Universidade de Aveiro Departamento de Eletrónica, Telecomunicações e Informática
2019

Luís Filipe Sobral Silva

Dignitas - Uso de reputação como moeda para avaliar a sensorização humana em Cidades Inteligentes

Dignitas - Using reputation as a coin to evaluate human sensing in Smart Cities

Universidade de Aveiro Departamento de Eletrónica,
2019 Telecomunicações e Informática

Luís Filipe Sobral Silva

Dignitas - Uso de reputação como moeda para avaliar a sensorização humana em Cidades Inteligentes

Dignitas - Using reputation as a coin to evaluate human sensing in Smart Cities

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia de Computadores e Telemática, realizada sob a orientação científica do Doutor André Zúquete, Professor auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro, e do Doutor Carlos Senna (co-orientador), Professor Investigador no Instituto de Telecomunicações .

Ao meu irmão Tomás, por não me ter deixado crescer

o júri / the jury

agradecimentos

Gostaria de agradecer ao Prof. Zúquete e ao Dr. Carlos pela orientação - as ideias, as críticas, as vírgulas e, ainda, todas as agradáveis reuniões que tivemos. Aos meus amigos e ao meu irmão, que não me deixaram enlouquecer nos últimos metros desta maratona. Quero agradecer também à minha mãe, pelo apoio incondicional durante todo o meu percurso pela Universidade. E, finalmente, à Ana Margarida, por me aturar e ser alguém com quem posso sempre contar independentemente de tudo.

Palavras Chave

Blockchain, sistemas distribuídos, reputação, confiança, privacidade, cidades inteligentes, redes veiculares

Resumo

Vivemos num mundo cada vez mais digital, onde as cidades inteligentes passaram a ser uma realidade. Uma das caracteristicas que permite a estas cidades serem inteligentes é a capacidade de adquirir informação e agir sobre ela, melhorando a vida de todos os cidadãos. Neste trabalho apresentamos o nosso sistema, *Dignitas*, um sistema de reputação baseado numa *blockchain* que permite aos cidadãos de uma cidade inteligente avaliar informação relatada por outras pessoas. Esta avaliação é baseada numa aposta feita pelo relator, e por todos os que com ele concordam, em que põe em risco parte da sua Reputação no sistema. Este uso da Reputação como uma moeda é o que nos permite construir um sistema anónimo. O uso de uma blockchain permite-nos ter multiplas autoridades responsáveis, evitando por isso o uso de esquemas centralizados. O nosso trabalho focou-se em desenvolver a nossa idea, uma prova de conceito, e testar a viabilidade desta nossa nova solução.

Keywords

blockchain, distributed systems, reputation, trust, privacy,Smart Cities, vehicular networks

Abstract

We live in an increasingly digital world, where Smart Cities have become a reality. One of the chaacteristics that make these cities smart is their ability to gather information and act upon it, improving their citizens lives. In this work, we present our system, *Dignitas*. A blockchain-based reputation system that allows citizens of a Smart City to assess the truthiness of information posted by other citizens. This assessment is based on a bet that reporters make, and oll of those who agreed with him, that puts their gathered reputation at stake. This use of Reputation as a currency is a novel idea that allowed us to build an anonymous system. Using blockchain we were able to have multiple authorities, working with each other to make the system secure and thus avoinding centralized schemes. Our work was focused on developing our idea, a proof of concept, and testing the viability of our new solution.

Contents

List of Figures

List of Tables

Glossary

ØMQ	ZeroMQ
API	Application Programming Interface
BFT	Byzantine Fault Tolerant
CBOR	Concise Binary Object
DLT	Distributed Ledger Technology
DPoS	Delegated Proof of Stake
MANET	Mobile *ad hoc* networks
OBU	On Board Unit
OBU	On Board Units
P2P	Peer-to-Peer
PKI	Public Key Infrastructure
PoC	Proof Of Concept
PoA	Proof of Authority
PoS	Proof of Stake
PoW	Proof of Work
RSU	Road Side Unit
RSU	Road Side Units
REST	REpresentational State Transfer
SHA	Secure Hash Algorithm
TA	Trusted Actors
TP	Transaction Processor
UA	Untrusted Actors
V2I	Vehicle-to-Infrastructure
V2V	Vehicle-to-Vehicle
VANET	Vehicular Ad-Hoc Network
MVVM	Model-View-ViewModel
cli	Command Line Interface
DoS	Denial of Service
SDK	Software Development Kit

CHAPTER 1

Introduction

Nowadays, computers are everywhere, from the ones scattered around the world in the so-called cloud to the ones in everyone's pocket, our smartphones. However, the mass adoption of these technological wonders might not have happened if they did not possess one key feature, the ability to communicate with each other. Communication is the property that enables the creation of huge networks, interconnecting systems and facilitating constant flows of information and data all around the globe.

Today the information available through this distributed network is no longer solely human-generated data with the intent of informing other humans, there has been an increase on the amount of information being created by machines with the sole purpose of being aggregated and analyzed by another machine and then, finally, being presented to a Human in a distilled and resumed way. This new paradigm is usually used when there is a need to sense and watch over an environment and where the sensing is done by small machines (sensors) collecting small pieces of data about that same environment that when processed with other bits of information can build a bigger picture of what they are observing. Smart Cities are one contemporary application of this model.

Smart Cities is a concept that has been growing gradually over the past years, they are no longer an idea but have become a reality in our society. Cities have measurable advantages in becoming "smart", deploying a wide network of sensors allow for the city administrators to easily overview the well-being of their population. One way of attaining this knowledge is through the measurement of environmental data such as air and water quality, or by measuring car flows on the streets to identify possible circulation bottlenecks and accident-prone locations.

It is evident that the need for this sensing of our cities will increase. One way to handle this increment is to deploy more sensors. Another interesting approach is to use one element that is essential to cities, its citizens. Citizens are those for whom there are more advantages in this type of systems. Information about the place they live is priceless, enables them to act upon it and enhance their lives. For instance, it would be useful for a citizen to know in advance if there is an accident in the route they usually use to commute so they can take an alternative. If citizens are the ones that most benefit from this, it makes sense to induce them in this process of sensing their own city, to get people to act as sensors in this mesh of interconnected nodes.

However, *"with great power comes great responsibility"*[1] and if the purpose is to give citizens more power to participate in this process, there is also the need to ensure that somehow people are going to take this task seriously. This is where reputation comes into play. Since the dawn of human interactions, reputation has played a crucial part in the first assessment of new information. Humans tend to trust new information based on the person that is telling it. Today, in our everyday life, reputation systems are frequently used: specialized reputation systems, such as Tripadvisor, as part of online commerce systems, such as eBay and Amazon, or even in programming support systems, such as StackOverflow. Even current social networks can be seen as a reputation system, where the opinion of someone is rated not by its content but by the number of people paying attention to it.

Reputation systems are also inherently insecure in terms of user privacy because their whole purpose is to guide human interactions solely through an identity based process. Even with people being more aware than ever about privacy and its importance in our modern world, it is more common to hear about systems leaking private information about its users.

So, how can we build a secure and private reputation system that is able to shape the future of information and interactions in our Smart Cities ?

1.1 OBJECTIVES

This work main objective was to build a Blockchain-based private reputation system that could support an event reporting system for a Smart City.

A system that could aid in reducing the problems caused by the exponential growth of cities, enabling citizens to receive information, in real time, about their surroundings and allowing them to make informed decisions based on the information that the system provides.

Citizens should play a central role in their cities and our system should give them the power to make an impact in their communities. The reputation of participants should guide the final credibility of the information being spread. Individual reputation is used as a "currency", citizens bet with their reputation when giving an opinion, and get some reputation back when they are considered to be right.

However, there is a key principles that we should enforce: privacy by design, meaning that every citizen that would want to participate should feel secure doing so. Our objective is to create a system that could help and improve citizens lives and not the other way around.

The sense of community and helping others should be fostered and citizens could also be rewarded for their good behaviours, without necessary giving up on their privacy.

[1]Popular quote from Stan Lee's Spiderman comic books

In these types of systems, where one could leverage opportunities, generally, some problems arise such as: *"Quis custodiet ipsos custodes?"*[2]. If the integrity of the system is relying on a single entity how could we be sure of its impartiality?

With this in mind, our proposed system should not rely on a single, centralized authority entity. Our proposed distributed solution makes our system invulnerable to this types of problems. The reputation system should have the ability to be supervised by municipality stakeholders, which in turn have the role of giving a final decision on the quality of the information and, consequently, in the update of the reputations associated with. The blockchain technology is used as a natural solution to spread power among stakeholders when it comes to reputation management, ensuring a trustworthy dissemination of information.

For our work we also assumed the deployment of the system in a smart city scenario where it should rely on the existing infrastructure to support itself, meaning that it should be able to deal with devices that have limited operational capability.

1.2 CONTRIBUTIONS

The main contribution of this work is the novel idea of using reputation as a coin disassociating an individual reputation from its identity. Meaning that, in a reputation system based on this approach, users could preserve their anonymity while participating. This idea was present in a paper [49] that was accepted and present on the 2019 Fifth Conference on Mobile and Secure Services (MobiSecServ).

Following up on this work, we worked on a practical implementation of this idea in a Smart City scenario from which we developed a framework called *Dignitas*, our Proof Of Concept (PoC). This framework could be used in a Smart City scenario, but also in any other scenario that could take advantage of a reputation scheme. With it, we also successfully demonstrate yet another use for the distributed ledger technology (blockchain) in an environment requiring the existence of distributed trust.

During this work some spin-off contributions were also made to the open source community, namely on some library support for the novel language Rust.

1.3 DOCUMENT STRUCTURE

After this introductory chapter the reader will find Chapter 2, a small background review which presents some concepts and give some indispensable context in order to understand the rest of the work. In Chapter 3 the reader will find the state of the art where we present and discuss related works. Next, in Chapter 4 we begin the presentation of our solution discussing its main ideas, processes and actors. In Chapter 5 we deep dive into our system components explaining how they should operate and communicate with each other. Chapter 6 is dedicated

[2]Latin for "who guards the guards?"

to our system implementation, here the reader will understand the steps and decisions made while building our Proof Of Concept (PoC). In Chapter 7 we explain how we tested and what results we got from our PoC. Finally, in Chapter 8 the conclusions of this work are presented.

CHAPTER 2

Background Review

"Familiarity breeds liking." - Daniel Kahneman, Thinking, Fast and Slow

Since this works spawns across several different fields, this chapter's purpose is to give an overview of concepts that are indispensable to understand it. One of the main objectives of this work was to implement, above all, a secure system. For this reason, in this chapter the reader will find the description of security concepts important to understand our work. The Blockchain is another important foundation and tool of this work, we will explain how it came to exist, its main attributes, use cases and how we can position this tool in the broader ecosystem of Distributed Ledger Technology (DLT). Another topic we addressed briefly are smart cities; since they are one of the use cases of this work, we decided it was important to discuss some important concepts.

2.1 Security Principles

Being one of the core aspects of the our proposed solution, security has a big part in this work. In this section we present several concepts that are fundamental to understand the proposed solution in detail. Concepts such as public key cryptography, what it is and what role does it play in current security solutions. Also, security primitives used throughout this work, such as hash functions, are presented.

2.1.1 Public Key Cryptography

When there is cryptography, there are keys. To us, the idea of using a key to encode something is very intuitive. It is a process that has been employed since the times of ancient Greece. The privacy of a message depends entirely on the privacy of the key being used to secure it. If the necessity to share the keys with the party we want to communicate with is considered, then we have a problem; specifically, the need to find a secure channel for sharing the key in the first place.

In recent years, there has been a sprung of cryptographic systems based on more than one simple key. These solutions tried to tackle the problem described above, the dissemination of a key when there is no previous secure channel on whom to trust. In [12] Diffie and Hellman defined the basics ideas behind a system based on two different keys: one that must be always private, and other that can be public and shared with everyone. Both these keys are intertwined with themselves. The public key is generated from the private one, but it is infeasible to generate the private key from the public one. This generation of keys is based on the known intractability of certain mathematical problems, which are easy to state but difficult to solve, for instance, discrete logarithm problems or integer factorization [19].

These type of cryptosystems, also called Public Key cryptosystems, can have different purposes, such as key agreement or digital signatures. Key agreement is the problem stated above, the need to exchange a secrete key over an insecure channel. Another use for this systems are digital signatures. Digital signatures, similarity to their counterpart in the physical world, are used to verify the author of some information, providing authentication of messages. However, signatures from the digital world can do more than that, they can ensure that the message was not modified (provide integrity) and can even ensure the non-repudiation of messages, meaning that a person cannot deny having signed a specific information.

2.1.2 DIGEST FUNCTIONS

Digest functions are a special type of cryptographic construction [40]. They create a sort of summary, or fingerprint, for an arbitrarily sized piece of information.

Thinking in terms of a black box a digest function takes as input some block of information and deterministically returns a fixed sized representation of that data. They must possess some important properties for them to be useful and used in real scenarios. First, the computation should be fast (linear time). Next, the inversion of this computation should be infeasible (exponential time). Last, but not least, the digest function should be collision resistant, meaning that it should be hard to find two different documents with the same digest.

This tool can be used in digital signatures. Digitally signing a document is usually achieved by only signing the document digest, and not the full version.

Nowadays, the digest function more widespread and used is the Secure Hash Algorithm (SHA). SHA is actually a family of functions that are represented by SHA-n where n represents the fixed size in bits of the output, such as, SHA-256, SHA-512, SHA-1024.

2.1.3 ATTACKS ON REPUTATION SYSTEMS

Reputation systems, as any other type of system, are not impervious to malicious attacks. Even more so because attackers can personally benefit from such attacks. There are a plethora of different attacks than can be performed on a reputation system; however, based on [28] we can group them into different classes such as:

- *Self-Promoting:* attackers try to influence their own reputation with false reports;
- *Whitewashing:* attackers try to "reset" their reputation in order to avoid the negative effects of bad behaviour enabling them to continue;
- *Slandering:* attackers try to influence other people's reputation by falsely reporting;
- *Orchestrated:* attackers collude and perform one or more of the above strategies;
- *Denial of Service:* attackers affect the normal behavior of the reputation system by affecting the way reputation is disseminated;

These classes are just a guideline, and they can be subdivided into even more types of attacks, for instance, one that is important to be familiar with is the Sybill attack [33]. Sybil attacks occur when an attacker creates several identities and by doing so it can manipulate the system to gain unfair advantages over the other participants in the system

It is important that we know what we are up against when building our solution and to plan for it.

2.2 DISTRIBUTED LEDGER TECHNOLOGY

Blockchain is a household name bringing attention to many systems. However, it is inserted in a much broader ecosystem of architectures and solutions. In this section, this novel technological field is explored, namely, its origin and its properties. Ideas such as ledger and consensus are explored and explained in this section, so that we can understand why it was chosen as a tool in our solution.

2.2.1 BITCOIN AND THE ORIGIN OF BLOCKCHAIN

Bitcoin [42] by Satoshi Nakamoto was the system that brought Blockchain to the mainstream conscience. Because of that, usually, the technology (Blockchain) is usually mixed and associated indiscreetly with cryptocurrencies in general. Fun fact, in his paper about Bitcoin, Satoshi not even once uses the name Blockchain. The solution proposed by Satoshi focused on solving a problem: creating a version of electronic cash that allowed for users to send payments directly to other users without the use of a middleman to mediate the transaction.

His solution was based on a Peer-to-Peer (P2P) network of nodes that collaboratively maintain the network working. Each node of the network is responsible for maintaining the state of the system and keeping a copy of a ledger. The ledger was nothing more than a registry of every transactions of coins that users made to one another. Transactions leverage the power of public key cryptography, through the signing process. To make a transaction, the recipient's public key is needed. This public key is hashed with the previous transaction of the coin and the new recipient public key. The hash is then signed by the current owner

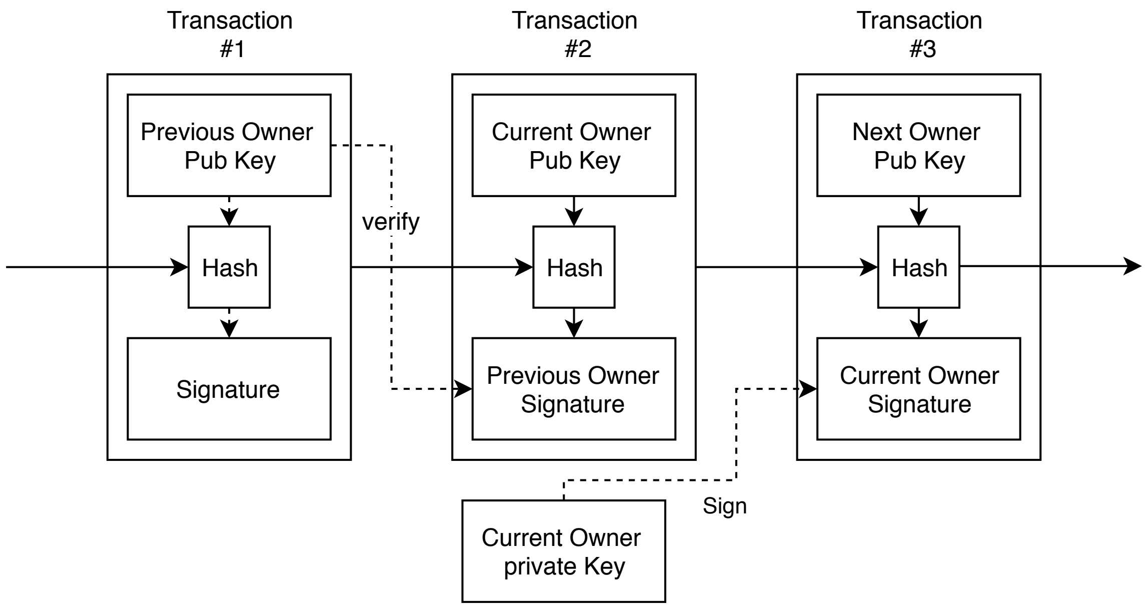


Figure 2.1: Schematic of transactions between three users

of the coin. This ensures several things: the traceability of a coin (through the hash of the previous transaction), the coin can only be claimed by a specific user and the non-repudiation of the transaction (the signing to spend a coin must be with the private key of the current coin owner). We can see this process schematically in Figure 2.1.

The problem is that these methods do not solve the problem of double spending. This is why a point of authority is usually used, this authority is responsible to review the transactions that are arriving and ensure that there is no problem with the transaction being proposed. For this process to work on a distributed manner, the author proposed the creation of a distributed time stamp server instead of centralized mint. To better understand this process lets understand the idea of a single time stamp server. The time stamp server is responsible to group transactions, time stamping them, and create a hash. This hash is then used when creating the next block of time stamped transactions. Through this, we can ensure that there is an order of transactions that existed at specific times. As we can see in Figure 2.2, this process creates a chain of blocks, hence the name blockchain.

Finally, the last major contribution of [42] is the Proof of Work (PoW) consensus mechanism. The time stamp server has to work on a distributed way. Because of this, some time stamp servers can be receiving transactions that did not yet arrive to other servers. The general objective is for the whole network to share the same state; so, the nodes must preserve the same ledger state, the same chain of blocks. Satoshi come up with a solution based on [4], using a cryptographic puzzle to moderate the creation of blocks. This puzzle must be difficult to solve, but easy to verify. The idea is that each server, when after grouping some transactions in a block, add a nonce to the block and create its hash. The problem lies in the need of the hash to have a particular structure, for instance, begin with some predetermined number of zeros. The server must then continuously try and hash the block with different nonces until finding a suitable nonce. This is similar to a lottery, in which the winner gains the right of

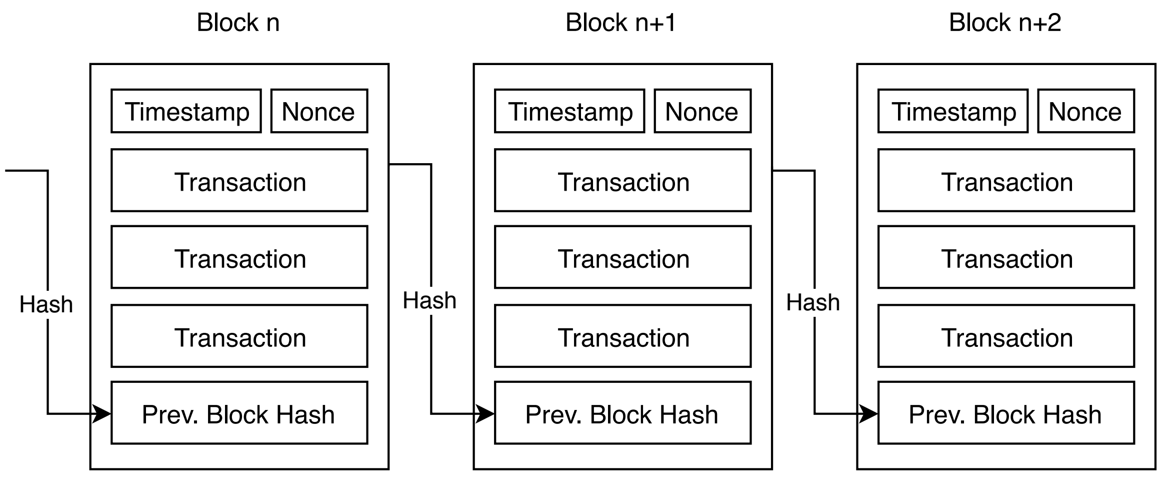


Figure 2.2: Chain of blocks schematic

adding the block to the chain. To ensure that there is a reward for performing this kind of search work to the network, the entity that successfully finds a solution for adding a block to the chain also gains the right to create some new currency, adding it to its account.

So, in conclusion, users create transactions and send them to timestamp servers. These servers verify the transactions, create a block through the PoW mechanism and announce the new block to the ledger holders. Other time stamp servers can easily verify that everything is alright with the proposed block, add it to their copy of the chain and start working on the next block.

With Bitcoin we can easily identify 4 basic principles of the blockchain as a tool [31]:

- Distributed Database
  - The ledger can be seen as a distributed database that every nodes has access to. There is no middleman and every node verifies everything.
- Peer-to-Peer Communication
  - The P2P network provides a base for communications that does not need a central node to work.
- Transparency with Pseudonymity
  - The transactions are all recorded and are visible to every actor of the network, providing full transparency. However, the records only register transactions between public keys, which, in the absence of more information do not provide any useful information about their real owners.
- Irreversibility of records
  - The linked nature of the records in the blockchain ensures that once a record is deemed true, it will be very difficult to alter that.

2.2.2 BLOCKCHAIN NOWADAYS

As it is well known, Blockchain became a household name that has rally many different opinions over the last few years. Some people believe that blockchain is a concept that will solve all the world problems. Others understand that it is only a tool that can useful when used in the right use case. Various platforms have leverage the success of bitcoin to launch themselves frameworks based on blockchain. These platforms aim to provide the industry with blockchain-based solutions.

If we look to the bitcoin use of blockchain, we see that these concepts are already present. In bitcoin, each node verifies a set of rules to ensure that the transactions being presented are valid. However, these rules are not mutable and cannot be changed (if they are, the node can no longer be a part of the network). Bitcoin and financial transactions in general are one of the use cases for the blockchain technology.

Lately, several use cases have been proposed, where the blockchain could be put to good use. Supply chain management is one of the recurrent novel use for blockchain. There is a lack of transparency in the course of products along supply chains, namely when there are several entities involved. Consumers do not always know the provenience of the products they are buying. Blockchain properties, such as the immutable transactions and distributed nature, can help mitigate this problem. Several other use cases, such as Digital Identity, Voting, Healthcare and Notary can also leverage the power that this technology provides [55].

The industry needed tools in which it could implement their own rules in terms of transaction validation and business rules that manage the network. Several new frameworks started to appear which provided the flexibility needed, bringing Distributed Ledger Technology (DLT) to the industry world.

One of the most known solutions is the Ethereum framework [9] where Vitalin introduces us to the concept of smart contracts. Throughout the different frameworks available, smart contracts receive different names; however, the main ideas are transversal to them. Smart contracts are the way the basic features of the blockchain are expanded. While in Bitcoin the program that runs to verify each transaction is pre-defined with smart contracts it is possible to specify any program to run, to specify transaction validation rules in order to implement any business logic.

2.2.3 CONSENSUS IN A DISTRIBUTED LEDGER

Consensus is a very important concept in a distributed ledger architecture. So far, the only one mentioned was the PoW mechanism sugested by Satoshi for the bitcoin application. Before going into more detail, lets first arrive to a definition of consensus. When we have a distributed network of peers, and they all need to share a state that can mutate, we say that there is a consensus when all the peers agree on the same state, in other words, consensus is the same thing as agreement.

The Two Generals' Problem [1][25] is a classic consensus problem, well-known in distributed systems. In it, two generals must coordinate an attack, however, there are no guarantees that the messages they try to exchange actually reach their destination (Figure 2.3). This problem has been proved impossible to solve in a finite way, meaning that there is no finite protocol that can solve it. The system will never reach a state in which there is full certainty that both generals agree with the attack plan.

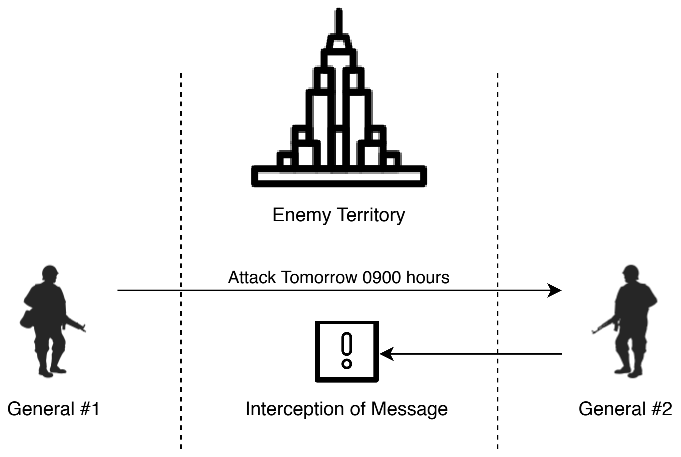


Figure 2.3: Two Generals Problem

The Two Generals' Problem was expanded by Lamport, Shostak and Pease in [37]. The new version of the problem contained more than two generals trying to reach an agreement on the plan of action. However, they add another layer of problems by allowing the presence of traitor generals (generals that can lie about their choice). The authors called this new version the Byzantine Generals' Problem. A algorithm that solves this problem must allow that all loyal generals decide upon the same strategy and that a small number of traitor generals cannot jeopardize the attack by causing the loyal generals to adopt a bad plan. In a more practical, way we can map this problem in a distributed system where the peers (generals) are trying to reach a consensus about something (plan of attack) while malicious node are trying not to (traitors). Algorithms that can reach consensus withstanding faults, such as malicious actors, are known as Byzantine Fault Tolerant (BFT).

With bitcoin PoW, there is a probabilistic solution to the Byzantine Generals Problem. Other consensus protocols that exist must all be BFT. This property is important to ensure that malicious nodes (if they are contained to a specific range) cannot influence the network in a bad way.

The PoW consensus mechanism consumes a high amount of resources, because of the way it was designed. The nodes are constantly looking for a suitable nonce for a block of information, thus hashing over and over again, trying to win the lottery of adding a new block. Being this process the bottleneck for a new transaction being accepted, in order for the network to grow this process must also scale. Meaning that, there is a ever growing need for electricity to power this process. In its peak, the bitcoin network was using the power equivalent to a country [35].

Because of this, other consensus mechanisms have started to appear. Proof of Stake (PoS) based mechanisms are proofs that require that nodes that wish to add new information to the system to put at stake some of their wealth. There are different type of PoS mechanisms; however currently there are two major flavours: chain-based and BFT-style. In chain-based protocols the validators of blocks lock away some of their wealth and are assigned to add a new block to the chain. If they act in a badly manner, their wealth is not returned to them. In BFT-style protocols, also called Delegated Proof of Stake (DPoS), several validators are chosen and together (after locking up some of their wealth) they decide on which block to add to the system. PoS-based consensus is clearly more eco-friendly than PoW, because it is not based on work-intensive puzzle solving.

Another example of these novel consensus mechanisms is the Proof of Authority (PoA). This protocol leverages identity in order to save on computing resources. In it, there are some nodes which are deemed trusted *a priori*, they are called validators and are the ones with authority to add new information to a specific ledger in a specific network. Usually, PoA networks are used in private/semi-private systems, where we can clearly identify the authorities. Note that in order to fully obtain all the benefits of a decentralized solution there should always be different authorities. In these systems, nodes will never battle for authority, usually the power to add new information to the ledger is alternated between all the validators using for instance, a round-robin approach. Because of this, these systems are fairly faster and less resource intensive.

There are several ways of achieving consensus in a blockchain system. One must access all the benefits and disadvantages in order to choose in the best way possible.

2.3 SMART CITIES

The definition of what a Smart City is is still a open debate. In the eyes of Harrison *et al.* [26] the term Smart City stands for a "instrumented, interconnected and intelligent city" (Figure 2.4). Instrumented refers to the ability of capturing live, real-world data, achieved through the use of sensors. Interconnected since its sensors should form a connectivity mesh supporting the constant sharing of information. Finally, intelligent in the way that we must be capable of handling this complex processes autonomously through advance modeling, visualization, all with the purpose of enhancing the operational decision making process. This definition is going to be maintained throughout this work.

Smart Cities are a growing trend. In Europe, for instance, 75% of the population lives in urban areas and this is a number that is expected to grow up to 80% until 2020 [2]. The main takeaway is that the growth of cities is the driving force behind the concept of Smart Cities and innovations associated with them. Currently, this is a field with a fast paced race to improve urban sustainability with a anthropocentric approach, enabling those in charge to respond more efficiently to their citizens needs.

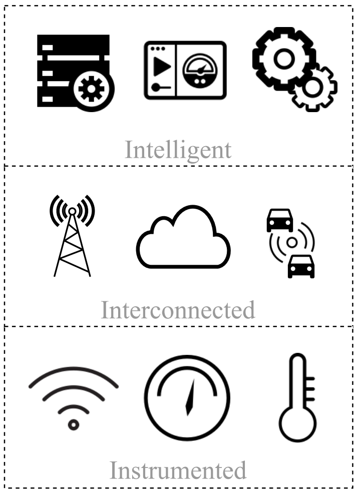


Figure 2.4: Three major areas of a Smart City

2.3.1 VANET

A Vehicular Ad-Hoc Network (VANET), as the name itself says, is a special type of network architecture. It relies on vehicles to relay information, enabling Vehicle-to-Vehicle (V2V) and Vehicle-to-Infrastructure (V2I) communication. VANETs are a type of *ad hoc* networks and considered a subclass of Mobile *ad hoc* networks (MANET) with some special characteristics such as high node speed and highly volatile topology. They have several applications such as: safety applications, using V2V to disseminate emergency messages for making the roads more safe; traffic optimizations, using real-time data of traffic for enabling people/autonomous cars to better plan out and optimize routes; improve visibility, for instance in intersections where physical visibility is impossible, the use of this network can warn about incoming cars or other invisible dangers; and many others, such as encouraging cooperative behaviors, enabling advanced driver support and overall improvement on communication.

When we talk about a VANET infrastructure there are some important entities on which these networks are built upon (Figure 2.5). Namely, the On Board Units (OBU) and Road Side Units (RSU). The Road Side Unit (RSU) is an entity fixed by nature; deployed in strategical places, these equipments are the intermediary between the mobile nodes of the network and the infrastructure and/or the Internet (V2I communication). A On Board Unit (OBU) is installed in the vehicles and is the mobile entity of the network. It is the equipment responsible for creating intra-vehicle networks and must be capable of V2V and V2I communication through the use of some wireless communication technology.

Although beyond the scope of this work, wireless communication technologies are an important component in this type of architectures. The mobiles nodes must have the capability to communicating using existing protocols such as the traditional WiFi (802.11), or through the use of other protocols, such as the ones in the core of the WAVE architecture ?? . Built upon a stack of standards, it has the goal of supporting low-latency communications among vehicles in mind.

However, the principal idea in a VANET is to create a multi-technology environment with

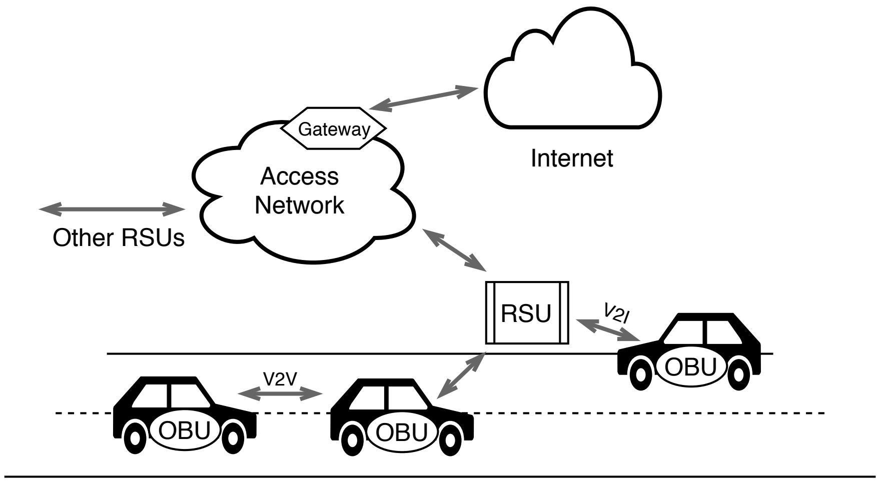


Figure 2.5: VANET architecture overview

nodes being able to communicate with different technologies and where applications must overcome the mobility induced problems of such a network topology.

CHAPTER 3

State of the Art

"We learn from failure, not from success!" - Bram Stoker, Dracula

No work is a standalone project and reputation systems make the base for ours. In this chapter are presented in a top down approach the works that based and guided our process. First, it is important to understand what has been done relatively to reputation systems: understand where they are usually used, common problems they present and solutions usually employed. Next, we tried to understand how this work was being applied in the scenario of Smart Cities, problems that arise and usual methodologies to solve them. Finally, we searched for similar solutions that might employ some of our mechanisms, such as blockchains.

3.1 TRADITIONAL REPUTATION AND TRUST SYSTEMS

Nowadays, reputation systems are ubiquitous. Sometimes, more than we are willing to accept, for instance, in the case of governments trying to keep a reputation system to evaluate all of their citizens actions [50]. This continuous interest ensures that there is a ever-growing literature around this thematic.

In [34], Jøsang *et al.* evaluated and explained the need for reputation systems in online environments, namely, in service provision through some existing solutions. They identified core concepts such as Trust and Reputation. Trust is presented as two different definitions. *Reliability Trust* referring to the expectation that certain individual behaves in a certain way and *Decision Trust* to which they defined as being the willingness of someone to depend on something or someone and feeling relatively safe doing so. Trust can be seen as a subjective value intrinsically present in human interactions. They presented the concept of reputation as one being closely linked to trustworthiness. We use the same definition for *Reputation* as being what is generally said or believed about a person's or thing's character or standing. The reason for the two concepts being inherently entwined is that the trust we feel towards something depends on their reputation. And the other way around also works, we might discard sometimes reputation if we already have a predisposition to trust it. With focus on the architectures, they concluded the existence of two main architectures for reputation systems: *Centralized* reputation systems, on which there is a central authority, and *Distributed* reputation systems environments, where information about reputation can be distributed on different stores or each participant is responsible to store information needed to compute reputation. Our system is included in the later type, increasing service availability, robustness and security. In their conclusion they pointed some problems on existing reputation systems. Problems such as, the possibility for entities to manipulate reputation scores. In our solution, we try to tackle this problem through the use of the Blockchain and its tamper resisting properties. Another problem pointed out is the lack of incentives to provide ratings. We try to counter this by making the process of rating transparent to users, but extract reputation through their actions on the network, namely, their behaviour while voting.

In a more recent survey [27], Hendrikx *et al.* reiterated the important distinction between trust and reputation. One distinction that we maintained throughout this work. In it, they classified a reputation system as a system that can collect and aggregate data about an entity in order to characterize and predict future behaviour from that entity. Another key point of reputation systems is that they must be able to share this reputation amongst other users, enabling the emulation of a long term relationship even if there was no previous interaction between them. In their taxonomy, the first level they presented devises reputation systems in *explicit* and *implicit*. Implicit reputation systems are the ones that have almost no structure, the ones that simply appear. One example of this type of systems is, for example, the one that derives from our natural human interactions, we know which baker to buy bread from, or where to find the best fruit and even when not to trust the travelling salesman. These intuitions arise from our day-to-day interactions, and they form a reputation system without any real structure. In a more recent example, social networks are also a type of implicit reputation system. There is no formal reputation system in place, however, people tend to make their own conclusions about someone influencing their behavior with them. The system present in this work is included in the second type of systems, it is an explicit reputation system. There are mechanisms in place to evaluate and gather information about users and also a way to share this information with other users.

Traditional reputation systems usually separate the user interaction and the trust building process. When a user interacts with another user, and a traditional reputation system is in place, the interaction is separated from the reputation. It is a two step process, first the user access the reputation of the user we will interact with and then, based on that information it decides if it will proceed or not. Privacy is usually not a big concern of these systems. Our system innovates in a way that the reputation is implicit in the interaction itself, it only takes one step, when a user starts to interact it also presents theirs' reputation. Users do not need to go and consult anything or anyone about that user's reputation to decide on either trust or not the information. The "trustworthiness" is implicit in the message/interaction. This implies that the interaction can be private, and users do not have to know each other to trust in the information they provide.

Recently, with growing concerns about privacy and data protection, users are more aware of the dangers of using a not so private system. Because of this, there is a growing number of contributions in improving the privacy of systems. However, as said before, most of the literature in this area is focused on online marketplaces and so is the work that tries to tackle the privacy problem.

The work in [7] by Blömer *et al.* is a classical example of such premise. Their main focus was to build a private reputation system to govern online transactions. Built around a centralized architecture, they presented a system on which a user must register in a central authority and then ask for tokens to the providers in order to rate a provider's product. Secured with cryptography operations such as Zero Knowledge Proofs and a Public Key Infrastructure (PKI), the users can post their opinions, in a secure way, once per product (if they post more than one opinion per product they endanger their privacy). This types of solutions are not suitable to our goal because they depend on a central authority and, because of, that they cannot scale.

In [3], [5] Bag, Azad, and Hao presented a system that also has privacy as the main concerned and claims a decentralized nature. Fueled by public concerns about an increase in fraud over online marketplace transactions, their proposal was built upon concepts such as Zero-Knowledge Proofs and other cryptography primitives to secure their system. The main takeaway is their use of a public board that registered the users opinions. Although public, the remarks enclosed on the board were cryptographically secure and ensured the privacy of the post creator. Another key point of their architecture was that, in order for a user to be able to post in the public board, the marketplace regulator must issue tokens for the user to use while posting their opinion. Several problems rise from this type of solutions. One of them, and the most common in similar solutions, is the need for a trusted intermediary; in this case, this dependency surges with the tokens that are issued by a central authority, meaning that they have control of what is being published in the board. Although claiming a decentralized architecture, they presented a centralized system scaled into a decentralized pattern by multiplying the centralized idea.

3.2 REPUTATION SYSTEMS IN SMART CITIES

With any reputation system it is important to understand the scope and what is being evaluated. So far we have seen that a common implementation of reputation systems is in online marketplaces. In those systems, usually, the users are categorized as either being trustworthy or not in order to influence future interactions.

Even when we change the formula of a reputation system in order for it to be applied in a Smart City many concepts are transferred into this new domain: there is still going to be users; the main objective is still to influence behaviors based on previous interactions; and usually, there are messages and informations that are being distributed throughout the network. Applying reputation concepts to this kind of scenario brings the same advantages as in the previously discussed ones. One can consider to act upon a new information or not, based on reputation (e.g I decide to go through a specific route in order to avoid traffic, since I trust the report of traffic, henceforth I change my original action).

Reputation in Smart Cities is a broad topic. There are several works that use reputation to, for instance: find badly behaving nodes in the networks, since Smart Cities are heavily based on sensor meshes it is important to filter sensor information [36]; to correctly route messages in the network based on node reputation [38][39]. In this section, we narrowed the search down to the works that directly could help us find our final solution, namely, the ones that could help with taxonomy, that had message passing and that allowed for people input into the system.

From a systematic review done by Soleymani, Abdullah, Hassan, *et al.* [52] it is possible to extract important taxonomy. They identify three major types of reputation systems in a VANET environment:

- Identity based, when there is the idea of identity and individual. Usually, these models and solutions heavily depend on information about their peers in order to being able to compute the trustworthiness of their messages. This dependency is their Achilles heel since, most of time, because of the highly dynamic environment, the nodes fail to gather the information necessary to correctly assess their peers.
- Data based systems, when the main objective is to simply assess the trustworthiness of the data/message being delivered (ignoring the sender). The need for the message to be self-contained, meaning that its trustworthiness cannot depend on external parties, can increase the message size and can be a problem in high density networks.
- Combined systems, which as the name says are a composition of the ones above. Initially, they may depend on identity information but they try to limit it to a minimum.

Unfortunately, our proposed solution does not fit well into these definitions. On one side it does not use identity to compute the trustworthiness of a message but on the other side it allows for certain interaction based that are based on pseudo-identities, for instance, the rewarding system.

An early example can be found in [13], where Dotzer, Fischer, and Magiera propose a reputation system for a VANET in order to generate trust towards messages being exchanged. In their work, they specified the difference between an event, a decision and distribution areas. The event area is where the trustworthiness of the message can be assessed. The decision area is the area on which a decision must be reached. Finally, the distribution area is where we can find people interested in the message. The system relied on the calculation of the message trustworthiness, for which they rely on several inputs: direct observation when the car is on the event area; indirect observation through the trust the vehicle has in their neighbors; opinion piggybacking, with opinions being added to the message while it transverses the network. Because of the early nature of the work, there are no security concerns mentioned and no privacy aware measures in the system, a problem tackled by us.

Gómez Mármol and Martínez Pérez [22] work presented us a new solution and establishes comparisons to other systems. A scenario is presented with many of the components of a VANET, such as RSUs. The main objective is to reliable identify vehicles that are not trustworthy and, henceforth, ignore their messages. They define the main requirements for a trust and reputation model:

- Simple, light, fast in order to provide reliable feedback to its users.
- Accurate so that the model does not add entropy to the system.
- Resilient to security and privacy threats, since more than ever it is on the users minds its importance.
- Scalable, in order to provide space for cities to grow.
- Not dependent on mobility patterns, the system should not be dependent on the network characteristics.

About the system presented, when a message was received the vehicle was then responsible for calculating the reputation score of each node. This was done by taking information from different sources (as the previous work). This reputation score was then compared against a scale to decide if the message was worth taking into consideration, if it should be passed along or discarded. The authors conclusion is that their system fulfills all the objectives that they proposed for a reputation and trust model. However, it appears to be an erroneous conclusion. For instance, they write about the importance of being independent from mobility pattern and then, in the formula they use to calculate the reputation score, 0.7% of the weight is based on interactions with other vehicles that they previously had interactions with. Also, they do not provide any privacy or security mechanism but claim to be partially resilient to this types of threats. In our work, we agree with the objectives proposed by Gómez Mármol and Martínez Pérez and we believe we actually fully achieve them.

A new trend emerged in this type of scenarios: the idea of soft sensing and crowdsourcing. From the work of Vakali, Angelis, and Giatsoglou [54] we get a simple but innovative idea: complement the objective input from sensors and machines with subjective input from people's opinions. The authors presented a system that uses social networks in order to gather human opinions. Sensors in the network are responsible to present the facts, these facts are then posted into a citizen accessible forum (social network) where they can debate the situation and give even more insights to the situation. This idea of harnessing the power of people's opinions plays a major role in our work.

As we can see, reputation systems are not a new concept in a Smart City scenario. However, the existing solutions are lacking either in terms of privacy or even scalability. Our system is a new approach to this problem, leveraging solutions and avoiding pitfalls that others suffer.

3.3 BLOCKCHAIN IN REPUTATION SYSTEMS

As we saw before, a blockchain can be a powerful tool to enforce certain properties in a system. Because of that, its no surprise that there is already work that tries to harness the power of tool, namely, in reputation systems.

Dennis and Owen [11] proposed a blockchain-based reputation system, presenting it as a generic reputation system that can be deployed in different scenarios such as, e-commerce; in the presented work the focus is on a P2P network. They proposed the creation of a "new blockchain" built specifically to handle the registration of reputation scores, which can either be a one or a zero.

The miners (nodes that want to add new blocks to the blockchain) are responsible to connect to both peers that took place in the transaction and ask for cryptographic proofs that they both have the correct file. After the miner validate these, the block is added to the blockchain in a similar fashion as bitcoin. Dennis and Owen system is presented as a generic reputation system, however, their blockchain is tightly coupled to the requirements/properties of their specific file sharing in a P2P scenario. Nevertheless, it gives an idea of how the blockchain can be used to enhance the security requirements of a reputation system.

Another example using a blockchain can be seen in [48]. Sharples and Domingue present solutions for education systems: as a permanent digital record; as a proof of intellectual work; or even as intellectual currency.

CHAPTER 4

Reputation as a Coin

"Most people are not prepared to have their minds changed" - Iain M. Banks, Use of Weapons

In this chapter we present our solution. First, we present a small overview of the main idea and how the system would work. We then explain the main entities present in our system and through which processes they interact with each other. Finally, we present our proposed architecture and the main components that take part of our solution.

4.1 OVERVIEW

Using reputation as a coin is a novel way to deal with reputation, one that allows for the separation of identity and reputation. This allows for a new generation of system which can use reputation without compromising their users privacy.

To clarify our idea, we will use the following scenario to present the system that from now on will be referred as a whole by the name *Dignitas*:

Olivia is a citizen that on her way to work sees a car crash. Immediately, she alerts the authorities using *Dignitas*. Other citizens in the area also report on this accident, the competent authority is alerted for this and based on the opinions of people has total trust in deploying the necessary means to the reported place. Bob is a citizen, he likes to plan his day meticulously, before leaving for work he accesses *Dignitas* and is now aware of a massive crash in is usual route, for that reason he decides to take another route in order to arrive to work on time. Olivia is a very active citizen in her community, she likes to report on neighborhood problems, such as broken water pipes and potholes, competent authorities see these reports and since Olivia has a good reputation they are confident that they are not wasting resources when they dispatch them to resolve those issues and can react much quicker to any problems that arrive. All of the above interactions have a interesting caveat. All those messages are exchanged anonymously, and although we name the characters in the story, in the system they are invisible to each other.

With this, we can empower citizens in a city to cooperate and report on events anonymously and getting valuable information for their everyday life. We can also see the several benefits of a system like this, not only to the citizens but also to responsible entities that are able to gauge their city and act upon disturbances more promptly. Offering such functionalities is the main purpose of *Dignitas*.

4.2 ACTORS

In *Dignitas*, we have two types of actors that can take advantage of our solution. Those two different types of actors have different obligations and benefits from a system like this. In this section both of them are explained in detail.

4.2.1 Untrusted Actors (UA)

Untrusted Actors (UA) are composed by the vast majority of citizens. They are the ones who can benefit the most from a system like this and, because of that, they are also the ones with greater responsibility in maintaining this system. They can interact with the system in two different ways:

- *Actively:* by generating event reports. These events can be anything from car accidents to a broken water pipe, or even use it to report on other types of events, such as social gatherings.
- *Passively:* citizens can also use the information provided to make plans in advance taken in consideration these previous reported events (e.g. taking an alternative route to work if there is an accident in their usual one).

The reason they are called untrusted is because the information they provide to the system can either be true or false, there are no *a priori* guarantees. In our anonymous system, there is no identity that we can bind to the users, all that we can use is their reputation. A UA can be a reputable citizen that behaves accordingly and only reports truthful events, but it can also behave in a malicious way and falsely report events for whatever reasons.

Because of this, the reputation value assumes an important role in keeping the system useful. UAs have assigned to them a reputation score that they can use to give credibility to their claims. They do this by using this reputation to generate reports and also to reinforce other UAs reports.

In turn, the system rewards UAs that were being truthful.

4.2.2 TA

Trusted Actors (TA), on the other hand, are composed of already trusted entities in a city ecosystem, namely, law enforcement forces, firefighters, emergency medical services agencies or even town hall representatives. These entities can use this system in order to better manage their deployments and have a sense of what is happening in a city, collecting and analysing data to enhance their services.

TAs that want to use the system have responsibilities too, they must monitor the system and react upon when they are needed. Different TAs have different responsibilities, since the reports can span multiple areas of intervention.

They can also report events in the system, like the UAs can; however, these entities are deemed honest and, because of that, everything they say is considered true by definition. Because of this particular power, they are also able to "close" event reports, meaning that, thereafter the truth about that event has been set by authorities. This is also the power that enables the system to reward well-behaved citizens that reported truthfully on events.

4.3 PROCESSES

The system actors interact with our system through two main processes. One of the processes is the ability that citizens have to report on real-world events, bridging the real-world with the digital one. The other one is the opportunity that citizens have to evaluate these reports by voting on the truthiness of the events. All citizens (anonymously) registered in the system maintain a value of reputation that is essential to their ability to participate in those processes. For that, citizens will use their reputation as a currency traded inside the reputation system: putting their reputation at stake when they wish to participate, and being rewarded for good behaviour.

Using reputation as an asset that the user can use to interact with other users is what allows to detach the identity of the user and create anonymous trust-based decisions.

4.3.1 REPORTING PHASE

A citizen, when confronted with a real-world situation, has the ability to broadcast that information to other citizens through our system. To do so, they must use their reputation to back their claim. In that sense, they put at stake their reputation by betting part of it. Thus, each message broadcast on the system has associated with it a discrete amount of reputation. This value of reputation, associated with the message, is what other users will use to discern between worthy messages and fake ones. Using this approach, we can ensure the anonymity of the reporters, since the information being broadcast has no personal elements. A person does not need to know the other person to trust in what they are saying, they only have to infer that from the reputation being putted at stake.

For this actions we use asymmetric cryptography, UAs generate a key pair in order to participate in *Dignitas*. They then use these keys to digitally sign their reports and the transaction of reputation associated with it. This information gets published on a public ledger that anyone can access. Digital signatures ensure the non-repudiation of information, meaning that once it is published, the user as effectively put part of their reputation at stake and they relish control over the amount they bet. They also ensure that the information that is published cannot be tampered with.

The goal of using part of the reputation as a bet is twofold. First, citizens have a limited capacity to broadcast reports, bounded by their amount of reputation, therefore, they cannot flood the system with messages. Second, higher bets will provide higher rewards (or penalties), and, because of that, will naturally be associated with true events since users are risking more. This somewhat ensures that reports backed by high reputation bets will probably be true.

4.3.2 EVALUATION PHASE

The reputation of a message is not solely dictated by the reputation the original announcer was able to put at stake. Other UAs can receive a message and re-enforce it, positively or negatively, but always by putting some of their reputation at stake as well. Thus, the "trustworthiness" value a message has inside the network can vary over time. Also, it is not the number of people that agree that matter, is the amount of reputation that they are willing to risk vouching for some piece of information. This process generates a type of voting situation, where each citizen can vote about some report veracity with their reputation.

This process can generate some "fuss" while the network tries to decide whether a piece of information is false or not. If these types of report fall under some trusted entity jurisdiction they can be closed by one of the TA responding to the commotion, ensuring that the vote has an end and that these entities only respond to critical situations. These entities are naturally trusted by the system, so they can close the voting on a particular message and decide about its veracity.

After this, two things happen. First, all citizens will see the quality of the message as decided by the trusted entity (either true or false). Second, all citizens that bet an amount of reputation on it will receive some reward back or lose the bet accordingly with their positions and the final result. For instance, if the message was deemed true, those who supported this position will receive their reputation back and be rewarded with some more, whilst those that supported a opposing opinion will lose all the reputation that they bet on it.

To support these interactions asymmetric cryptography is also employed. TAs have known public keys, meaning that one can always differentiate their votes and actions from the other citizens. Those keys, besides being used to vote/close reports of events, are also the only ones that are valid to reward users with new reputation.

4.4 REWARDING

Rewarding plays a major role in our solution and has multiple objectives. First, it ensures that UA keep caring for the system, since it mostly depends on their participation in order to properly function. The reputation gathered could be used outside the system, for instance, in real-life rewards.

Besides penalizing erroneous reporters, the rewarding scheme also plays a role in keeping the system secure. In Chapter 2, we talked about the possible attacks on a reputation system, we use the rewarding process in order to mitigate some of them.

One usual attack that systems where identity is obfuscated suffer are Sybil Attacks. As previously mentioned, in Sybil Attacks the attacker generates multiple identities, which he uses to gain advantages over other users. In our system, having many identities does not help you passing a report as truthful, since, what really counts into assessing the veracity of reputation is the total amount that was at stake and not the number of people voting. A malicious UA can create several fake identities in order to poulte the system with fake reports, however, a well behaved citizen with lots of reputation to swing the report to the right direction.

Whitewashing should also be mitigated by making new identities start with the minimum amount of reputation possible, which makes it impracticable "resetting" an identity.

"Bounty Hunting" is another type of behavior that we should try to mitigate. This behaviour is characterized by users join voting late in order to accumulate reputation without actually putting much at stake. In order to do so, our rewarding scheme varies with time, meaning that, voting on existing reports with already many participants is not beneficial. The rewarding amount percentage drops with the amount of participants. Our system highly rewards "risky" bets, as any other bet.

These types can be damaging to the system, with the rewarding scheme we must ensure that an attacker can not really take any advantage, it should be more easy to play by the rules than to behave badly.

CHAPTER 5

Architecture

"If you're afraid to change something it is clearly poorly designed." - Martin Fowler

In this chapter we present our proposed system architecture, we then explain some decisions that were made in order to successfully deploy the reputation as a coin in a Smart City scenario.

However, the contribution of this work was always intended to be of a reputation system that was agnostic to the implementation, meaning that it could be deployed over different architectures and scenarios. For this reason, although taking in consideration Smart City characteristics, we also explain the reasoning and behaviours some entities and components must have present in any type of implementation.

In Figure 5.1 it is possible to find, in a schematic way, all the components that take part in our system and how they interact with each other. The system can be seen has three independent sections: the network that contains the public ledger; the untrusted side, the components that refers to UA interactions, namely, the ledger proxy and the UA application; and the trusted side, that refers to the components that TAs interact with, the secure server and the TA application.

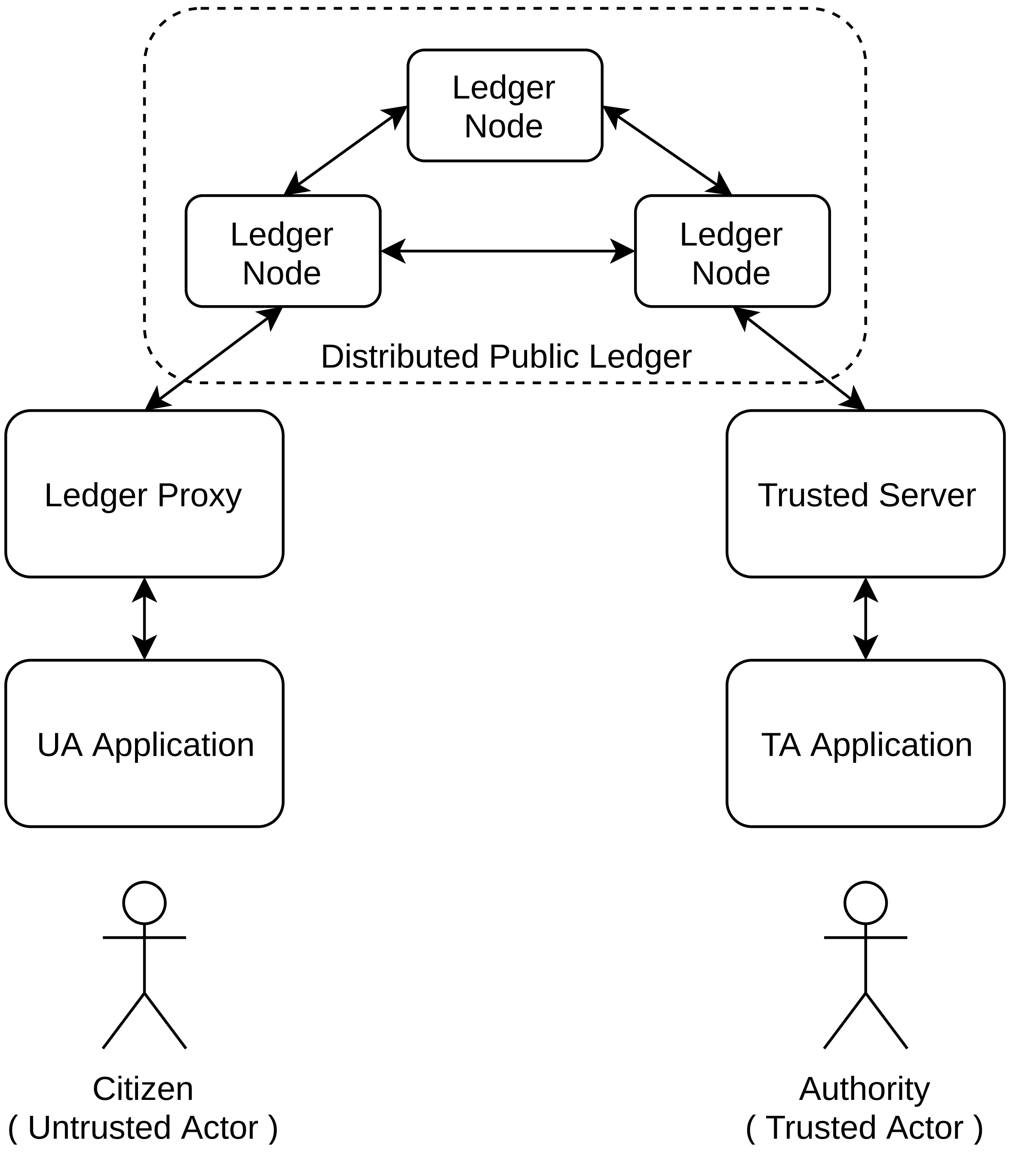


Figure 5.1: Architectural schematic of *Dignitas*

5.1 THE NETWORK

The network part of this system is responsible, solely, for the transport of the messages being exchanged. Besides that, for our system to work the network does not need any more special characteristics. Functionalities such as secure channels, tamper resistant transportation protocols or other security measures are dispensable for the system to work, since our protocols rely on cryptographic primitives where this type of security is already embedded. However, the added security in the network allows for a smoother experience.

For these reasons, the network used can be one of the already deployed, as the public Internet, but there is also space to use other type of networks such as, for instance, a city VANET. Other components of the architecture take this into consideration, since they must be the bridge and entry point to any type of network used.

5.2 THE LEDGER

The ledger is the component where all the information is stored. Like the other components, it must also have some characteristics in order to be employed in our system. First and foremost, should be a public board, where anyone can post anything without restrictions. The only restrictions that should exist are the ones associated with the "cost" of publishing something, namely, the cost of putting reputation at stake when stating something. The use of reputation should be the only regulatory element of this open board. This open concept allows for process transparency and greater audit capability.

However, the ledger public nature should not compromise its integrity. The board should be secure, meaning, should be tamper-resistant (a user should not be able to tamper it by changing things in the board) and timely-immutable (anything that is written in the board should be able to change over time, however, it should not be possible to reverse time and change something that occurred in the past). Users should also be unable to repudiate information that they had post. This can be achieved through cryptographic primitives, as hash-linking and digital signatures.

Another important characteristic that this ledger should have is being of decentralized nature. This is important for several reasons. First, availability; in a system that is supposed to be deployed in large environments, a centralized architecture is a bottleneck that can lead to catastrophic failures. Another important reason is due to the fact that this system is supposed to be regulated by several entities (e.g firefighters, police , etc) and a decentralized architecture allows for a wide variety of responsible parties that do not have to be under the rule of one central authority governing the public board.

All of this characteristics can be found in contemporary solutions based on Distributed Ledger Technology (DLT), the so called blockchains. Because of this we chose to employ a blockchain in our solution and harvest the full potential it offers. In Chapter 2 in Section 2.2 there is an in-depth explanation of how these systems usually work and which concepts are related to them. Here, we will explain how we can leverage these properties in our solution.

The nodes of the ledger can be distributed across the city under different jurisdictions, there is no need for a central authority. Anyone can run its own node and help increase the security and usability of the system, even if only a few are authorized to add new information. These nodes are responsible for receiving reports and add them to the ledger, they will then try and reach a consensus with all the other nodes in the network by broadcasting the new information to their neighbours.

5.3 THE PROXIES

In earlier drafts of the project architecture these components did not exist. The idea was, for instance, the UA application to communicate directly with the ledger nodes. However, depending on the number of nodes deployed, and their geographical location, it could not be an ideal scenario, since it could create a bottleneck scenario if several UAs tried to send reports at the same time to a node. It is important to remember that ledger nodes, although being highly performant, they have important requirements in term of computation power, since they ensure that our network stays secure. And for this reason, it is important to offload some tasks away from them, and that is why proxies came to exist.

In our system we have two types of proxies, one for each type of user. However both behave as a buffer to the ledger, thus an intermediate step between the UA/TA applications and the distributed ledger.

5.3.1 LEDGER PROXY

The ledger proxy is the one who receives the reports and proxies them to the distributed ledger. This approach, besides the previously discussed performance improvements, has another important advantage: allow us to regionalize the reports. This means that this proxy acts as a regional manager. If correctly distributed across a geographical area this would allows us to do some regional delimitations, thus enhancing even more the system performance.

For instance, without this proxy all the users would connect to a ledger node in order to report something. With this approach, the UAs would send their requests through the proxies and those have the ability of, for instance, batch several reports together and send them all at once to the public ledger, minimizing network impact. This batching mechanism does not compromise security, since the reports are signed by the user, it can even improve it if some verifications are ran, for instance, we can deter UAs from reporting events that they could not possibly be aware to report, based on the geographical provenience of the requests. Another improvement of this technique is that, while fetching reports present in the distributed ledger, we could fetch only the relevant ones to our geographical area instead of the whole set of reports, and proxies could even cache the reports off their geographical responsibility in order to improve response time to UA applications.

In a smart city scenario, this type of proxy could be implemented in already used components of a VANET network, specifically in OBU and RSU, which are geographically distributed across the city and could work as the entry point for our network.

5.3.2 SECURE SERVER

Another type of proxy present in our system is the one TAs use in theirs application. This proxy is similar to the previous in some aspects, as the batching of requests. However, it also has some different characteristics, for instance, the geographic properties do not make sense anymore.

This secure server has more responsibilities than the simple ledger proxy, besides proxying it will also be responsible for creating, batching, signing and do everything related with the TA. This does not mean that should be only one secure server for all the TAs, it means that each TA should have its own secure server that it uses to connect themselves to the network. It should monitor the distributed ledger, searching for important reports that should be dealt with by the specific TA in charge and send that information in real time to the TA application.

5.4 THE USER APPLICATIONS

For a system that allows people to sense their environment and report back some data, there has to be a bridge between these two worlds. This can be done through applications that people use. First, there are two different kinds of applications at stake: the one that is used by the UAs to report and endorse events; and another to be used by the TAs to monitor the system and respond to events. These applications must met some requirements. They both must have cryptographic capabilities in order to be able to follow the system protocol. Also, for UAs the application must not be resource intensive, since it will mainly be present in mobile hardware that may not have the best capabilities.

5.4.1 NATIVE OR WEB

When we talk about building applications for users there are mainly two ways of doing it, each with its own subset of advantages and disadvantages: we either build a web-based solution, or a native application.

We talk about web-based solutions when we have the application available through a web browser. This is a big strength for this type of applications, using the browser makes them extremely portable since it will be available in every platform that can access the Internet. This type of solution also has easier implementations and faster prototyping time. However, using the browser to build an application comes with one major drawback: you are restricted by what the browser can offer you. This means that you need to use the browser APIs to build your application. Usually, this is not a problem for many applications, since the browser is mainly used to present information while a backend computes and makes information available for the frontend to consume.

On the other side of the coin we have native applications. These applications are built with a specific environment in mind, making them only available in that specific environment and binding the programmer to the tools the manufacturer considers fit in order to program its devices (increasing development time in order to accommodate a new platform). However, native applications come with a big advantage, allowing the programmer to interact directly with the devices APIs, giving them more freedom and processing power. For instance, easier access to cryptographic primitives such as digital signatures; in handling media formats like video or audio; and even storage options.

5.4.2 UA application

Initially we wanted that the UA application to be the most simple it could be. Ideally, it would not perform any major action, such as digital signatures or any other similar cryptographic action. For the UA application, an easy solution would be a web-based solution, a website where a UA would, somehow, securely login and post his findings.

However, this approach would not work, because if we want people to be able to sign reports anywhere they are, then, they need to have their keys with them when using the system. We thought about relegating the responsibility of creating the cryptographic reports to another entity, but again, because of the keys needed to do this, it would not make sense to have any other entity creating the reports other than the UA application.

For this reason, the UA application needed to be able to perform cryptographic actions, such as signing the reports it builds and sends to the system. Nowadays, web APIs do not fully enable us to create signatures in a efficient way, namely, it would not be viable to load a user private key from their device and using a centralized approach to store users keys would defeat the purpose of our decentralized solution.

Then, the UA application should be a native application, for instance, an Android application. In our case this, enable us to more easily create reports, digitally signing them and also storing cryptographic secrets in the end-user devices without compromising the distributed way of our system. This application is then going to communicate with the ledger proxy, passing to it the reports created by the user.

5.4.3 TA application

With the TA application, we can follow the previous mantra of keeping it as simple as it can possibly be, since there is no problem of delegating the responsibility of creating new reports to the proxy. In this case this makes sense, because there will be more than one single person impersonating a responsible entity, for instance, a police precinct will only have one key pair associated with it but multiple police officers can close out votes.

So, the TA application can be a web-based portal where users interact with the real time reports being broadcast in the network. The application works as a remote control, issuing instructions and actions to another place. However, it should also meet some requirements, as the ability to broadcast and present information in real-time to the TAs.

5.5 SCENARIO

In figure 5.2 we can see a possible deployment scenario. In it, we can see that there is no central authority but rather trust in a distributed ledger. Also, we see the possibility of interconnecting our system with VANET components, namely, OBUs and RSUs. In different colors we see possible areas of ownership, each TA should have its own secure server and ideally should be in charge of at least one ledger node.

For the purpose of this work we devise some basic operations that our Proof Of Concept (PoC) users should be able to do. We also devised some use cases for our systems users, however, since there are different types of users, they also have different basic use cases. For instance, in a UA perspective, several things should be possible in a PoC:

- **UC-1** Create their own "account", getting a reputation wallet;
- **UC-2** Report on events that are taking place in the real world;
- **UC-3** Vote on existing events that are in discussion;

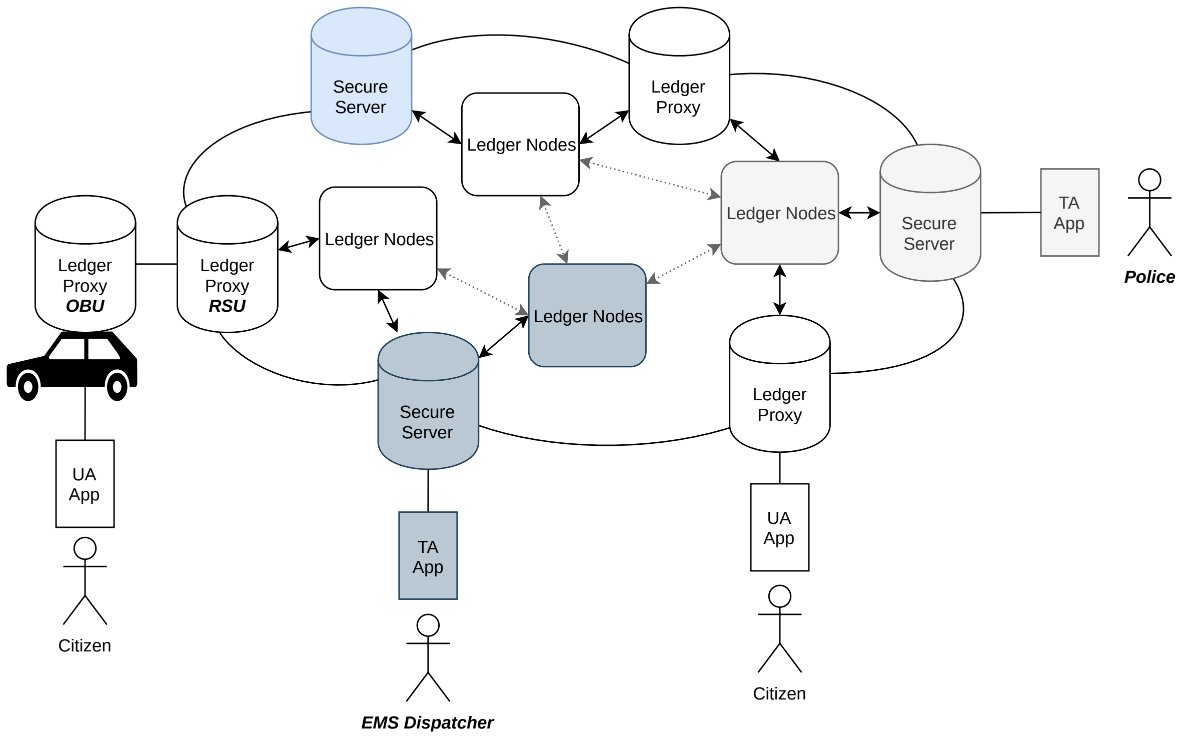


Figure 5.2: Possible scenario deployment of *Dignitas*

- **UC-4** Be able to know how much reputation has gathered on the network;
- **UC-5** Scout for events in specific geographic areas to gain information;

Another user perspective is the TAs one, and they should be able to:

- **UC-6** Be notified in real time of new events that people report;
- **UC-7** Close votes based on the information they gather;

These use cases were the base on which we built our scenarios and the needed components.

CHAPTER 6

Implementation

"Talk is cheap. Show me the code." – Linus Torvalds

This chapter introduces the details of our Proof Of Concept (PoC) implementation. As discussed in the previous chapter, this work is only possible with the interconnection of several different components. All of the components presented were modelled and tested successfully.

In Section 6.1 we talk about one of the programming languages used in the project, Rust, and why we chose it. Next, in Section 6.2 we dive into the ledger details, how we chose our distributed ledger platform, how it works and all the changes that we needed to implement in order for it to work with our solution. Section 6.3 is where we present the implementation of the UAs proxy, the ledger proxy. Followed by Section 6.4, where the UA application is presented. In Section 6.5 we present the TAs proxy, and the secure server. Finally, in section 6.6 we present the TA application.

6.1 RUST PROGRAMMING LANGUAGE

Currently, there is a plethora of programming languages that programmers can use. They are nothing more than tools that programmers can choose to perform their job, and like any other job, it is important to choose the correct tool. Before choosing a programming language it is important to understand the requirements of the application we are trying to build. When we look at some of the components of our system, as the ledger and the proxies, we see that depending on the deployment scenario they can experience high request loads. These types of components must be extremely reliable and are expected to be highly performant.

Interpreted languages (e.g. Python and Javascript) are still big candidate choices when it comes to prototyping a solution, since they are generally easier to implement. However, they tend to lack in performance, are prone to runtime errors, and because of their interpreted nature they cannot benefit from compile optimizations. Because of this, interpreted languages are not ideal for critical components in our PoC.

Usually, when it comes to high performant code the choice falls onto C/C++, a compiled language that has beaten the test of time. However, being highly performant comes with a price: developers are expected to know very well what they are doing. Compilers for C/C++ have been improving continuously to try and protect the developer from unsafe code; however, there are still many instances where C/C++ can produce unsafe code with undefined behavior, causing ambiguous and hard to predict bugs.

For some of our most critical components we ended up using a relatively new language: Rust [47]. Rust is focused on programming safety, while maintaining high performance. It is a multi-paradigm compiled language created by Mozilla Engineers to power the Firefox browser engine.

Syntactically, Rust is very similar to C/C++, separating statements by semicolons, using curly braces to delimit blocks of code and the same control flows operands. It introduces some new operands, such as *match*, a powerful pattern matching operand, and a *for* loop that works more like a for-each type of loop. It is also a statically typed language, meaning that variables types are know at compile time. However, unlike Java, it does not require the programmer to explicit write these types because they can be deduced with *type inference*, similar to languages like Haskell and OCaml. Dynamic typed languages, as Python, only infer variable types at runtime and it is a common origin of bugs.

Some language, as Java, use garbage collectors to automatically handle memory management, freeing memory that has been allocated but is no longer in use. This method has some advantages, mainly, it is a safer approach than relying on programmers to do it manually. There are also some drawbacks: it increases the resources needed, affecting the performance of applications namely when dealing with real-time requirements or low-power devices; it is non-deterministic, since the garage collection can run at any time, leading to unpredictable stalls on program execution; also there is no guarantee a resource is freed when the programmer wants it to be.

One of the more interesting features of Rust is its ownership system to handle memory management, ensuring memory safety without using a garbage collector. The ownership system allows for the compiler to check at runtime that the program is free of memory safety errors, such as dangling pointers, double frees and others. It does this by restricting the way the programmer can use pointers based on three simple rules: every value has a owner, there can only be one owner for a certain value at a certain time, when this owner goes out of scope so goes the value it owns.

By enforcing some rules to the programmer when dealing with variables and references, all memory management errors can be detected at compile time, which ensures run-time safety.

Rust characteristics allows us to build performant components without compromising on system security or reliability. In our PoC we used Rust to build some internal components of the ledger and also in the UA proxy, two of the components more likely to receive high loads of requests.

6.2 DISTRIBUTED LEDGER IMPLEMENTATION

One big part of our system is the distributed ledger. In Chapter 5, Section 5.2 we talked about its importance and which characteristics our ledger should have. Throughout this section we explain the choice process for the ledger used, the characteristics and implications of it and how we used them to ensure it met our system requirements.

6.2.1 CHOOSING A DISTRIBUTED LEDGER PLATFORM

When it comes to choosing a blockchain platform, there is also a plethora of options to choose from. Since the bitcoin network became a success and its cryptocurrency widely adopted, companies have tried to tip their toes on a market they all saw as being profitable and revolutionary. However, not every goose gives golden eggs, in fact, the vast majority does not.

Eight platforms were chosen for analysis. This analysis was conducted based on the information and documentation the authors provided, with the objective of finding the best option for this work use case. The platforms were analysed in two distinct areas, first in terms of capabilities and then in terms of usability. In terms of capabilities, things like the type of consensus mechanisms, the possibility for smart contracts, type of permissions, scalability and security were taken into consideration. For the usability part, documentation, community behind the project, activity, deployment and licensing were the focus.

Bitcoin [6] can be considered the original distributed ledger, designed to support the bitcoin cryptocurrency is a sturdy platform that has the benefit of being the first one and having a wide community supporting its mechanisms and growth. In terms of capabilities, it has the possibility for some kind of smart contracts but is limited. Uses the PoW consensus, known for its massive amount of processing power and energy needed. Is a permissionless blockchain, meaning that anyone can participate and join the network at any time.

Ethereum [15] belongs to the new generation of blockchains. It introduces the concept of smart contracts in a more generalized way, enabling the creation of truly decentralized applications. It still uses the PoW consensus, however has plans to move to PoS in order to deal with the problems presented above. It also behaves as a permissionless Blockchain just like Bitcoin.

On top of Ethereum was built a framework called Hydrachain [29]. This framework leverages all the benefits of Ethereum and adds a control layer that enables the creation of permissions in this ecosystem.

Another option available is the Exonum [16] platform, built to provide a skeleton for blockchain applications. It is a bare bone framework that gives a lot of flexibility to the developer. However, one of the main problems is the team behind the project: a small team without any major name backing the idea, which can be a problem in the long term.

Speaking of teams, another framework that was a possibility was the OpenChain Framework [43], a small, simple and modular framework, ideal for deploying a prototype. However, after close inspection, it was detected that the company behind the project was over and because of that the project was not maintained anymore.

Corda [10] is a framework with some special characteristics. There is no need for the global broadcast of data across the entire peer to peer network. It has a modularity that enables it to plug any kind of consensus algorithm. It is a sturdy platform with a clear business focus.

Based on Bitcoin we can find one more solution, the Multichain [41]. The main difference between them is the possible creation of multiple assets, multiple currencies.

At last, but not least, there is the Hyperledger project [30]. This project can be seen as an "umbrella" project, because it is composed of different projects, each of them independent from each other. Creation of the Linux Foundation Project, with members like IBM, Intel and Cisco; all the projects are well-backed by these teams. The most popular project, the Hyperledger Fabric, is a private blockchain with all the features a regular blockchain offers, such as smart contracts and a distributed ledger. Another project under this "umbrella" is the Hyperledger Sawtooth, a versatile framework that enables different applications without having a cryptocurrency automatically associated with the blockchain.

All this information is summarized in Tables 6.1 and 6.2.

	Consensus	Smart Contracts	Permissions	Scalability
Bitcoin	PoW	Limited	Open	Limited
Ethereum	PoW/PoS	Supports	Open	Limited
Hydrachain	PoW/PoS	Supports	Open	Limited
Exonum	BFT	Supports	Open	Limited
OpenChain	Direct Validation	Limited	Supports	Limited
Corda	Plugable	Supports	Supports	High
Hyperledger Fabric	Direct Validation	Supports	Supports	High
Hyperledger Sawtooth	Plugable	Supports	Supports	High

Table 6.1: Capability analysis of blockchain frameworks

	Open-Source	Community	Activity	Documentation
Bitcoin	Yes	Big	In Development	Extense
Ethereum	Yes	Big	In Development	Extense
Hydrachain	Yes	Big	In Development	Extense
Exonum	Yes	Small	In Development	In Development
OpenChain	Yes	Small	Closed	In Development
Corda	Yes	Closed	In Development	Extense
Hyperledger Fabric	Yes	Big	In Development	Extense
Hyperledger Sawtooth	Yes	Big	In Development	Extense

Table 6.2: Usability analysis of blockchain frameworks

In the end, the choice has fallen in the Sawtooth Distributed Ledger of the Hyperledger "umbrella" project.

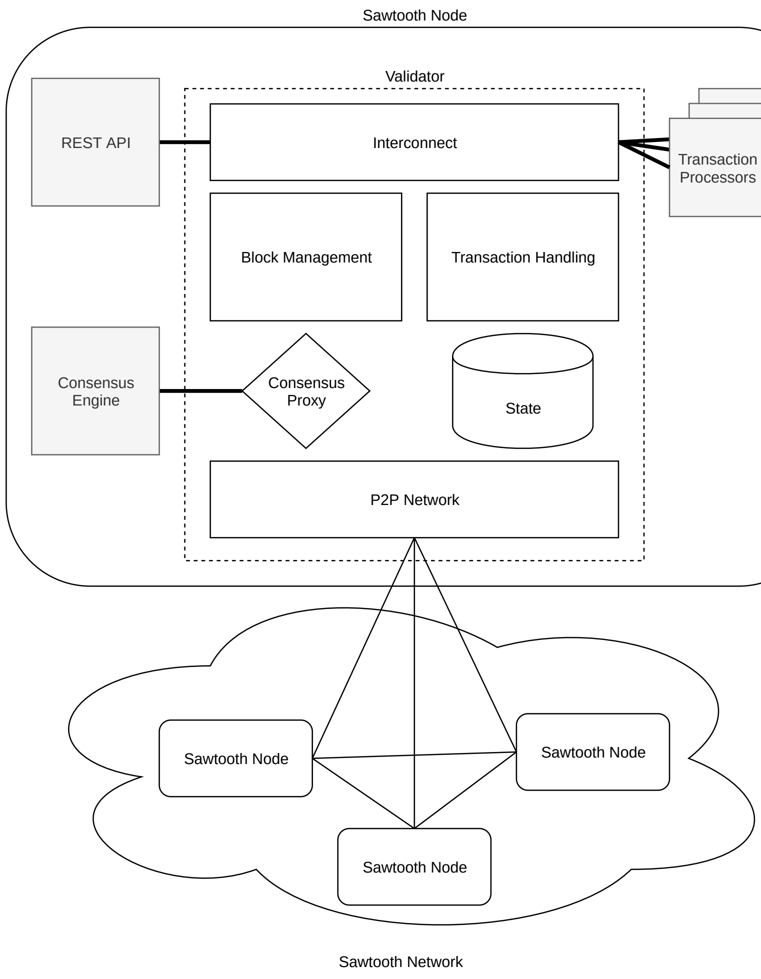


Figure 6.1: Sawtooth architecture overview

6.2.2 SAWTOOTH ARCHITECTURE OVERVIEW

One of the more interesting parts of the Sawtooth ecosystem is its components modularity; every part of the system is built in a modular way, allowing a more detailed control. It separates core system modules, such as the communication between peers and the P2P protocols that are the base for a distributed system, from application domain, i.e, the applications running on top. This modularity allows for a fine design while developing applications, there is no need to compromise in order to accommodate application-specific constraints.

Figure 6.1 presents a schematic overview of a Sawtooth network deployment. The Sawtooth network is comprised of several interconnected nodes, which are the central pieces of Sawtooth deployments. Each of these nodes is composed of several smaller elements. One of those elements is the validator component itself, it is one element that is always present no manner the deployment and always works the same way. The other components function on a "plug-and-play" manner, for instance: the consensus engine, that allows different consensus algorithms to be used; the transaction processors, that implements the business logic of the system.

In the following sections these nodes will be explained in more detail, in order to understand the changes and decisions made to accommodate our project and PoC. We subdivided the rest of this section in three parts: Section 6.2.3 explains the networking principals used within this framework, the protocols and messages; Section 6.2.4 presents what we called our data layer; this layer encompasses the methods used to store and how to store information in this P2P network and also how this information is allowed to mutate; finally, Section 6.2.5 presents the application layer, namely, the transaction processor implementation which is the component responsible to handle our application business logic.

6.2.3 NETWORK LAYER

As with every distributed ledger, it guarantees a distributed architecture with every participant in the P2P network having access to the same information. This connectivity is the responsibility of an abstract layer that we called the network layer. This layer is responsible for several things that are the backbone of the whole system: initial connectivity, peer discovery and message handling. The system nodes use specific interface and ports to listen to connections.

After the initial connection the nodes exchange messages with each other based on an epidemic protocol. Epidemic protocols, also known as gossip, are used to disseminate information efficiently in distributed networks. Such protocols can also be used to provide failure detection and to self-organize complex network topologies.

The Sawtooth network uses ZeroMQ (ØMQ) to handle communications which is a platform-agnostic network library. It provides a framework that offers sockets that carry atomic messages over diverse transport layers, from local message passing to TCP and multicast. I/O operations are modeled in an asynchronous way, enabling the creation of scalable applications based on message passing, ideal for a P2P network. Within ØMQ, an asynchronous client/server pattern is used, an N-to-1 architecture, where various clients talk to a single server, asynchronously. But, since a node in the network works simultaneously as a client to different servers and as a server to different clients (see Figure 6.2), the generated connection are, in fact, N-to-N between all the peers.

Each node can be seen as a state machine, changing between 3 different states: unconnected, connected and peered. They define the state of the connection between any two nodes. The nodes are *unconnected* when there is no connection between them, *connected* when there is a communication between tho nodes and *peered* when both nodes establish the client/server pattern between themselves. In Figure 6.3 we can see the available protocol messages and how the node transits between states based on those messages.

Most of the times, in complex scenarios there is not a full mesh connectivity, but it is ensured that each node achieves a minimum of connectivity with n other nodes. The bi-directional peering based on this neighboring construction ensures a reliable connectivity throughout the network.

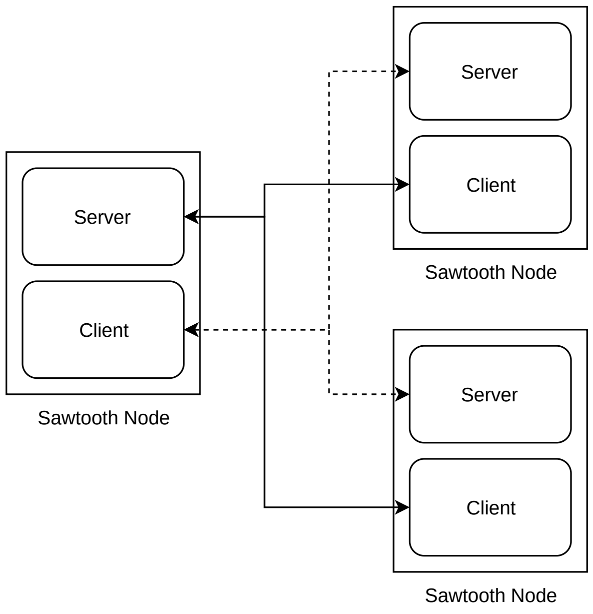


Figure 6.2: Network layer architecture

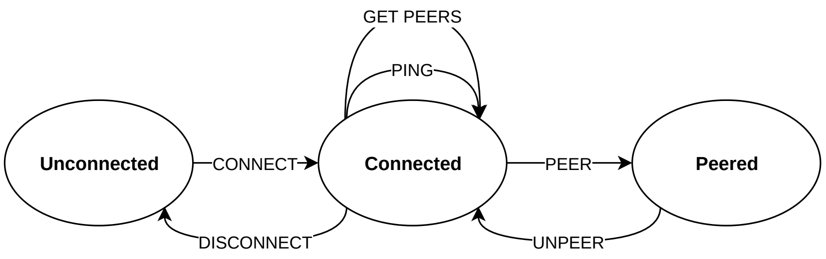


Figure 6.3: Node states

6.2.4 DATA LAYER

As we previously mentioned the main objective of DLTs is to, as the name implies, distribute a ledger. This ledger is nothing more that a type of database that all nodes should be able to access in a consistent manner.

Classical DLTs, distribute information in the form of transactions. Each node does not store a current state, for instance, the only way to know for sure the amount of bitcoin a person has is to iterate all of the blocks and find transactions from and to a specific person. Imagine a bank statement that instead of giving you the transactions and the amount you have, only gave you the list of transactions and you were responsible to add it all up.

More recent DLT have a state and record transactions that alter that state, so if all the nodes follow the same transactions, all the nodes have the same final state. The state allows for the implementation of more complex use cases and applications, such as smart contracts. In Sawtooth this is called global state and every node is responsible for its copy of the state.

In terms of implementation, the state is an addressable Merkle-Radix tree. This is a Merkle tree, because it stores a hash that corresponds to the hash of all hashes of the tree nodes. This process starts at the leaf nodes, where the hash label corresponds to the data hash, all the way to the root node, passing through all the intermediate nodes, which in turn compute their label hashes by adding and hashing the hashes of their respective child nodes. The root hash of the tree is used in the block header to gain consensus not only on the chain of blocks but also in the state of the global state. The global state is also a Radix tree because its addresses allow for the identification of the path to a leaf node in a unique way, property which we take extensive advantage of.

So, to store information in the global state we just need an address. We can think of the global state as a big dictionary that uses addressing to store information. The addresses available to use are 35 byte long. In Sawtooth, the first 3 bytes are special, they represent the namespace of the address. We already talked a little about the transaction processors that are responsible to implement the business rules of the system. The namespace allow us to have multiple transaction processors, each responsible for their own namespace and also, as a security feature, transaction processors cannot access addresses that are not under their namespace.

The data stored in each node of the tree state is a byte array that is transparent to the node. Here, we can see an example of a leaf node full address that can be used to store information in the global state:

$$\overbrace{\underbrace{\texttt{657374}}_{\text{Namespace prefix (3 bytes)}}\,\texttt{6F752070656C6F7320636162656C6F7320636F6D2065737461207465736520202E}}^{\text{Example global state leaf node address (35 bytes)}}$$

For our scheme there are two things that we choose to store in this ledger. First, and the most easy to understand are the reputation wallets, which are a simple integer value representing the amount of *dignitas* each UA has; the second one are the event reports made by the participants. In order to do this an we needed a addressing scheme that followed some rules:

- Have a namespace prefix;
- Be deterministic;
- Be collision resistant.

Having a namespace prefix and a determined size are restrictions enforced by the Sawtooth framework. They need to be deterministic, because different nodes and clients must calculate the address in the exact same way, any time they are needed to be accessed. Also, ideally, it should be collision resistant; if not, the consumers of this addresses should know how to handle collisions, also in a deterministic way.

As previously stated, there are two main types of addresses, however, they share the first 3 bytes, the namespace.

$$\text{Namespace prefix} = \text{First 3 bytes hash}_{256}(\texttt{"dignitas"}) = \text{CE9618}$$

The next byte in our scheming address is used to discern between the two types of addresses: 00 for the reputation wallets, and 01 for the reports. Let us start with the first one, wallet addresses. These addresses are used to store the reputation value of each UA, a integer value that represents the amount of reputation they gathered so far. These type of addresses have, besides the namespace and the discerning byte, 31 remaining bytes used to identify the user. These bytes are the first 31 bytes of a 521-bit digest of the UA public key (k_{pub}).

$$\text{Wallet address} = \texttt{CE9618} + \underbrace{\texttt{00}}_{\text{Discerning byte (1 byte)}} + \overbrace{\text{hash}_{512}(\texttt{pubK})}^{\text{User Id (31 bytes)}}$$

The addressing scheme is used to store each Wallet or Report in the correct location in the ledger state but also to enable their retrieval in a deterministic way. Because of the radix nature of the global state tree we can use our addressing scheme to, for instance, retrieve all the addresses that correspond to wallets. The address `CE961800` represents all the addresses that start by that sequence, in this case all the Wallet addresses. This property is very powerful and it is used to a greater extent in the Report addresses.

First, each Report on our system has a specific address that can only be used by one Report at a time. The Report address has the same initial bytes of their Wallet counterparts, the namespace using the first three bytes and also the discerning byte. The remaining bytes are used to uniquely identify the report. In our PoC we use a geohash in the next 6 bytes.

$$\text{Report address} = \texttt{CE9618} + \underbrace{\texttt{01}}_{\text{Discerning byte (1 byte)}} + \overbrace{\text{GeoHash}(\texttt{Vote Location})}^{\text{Vote Id (6 bytes)}} + \text{Remaining 25 Bytes}$$

GeoHash [20] is a hashing algorithm built for geographic coordinates, a geocoding system. It encodes geographic coordinates into bits and it decodes bits into geographic coordinates. The public domain geocoding system GeoHash encodes the bits in a strange alphabet that takes into consideration the original use of this encoding method, replacing the Postal Code System and enabling a global code that could be used to send mail to any geographic coordinate. For this reason, since the code was designed to be written in letter envelopes and read by mailmen, some characters are not used so that they could not be misinterpreted as numbers (e.g the o and the 0). Because of this design quirk we had to build our own library to encode geographic coordinates using the standard hexadecimal notation as the final encoding method. It works in a very simple way: it starts by dividing the globe in two halves, the algorithm adds a 1 or a 0 based on which half the geographic point is on, this process keeps repeating until the necessary precision is achieved.

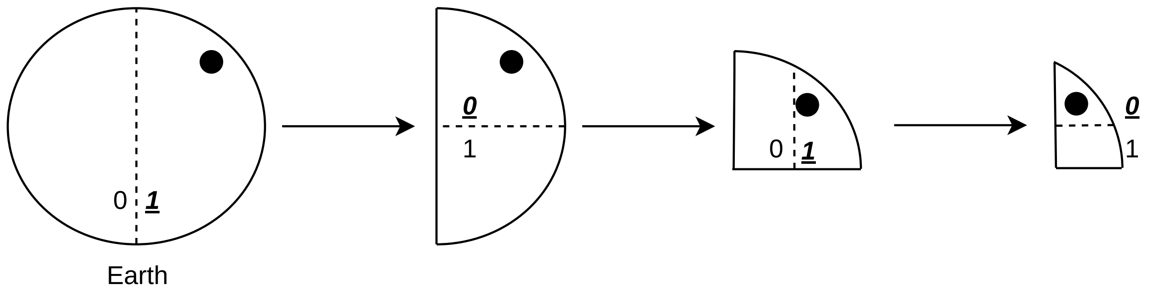


Figure 6.4: Geohash algorithm in 2D

In Figure 6.4 it is shown in a simplified way the algorithm: in that example the resulting bit string would be 1010 that would translate to `0xA`. This encoding method has variable precision, every string of bits will always represent a curved surface on earth, with more bits this surface area is smaller and represent a more precise location, another important characteristic is that, the longer the prefix two places share, more likely is for them to be closer together. Both of these properties allow us to retrieve votes based on their location without any major calculations.

We previously stated, that the GeoHash algorithm itself has variable precision, however we use a fixed number of bytes to represent the hash of a Report location. This precision is important since we use it to prevent "deja-vu" events, a UA is not capable of repeating events in the same location; at least, not until they are resolved. Since area calculations on non-spheric planes is out of scope for this work we calculated our precision based on existing values for the original GeoHash. In our solution we use 6 bytes of hexadecimal characters, meaning that in total we have forty height bits encoding our GeoHash. In the original GeoHash, bits were encoded using a thirty two character alphabet, which means that it would be able to encode our forty height bits into at least nine characters. In [21] we see that for our number of characters we can have a rectangle of approximately three by three meters, which makes it the precision of our Reports.

The information stored on each leaf node of the tree is nothing more than a group of bytes to the validator node but it must yield some meaning to the clients that want to use that information; in this case, the Transaction Processor (TP), which is the validator component that stores and reads information from the global state (more details in the next section).

Since the information stored in the global state has no intrinsic meaning associated with it, it can be anything, since we are only storing bytes. However, one must be able to store something and retrieve it in the same exact way, extracting the same exact information.

With the Wallets this is a simple problem, since we are only storing the reputation value value of the UAs, which is a simple integer, it is how many *dignitas* a user has. The encoding method used to store this value is simply convert the value to bytes and store it in the respective address. When the value is retrieved is decoded from the bytes and interpreted again as a integer.

Problems start to arise when we want to use more complex structures. With Reports we need to take into consideration that we want to store more fields, and not only a numerical value (e.g the bets of other users). For this, we use objects to represent this types of structures. However, we needed to be careful with the object we choose to store this information. Since what is stored in the ledger is only a collection of bytes, it is critical to use a serialization mechanism which is deterministic across platforms and across time. The use of *Maps* or *HashMaps* is not advised, since they produce non-deterministic byte arrays and could make the network useless, since some validators would consider a state valid and then other validators might not be able to find the same result. Because of this serialization concerns, we used a serialization method that can be reproduced across time, platforms and allows the use of complex objects, the Concise Binary Object (CBOR) [8]. CBOR is a data format specified in

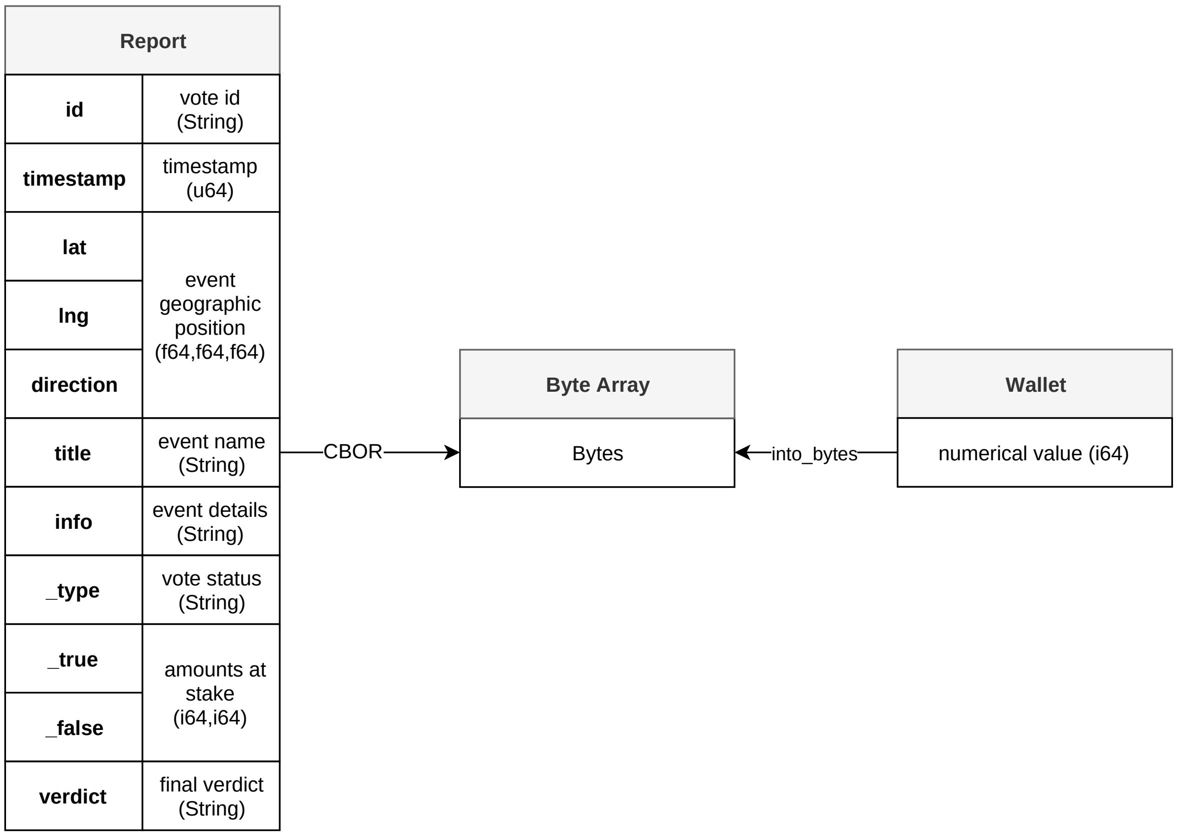


Figure 6.5: Report and Wallet data structures

IETF RFC 7049, a format that was designed to be stable for decades. It is build around the JSON data model, with a key-value design without the need for a schema; basically,this is everything we needed. This data format allows us to store complex information in a binary format while enabling others to retrieve the same information and decode it in the same exact way.

In Figure 6.5 we can see all the fields that compose each Report and each wallet as well as the encoding method used for each one. Those are the fields stored in the distributed ledger, the ones that are stored in the Report and Wallet addresses, respectively. These are the base structures that act as the ground truth for the rest of our system.

Next, it is explained how the TP works and how it interacts with the data stored in the ledger. The ledger information, with the global state, behaves as a dictionary where each entry as a specific address and each value a specific encoding. The information stored in this dictionary (the global state) is only modified through transactions. Transaction can perform multiple things, such as changing the data stored in a specific entry or creating new ones. Transactions arrive at the nodes wrapped around in Batches. A Batch is the SQL equivalent of a SQL-Transaction, meaning that all the operations inside a batch are performed or none is. The Batch is the atomic unit of state change. These mechanisms are explained in detail in the next Section.

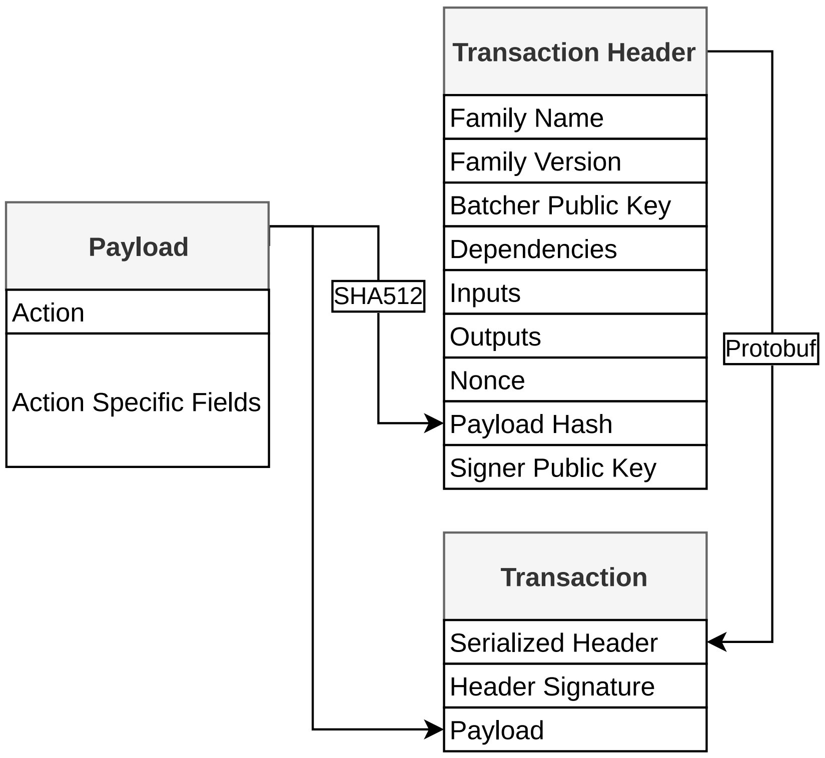


Figure 6.6: Fields of a Sawtooth Transaction

6.2.5 APPLICATION LAYER

In Section 2.2, we talked about the core concepts of DLTs, one of them being the process that a requester has to go through in order to submit an update into the Ledger. It starts by creating a transaction where encapsulates the desired change, then submitting that transaction to a validator node that has the responsibility of verifying the conditions of that transaction. As previously stated, the Sawtooth Distributed Ledger abstracts the application-specific verifications from the core system in a subsystem called Transaction Processor (TP).

In Sawtooth, Transactions and Batches are the basic units for state change and the component responsible for handling and implementing these changes is the TP.

In Figure 6.6 we can see the different fields that are used to generate a transaction. First, the payload is where the action we want to apply is specified; more on that later. The payload is then encapsulated in a transaction, this transaction is made of a header, its signature and the payload. The header has several fields that are necessary in order for our distributed ledger to work in a trustworthy and safe way:

- The family name and family version are just fields used by the validator node in order to send the transaction to the correct TP;
- The batcher public key is a field that indicates the public key of the transaction batcher, meaning that one entity can create the transaction and another could be responsible to batch them together. However, not any batcher can grab transactions and batch them together, only the one that has the private key to this public key can do it;
- Dependencies is where we can specify transactions on which this one depends. Since

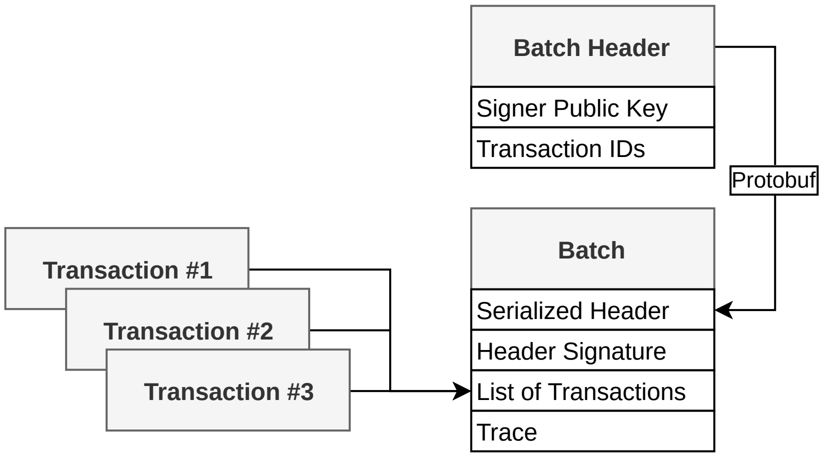


Figure 6.7: Fields of a Sawtooth Batch

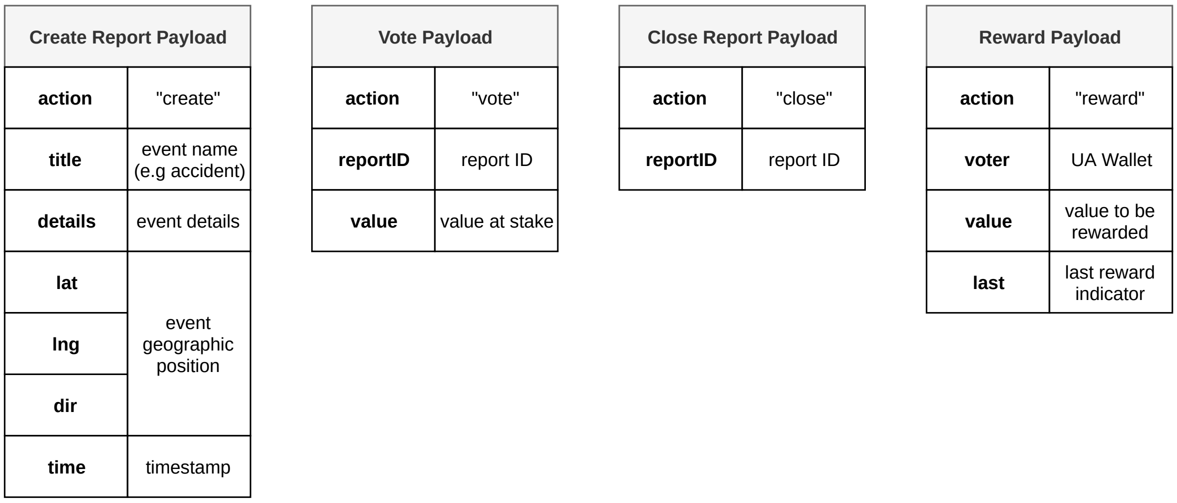
this is a distributed process, there is no guarantee that transaction will be applied in the order they arrive to the node. For this reason when a causality dependence exist, we need to specify it;

- Inputs and Outputs represents the global state addresses that are going to be used while performing the payload operation. If a transaction tries to access addresses not specified in this field it will fail;
- The nonce is used in order to guarantee that even if everything is the same, we can still differentiate two transactions
- The payload hash is a SHA_{512} digest of the payload;
- The signer public key is the public key of the one who created this transaction.

These transactions are then wrapped around in a batch; even if it is just one transaction, it must always be wrapped around in a batch. In Figure 6.7 we can see what constitutes a batch. This structure also uses a header to ensure cryptographic safety through signatures. In the header is present the public key of the signer and a list of the transaction IDs that are present in the batch. This way, attackers cannot hijack batches adding transactions to the batch illegally. The batch body is composed of the elements necessary to ensure the signature works and the list of transactions. The Trace is a field that a developer can use in order to trace the batch through the blockchain for debugging purposes.

One of the purposes of transaction/batches is to deliver in a safe way the payload to the TP, so it can alter the global state of the system. The validator node receives the batches from its clients, run validations, schedules them and sends them to the respective TP. The TP receives the transactions, runs verifications and applies the changes to the global state. It is the one responsible to apply all the application logic.

Within our TP there are 4 main types of actions allowed, which include the creation of reports and voting on them by the UAs and also closing votes and reward users by the TAs. These actions have specific associated payloads generated by the system actors when they want to perform each action. The payload is a key-value structure that instructs the TP what action should be performed and with which values (see Figure 6.8).



Create Report Payload	
action	"create"
title	event name (e.g accident)
details	event details
lat	event geographic position
lng	
dir	
time	timestamp

Vote Payload	
action	"vote"
reportID	report ID
value	value at stake

Close Report Payload	
action	"close"
reportID	report ID

Reward Payload	
action	"reward"
voter	UA Wallet
value	value to be rewarded
last	last reward indicator

Figure 6.8: Payloads of different actions

After the validator receives a batch it will schedule its correspondent transactions into their respective TPs. The TP will then extract the payload from the transaction, namely, the first field that dictates the respective action to be made. After determine the action, it calls the function that is supposed to handle that action.

Let us start by the process of creating a Report. After determining that the payload objective is to create a Report, the program will try to extract the rest of the fields that it expects to be present in the payload, which are:

- the title, where the user specifies the title of the event (e.g. “accident”, “party”, “broken water pipe”);
- the details, where the user can give more informations about the event;
- the lat, lng and dir that correspond to the event latitude, longitude and direction or heading, respectively;
- the time field, that corresponds to a timestamp.

If any of these fields does not exist, the transaction fails and is not applied. With this information the TP has the conditions to create a new vote in the global state of the ledger. In order to do this, the TP calculates the address of this vote and stores the vote information as explained in Section 6.2.4. Creating a new vote is a fairly simple ordeal in the TP perspective.

The voting process (i.e putting reputation at stake) goes along in a very similar way, it starts by extracting the payload fields: reportID that represents the ID of the Report on which the UA wants to vote; and the value that it wishes to put at stake. However, it must perform some validations and side action in order to successfully vote, for instance, it has to ensure the UA has sufficient *dignitas* in order to place that Vote on a Report; It is important to note that we learned in section Section 6.2.4 that the Wallet addresses are computed through the public key of the UAs. So, in order to verify that a specific user has sufficient funds to perform a specific vote, we need the public key of that voter. However, there is no field for the voter public key in the payload of the vote action. This happens because that public key is already present in the transaction itself in order to verify signatures. This enables us to access it and to use it while also ensuring that users are not able to vote with somebody else's *Dignitas.* After ensuring the voter credit, the TP performs several actions. First it updates the voter balance, removing from his account the value they want to put at stake, and then, updates the Reports _true or _false fields in order to reflect the opinion of the voter. If any of these steps fails the whole transaction is discarded.

Both of the previous actions can be made by any UA, with the only pre-requisite being that while trying to vote it must have sufficient credit. The next two are special actions only accessible to TAs, the closing and rewarding actions.

They both are very similar they both update state in a simple way, the closing action simply updates the Report state and the rewarding updates the Wallet values of the UA its payload specifies.

We implemented this TP using Rust, as said before. The TP behaves as a plugin and, although being inside the Validator node, is a subsystem that communicates through messages with the core of the node. We used the Software Development Kit (SDK) provided by Sawtooth which allows for a straightforward implementation of the topics above. From our work, we were also able to contribute to the Sawtooth project itself: There was no documentation for the Rust SDK and, since we were able to use it successfully, we decided to write some documentation for it and share it with the community [53].

6.3 UNTRUSTED SIDE LEDGER PROXY

The UA Proxy is a REST server built in Rust. Rust was chosen because of some characteristics that had to be taken into consideration while building this proxy, namely:

1. The need for it to be small, since it was meant to be deployed in components of a VANET network, specifically in OBUs and RSUs, which may have limited processing capabilities;
2. The need for it to be highly performant, since if we expect to deploy this system in a city we should also expect incredibly high amounts of connections to this proxy at any given time;
3. Rust programming language offers both of these, giving us great performances with small overheads.

In this PoC we used the Rocket library [46], a web framework that allows quick prototyping of web applications. Since this is just a PoC, it is important to build code that allows evolution, namely when it comes to communication forms, because our system should be able to be deployed over any type of network and be agnostic to the underlying network implementation.

In Figure 6.9 we can see the way the proxy was built that allows for a very modular approach. The `bin` folder holds the entry points of our application, is a very simple file that only starts the web server, making our application listen for HTTP requests on a specific port. The `lib.rs` file is where we define how we process transactions, how we should handle cryptographic elements, basically, it is the file that holds the "business" logic of the ledger.

```
/proxy_src
├─ /bin
│  └─ main.rs
├─ lib.rs
├─ /comns
│  ├─ api.rs
│  └─ out.rs
├─ /data
│  └─ schemas
└─ /util
   └─ transaction_helper.rs
```

Figure 6.9: UA proxy source code structure

This file is supported by other files that help subdivide the problem in sections, in order to better scale this solution. In the `comns` folder two important files can be found: `api.rs` where we define the exposed Application Programming Interface (API) for the UA client application to use; `out.rs` is where the communication with the ledger is defined; in our PoC is also established through HTTP. The way we built this should ensure that changing communication protocols should be trivial. In the `data` folder we can find the data schemas of objects that travel through the ledger, since Rust is also a strongly typed language, every object must have a type and its size must be known at compile time. This ensures type safety, but also forces us to think very well about the data we will be handling. At last, but not least, there is the `util` folder, where we can find the `transaction_helper.rs` a module which sole purpose is to handle operations that relate to Sawtooth transactions, such as creating batches with the client transactions and cryptographically sign these batches before sending them to the ledger.

6.3.1 ENDPOINT SPECIFICATION

This specification corresponds to the publicly available API that is present in our ledger proxy. All endpoints are prefixed with `/api/v1`.

POST /TRANSACTION

This endpoint is the most used one, it is through it that the UAs can submit transactions to the ledger. This endpoint accepts a HTTP POST that should have in its body a serialized version of a transaction. The proxy creates a batch from the incoming transactions, and signs them with its private key. We use this to enhance the security of our system, because a UA can specify the public key of a transaction batcher, meaning that the UAs can choose to which proxy to send their transaction, and we can securely guarantee that these transaction are not hijacked and used somewhere else. If, for any reason, the transaction sent is not valid, this operation will fail. Another advantage of having a proxy is that we can keep this errors from the main ledger, meaning that a Denial of Service (DoS) attack could be stopped in the proxy layer without interfering with the ledger. After creating the batch, it is sent to the ledger. This part functions on a best effort basis. Because of the nodes high mobility, it is impractical to maintain the notion of state on the proxy. So, instead of waiting for a response from the ledger, we respond with a OK if this process of sending the transaction to the ledger went as planned. The OK from the proxy does not mean that the transaction arrive to the ledger or that it will be added to the global state; it just means that the proxy did its job of proxying. The responses from this endpoint (see Fig. 6.10) are simple JSON messages signalling if the process went as expected.

```
{
  "status" : "ok"
}
```

Figure 6.10: Response from the POST /transaction endpoint

GET /REPORTS

This endpoint is the one used by UA applications to retrieve the reports from the ledger. The client application just needs to make a simple GET HTTP request. As discussed before the proxy could act as a geographic filter, only dealing with requests of a predetermined area. This does not imply that a proxy cannot retrieve reports from other geographical areas, since the ledger is public and all the information is available. The proxy can also choose to cache the information it retrieves from the ledger. This information is always timestamped and digitally signed by the ledger validators so UAs can always verify if the information they receive from the proxy has been tampered with.

The endpoint can also decode the responses from the ledger, for instance, if the proxy is deployed in a OBU, we could assume that it is a trusted environment and that communications between the OBU proxy and the on-board UA application can be considered secure. If we assume this, we can offload the UA application even more by decoding the byte array received from the ledger to a format simpler for the user application to understand avoiding having to deal with serialization in the UA application side. In this cases the response can be a simple JSON that sends the votes information to the user application.

```
{
    "timestamp": timestamp in seconds,
    "votes" : [list_of_votes]
}
```

Figure 6.11: Response from the GET /votes endpoint

GET /BALANCE/<WALLET>

This endpoint, as the name says, retrieves the *dignitas* balance for a specific wallet. The user application should make a GET HTTP request with the wallet address it wishes to receive information about. Like the previous endpoint we can simplify the task of the UA application by deserializing responses from the ledger and only sending the valuable information to the client. The wallet address is the address where the wallet is stored in the ledger.

```
{
    "timestamp": timestamp in seconds,
    "value" : amount of dignitas on the <wallet>
}
```

Figure 6.12: Response from the GET /balance/<wallet> endpoint

6.4 UA android application

For a system to be useful, its users must have a easy way to interact with it. In this PoC we used Kotlin to develop an application targeting Android devices. The application was built using the Model-View-ViewModel (MVVM) architecture. This architecture helped us build a sustainable and scalable application, while also being considered heavily favoured by the Android community because of the available Android components that enable it.

In MVVM, we have the Views that are responsible for the immediate interactions with the user. They do not execute any business logic; their main purpose is to take care of the UI and dispatch events to the level below. The level below, the ViewModel, serves as a bridge between the UI and the business logic of the application. Finally, there is the Model, where the domain-specific data appears in the form of a repository, which is seen as the single source of truth for the system. The repository itself can gather information from multiple sources, but as far as the ViewModel knows, there is only one place to fetch information from. All of these components are empowered by a new data type in the Android ecosystem, the LiveData, which is an Observable data type that allows changes from data to seamless flow from the Repository to the View. The Views observe LiveData on the ViewModels, than in turn observe them in the Model (i.e the Repository). When some value is changed in the repository, a chain reaction is triggered updating all the values all the way up to the UI. In Figure 6.13 we can see the schematic architecture of our application, where all of this components are present.

The application was built in order to explore our API and enable the user to interact with the system. We achieved this by using the Retrofit [45] and Gson [23] Libraries, simple yet powerfull solutions. We also take advantage of Kotlin coroutines to handle asynchronous, such as for fetching remote data. In order for the application to work even when offline some information is cached in the Local SQLite Android Database using the ROOM library allowing a nicer user experience even when our proxy cannot be reached. The application also uses

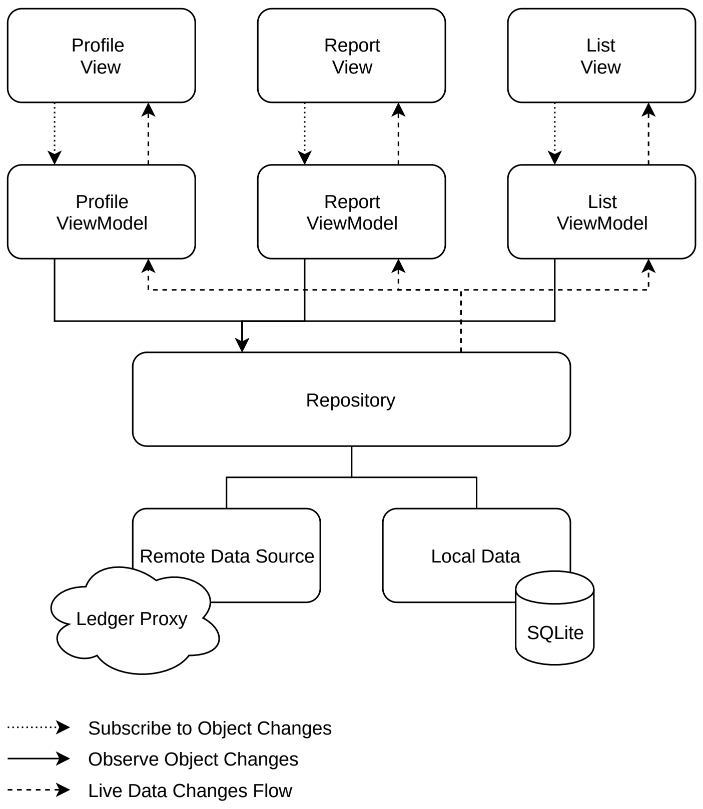


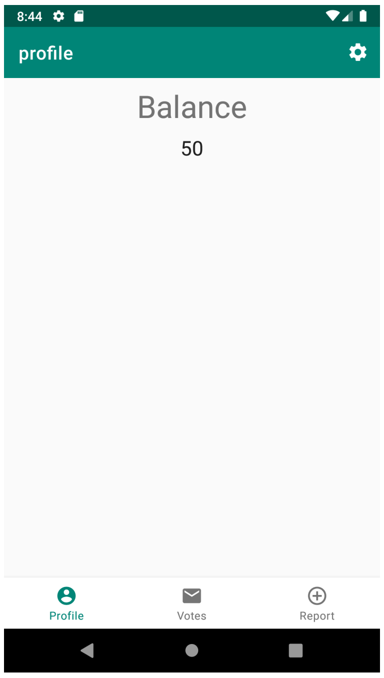
Figure 6.13: Dignitas Android application architecture

the Kodein Library, that allows us to perform dependency injection, in order to enhance the maintainability of our code.

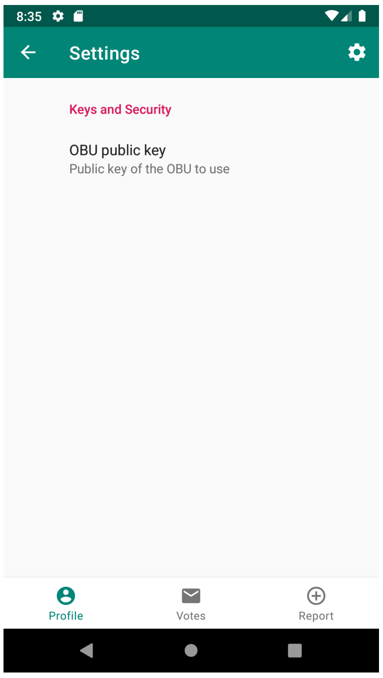
Figure 6.14a show us the profile page of the application; this is the entrance point of the application, in our PoC we just added in this view the current balance of *dignitas*. Going forward, this view can be reused to show more information, for instance, the history of transactions the user has done so far. In Fig.6.14b we see the settings page, where the user can input the desired batcher address, for instance, to lock its application to communicate only with the proxy ledger that is inside the OBU of its vehicle.

Figure 6.15a is the view that the user uses to find nearby events, the events appear as a overlay of points in a Map, this map should be centered around the current location of the user and show all of the events in the area, while also showing which of them have been resolved and are closed and the ones that are still open for voting. When a user clicks on a particular event, he will be able to see its details in the detail page that we see in 6.15b. This view also enables the user to see positive or negative opinions (votes) on the event, while also being able to cast his vote.

Finally, in Figure 6.16a we see the report page, this is where a user can report new events by filling up some fields, such as the title and detailed info. GPS information such as position and heading can be inferred from the device when available.

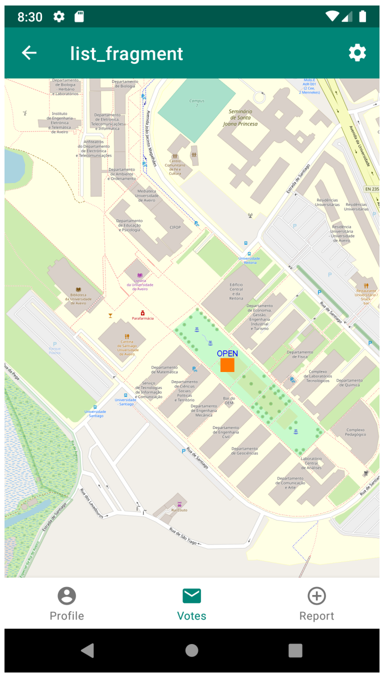


(a) Profile page

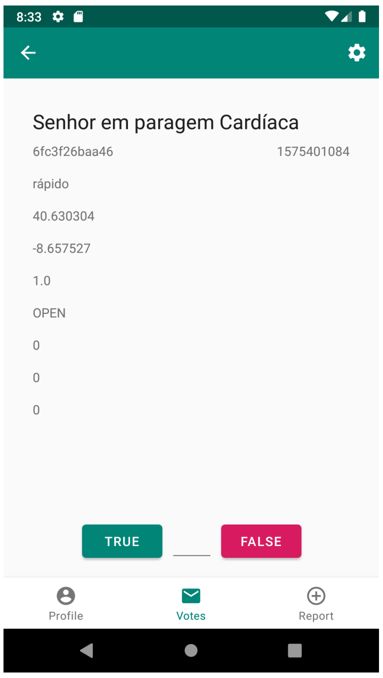


(b) Settings Page

Figure 6.14



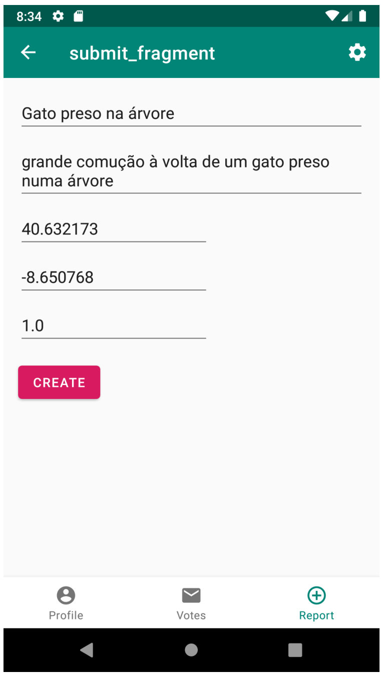
(a) Nearby events page



Senhor em paragem Cardíaca
6fc3f26baa46 1575401084
rápido
40.630304
-8.657527
1.0
OPEN
0
0
0

(b) Event details page

Figure 6.15


8:34

submit_fragment

Gato preso na árvore

grande comução à volta de um gato preso numa árvore

40.632173

-8.650768

1.0

CREATE

Profile Votes Report

(a) Report page

Figure 6.16

6.5 TRUSTED-SIDE SECURE SERVER

The secure server is a NodeJS application. Unlike its counter part from the untrusted side, it does not have tight restrictions in terms of performance, since it can be deployed in more powerful hardware. This component also performs more actions than its counterpart, it is more than just a proxy and it is actually responsible to create transactions for TAs, for instance, the transaction that closes out reports and redistributes *dignitas*.

In Figure 6.17 we can see how we divided and built the secure server. Since one of the main objectives for this server was to relay information in real time to the authorities, it communicates with the main ledger through a websocket connection that keeps the server updated with every new piece of information that appears on the ledger. It is also possible to only be listening for some kind of events in a pre-determined area, achieving the regional properties that we previously mentioned. This information, captured on the public ledger, is

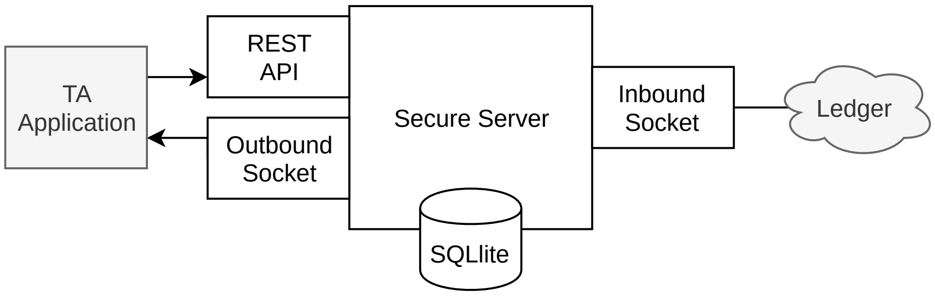


Figure 6.17: Secure server implementation

directly piped to the TA application also through websockets. For websockets operations we used the socket.io library [51].

Other than proxying information from the ledger to the TA application, there is one more action that the component must perform, which is creating rewarding transactions. These transactions are triggered by the TA application calling a specific HTTP REST endpoint. We use the express library [17], a minimal web framework ideal for either prototyping and big scale projects. In order to support the rewarding part of the system, we have a local database that stores IDs of reports being made in the network in order for us to just query this local database when we want to reward people for their behavior. For testing purposes we use SQlite, where the system stores references to UA reports in order for it to be able to trigger rewards without having to transverse through the ledger.

6.5.1 ENDPOINT SPECIFICATION

These endpoints correspond to the exposed API of the secure servers. Unlike the other proxy there is not only HTTP REST endpoints available, there is also a websocket endpoint.

POST /CLOSE/<REPORTID>/<VERDICT>

This endpoint is the one used when a TA wants to close out a Report. The information that should be provided is the reportId and the verdict from the TA. After this, the secure server will fetch the UA report that it has been storing locally, calculate how much the reporter and supporters should get, create several transactions to mirror these changes, batch all of them and then, finally, send this batch to the ledger. The way we are redistributing *Dignitas* is trough using the algorithm discussed in Section 4.4, in practice, we are redistributing the *dignitas* wrongly staked by UAs that correctly voted on the report.

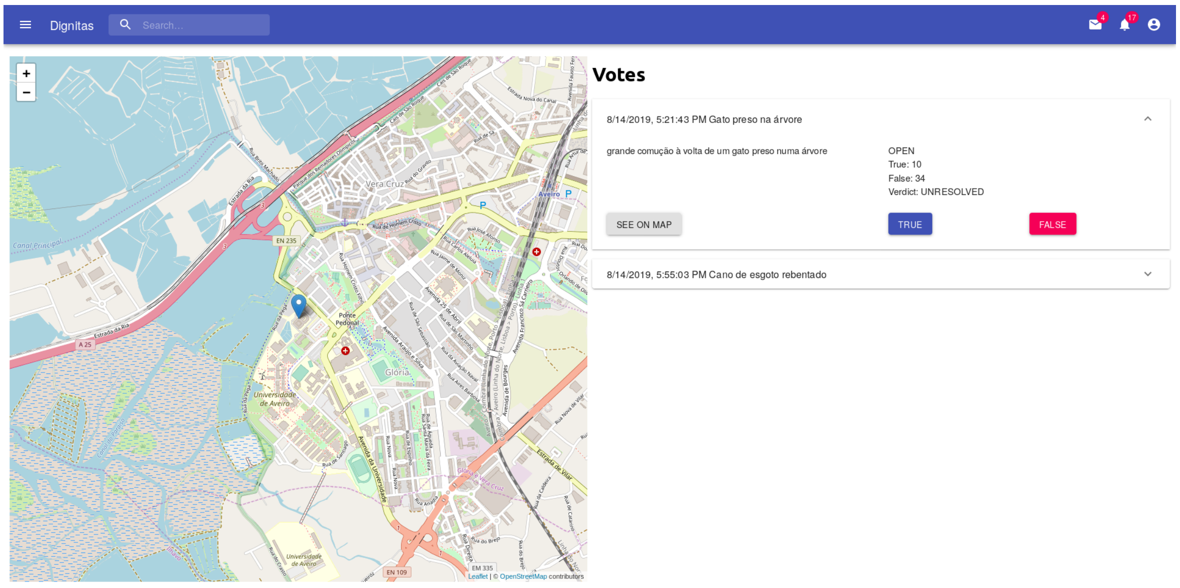
WS://<SERVER>:<PORT>

This endpoint is the websocket that the TA application uses to connect and receive live events' data from the ledger. This endpoint is very simple because it simply holds a connection to the application and everytime we get a new event from the socket ledger we directly pipe it into this websocket, so that it could flow to the application.

6.6 TA APPLICATION

Last, but not least, we have the TA application, which is a simple React [18] application using Redux [44] to manage application state. It receives new information in real time through web sockets and instructs the secure server through the provided REST API.

In Figure 6.18 it is possible to see how it looks like. It provides information about real time



Votes

8/14/2019, 5:21:43 PM Gato preso na árvore

grande comução à volta de um gato preso numa árvore

OPEN
True: 10
False: 34
Verdict: UNRESOLVED

8/14/2019, 5:55:03 PM Cano de esgoto rebentado

Figure 6.18: TA application

reports by showing a list of events. Besides the list view, it can also open up the report details where they can for instance see the report position on the map or trigger the reward process. Its only responsibilities are to provide a visualization of the data and to give instructions to the secure server which makes this a very simple component.

Chapter 7 Results

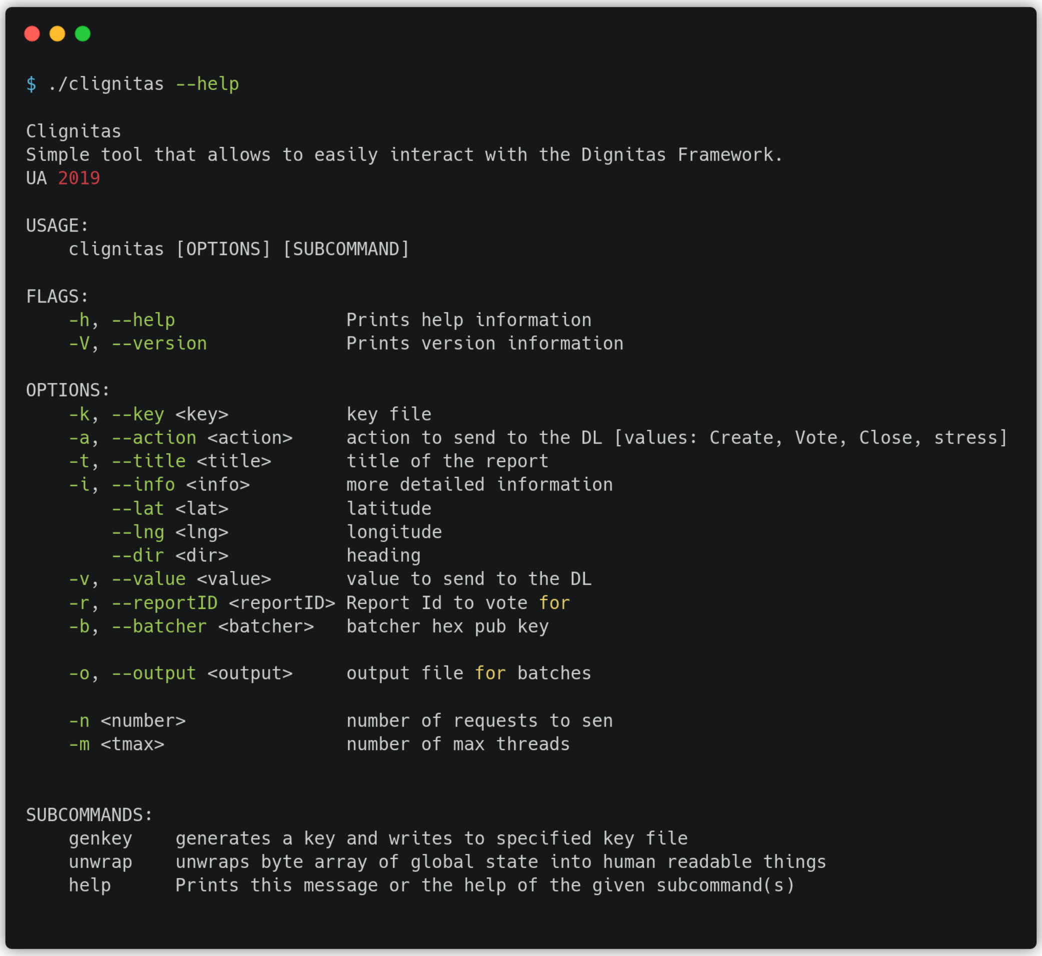
"...logical validity is not a guarantee of truth." — David Foster Wallace, Infinite Jest

Every solution looks perfect while it sits on paper. In order for us to confidently write that our solution works, as we proposed it would, we needed to properly test it. Testing big projects like this one is always a challenge, since there are many levels that you can test and account for; testing individual components, testing communication or even system-wide tests are all different possible degrees for testing our platform. In this chapter we explain how we approached this task. First we present some tools used while testing the system, tools that we built and existing one that we took advantage of. Next, we talk about the system-wide tests that were done and the extracted metrics. Finally, we also discuss some of the results, in order for us to understand if they make sense or if they make our solution unviable.

7.1 Testing tools

Early into the development of *Dignitas* we felt the need to build yet another component. This need arose because we found it inadequate to use the user applications, either the UA or the TA one, in tests. Although their sole purpose of existence is to easy the use of our system to users, while testing, they cannot provide the level of productivity that we desired. For instance, it would be impractical for us to use the UA Android Application everytime we wanted to test sending a Report into the ledger, or even to start the TA React application whenever we wanted to close out a Report while testing. Even more given the platform diversity of our components. We needed a simple tool that would help us interact with the system without the need for the user applications, and that is how the *clignitas* was born.

clignitas is a simple Command Line Interface (cli) that, basically, mimics the behaviour of the user applications, allowing us to quickly interact with *Dignitas* through the terminal. This small program was built in Rust and offer several capabilities, such as creating new



```
$ ./clignitas --help

Clignitas
Simple tool that allows to easily interact with the Dignitas Framework.
UA 2019

USAGE:
    clignitas [OPTIONS] [SUBCOMMAND]

FLAGS:
    -h, --help                 Prints help information
    -V, --version              Prints version information

OPTIONS:
    -k, --key <key>            key file
    -a, --action <action>      action to send to the DL [values: Create, Vote, Close, stress]
    -t, --title <title>        title of the report
    -i, --info <info>          more detailed information
        --lat <lat>            latitude
        --lng <lng>            longitude
        --dir <dir>            heading
    -v, --value <value>        value to send to the DL
    -r, --reportID <reportID>  Report Id to vote for
    -b, --batcher <batcher>    batcher hex pub key

    -o, --output <output>      output file for batches

    -n <number>                number of requests to sen
    -m <tmax>                  number of max threads


SUBCOMMANDS:
    genkey    generates a key and writes to specified key file
    unwrap    unwraps byte array of global state into human readable things
    help      Prints this message or the help of the given subcommand(s)
```

Figure 7.1: Clignitas help information

Reports, vote in existing ones and also closing them. It also has implemented several "quality of life" tools, for instance, it allows the easy creation of key pairs, it can decode byte arrays extracted from the ledger into human readable structures. One other important feature is the possibility of performing automated tests onto the *dignitas* system, more on that later. Overall is a swiss army-knife that helped us develop our PoC In Figure 7.1 we can see the help command output of this tool.

One other tool that helped us bringing *Dignitas* to life was Docker [14]. Docker is a virtualisation software that enabled us to build the different components in self-contained environments. This process allowed for easy deployments of, for instance, Sawtooth Nodes in a predictable and deterministic way, very useful when dealing with such complex systems.

There is one more tool which is important to mention, Grafana [24]. Grafana is an open source, general purpose dashboard, which allowed for the visualization of different metrics on our system. However, in order to use Grafana with our system we had to integrate *Dignitas* with InfluxDB [32]. InfluxDB is a time-series based database that acts as a sink to our system

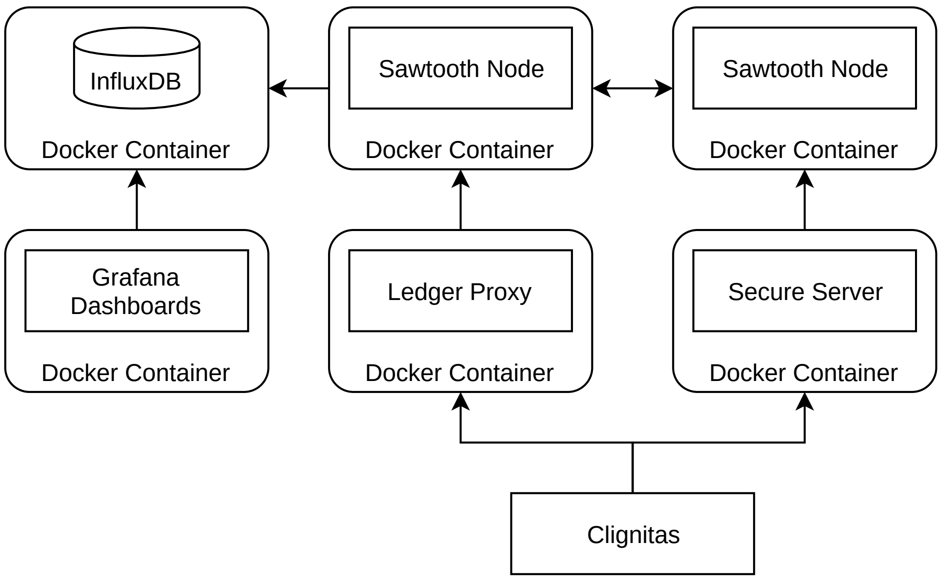


Figure 7.2: Test tools integration

metrics and logs. Grafana is then responsible to build dashboards and visualizations from the values it extracts from the tables in InfluxDB.

In Figure 7.2 we can see how all this work with each other in our tests.

7.2 TESTING

When it comes to testing the system we had a very pragmatic approach. First, all of our system components were tested individually to ensure they worked as expected. We also ensure that every part was correctly communicating with each other. We made sure that the use cases we set for ourselves in the beginning were achieved in the final version of *Dignitas*.

However, we tried to test our solution even better. For instance, we try some stress tests to see how our solution would cope. For these tests, we were mainly interested in how the distributed ledger would handle a high volume of requests. We choose the distributed ledger since its use to store reputation is the main novelty of our work. We considered traffic times between components, and cryptographic operations duration to be either negligible, or out of scope for this work. Next, we will explain how this test was performed, and then discuss some of the results that we had.

The test consisted in making a huge amount of requests (i.e creating new Reports or randomly vote on existing ones) to the proxy ledger in order to simulate high traffic volume. In order to do it we used our *clignitas*, which allowed us to perform stress tests with predefined parameters. It works by creating several threads in order to maximize the amount of parallel requests that are done to the proxy server, in our case we used four, which allowed us a throughput of around eighteen requests per minute (which should represent an intense throughput of Reports in our system). The test consisted of:

1. Sending ten transactions
2. Waiting thirty seconds
3. Sending one hundred transactions
4. Waiting thirty seconds
5. Sending one thousand transactions
6. Waiting thirty seconds
7. Sending ten thousand transactions

The whole stress test took about ten minutes to run. After running the test, we went to Grafana and extracted some results that we found were interesting to discuss in this work. All the function graphs present were built using Grafana, and since they are based in time-series of events the x axis always represents a timestamp. Since all of the graphs were taken from the same test, we can see that the x axis coincides in all of them.

In Figure 7.3 we find the first graph that we found interesting to analyze. In it, we can see the number of blocks published during the test, these are the blocks of batches that were added to the ledger. It is possible to see that were published almost five thousand blocks into the ledger. This, at first sight, could raise suspicions about the amount of disk space that a solution like this would require. However, it is important to note that more than ten thousand Reports were created or updated, this value is massive compared to the actual numbers that we could expect from a reporting system. We can see, in Figure 7.4, that the number of blocks published is equal to the number of blocks considered. This means that no block was discarded during the stress test. Block considered are, as the name says, blocks that the nodes consider adding into the ledger. These blocks can either be added and end up as published or they can be discarded when there is something wrong while validating its data or during the consensus process.

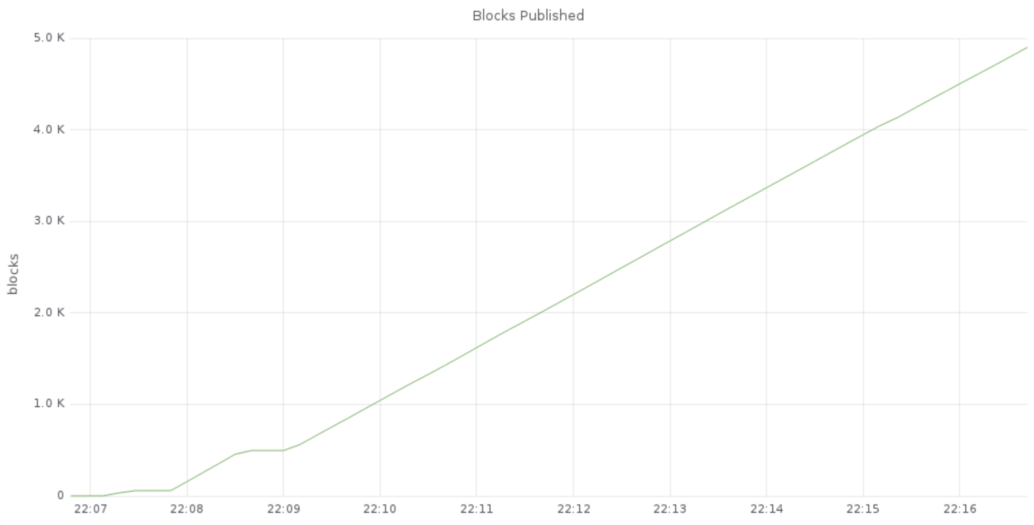


Figure 7.3: Number of blocks published during the stress test

However, the number of blocks might coincide with each other but it does not mean that all the information we sent was registered into the ledger. If we look into Figure 7.5, we can

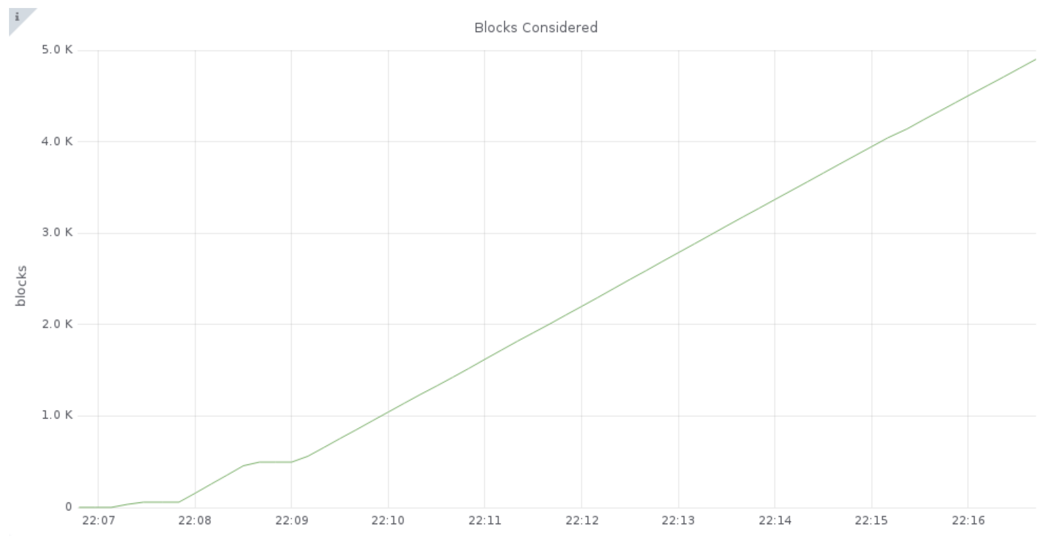


Figure 7.4: Number of blocks considered during the stress test

see that the number of transactions processed during the test is a little above five thousand. First, we see that there is some discrepancy between the number of transactions added and the number of transactions sent, which were over ten thousand. There are a couple of reasons for this to be happening, however it is not very significant. Since *clignitas* does not use any type of retry mechanism, and only works on a best effort basis (like the rest of the system), we can assume that these results are to be expected and can be easily improved upon.

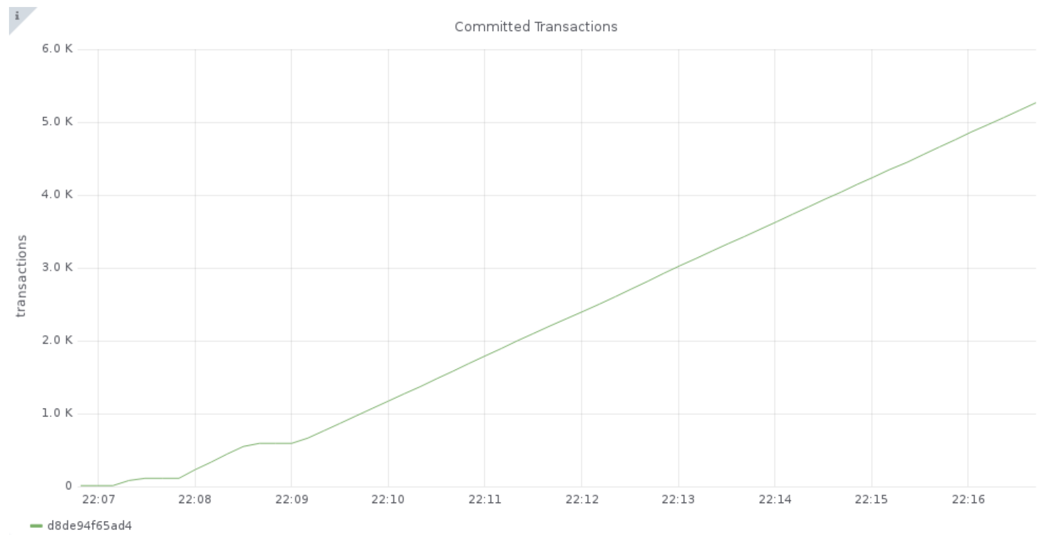


Figure 7.5: Number of transactions committed to the ledger during the stress test

Analysing in terms of time, the transactions that actually got processed we see in Figure 7.6 that in the ninety-ninth percentile the transactions all took under twenty milliseconds to be processed. This is the time it takes for the validator to be unpacked the transaction from its batch (making all the necessary validations), to send it to the TP, the TP to perform the action inside the transaction (it can be creating a new Report in global state, or voting); and return it to the validator. Under twenty milliseconds is an acceptable processing time, taking into consideration all the steps it took.

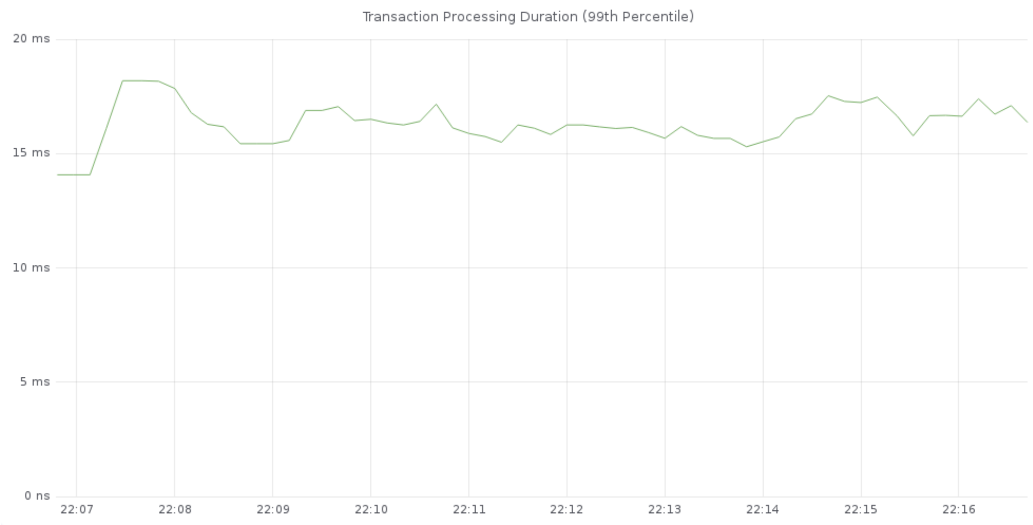


Figure 7.6: Processing time of the 99th percentile of transactions committed to the ledger during the stress test

After all of this, once again, we arrived at the same conclusion. *Dignitas* works as intended and it allows for a highly performant reporting system, which is also secure and anonymous. The use of a distributed ledger does not appear to be a bottleneck in the implementation of our solution.

CHAPTER 8

Conclusion

In this work we presented *Dignitas*, a blockchain-based, anonymous reputation framework. *Dignitas* and the supporting protocols are built around the novel idea of using Reputation as a Coin, which allows users to preserve anonymity while participating.

To demonstrate the effectiveness of the proposed solution, we used it as the backbone for a reporting system in a Smart City scenario. In this scenario, citizens are able to provide reports on real-life events. This information is then disseminated throughout the city, through already existing infrastructure. This allows citizens to be used as sensors in their own cities. Citizens can vote on existing Reports, using their own accumulated reputation to back someone else's claim. This process works as a bet, if they end up backing up a false Report they will lose the Reputation they have put at stake. Through this, citizens are able to discern between truthful and erroneous reports by analyzing the reputation a Report has backing it up. Our solution does not require a central authority. The system is administered by a set of already trusted authorities in the context of a city, such as firefighters and police forces, which have the final word when it comes to determine the veracity of a Report, and are able to reward good behaving citizens.

We built a prototype of the system that works as a proof of concept. Our prototype was architected in order to be the most scalable possible solution. On top of which, we run stress tests in order to determine the viability of the solution, using our own tools.

From this work we can take several conclusions. First, the viability of blockchains and distributed ledgers in general. Right now, these tools gather different and opposite opinions. Some think it will revolutionize the world, while others think it is a bogus tool riding on the "hype" of bitcoin. However, we found that the main problem with this tool is the incorrect use-cases on which it is used. As with every tool, blockchains , and DLTs in general, have a specific purpose and an ideal scenario where they should be employed. We found that scenarios where multiple authorities exist is ideal. The idea of distributed trust is the main power of a blockchain-based architecture. Other critics say that DLTs need many resources, however, we found that with some careful planning is possible to leverage all the benefits, without worrying about performance issues.

We proved the viability of this technology, and that a Smart City is one ideal scenario for it. Because, the reality is that there already exist multiple authorities, it is already a distributed scenario.

Using our system, Cities could improve their ways, turning citizens lives easier. Fostering a sense of community and responsibility, *Dignitas* enables Cities to really embark into the future with the right foot.

However, there is still room for improvement in several areas. One of which is the rewarding part of the system, we had a very naive approach to it and it could benefit from an in depth study, maybe using game theory in order to maximize people interactions and sense of accomplishment. Another meaningful work that could sprung from this base is, a more detailed evaluation of the impacts of different consensus protocols, leveraging the fact that using Sawtooth they are a "plug-in" component.

This work also opens up a new frontier for application in a Smart City scenario, one that may be worth pursuing for a better tomorrow.

Bibliography

[1] E. A. Akkoyunlu, K. Ekanadham and R. V. Hubert, “Some constraints and tradeoffs in the design of network communications”,

[2] V. Albino, U. Berardi and R. M. Dangelico, “Smart Cities: Definitions, Dimensions, Performance, and Initiatives”, *Journal of Urban Technology*, vol. 22, 1 January 2015.

[3] M. A. Azad, S. Bag and F. Hao, “PrivBox: Verifiable decentralized reputation system for online marketplaces”, *Future Generation Computer Systems*, vol. 89, December 2018.

[4] A. Back, “Hashcash - A Denial of Service Counter-Measure”,

[5] S. Bag, M. A. Azad and F. Hao, “A privacy-aware decentralized and personalized reputation system”, *Computers & Security*, vol. 77, August 2018.

[6] *Bitcoin - Open source P2P money.* [Online]. Available: `https://bitcoin.org/en/` (visited on 20/03/2019).

[7] J. Blömer, J. Juhnke and C. Kolb, “Anonymous and Publicly Linkable Reputation Systems”, in *Financial Cryptography and Data Security*, R. Böhme and T. Okamoto, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg.

[8] C. Bormann and P. Hoffman, *Concise Binary Object Representation (CBOR)*, en. [Online]. Available: `https://tools.ietf.org/html/rfc7049` (visited on 10/12/2019).

[9] V. Buterin, “A next generation smart contract & decentralized application platform”,

[10] *Corda | Home.* [Online]. Available: `https://www.corda.net/` (visited on 20/03/2019).

[11] R. Dennis and G. Owen, “Rep on the block: A next generation reputation system based on the blockchain”, in *2015 10th International Conference for Internet Technology and Secured Transactions (ICITST)*, December 2015.

[12] W. Diffie and M. E. Hellman, “Multiuser cryptographic techniques”, in *Proceedings of the June 7-10, 1976, national computer conference and exposition on - AFIPS ’76*, New York, New York: ACM Press, 1976.

[13] F. Dotzer, L. Fischer and P. Magiera, “VARS: A vehicle ad-hoc network reputation system”, in *Sixth IEEE International Symposium on a World of Wireless Mobile and Multimedia Networks*, Jun. 2005.

[14] *Enterprise Container Platform.* [Online]. Available: `https://www.docker.com/` (visited on 10/12/2019).

[15] *Ethereum Project.* [Online]. Available: `https://www.ethereum.org/` (visited on 20/03/2019).

[16] *Exonum — A framework for blockchain solutions.* [Online]. Available: `https://exonum.com/` (visited on 20/03/2019).

[17] *Expressjs/express*, original-date: 2009-06-26T18:56:01Z, December 2019. [Online]. Available: `https://github.com/expressjs/express` (visited on 05/12/2019).

[18] *Facebook/react*, original-date: 2013-05-24T16:15:54Z, December 2019. [Online]. Available: `https://github.com/facebook/react` (visited on 05/12/2019).

[19] S. D. Galbraith, *Mathematics of Public Key Cryptography.* October 2018.

[20] *Geohash*, December 2019. [Online]. Available: `https://en.wikipedia.org/w/index.php?title=Geohash&oldid=929284482` (visited on 10/12/2019).

[21] *Geohash encoding/decoding.* [Online]. Available: `https://www.movable-type.co.uk/scripts/geohash.html` (visited on 10/12/2019).

[22] F. Gómez Mármol and G. Martínez Pérez, "TRIP, a Trust and Reputation Infrastructure-based Proposal for Vehicular Ad Hoc Networks", *J. Netw. Comput. Appl.*, vol. 35, 3 May 2012.

[23] *Google/gson*, original-date: 2015-03-19T18:21:20Z, December 2019. [Online]. Available: `https://github.com/google/gson` (visited on 03/12/2019).

[24] *Grafana/grafana*, original-date: 2013-12-11T15:59:56Z, December 2019. [Online]. Available: `https://github.com/grafana/grafana` (visited on 10/12/2019).

[25] J. Gray, "Notes on Data Base Operating Systems", in *Operating Systems, An Advanced Course*, London, UK, UK: Springer-Verlag, 1978.

[26] C. Harrison, B. Eckman, R. Hamilton, P. Hartswick, J. Kalagnanam, J. Paraszczak and P. Williams, "Foundations for Smarter Cities", *IBM Journal of Research and Development*, Jul. 2010.

[27] F. Hendrikx, K. Bubendorfer and R. Chard, "Reputation systems: A survey and taxonomy", *Journal of Parallel and Distributed Computing*, vol. 75, January 2015.

[28] K. Hoffman, D. Zage and C. Nita-Rotaru, "A Survey of Attack and Defense Techniques for Reputation Systems", *ACM Comput. Surv.*, vol. 42, December 2009.

[29] *HydraChain/hydrachain: Permissioned Distributed Ledger based on Ethereum.* [Online]. Available: `https://github.com/HydraChain/hydrachain` (visited on 20/03/2019).

[30] *Hyperledger – Open Source Blockchain Technologies.* [Online]. Available: `https://www.hyperledger.org/` (visited on 02/04/2019).

[31] M. Iansiti and K. R. Lakhani, "The Truth About Blockchain",

[32] *Influxdata/influxdb*, original-date: 2013-09-26T14:31:10Z, December 2019. [Online]. Available: `https://github.com/influxdata/influxdb` (visited on 10/12/2019).

[33] R. John, J. P. Cherian and J. J. Kizhakkethottam, "A survey of techniques to prevent sybil attacks", in *2015 International Conference on Soft-Computing and Networks Security (ICSNS)*, February 2015.

[34] A. Jøsang, R. Ismail and C. Boyd, "A survey of trust and reputation systems for online service provision", *Decision Support Systems*, vol. 43, 2 March 2007.

[35] S. Küfeoglu and M. Özkuran, "Energy Consumption of Bitcoin Mining", Faculty of Economics, University of Cambridge, Working Paper, May 2019. [Online]. Available: `https://www.repository.cam.ac.uk/handle/1810/294129` (visited on 08/12/2019).

[36] N. Kumar and N. Chilamkurti, "Collaborative trust aware intelligent intrusion detection in VANETs", *Computers & Electrical Engineering*, vol. 40, 6 August 2014.

[37] L. Lamport, R. Shostak and M. Pease, "The Byzantine Generals Problem", *ACM Transactions on Programming Languages and Systems*, vol. 4, 3 Jul. 1982.

[38] X. Liu, N. Xiong, N. Zhang, A. Liu, H. Shen and C. Huang, "A Trust With Abstract Information Verified Routing Scheme for Cyber-Physical Network", *IEEE Access*, 2018.

[39] Y. Liu, M. Dong, K. Ota and A. Liu, "ActiveTrust: Secure and Trustable Routing in Wireless Sensor Networks", *IEEE Transactions on Information Forensics and Security*, vol. 11, 9 Sep. 2016.

[40] A. J. Menezes, P. C. Van Oorschot and S. A. Vanstone, *Handbook of applied cryptography*, ser. CRC Press series on discrete mathematics and its applications. CRC Press, 1997.

[41] *MultiChain | Open source blockchain platform.* [Online]. Available: `https://www.multichain.com/` (visited on 20/03/2019).

[42] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System",

[43] *Openchain - Blockchain technology for the enterprise.* [Online]. Available: `https://www.openchain.org/` (visited on 20/03/2019).

[44] *Reduxjs/redux*, original-date: 2015-05-29T23:53:15Z, December 2019. [Online]. Available: `https://github.com/reduxjs/redux` (visited on 05/12/2019).

[45] *Retrofit.* [Online]. Available: `https://square.github.io/retrofit/` (visited on 03/12/2019).

[46] *Rocket - Simple, Fast, Type-Safe Web Framework for Rust.* [Online]. Available: `https://rocket.rs/` (visited on 02/10/2019).

[47] *Rust-lang/rust*, original-date: 2010-06-16T20:39:03Z, December 2019. [Online]. Available: `https://github.com/rust-lang/rust` (visited on 10/12/2019).

[48] M. Sharples and J. Domingue, "The Blockchain and Kudos: A Distributed System for Educational Record, Reputation and Reward", in *Adaptive and Adaptable Learning*, K. Verbert, M. Sharples and T. Klobučar, Eds., Springer International Publishing, 2016.

[49] L. Silva, C. Senna and A. Zúquete, "Using Reputation as a Coin to Bet on Information Items Distributed in a Smart City", in *2019 Fifth Conference on Mobile and Secure Services (MobiSecServ)*, March 2019.

[50] *Social Credit System*, May 2019. [Online]. Available: `https://en.wikipedia.org/w/index.php?title=Social_Credit_System&oldid=898779809` (visited on 29/05/2019).

[51] *Socketio/socket.io*, original-date: 2010-03-11T18:24:48Z, December 2019. [Online]. Available: `https://github.com/socketio/socket.io` (visited on 04/12/2019).

[52] S. A. Soleymani, A. H. Abdullah, W. H. Hassan, M. H. Anisi, S. Goudarzi, M. A. Rezazadeh Baee and S. Mandala, "Trust management in vehicular ad hoc network: A systematic review", *EURASIP Journal on Wireless Communications and Networking*, May 2015.

[53] *Using the Rust SDK — Sawtooth v1.2.3 documentation.* [Online]. Available: `https://sawtooth.hyperledger.org/docs/core/releases/latest/app_developers_guide/rust_sdk.html` (visited on 10/12/2019).

[54] A. Vakali, L. Angelis and M. Giatsoglou, "Sensors talk and humans sense Towards a reciprocal collective awareness smart city framework", in *2013 IEEE International Conference on Communications Workshops (ICC)*, Jun. 2013.

[55] K. Wüst and A. Gervais, "Do you Need a Blockchain?", in *2018 Crypto Valley Conference on Blockchain Technology (CVCBT)*, Jun. 2018.