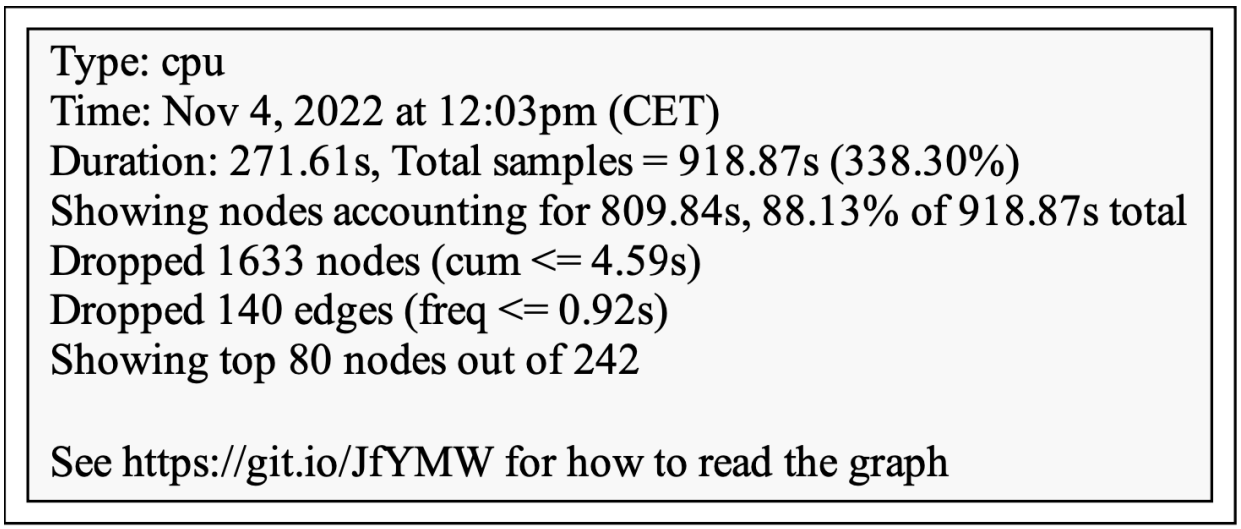


Figure 4.1: Header of the profiling

As you can see the total time sampled for this profiling is around 918s. The resulting graph was not really satisfying since it was not clear at all and contained a lot of irrelevant data because most of the time is spent doing cryptographic operations in DELA. But something seemed odd as you can see in the image below:

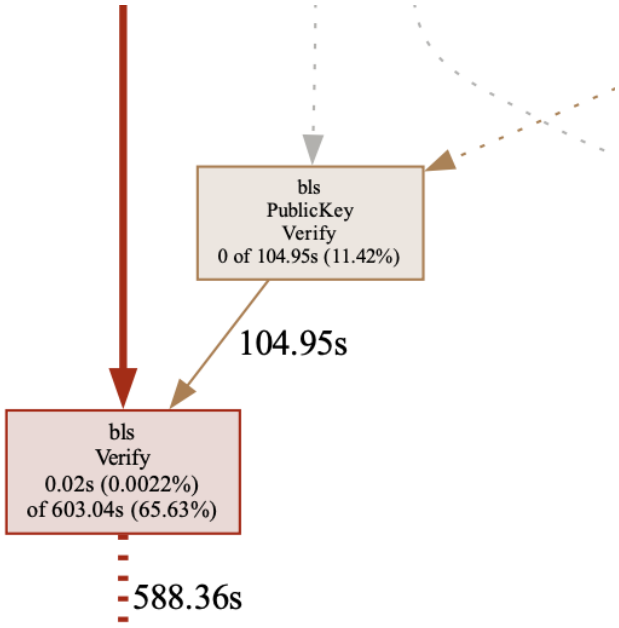


Figure 4.2: Extract of the profiling

The results of the profiling showed that a significant amount of time (104.95s, or approxi-

mately 11.4% of the total time) was being spent on a particular section of the code. This section of the code was responsible for generating a proof from the blockchain, and it required the system to retrieve a proof from every block of the blockchain, as well as a proof for every transaction in those blocks.

This was problematic for several reasons. First, the process of retrieving a proof from every block and transaction was time-consuming, which slowed down the overall performance of the system. Second, the high volume of data that needed to be processed made it more difficult for the system to handle a high volume of transactions, which could impact its reliability in a production environment.

Overall, the results of the profiling indicated that the process of generating a proof from the blockchain was a significant performance issue that needed to be addressed in order to ensure the efficiency and reliability of the system.

Upon further investigation, we determined that the issue with the performance of the system when generating a proof from the blockchain was due to the way that the DELA blockchain was designed.
Fortunately, this issue has since been addressed and fixed by the DELA team.

4.3 Authorization

4.3.1 Problem

Our authentification and authorization system is based on Tequila, hence, we can use the SCIPERs to check if a user is indeed a member of the EPFL organization or not. Nevertheless, we managed to discover that during the rights lending process, an administrator could give rights to any number as a SCIPER and that no checks were made on this number passed as a SCIPER. This could pose problems since we could have at some points numerous admins that are not even real and elections runned by no one.

4.3.2 Fix

In order to fix this issue, we decided to check the SCIPER before all. Thus, we first started by checking it with the size of the number given in the input then established a direct communication with the Tequila API and used the results of this communication to conclude on the validity of a SCIPER.

```
1
2   axios
3     .get(`https://search-api.epfl.ch/api/ldap?q=${sciper}`)
4     .then((response) => {
5       if (response.data.length === 0) {
6         res.status(400).send('Unknown Sciper');
7         return;
8       }
9
10      usersDB
11        .put(sciper, role)
12        .then(() => res.status(200).send('Role added'))
13        .catch((error) => {
14          res.status(500).send('Failed to add role');
15          console.log(error);
16        });
17    })
18    .catch((error) => {
19      res.status(500).send('Failed to check Sciper');
20      console.log(error);
21    });
```

4.4 Transaction Mechanism

4.4.1 Problem

During our testing of the system, we encountered a problem with the transaction mechanism. Upon conducting a load test by sending 2 votes per second for 60 seconds, we observed that the number of ballots the client perceived as being accepted did not match the number of ballots that were actually added to the blockchain. Upon further investigation, we found that the issue was caused by the way transactions are handled by the proxy.

This issue is significant because it can potentially lead to discrepancies in the voting process, affecting the integrity of the results. To address this problem, we performed an in-depth analysis of the transaction mechanism, as illustrated in the following diagram:

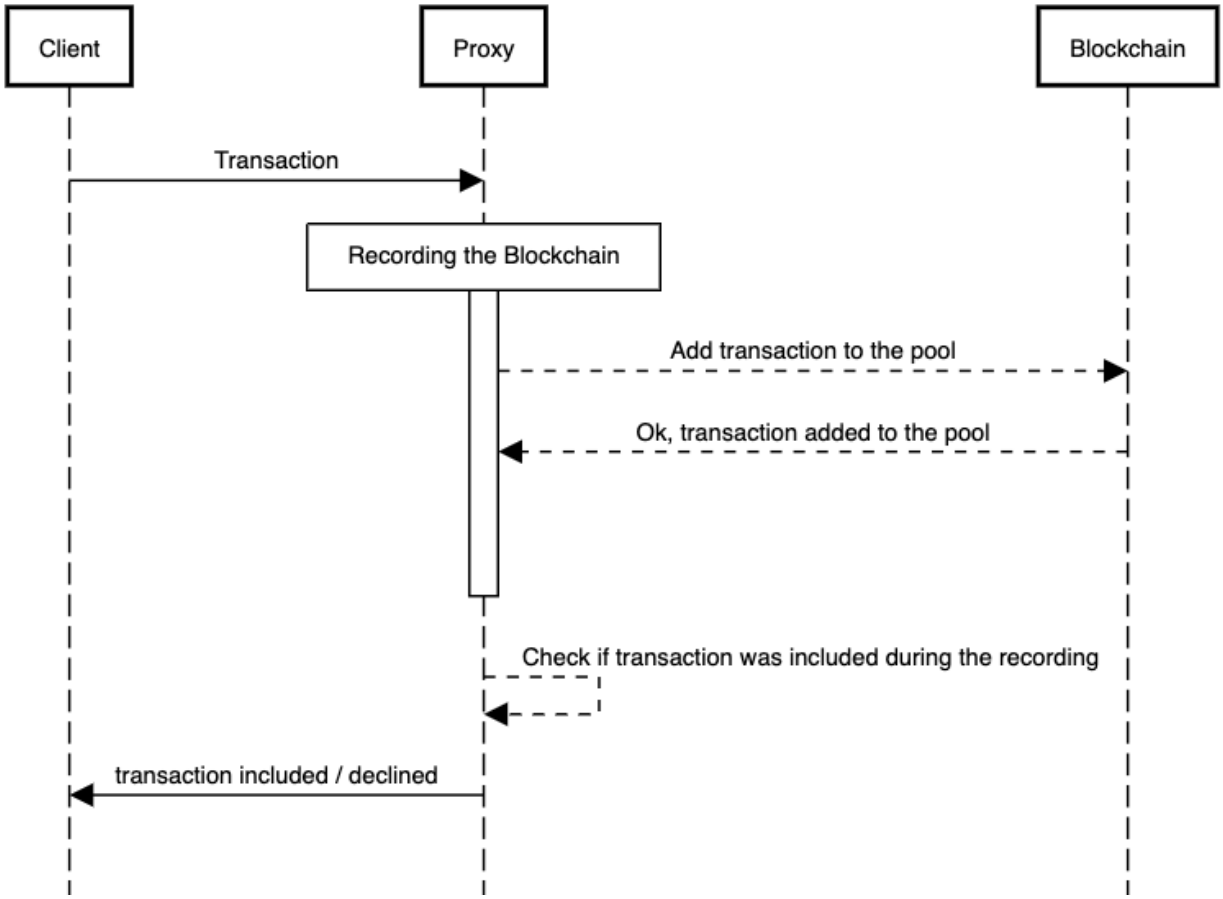


Figure 4.3: Previous Transaction System

As shown in the graph, the proxy waits for the transaction to be added to the blockchain before sending a response to the client. This can lead to delays for the client, even if the transaction is quickly added to the blockchain. Additionally, if the transaction takes too long to be added to the blockchain, the proxy will not be able to record it and the client will receive a declined response, leading the client to believe that the transaction will never be added to the blockchain when it may still be able to be added at a later time.

Another issue with this system is that while the proxy is busy recording transactions to the blockchain, it is unable to handle transactions from other clients, leading to a lack of scalability and making it vulnerable to denial of service attacks.

4.4.2 Fix

To address the issues with the current transaction mechanism, we have proposed a solution to change the way the proxy handles transactions. Under the new system, the proxy will no longer

maintain a state of the transactions and will instead respond to the client more quickly. This will eliminate delays for the client and improve the overall efficiency of the system. The new transaction mechanism is illustrated in the following diagram:

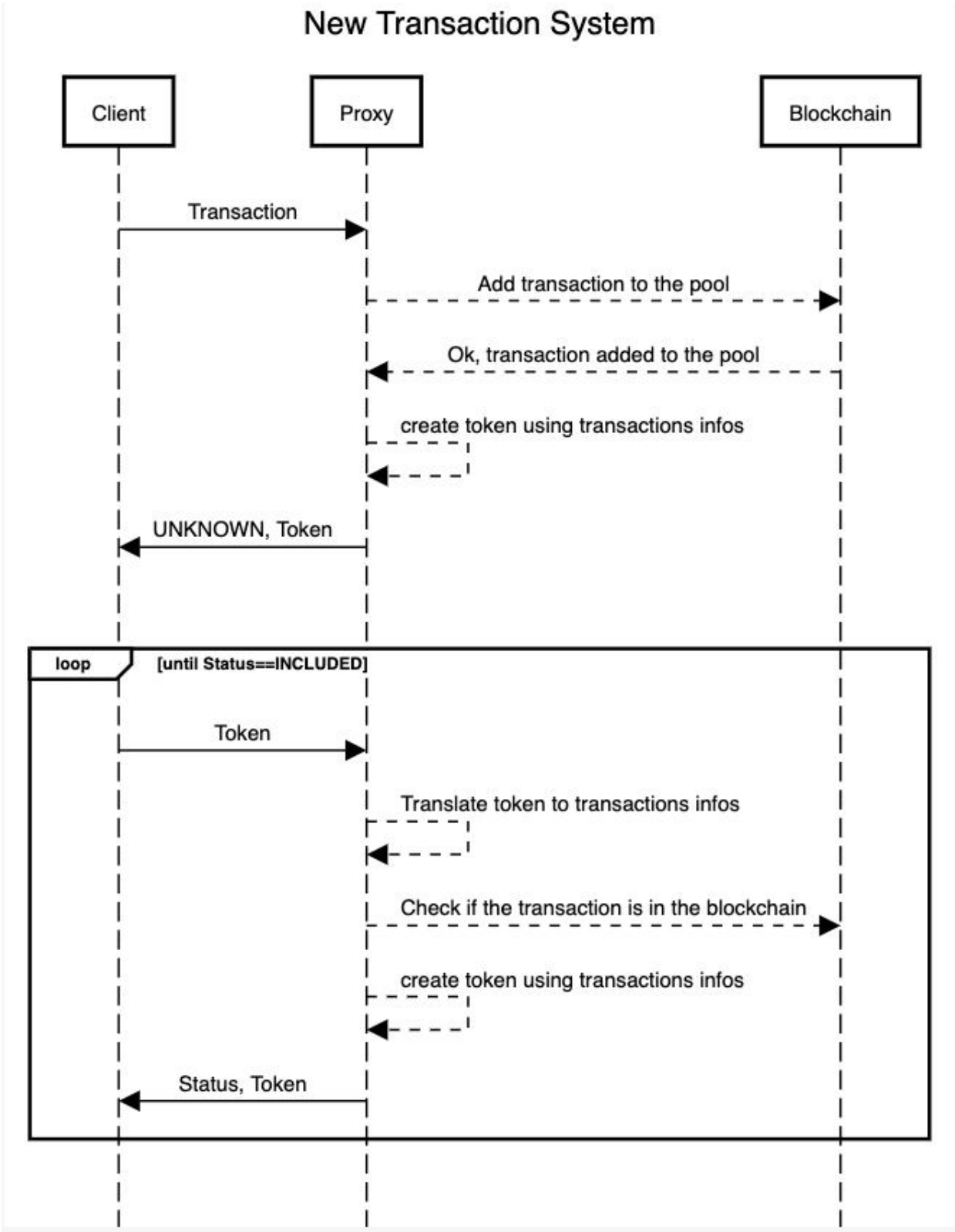


Figure 4.4: New Transaction System

Under the new transaction mechanism, when a client wants to send a transaction to the proxy, the proxy will verify that the transaction has been added to the blockchain pool and respond to the client with a token. This token is the URL encoding of a JSON object, which includes the following information:

```
{
    "Status"      :   TransactionStatus ,
    "TransactionID": []byte,
    "LastBlockIdx" :  uint64,
    "Time" :          int64,
    "Hash" :          []byte,
    "Signature"   :   []byte
}
```

- Status: the status of the transaction (UNKNOWN, INCLUDED, DECLINED)
- TransactionID: the ID of the transaction
- LastBlockIdx: the index of the last block before the transaction was added to the pool
- Time: the time when the transaction was added to the pool
- Hash: the hash of all the previous fields
- Signature: the signature of the hash

The Time field can be used to prevent a client to ask for the inclusion of a transaction submitted too long ago and that will force the proxy to read a big part of the blockchain, which would take a lot of time.

The Signature is here to ensure that the token was indeed created by a proxy.

The Status is UNKNOWN because the proxy does not yet know if the transaction has been added to the blockchain. The client can then send this token back to the proxy to check the status of the transaction. The proxy will verify the validity of the token and determine whether the transaction has been added to the blockchain. If the transaction has been included, the proxy will respond to the client with the status INCLUDED. If the transaction has not yet been included, the proxy will respond with the status UNKNOWN. If the transaction has been declined, the proxy will respond with the status DECLINED.

The proxy will also send an updated token to the client with the new status, the new LastBlockIdx (which is now the last checked block), the new Time (which is the time when the transaction

was checked) and obviously new Hash and Signature. The client can then repeat this process until the status of the transaction is INCLUDED (or DECLINED) or until a timeout is reached.

One of the key advantages of this new approach is that the signer is shared among all proxies, allowing the client to send the transaction to any proxy and for the proxy to check the status of the transaction. This increases the flexibility and reliability of the system. Additionally, by not maintaining a state of the transactions, the proxy is able to more efficiently process requests and provide faster responses to the client, improving scalability and reducing the risk of denial of service attacks.

Chapter 5

Evaluation

5.1 Transaction Mechanism

The new transaction mechanism has improved the system in several ways.

First, before the new mechanism was implemented, the proxy had to wait for 10 seconds to determine whether a transaction had been added to the blockchain. This could lead to significant delays for the client. Under the new system, the proxy no longer has to wait and is able to check each block in around 40 milliseconds, resulting in faster responses for the client.

Second, before the new mechanism was implemented, the client was not always up-to-date with the votes that were being counted. For example, during a load test where 120 votes (2 votes per second for 60 seconds) were sent, the client may have believed that only 118 votes were included, when in fact all 120 votes were added to the blockchain. This could have led to discrepancies in the voting process and compromised the integrity of the results. With the new mechanism in place, the client is able to more accurately track the status of the transactions and can be confident that all votes are being accurately counted.

Overall, the new transaction mechanism has significantly improved the performance and reliability of the system, ensuring that the voting process is accurate and efficient.

5.2 Scenario test

To further test the performance and reliability of the system, we conducted a scenario test with 100 ballots and 10 nodes. The test involved sending 100 votes to the system simultaneously and measuring the time it took for the transactions to be processed and added to the blockchain.

The results of the test are shown below:

Create form	Setup DKG	Open form	Casting 100 ballots
1.08s	3.56s	1.16s	138.39s

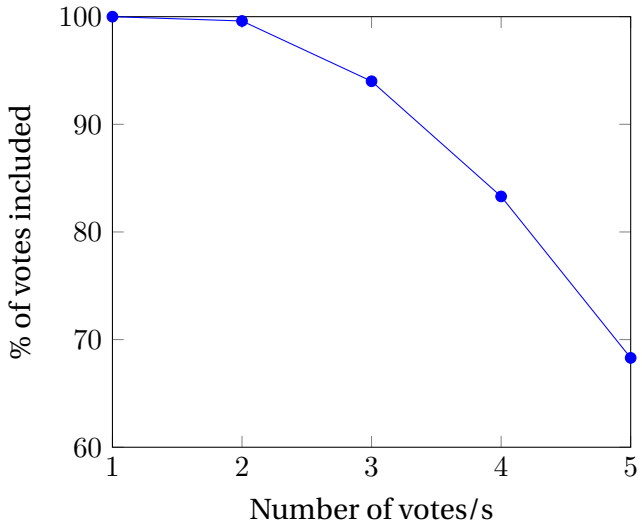
Close form	Shuffle ballots	Decrypt ballots
1.56s	31.3s	7.9s

The results presented by these tests when tested on smaller samples were overall all satisfactory since it didn't expose any new vulnerabilities or bugs in the code and the incomprehensible failures of the initial integration test were corrected. Furthermore, the time spent on every part of the election was correct and aligned with our expectations. Note that the time spent casting ballots is long because in the scenario test you have to wait until a vote is included to send the next one.

5.3 Load test

As part of our efforts to understand the system, we conducted a series of load tests with 10 nodes and different numbers of votes per second over a period of 60 seconds. The results of these tests are shown below:

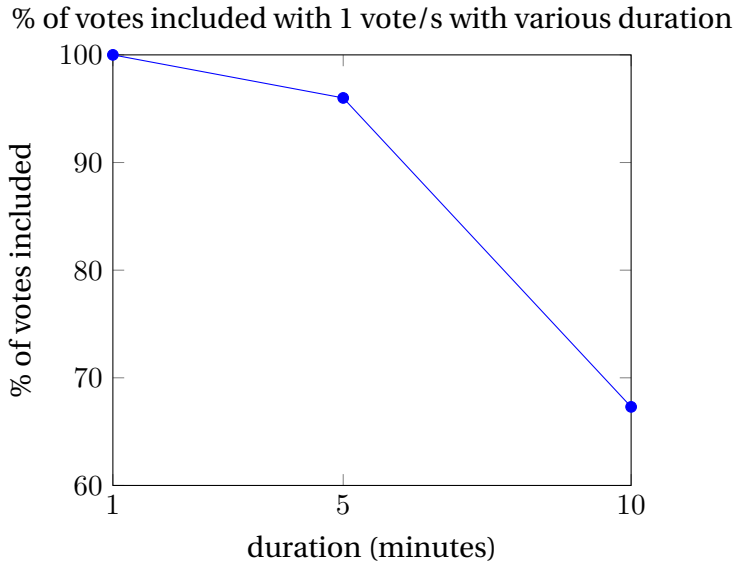
% of votes included as a function of the number of votes per second for 60 second



From the data collected, it appears that the system becomes less resilient as the number of votes per second increases. Specifically, when there are 2 votes/s or less, the system is able to include nearly all of the votes (99.6% and 100%, respectively). However, as the number of votes/s

increases to 3 and beyond, the percentage of votes included decreases significantly (to 94% and lower).

But, it is expected that votes at EPFL will induce less than 2 votes per second. It is also possible that votes at EPFL may induce 1 votes per second for a longer period of time. In order to test the system's resilience in this scenario, it is necessary to conduct tests with a variable duration to see how the system performs under different conditions. This will allow us to determine the system's limits and identify any potential weaknesses that may need to be addressed.



This plot shows that the percentage of votes included decreases as the duration of the test increases. This suggests that the system may not be able to handle a high volume of votes over a longer period of time. It may be necessary to implement measures to improve the system's resilience and ensure that all votes are properly included.

Based on the data and the plots, it is clear that the system has some limitations in terms of its ability to handle a high volume of votes in a short period of time as well as handling a reasonable load over a longer period.

Chapter 6

Future Work

6.1 Omitted transactions

During the course of our development project, we encountered a problem where some ballots were not being properly recorded and included in the blockchain when a large number of ballots were cast as showed in the Load Test Evaluation section . This was a critical issue because it could potentially impact the accuracy and reliability of the voting process. If some ballots are not properly recorded, it could lead to inconsistencies in the results and raise doubts about the integrity of the entire voting process.

In order to understand the root cause of this problem and develop a solution, we conducted a thorough investigation of the system and the data stored in the blockchain. As part of this investigation, we examined the behavior of the system under different workloads and analyzed the data stored in the blockchain.

One of the key findings of our investigation was that the size of the blockchain pool was never larger than the number of nodes, which was unexpected and raised concerns about the integrity of the voting process. We are currently working to understand why this is occurring and to identify a solution that will ensure that all ballots are properly recorded and included in the blockchain.

Overall, we believe that addressing this issue is critical in order to maintain the integrity and reliability of the voting process. We are committed to finding a solution that will ensure that all ballots are properly recorded and included in the blockchain, and we will continue to work towards this goal as part of our ongoing development efforts.

6.2 Proxy Split

As part of our efforts to improve the performance and reliability of the system, we are considering splitting the proxy component for the transaction verification into two parts: one to handle authentication of the token and one to handle all other tasks. This would allow us to optimize the performance of each component and potentially improve the overall efficiency of the system.

One potential benefit of this approach is that it would make the system more resilient to denial of service (DoS) attacks. By separating the authentication process from the rest of the tasks performed by the proxy, we could ensure that the authentication component had sufficient resources to handle a high volume of requests as it can for example easily run on the cloud for a cheap price and we could automatically adapt the number of instances running.

This would reduce the risk of the authentication process being overwhelmed by a DoS attack, which could disrupt the operation of the system.

Overall, we believe that this approach has the potential to significantly improve the performance and reliability of the system.

6.3 Batch verification

In order to ensure the scalability of the system, we are considering implementing a strategy for batching the verification of transaction inclusion. This would involve grouping multiple transactions together and verifying their inclusion in the blockchain in a single operation, rather than verifying them individually.

There are several potential benefits to this approach, here are the main ones:

- First, it would reduce the number of requests that need to be made to the blockchain, which would reduce the overall load on the system and improve its performance.
- Second, it would allow us to verify larger batches of transactions more efficiently, which could further improve the scalability of the system.
- Finally, it would allow us to process transactions more quickly, which could improve the user experience and overall usability of the system.

Overall, we believe that batching the verification of transaction inclusion has the potential to significantly improve the scalability and performance of the system, and we are excited to explore this approach further as part of our ongoing efforts to optimize the system.

6.4 Front-end testing

Our work regarding tests was majorly directed on testing the back-end functionality of the system. Hence, only a small part of the front-end could be tested. Nevertheless, Front-end testing is an important aspect of software development that ensures that the user interface of a website or application is functioning correctly and effectively. It involves testing the visual design, layout, and interactions of the D-Voting website to ensure that it meets the requirements and expectations of the end user.

Chapter 7

Conclusion

In conclusion, our Production Readiness project for the decentralized voting system has been a success in many regards. We were able to make the system easier to use, identify and fix a number of vulnerabilities, and enhance the overall quality of the code. In addition, we have conducted tests under real-world conditions that have helped us to understand the current limitations of the system and to identify areas for further improvement.

However, we did encounter one major vulnerability that caused some transactions to be omitted. While we were able to implement an alternative transaction system that partially addressed this issue, it did not fully resolve the problem.

Despite this challenge, we are confident that the decentralized voting system is now in a much stronger position to handle real-world use cases. The system can easily be monitored in the future and the work to identify and address any remaining vulnerabilities has been eased. Overall, after the patch of this problem, we believe that the decentralized voting system can be ready for production use, and we are excited to see it being used to facilitate secure and transparent elections.

Bibliography

[1] Noga Cohen. *Performance Testing vs. Load Testing vs. Stress Testing*. https://www.blazemeter.com/blog/performance-testing-vs-load-testing-vs-stress-testing#load. 2022.

[2] Google. *Pprof*. https://github.com/google/pprof. 2016.

[3] DEDIS LAB. *D-Voting GitHub Repository*. https://github.com/dedis/d-voting. 2021.

[4] DEDIS LAB. *DEDIS Ledger Architecture GitHub Repository*. https://github.com/dedis/dela. 2020.

[5] C.Andrew Neff. *A Verifiable Secret Shuffle and its Application to E-Voting*. 2001.

[6] Torben Pryds Pedersen. "AA Threshold Cryptosystem without a Trusted Party". In: (1991), p. 5.