POLITECNICO DI MILANO

Facoltà di Ingegneria

Scuola di Ingegneria Industriale e dell'Informazione

Dipartimento di Elettronica, Informazione e Bioingegneria

Master of Science in

Computer Science and Engineering

Feasibility Study of a Blockchain-based Framework for decentralized Traffic Monitoring

Advisor:

PROF. STEFANO ZANERO

Co-advisor:

STEFANO LONGARI

Master Graduation Thesis by:

DAVIDE MAFFIOLA
Student Id n. 913722

Academic Year 2020-2021

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LISTINGS

ACRONYMS

FCD	Floating Car Data
DSRC	Dedicated Short Range Communication
C-ITS	Cooperative Intelligent Transport Systems
C-V2X	Cellular Vehicle To Everything
V2X	Vehicle To Everything
PBFT	Practical Byzantine Fault Tolerance
RSU	Roadside Unit
GPS	Global Positioning System
FCC	Federal Communication Commission
IVI	In-Vehicle Infotainment
(D)DoS	(Distributed) Denial of Service

ABSTRACT

This thesis summarizes the research that we carried out on the feasibility study of a novel decentralized and blockchain based framework that can be sued to collect real-time traffic information.

We present and analyze the state-of-the-art methods and currently used services for traffic information gathering, and we characterize the advantages and the disadvantages that they offer. The centralized nature of commonly used traffic information gathering services like Google Maps, Waze or those offered by local authorities, leads to the possibility for the service provider of arbitrarily restricting the access to the traffic information.

The proposed framework aims at offering a decentralized alternative to centralized traffic information gathering services. For this reason, we defined a reasonable threat model and based the definition of non-functional security requirements on it.

We illustrate how we designed the consensus mechanism that regulates the blockchain data structure by defining how distributed consensus is reached and the rules to enforce the validity of the information content of the blocks that form the blockchain. Through a mathematical approach we show the security properties of the adopted consensus algorithm.

We describe and implement a new simulator based on OMNeT++, Veins and SUMO to test the proposed framework and study its feasibility in a realistic scenario.

We show the experiments that we conducted on the proposed framework in order to study how its main parameters affect the performance and to determine if it can actually resist the attacks defined in the threat model.

Finally, we discuss the results of the experiments and comment why the obtained behaviour is consistent with the intentions that drove the design phase.

SOMMARIO

Questa tesi riassume il lavoro di ricerca effettuato sullo studio di fattibilità relativo ad un nuovo framework decentralizzato basato sulla blockchain che sia in grado di raccogliere informazioni sul traffico in tempo reale.

Vengono presentati ed analizzati i metodi più avanzati e i servizi attualmente utilizzati per il raccoglimento delle informazioni sul traffico e discutiamo i vantaggi e gli svantaggi che essi offrono. La natura centralizzata dei servizi più comunemente usati per il raccoglimento delle informazioni sul traffico, come Google Maps, Waze e i servizi offerti dalle autorità locali, porta alla possibilità da parte del fornitore del servizio di limitare l'accesso ai dati raccolti in maniera arbitraria.

Il framework proposto punta ad offrire un'alternativa decentralizzata ai servizi centralizzati per il raccoglimento delle informazioni sul traffico. Per questo motivo, è stato definito un threat model coerente con il contesto del framework su cui è stata basata la definizione dei requisiti non funzionali relativi alla sicurezza.

Viene illustrato come è stato progettato il meccanismo di consenso che regola la blockchain attraverso la definizione di come viene raggiunto il consenso distribuito e delle regole che impongono la validità del contenuto Informativo dei blocchi che compongono la blockchain. Le prorietà relative alla sicurezza dell'algoritmo di consenso vengono dimostrate con un approccio matematico.

Un nuovo simulatore basato su OMNeT++, Veins e SUMO viene descritto ed implementato al fine di testare il framework proposto e di studiare la sua fattibilità in uno scenario realistico.

Vengono presentati gli esperimenti che sono stati condotti sul framework per studiare come i suoi parametri principali influiscono sulla sua resa e per determinare se il framework è effettivamente in grado di resistere agli attacchi descritti nel threat model.

Infine vengono discussi i risultati ottenuti tramite gli esperimenti, e vengono anche commentate le ragioni per le quali il comportamento del framework proposto ottenuto negli esperimenti è coerente con le intenzioni che hanno guidato la fase di progettazione.

1 INTRODUCTION

Road traffic management is a fundamental service for mobility and transport and it can affect both individuals and companies especially in urban environments. Modern navigation systems are a prime example of this, as they also take into account traffic congestion for route planning and can select the best route dynamically. However, in order to provide high quality traffic management services it is necessary to have a reliable source of real-time road traffic information. Unfortunately, precise traffic information is difficult to measure and estimate and the available sources are often partial or not easily accessible.

The systems that are typically used to collect traffic information rely on onsite sensors like cameras or magnetic loops or on the floating car data approach, where onsite sensors are substituted by the Global Positioning System (GPS) modules on connected devices, which are usually smartphones [4]. Popular traffic management services such as Google Maps and Waze primarily use floating car data to gather the traffic information they need for their services.

Google Maps and Waze use a centralized approach where users send traffic information to a server, which then processes the information, adds it to its database and eventually redistribute it upon request. These systems have thus a centralized structure and so the entity that owns the servers has full control over the traffic information. Centralization also implies that there is a single point of failure and that the access to data is bound to the policy regulations of the service provider, who can restrict access at their discretion.

Vehicles are now equipped with fairly computationally powerful processors (e.g., Samsung's Exynos 8890 Auto on Audi A4 models [7]) and vehicle manufacturers are investing on vehicle-to-everything communication, either based on cellular network [29] or on short range communication technologies [31]. Because of this, vehicles can perform complex operations and participate in a network, so it is now possible to have a distributed system that runs on vehicles.

In this work, we propose a new framework that can be used to collect real-time traffic information in a decentralized fashion. Vehicles generate the traffic information through short range communication and the Global Positioning System (GPS) by stating that another vehicle is close to their position. The traffic information is then collected in a blockchain, which is entirely managed by the participant vehicles and possibly some support Roadside Units (RSUs). To be able to work in a trustless environment, where malicious entities may try to tamper with the traffic information, the blocks contained in the blockchain are validated and possibly accepted through a consensus mechanism. The algorithms and the concepts used in the consensus mechanism have been inspired by other widely used consensus mechanisms, which they have been tailored for the specific use in the context of the proposed framework.

To validate the proposed framework, we implemented a network simulator based on OMNet++ [23] and Veins [30] that relies on SUMO [28] for realistic traffic simulation. The results obtained through the simulations show that the framework can function in a stable way in realistic scenarios, while also being capable of maintaining a high quality of traffic information even in presence of various malicious entities.

The contributions of this work are so summarized:

- A new decentralized framework that is capable of collecting real-time traffic information in a large area and investigate how to mitigate possible attacks that aim to degrade the validity of traffic information.

- An efficient and flexible simulation tool that can be used to simulate the proposed framework and its possible variations. The simulations then allow to study the behaviour of framework though the analysis of the generated result files and the log file containing the full blockchain.

- An evaluation of the feasibility of the proposed framework, its performance and its stability in case of normal functioning and in case of specific attacks.

The rest of the thesis is structured as follows:

- *Chapter 2* explains the problem statement and presents a brief overview of the state of the art of the technologies used or addressed through this work. The overview includes the description of commonly used traffic information gathering methods, how traffic management services are provided and the description of the blockchain data structure together with distributed consensus protocols that are widely used in blockchain based systems.

- *Chapter 3* describes what has been done in this work while providing justification for the choices that were made. The proposed framework is broken down into simpler, loosely-coupled logical units in order to manage the complexity of the framework and to replicate how the research was actually carried out. This chapter contains the specification of the blockchain data structure and the threat model and the algorithms that we choose to adopt in the proposed framework, namely the consensus algorithm and the validation algorithm.

- In *Chapter* 4, we describe the architecture of the simulator that was used to validate the proposed framework. The simulator architecture is presented at both high level and component level, together with the description of the implementation of the most important components and the optimizations required to have an acceptable performance.

- *Chapter* 5 contains the description and the results of the experiments performed on the framework with the simulator. The validation tests include parameter studies in scenarios without attacks, unorganized attack scenarios and organized attack scenarios. The results of the experiments are then discussed in order to assess the feasibility of the proposed framework.

- Finally, *Chapter* 6 presents the limitations of this work and the challenges encountered while carrying out the research. Also, it gives some proposals and hints about how to cope with the limitations and it summarizes the goals, the approach and the achievements of this work.

2 BACKGROUND

2.1 THE NEED FOR REAL-TIME TRAFFIC INFORMATION

Traffic information are a valuable asset in the context of mobility and transport and both privates and companies can benefit from traffic management services the use traffic information. A very trivial example is the dynamic route planing performed by most of the modern navigation systems. Also, local authorities can take advantage of traffic information, for instance they can decide to temporarily modify the viability without creating too much traffic congestion. To have the best benefits, all these applications require that traffic information is always fresh and up-to-date and that users have access to it, but traffic management applications also demand that the traffic information source is reliable and has a good coverage of the territory (for example, knowing the traffic situation of very few streets in a city is almost useless in order to plan the fastest route). Because of all these reasons, it's important that systems that gather traffic information can operate in a real-time fashion and can cover large areas.

Collecting traffic information involves either the use of infrastructures with high installation and maintenance costs that count the number of vehicles in a certain location or the use of devices like smartphones to obtain the approximate positions of vehicles through the Global Positioning System (GPS) and the cellular network. The data are then sent to the traffic information server and then distributed to the users when requested. Two popular services that also collect traffic information are Google Maps and Waze. These services rely on a centralized model where only the service provider has access to all the information, while users only know the information they generated and the aggregate information retrieved from the server. This means that only the service provider has full control over the collected information and can arbitrarily apply access regulation policies, enforce access fees or shutdown the service altogether. Moreover, because these services are based on a centralized model they suffer from known centralization issues, the most relevant being that they have a single point of failure, that they can violate the user's privacy and that they can become the target of cybercrime.

The research carried out in this thesis aims to explore the feasibility of a fully decentralized framework for traffic information collection based on a blockchain data structure. The main objective of the framework is to provide reliable real-time traffic information with a high coverage using on-vehicle resources. In fact, vehicles are now equipped with fairly powerful processors [7] and can participate in a network [29, 31], so they can perform the basic set of activities required in a blockchain, i.e., broadcasting messages and performing fairly complex data validation operations. Yet, decentralization raises a lot of new issues as it has to operate in a trustless environment and presents a larger surface that can be attacked by malicious users. Those new issues are further examined and analyzed in the next chapter. To the best of our knowledge, there is no prior work on a fully decentralized framework like the one that is proposed.

2.2 STATE OF THE ART

In the following sections, the state of the art of the researches and technologies that are relevant for this work are briefly presented and discussed. The choices that has been made during the research have been considerably affected by the current state of the art, as it has given context to the proposed framework and it has provided a starting ground for the design of the proposed framework. The goals of this work have been defined also by keeping into account to the state of the art. This work is based on the idea proposed by Legler in his master's thesis [18], although the approach adopted in this work is radically different. Legler's thesis used a more qualitative approach to design and evaluate his framework, while this thesis aims at using a realistic context to first design and then evaluate the proposed framework and at assessing the performance of the proposed framework though quantitative analysis.

2.2.1 *Traffic Information Collection*

Collecting real-time traffic information is not an easy task due to the large number of vehicles and the vast area over which they span. The most commonly employed methods are either to estimate the traffic condition with onsite stationary sensors like traffic cameras or induction loops or to collect the information through Global Positioning System (GPS) equipped devices connected to the cellular network like smartphones [4]. The first method works by estimating useful traffic properties, such as the number and the speed of vehicles, by means of one or more sensors. Clearly, this method is limited by the operating range of the onsite stationary sensors. For example, an ultrasonic sensor would measure the number and the speed of vehicles passing close to it, while an induction loop would only be able to count the number of vehicles passing on it. Onsite sensors have been used for a long time, but they present some issues nevertheless, in fact they can be damaged, their measurements can be affected by meteorological conditions like poor visibility or heavy rains and their coverage is limited by their operating range. The problem of having a limited range is further worsened by the cost of the infrastructure since every onsite sensor has installation, maintenance and repair costs. This method is often used by local authorities for traffic control.

The second method used to collect traffic information is Floating Car Data (FCD) and instead of using expensive stationary onsite sensors, it exploits Global Positioning System (GPS) equipped devices present on the vehicle or owned by the driver. The way this method works is by periodically collecting the position and other relevant information like the current speed to a central server. To obtain their position, the devices primarily use the Global Positioning System (GPS) or in case it is not available it's possible to use triangulation, although its precision is lower. The strength of this method is that its costs are drastically reduced, as it does not require expensive onsite sensors and the devices that produce the traffic information are not owned by the entity that collects traffic information. Well-known services like Google Maps and Waze have adopted this method. The success of these traffic management services is due to their vast user base which continuously provide them with massive amounts of real-time data [3]. Because of how these services work, users only know their data and the aggregated information provided by the service provider.

The systems that implement these methods use a trusted central server that collects, processes and stores the gathered traffic information, meaning that they follow a centralized model. While centralization greatly simplifies data management, the decision processes and trust management, it also gives complete control over the data to the service provider, since the full database is stored only on the servers owned by the service provider. The provider can then arbitrarily enforce or change data access policies or restrict the access to certain groups of people or categories. Furthermore, it was shown that it is inexpensive for a single attacker to produce false traffic information on Google Maps and generate a fake traffic congestion [32]. The success of the attack is due to the fact that the devices that send the information to the server (in this case the smartphones) are not bound to vehicles and can be purchased or rented for a cost that is affordable even to individuals.

2.2.2 *Blockchain and Blockchain-based Systems*

Decentralized systems, as opposed to centralized ones, don't have a trusted intermediary that can guarantee the immutability of past data and decide who is a trusted entity and who is not. In fact decentralized systems operate in peer-to-peer (P2P) networks, where no node has complete control over the network and the decision process. To cope with these issues, Nakamoto [19] proposed the adoption of the blockchain data structure together with a suitably designed consensus mechanism.

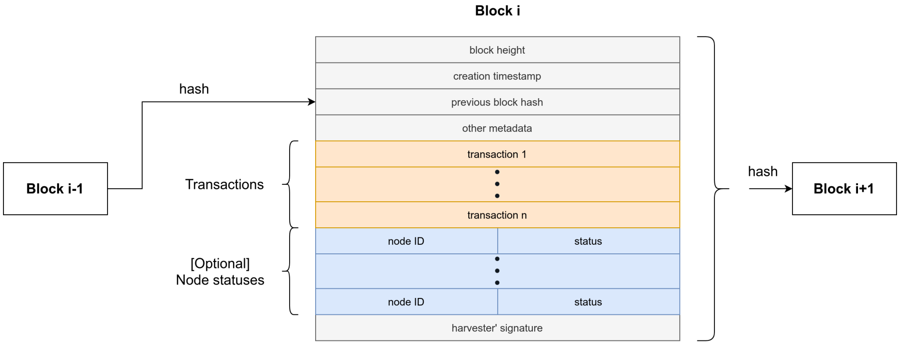


Figure 2.1: The internal structure of a generic blockchain.

The blockchain itself is not a technology, but rather a data structure usually employed in distributed systems to obtain an immutable and verifiable list of blocks. The basic idea behind the blockchain data structure is to collect the transactions generated by the participant entities into a unit of data called block and then chain the blocks together through their hash value so that it's possible to determine if a block hash been tampered with. To grant data authenticity, transactions, as well as blocks are digitally signed. Figure 2.1 shows graphically how the blockchain data structure is built.

The blockchain data structure is not actually bound to a centralized or decentralized network model, but it had significant success in decentralized models, as it is made clear by the vast number of proposed blockchain-based systems [12, 33, 13]. The core components of a blockchain-based system are:

- **Nodes**: are the participants of the network and the only entities that contributes to the growth of the blockchain data structure.

- **Transactions**: are the representation of the interactions between the nodes. For example, in blockchain-based cryptocurrencies transactions represent monetary transactions among participants.

- **Blocks**: are a collection of transactions collected over a certain period of time together with the metadata required for the functioning of the system. A block is linked with the previous block in the blockchain through the hash value of the previous block, creating a structure similar to a forward list.

- The **consensus mechanism**: is the component that builds trust over a trustless network and enforces the validity of the data. It's possible to further divide this component in the **consensus algorithm** and the **validation algorithm**. The former is used to achieve distributed consensus, while the latter is the algorithm that checks if the content of a block is consistent with the domain-specific rules.

- The **signature scheme**: grants the integrity and authenticity of transactions and blocks.

The key component that allows the blockchain data structure to successfully operate in decentralized networks is the consensus mechanism, which effectively defines the characteristics of the blockchain-based system. While the validation algorithm in the consensus mechanism is domain-specific, the consensus algorithm is not so it can be reused in other consensus mechanism with minimal modifications. The consensus algorithms commonly used in blockchain-based systems are described in the next section.

The advantages that blockchain-based systems provide include transparency, immutability, anonymity, decentralization, persistence and the possibility to remove intermediaries [21]. Although these features are desirable, blockchain-based systems have also significant drawbacks. In fact, blockchain-based systems are highly redundant, as every participant must have a copy of the entire blockchain, meaning that there is also a significant memory consumption. Moreover, blockchain-based systems require significant computational power to carry out the validation of the block and the consensus algorithm introduces a significant overhead in terms of exchanged messages and latency.

Blockchain-based systems can be classified according to the policies they enforce to join the P2P network and access the data contained in the blockchain [25]. In a public blockchain system there is no restriction on what node gets to join the network and on what operations nodes can perform. Consequently, nodes have full read and write permissions on the blockchain, meaning that they can read the content of the blockchain and produce new blocks. This kind of blockchain-based system allows for full decentralization, as all nodes are equal and can give equal contribution to the growth of the blockchain. On the other hand, private blockchain systems are owned by organizations or groups of individuals and require the users to have their permission to join the network. Also, nodes do not have the same permissions, meaning that the content of the blockchain is available only to selected entities and only selected nodes can create new blocks. Because of these reasons, this model leans more toward centralization. Finally, consortium blockchains delegates join invitations distribution and permission management to a certain set of nodes. In this way the system is more decentralized than private blockchain systems, as selected nodes can operate freely and independently, while maintaining more control than public blockchain systems over what node is trusted and what node gets read and write permissions. Table 2.1 summarizes the characteristics of the blockchain systems.

	PUBLIC BLOCKCHAINS	CONSORTIUM BLOCKCHAINS	PRIVATE BLOCKCHAINS
Consensus participation	Any participant	A set of trusted nodes	The service owner(s)
Consensus efficiency	Low	High	High
Join invitation	Not required	Depends	Required
Decentralization	Full	Partial	Centralized
Access to data	Open	Open or restricted	Typically restricted

Table 2.1: Comparison of features of the main types of blockchain systems.

2.2.3 *Consensus Mechanisms in Blockchains*

The consensus mechanism is a crucial component of blockchain-based systems and it characterizes the properties of the system. The tasks of the consensus mechanism are to ensure that all nodes, or at least the majority of them, agree on the blocks to insert in the blockchain through a consensus algorithm and to establish the rules needed to validate the content of the blocks through the validation algorithm. While the validation algorithm is domain-specific and cannot be easily generalized, the consensus algorithms can be categorized depending on how they operate. In fact, since the initial proposal of a blockchain-based system [19] that introduced the proof-of-work (PoW) consensus algorithm, there has been many proposals for alternative consensus algorithms that tried to overcome the limitations of PoW and to optimize the consensus process for different applications [2]. The number of variants of consensus algorithms is large, as most systems adopt their own variant. In the rest of this section, the consensus algorithms that are most used or that are relevant for this thesis are discussed.

The very first consensus algorithm to be proposed for a blockchain-based system is PoW, which is used in Bitcoin. Its main characteristic is that it uses a computationally hard problem that has a solution that can be easily verified. Specifically for Bitcoin, the problem is to find a nonce so that when the hash value of the block is computed, it has a certain number of zeroes, which is defined through a parameter, to prove that the node that harvested the block has spent time in finding a solution. When a solution is found, the block and the nonce are broadcasted and the rest of the nodes check that the nonce is correct and the content of the block is valid. If so, the block is added to the top of the blockchain and the harvester is rewarded with some bitcoins, otherwise it is discarded. In this way, an attacker that wants to insert an invalid block would require 51% of the total computational power of the participant nodes in order to succeed eventually. This algorithm is very stable, but it is so slow and computationally intensive that the cost of finding the correct nonce is higher than the reward [22].

To reduce the impact on the computational resources, the proof-of-stake (PoS) consensus algorithm was proposed [15]. In PoS the computational power is substituted with an abstract resource which is chosen depending on the actual variant of PoS, although it is often an amount of money or a score derived form the contribution of the node to the blockchain. For instance, Peercoin, the first system to adopt PoS, uses its own cryptocurrencies as a resource for the PoS algorithm. A participant that wants to harvest the next block puts a certain amount of its resources at stake that can be lost if the block that the node proposed gets rejected. The more the resources the node puts at stake are, the higher the probability that it gets the right to harvest a block is. Contrary to PoW, the PoS consensus algorithm can suffer from the nothing-at-stake problem when they are not designed properly, because of which the nodes have nothing to lose if they accept multiple block candidates [2].

Two interesting PoS variants are proof-of-importance (PoI) used by the NEM blockchain [20] and the proof-of-believability (PoB) used in IOST [27]. In PoI an importance score is calculated taking into account both the amount of resources that the node invests and its contribution in the network, and each node has a probability equal to the ratio of its importance over the total to be selected as the next harvester. This approach forces potential attackers to positively contribute to the growth of the blockchain before they are even able to try to attack the blockchain. On the other hand, if the attackers manage to gather enough importance, they can easily carry out an attack. PoB, instead, uses the believability score, which is non-tradable, is destroyed after each validation and is distributed automatically depending on the node contributions, to randomly elect a believable subnet. The nodes in the believable subnet then produce an optimistically validated block, while the rest of the network uses an optimized voting scheme to agree on the validity of the new block.

Voting-based consensus algorithms are the last category that was considered. They are variants of the Practical Byzantine Fault Tolerance (PBFT) algorithm [6], which is a three-phase commit protocol originally designed to handle Byzantine faults, but it is also suitable to counter malicious nodes in a distributed system. The three phases used in the algorithm are *pre-prepare*, *prepare* and *commit*, which make Practical Byzantine Fault Tolerance (PBFT) capable of handling up to $f = \lfloor \frac{n-1}{3} \rfloor$ malicious users, where n is the number of nodes that take part in the voting. Figure 2.2 shows the normal functioning of the Practical Byzantine Fault Tolerance (PBFT) consensus algorithm. Practical Byzantine Fault Tolerance (PBFT) and its variants offer more scalability and higher throughput than PoW and PoS, but requires

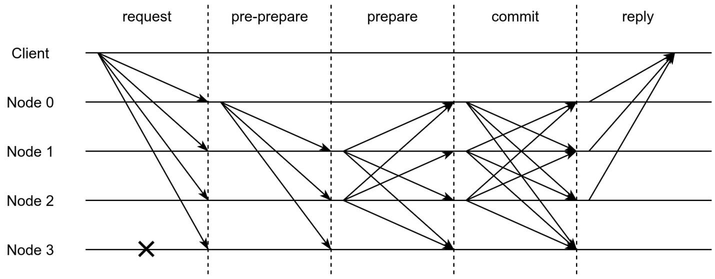


Figure 2.2: The normal functioning of the Practical Byzantine Fault Tolerance (PBFT) algorithm. The nodes are allowed to enter the commit phase and to generate the response when they have received $2f + 1$ messages from other nodes that are in the same phase.

a careful choice of the nodes that take part in the Practical Byzantine Fault Tolerance (PBFT) vote, making this category of algorithms more suitable for private and confederated blockchain systems. Hyperledger [13] is a popular system that is based on Practical Byzantine Fault Tolerance (PBFT), but it also allows selecting different variants of Practical Byzantine Fault Tolerance (PBFT) depending on the desired properties of the blockchain-based system.

Another important part of the consensus algorithm is the selection of the node that will be the next harvester. Block harvesting (or mining) is the process of creating and validating a candidate block that will be then verified by the other participant nodes and inserted on top of their blockchain if it is considered valid. It is important to introduce a set of rules so that nodes can agree whether to accept a candidate block from a certain node at a certain time or not. If no harvesting rules are enforces, any node can harvest a new block, but this situation is likely to cause the desynchronization of the replicas of the blockchain among the participants, thus causing a fork in the blockchain. Temporary and infrequent forks are tolerable, but they need to be solved by applying some policy, which is usually to keep only the longest blockchain. In the case in which there are no harvesting rules, forks are arbitrarily frequent and are created continuously, making the system utterly unusable.

In common blockchain systems two main methods for harvester selection are used, which are respectively based on a probabilistic approach, like the one used in the NEM blockchain [20] and in Bitcoin [19], and a deterministic approach, like the one proposed for the ByzCoin blockchain [16]. In the first approach, after a new block is added to the blockchain, each node computes a random value through a certain function and compares it with a threshold. In the case of Bitcoin the random value if the block and nonce hash value, while in the case of NEM, the function uses the node id, the importance score and the time as inputs. When the condition is met, the node gains the right to harvest the block. Since several nodes may simultaneously harvest a block, it is possible that the blockchain forks temporarily.

	DETERMINISTIC HARVESTING	PROBABILISTIC HARVESTING
Description	Harvester is *globally* selected based on a shared value	Harvester is *locally* selected based on a random variable
Harvest condition	$\#transaction \geqslant \mathsf{BLOCKSIZE}$ or $elapsed\_time \geqslant blocktime$	$H \leqslant f(\Delta t)$ where H is a random variable and $f(\Delta t)$ is a function depending on the elapsed time
Harvester uniqueness	Can be guaranteed, thus avoiding forks	Cannot be guaranteed, so forks can happen
Synchronization	Required	Not required
Guarantees	Block is always harvested	Block is eventually harvested

Table 2.2: Comparison between deterministic and probabilistic harvesting methods.

The deterministic approach instead uses a synchronized object to elect one or more leaders that harvest the block when it's full or after a certain amount of time. Because there is some form of synchronization to elect the leaders, forks can be prevented from happening. Table 2.2 shows a comparison of these two methods.

2.2.4 *Automotive Industry*

Vehicles are becoming more and more sophisticated and offer a vast range of services to their users. Two examples are the proximity sensors used when performing parking maneuvers and the integrated navigation systems. Another component present on modern cars is the In-Vehicle Infotainment (IVI) system which handles video and audio entertainment as well as in-car connectivity and the navigation system. In-Vehicle Infotainment (IVI) systems are powered by processors that are quite computationally powerful, like the Samsung's Exynos Auto 8890 processor that powers the In-Vehicle Infotainment (IVI) system of Audi A4 models [7]. These processors are comparable in terms of features and performances with mid-range smartphone processors, as it is shown in Table 2.3 where the Exynos Auto 8890 and Exynos 7884 processors are compared. The latter can be found in mid-range smartphones. The processors in In-Vehicle Infotainment (IVI) system have thus enough computational capabilities to allow a vehicle to execute the validation algorithm in a blockchain-based system.

	EXYNOS AUTO 8890	EXYNOS 7884
Cores	x4 Exynos M1 (Mongoose), x4 Cortex-A53	x2 Cortex-A73, x6 Cortex-A53
Frequency	2.3 GHz	1.56 GHz
Process	14 nm	14 nm
GPU	Mali-T880	Mali-G71 MP2
Memory	LPDDR4	LPDDR4

Table 2.3: Comparison of Exynos Auto 8890 and Exynos 7884.

Vehicle To Everything (V2X) is another technology that is increasing in popularity in the automotive industry. There are two main leading technologies: Cellular Vehicle To Everything (C-V2X) based on the 5G cellular network and short range communication technologies like Dedicated Short Range Communication (DSRC) in the USA or Cooperative Intelligent Transport Systems (C-ITS) in Europe. Although these technologies are still being developed and tested, many automotive companies are investing in them and exploring their usage [29, 31]. Short range communication, when paired with a Global Positioning System (GPS), enables vehicles to generate evidence of the closeness of another vehicle is close to their position. Moreover, both cellular networks and short range communication allows vehicles to join a network and thus broadcasting blocks and transactions, as well as consensus messages. Since vehicles are capable of carrying out the validation algorithm and consensus-related operation, it is possible for vehicles to actively take part in a blockchain-based system.

2.3 OBJECTIVES

The goal of this work is to propose and evaluate the feasibility of a new framework that exploits the computational capabilities and connectivity offered by vehicles to collect real-time traffic information into a blockchain. The proposed framework aims at begin able to provide a service at least comparable with popular and widespread services, but contrary to them it adopts decentralization and data availability policies. Another important issue that the proposed framework addresses and tries to mitigate is the possibility for an attacker to alter the quality of the traffic information and to create a fake traffic congestion. To assess the feasibility of this framework, we performed a study on the security guarantees it can offer.

As the proposed framework is supposed to be executed on the In-Vehicle Infotainment (IVI) systems in vehicles, the fact that the available resources are limited is taken into account for the design of the components of the framework. This implies that the validation algorithm should be as lightweight as possible and the number of times it is required to be executed should be minimized. Because of this, in many cases a faster heuristic is preferable over a computationally complex optimal algorithm even at the cost of generating a reasonable amount of false positives and false negatives.

Finally, this research can lay the foundations for further investigation toward decentralized blockchain-based vehicular systems, since to the best of our knowledge, there is no proposal for a decentralized traffic monitoring framework that can be used in a real environment. This fact has created difficulties during the research and hopefully this work will be used as a starting point for further research.

3 FRAMEWORK DESIGN

3.1 OVERVIEW

The framework proposed in this work is a complex system and in order to be able to better design and analyze it, the research was divided into loosely coupled logical units, which are described in better detail in the following sections. The logical units in which the framework was divided and their interactions and dependencies are shown in Figure 3.1. The order in which the logical units are presented in this chapter follows the dependencies among the logical units. This allows to have a clear understanding of the proposed framework, since the topics are presented in a more linear way.

The definition of blockchain data structure, its information content and how transactions are created directly or indirectly influences all the other logical units, so these concepts are presented first. This allows to give precise and concrete definitions that are then used in other components. The second logical unit, according to the dependencies, involves the definition of a semi-formal and reasonable threat model for the framework, so that the security requirements are clearly stated and can be referenced during the design phase of the algorithms used in the consensus mechanism, as well as during the planning of the experiments (see Chapter 5 for more details on the experiments). After defining a suitable threat model, we present how the consensus mechanism that powers the framework is designed. The consensus mechanism is further divided into the consensus algorithm and the validation algorithm. The reason behind this division is that the consensus algorithm has to be executed on a distributed system, thus requiring special care for network-related problems, while the validation algorithm is executed entirely on a single node and it has to deal with completely different issues, namely the local resource consumption and how to enforce a certain degree of consistency in the information content of blocks.

Finally, we discuss the remaining crucial parts of the proposed framework that have an impact on its functioning. More specifically, we deal with the problem of starting the blockchain, which is a common issue among all blockchain-based systems, and the reward system, which affects the evolution of the state of the proposed framework and can lead to the collapse of the framework when not configured properly. We also discuss the strategy used to decide whether to keep or drop a transaction, as its interactions with the consensus mechanism are quite complex.

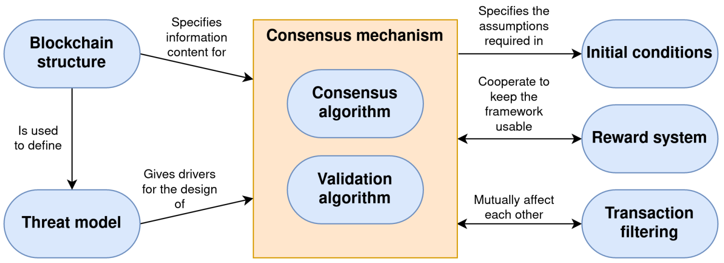

Blockchain structure
Specifies information content for
Consensus mechanism
Consensus algorithm
Validation algorithm
Is used to define
Threat model
Gives drivers for the design of
Specifies the assumptions required in
Initial conditions
Cooperate to keep the framework usable
Reward system
Mutually affect each other
Transaction filtering

Figure 3.1: The logical units in which the framework design process is divided and their interactions.

The base ideas that led to the definition of the proposed framework were described in a previous thesis [18], where a basic and simple decentralized blockchain-based framework similar to the one proposed in this work was proposed. With respect to the previous thesis, this work delves more deeply into the question of the consensus mechanism and uses a more analytical approach to design the framework.

3.2 THE BLOCKCHAIN DATA STRUCTURE

Regarding the blockchain data structure, the elements that are necessary to describe are the information content of the transactions and the blocks, how node identities and cryptographic keys are managed and how the main memory is used to store the blockchain. The most basic element in the blockchain is the transaction, which represents an interaction between two participants. In the context of this framework, two nodes interact with each other when they are close enough so that they can receive a probe message and send a reply through Dedicated Short Range Communication (DSRC) or Cooperative Intelligent Transport Systems (C-ITS), thus becoming aware that the other vehicle is close by. The probe message is generated and broadcast by each participant periodically. They will then generate and broadcast to all the other participants a transaction containing:

- The identity of the node that originated the transaction, which is referred as the **sender** throughout the rest of this work.
- The identity of the discovered node, which is referred as the **target** throughout the rest of this work.
- The **sender position** obtained though Global Positioning System (GPS).
- A **timestamp** indicating the moment in which the interaction has happened.
- The digital signature of the node that originated the transaction.

It's important to notice that the transactions don't contain an estimation or an upper bound for distance between the vehicles. Despite distance bounding protocols does exist and have been studied for a long time [5], the communication overhead they introduce does not justify the possible benefits. In fact, although a stricter bound can potentially improve the precision and the accuracy of the validation process, if the maximum detection range is small the performance loss is not significant. According to US Federal Communication Commission (FCC) classification, a low-power class A Dedicated Short Range Communication (DSRC) device has a communication range of 15 meters [14], meaning that stricter bounds on the detection range are not required.

The second element that composes the blockchain is the block, which has to contain all the pieces of information that define the current state of the blockchain system. In the context of the proposed framework, a block includes:

- The transactions generated during this blocktime, which is the interval of time between the insertion of the previous block and the creation of the current one. The number of transactions is limited to a maximum value called blocksize and because of this reason some of the oldest ones may be dropped.
- A flag for each transaction that indicates its validity.
- The identity and the reputation score of all the participants. The reputation score is a measure of how trustworthy the node has proven itself to be.
- A flag for each participant indicating if it was active during this blocktime. A node is considered active if at least one of its transactions is evaluated as valid.
- The hash value of the previous block.
- The attempt number (see Section 3.4.2 for more details).
- A timestamp indicating when the block was created.
- The block height, which is calculated as the number of blocks between the current one and the genesis block. This value acts as a sequence number and paired with the hash value of the block, uniquely identifies it.
- The digital signature of the node that harvested the block.

Figure 3.2 graphically summarizes the structure of the blocks and the transactions.

The amount of memory required to store the whole blockchain is large and grows constantly. This represents a critical issue for embedded systems, such as In-Vehicle Infotainment (IVI) systems, where the

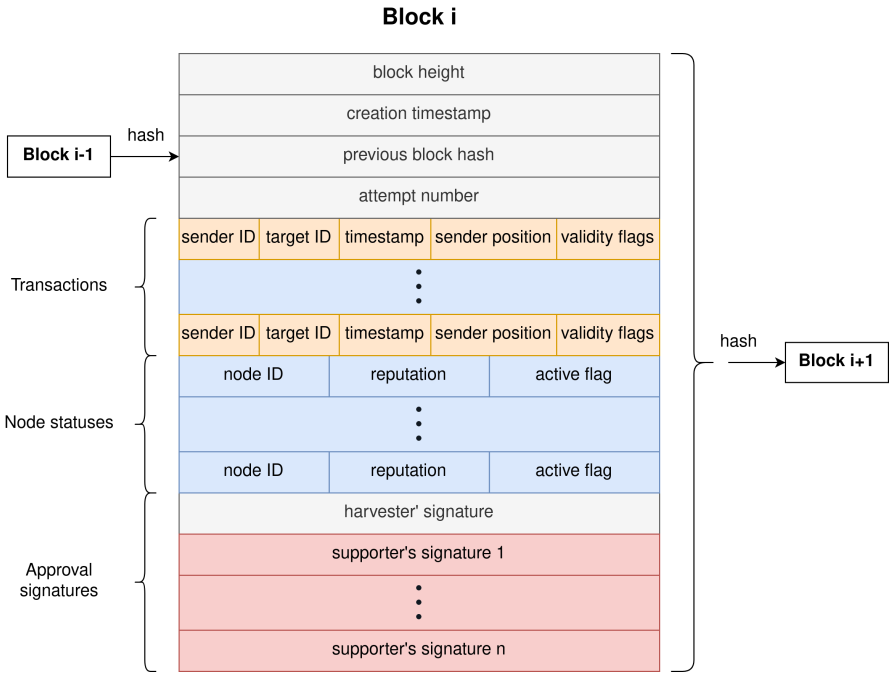


Figure 3.2: The structure of the blockchain used in the proposed framework and the information content of blocks.

amount of memory is strongly limited. A solution to this problem is to introduce the concept of state blocks [17, 27], which summarize the content of one or more blocks to significantly reduce the storage requirements, and they fundamentally act similar to checkpoints. State blocks only have to contain the state of the participant nodes and it is possible to use the same consensus algorithm as regular blocks to achieve consensus on state blocks. Using the state block optimization, a node has to keep in memory only the most recent blocks and the parallel blockchain defined by the state blocks. Also, new nodes that want to join the blockchain have to download and verify less data, granting a faster join procedure and only the nodes with more resources, like roadside units, have to store the full blockchain.

The problem of key distribution is a delicate question, because if cryptographic keys are not bounded to the ownership of a vehicle, attacker can generate an arbitrary number of keys and perform a similar attack to the one done against Google Maps [32]. On the other hand, binding keys to vehicles and delegating this task can make the proposed framework shift to a permissioned or even private blockchain model. An acceptable trade-off between openness and security is to give the task of binding asymmetric keys to identities to car dealers or vehicle manufacturers. In this way, it is possible to guarantee that each vehicle is associated with only one key. Another feasible solution is to let the departments of motor vehicles perform this task. In both cases the user privacy would be preserved through pseudonymity [24], as it is not possible to directly associate the vehicle positions to the user identity, but only with their public key. However, it is not among the objectives of this thesis to tackle or delve into this topic.

3.3 THREAT MODEL

The threat model is defined by the assets that are valuable to the framework and the stakeholders, the attack surface that can expose vulnerabilities and the threat agents that have interests in attacking the proposed framework. With this information, we present a list of meaningful attacks that the framework may undergo and state what is the impact of the attacks on the framework, as well as the risk that they represent for the assets.

The most important assets of the framework that we identified are the validity and the real-timeness of traffic information. They represent indeed the properties of the service that the framework is supposed to offer, so their importance is trivial. It is acceptable to have a reasonably small amount of invalid traffic information as long as the data provided by the framework have a sufficient quality to be usable. Another critical asset is the local resource consumption, since the framework is supposed to be executed in the In-Vehicle Infotainment (IVI) systems, which are actually embedded systems and, as such, they have limited resources in terms of memory and computational power as well as energy consumption. This means that if the proposed framework is too resource intensive or offers an easily exploitable amplification factor to attackers, the system can become unusable. Finally, the last valuable asset in the list is the reputation of the framework itself, since like other Floating Car Data (FCD) systems and blockchain based systems it requires a large user base to be effective. In fact, if the framework proves itself to be too unreliable or too weak against attacks, the user base is likely to move to another traffic management service.

The next step is to identify the attack surfaces that the proposed framework has. To be implementation-independent, the consensus mechanism is considered as a black-box that can perform two operations, which are to validate the content of a block and to achieve agreement on a block, while its white-box analysis is left for Sections 3.4 and 3.5. The attack surface was divided depending on what level in the blockchain the vulnerability is located:

- **Transaction level**: the nodes are responsible for the generation of transactions, so they can decide to forge them or to lie about some fields as they like. Because the transactions are signed, the identity of the sender is the only field that cannot be altered in any way unless private keys are not compromised. For the same reason, it is not possible for the nodes to tamper with someone else's transactions. The content of the transactions is not encrypted in order to guarantee the possibility to access to the traffic information to anyone, but in this way attackers have the chance to read the content of transactions before the block is harvested. Moreover, as the process of generating a transaction is delegated to nodes and therefore not controllable, nodes may decide to selectively generate transactions only toward specific nodes.

- **Block level**: blocks are harvested locally, meaning that the harvester node can attempt to insert wrong validity flags, passing invalid transactions off as valid ones or the other way around. Also, they may try to alter the statuses of the nodes or the block metadata.
- **Blockchain level**: this level mainly concerns the consensus algorithm used in the consensus mechanism. Attacker can in fact try to tamper with the blockchain by gaining the majority during the phase in which consensus is achieved or by deceive the harvester selection phase to candidate forged blocks.

Before considering the possible attacks that the system may undergo and completing the threat model, it is important to identify the feasible threat agents and the purposes of their attacks. The malicious entities that can have interest in attacking the proposed framework are single attackers, groups of attackers and competitors who offer a similar traffic management service and want to damage the framework reputation. It is reasonable to assume that individuals own exactly one vehicle, while corporations are able to gather few tens of vehicles. Attackers may want to lower the quality of the transactions in the blockchain by producing invalid transactions, try to spoof their real position, fake a traffic congestion or gain control over the blockchain by inserting a forged block into it. Table 3.1 shows the possible attacks and gives a risk level according to the threat model.

3.4 CONSENSUS

In this section, we present the analysis of the portion of the consensus mechanism relative exclusively to the consensus algorithm. The analysis of the other part of the consensus mechanism, i.e., the validation algorithm, is described in Section 3.5. The problem of selecting the harvester node or, more in general, leader election is strongly related to the consensus algorithm, so it is also described in this section. The two topics are analyzed independently so that their properties and behaviour could be studied more precisely.

ATTACK	DESCRIPTION	RISK	COMMENTS
Forged block insertion	Attackers insert a maliciously crafted block in the blockchain by gaining majority.	H	If the attack succeeds, the attackers gain full control of the framework. It's a type of Sybil attack [11].
Valid block rejection	Attackers prevent a valid block to be approved by making consensus fail.	M	It's a type of Sybil attack.
Fake traffic congestion generation	Attackers generate a fictitious traffic congestion.	M/H	If the attack succeeds in high-density areas often, the framework becomes unusable.
Fake position advertisement	Attackers try to spoof their own position.	M	
Teleportation	Attackers try to suddenly change their position so that they appear to be in an incompatible place.	M	
Copycatting	Attackers generate transactions by copying the content of someone else's transactions.	M	
Blockchain forking	Attackers force the blockchain to fork by tampering with the consensus mechanism.	M	Forking implies that the full validation algorithm is executed on more nodes.
Transaction spamming	Attackers produce an abnormally large amount of transaction.	L	This is basically a (D)DoS attack. The framework should not give an amplification factor to attackers.

Table 3.1: Description of the possible attacks and risk analysis for the threat model. Higher risk levels means that the problem is critical to the system, while lower risk means that it is tolerable to a certain extent.

3.4.1 *Consensus Algorithm*

For this work, we studied and analyzed several existing widespread consensus algorithms [19, 33, 16, 15, 27, 20, 13]. This operation led to the identification of three main categories of consensus algorithms used in blockchain systems:

- **Election based**: election based algorithms rely on a computationally difficult problem or on the amount of resources required to gain the right to harvest a new block. This category of algorithms is very popular in public blockchains as it scales well on networks with many nodes, even though its throughput is typically low. After being selected, the harvester executes the validation algorithm, cryptographically signs the block and broadcasts it, while the rest of the network only checks if the harvester is eligible to create the new block. In this way, only the harvester has to run the full validation algorithm to completion. The consensus mechanisms of Bitcoin (proof-of-work) and Peercoin (proof-of-stake) fall under this category. Sybil attacks can succeed on election based networks if the attackers own 51% of the resources used in the algorithm.

- **Voting based**: the consensus algorithms in this category are based on the Practical Byzantine Fault Tolerance (PBFT) consensus protocol [6] or one of its variants. A set of nodes, that can even be the whole network, takes part in the voting process, meaning that each node in the set has to run the validation algorithm at least once. Because of this, voting based algorithms don't scale well in large networks, but they have a high throughput and they outperform election based algorithms in small networks. Thanks to these features, voting based algorithm are frequently used in private and consortium blockchains, like Hyperledger. If the attackers own at least 34% of the resources, they can successfully perform a Sybil attack.

- **Hybrid**: this category of algorithms combine an election process with a voting scheme in order to improve the efficiency of voting by reducing the number of nodes that take part in the voting process. An example of a hybrid consensus mechanism is IOST, in which a part of the network is elected to create an optimistically validated block, while the rest of the network uses a voting scheme to reach consensus. The resistance of an hybrid algorithm against Sybil attacks depends mostly on how the underlying methods are hybridized.

Each category of consensus algorithms has some specific properties, advantages and drawbacks which may not be compatible with the requirements of the proposed framework. Ideally, the consensus algorithm should have a simple leader selection procedure and should require only few complete executions of the validation algorithm to achieve consensus, so that the framework can reach consensus with little effort per node on average. Election based algorithms only require one complete execution of the algorithm, making them the most resource efficient when the all the participants are considered, but they present important issues which are incompatible with the design goals of the proposed framework. The first problem is that adopting an approach similar to PoW would be unacceptable in an embedded system as it would simply require too much energy. PoS algorithms instead are more lightweight, but can lead to centralization when few nodes gain too many resources. More importantly, they have provable guarantees only in the case the stake does not change too often [10], which is almost impossible to achieve in a vehicular network where nodes join and leave frequently.

The voting based algorithms in which the full network participate in the voting scheme are utterly unsuitable due to the fact that every node must run the full version of the validation algorithm, and they also have a higher latency and huge network overhead because of the three phases commit. This method is also likely to consume the whole bandwidth of the channels used by the In-Vehicle Infotainment (IVI) systems as each block needs to be broadcast from every node to all the other ones at least once. A more centralized solution would be to restrict the voting process only to trusted entities like roadside units, but in this way the framework would fail to satisfy the decentralization requirement.

The hybrid algorithm that was used to perform the comparison is based on the ideas proposed by proof-of-importance (PoI) used in the NEM blockchain and proof-of-believability (PoB) used in IOST. A reputation score that is derived from their behaviour is assigned to each node and it is used to elect an harvester and a subnet of supporters. The harvester then creates the new block, while the supporters use Practical Byzantine Fault Tolerance (PBFT) to agree on the validity of the candidate block and each supporter broadcasts an approval signature to state that the block is valid according to them. Finally, the rest of the nodes checks the eligibility of the harvester and if there are enough eligible approval signatures before inserting the new block in their local blockchains. Including both the harvester and the supporters, $n = 3f + 2$ nodes have to be elected in total, where f is the fault tolerance parameter of the voting scheme as well as the number of required eligible approval signatures for each block. Because of how the consensus algorithm works, the validation algorithm is only executed by the harvester and the supporters, which are only a small portion of the whole network, while still guaranteeing full decentralization, provided that the election process is fair. All these features make this hybrid algorithm suitable for the proposed framework. Table 3.2 sum-

	VOTING	ELECTION	HYBRID
Number of exchanged messages	$O(n^2)$	$O(n)$	$O(f^2+n)$, $f << n$
Number of validations per block	n	1	$3f+2$
Malicious block insertion success rate	0 if malicious resources < 34% else 1	0 if malicious resources < 51% else 1 (eventually)	fails with high probability if if malicious resources < 34%
Valid block rejection rate	0 if malicious resources < 34% else 1	0 if malicious resources < 51% else 1 (eventually)	almost linear w.r.t. malicious resources in interval $[0; 15\%]$

Table 3.2: Comparison of the main consensus algorithm categories. The hybrid algorithm that was considered is described in Section 3.4.1. Election based algorithms only guarantee that the attacks eventually fail, while voting based ones can stop them immediately.

marizes the characteristics of all the considered types of consensus algorithms.

3.4.2 *Leader election*

Instead of inventing a brand-new harvester selection method that has to be evaluated and is likely to bring new issues which have not been explored yet, it was preferred to use one of the two widespread methods described in Section 2.2.3. The main driver for the choice was resource efficiency rather than complexity. Because of this a deterministic harvester selection method was preferred over a probabilistic one, as it can effectively prevent forking from happening provided that nodes are synchronized correctly.

The adoption of a deterministic harvesting method required a way to synchronize the participants and make them agree on a view number. This issue was solved by relying on the previous block (on which there is already agreement) and the assumption that the clock drift among the clocks on the In-Vehicle Infotainment (IVI) system of the participants is much smaller than the time between the harvesting of two subsequent block. This assumption is indeed reasonable because vehicles could keep their clocks synchronized through the Global Positioning System (GPS), which is anyway required to produce transactions. The view number v is thus defined by combining the hash value of the previous block with the attempt number, which is derived from the time elapsed since the creation of the last block though the formula $a = \lfloor\frac{now-last\text{-}block\text{-}timestamp}{blocktime}\rfloor$. The leaders are then elected using the view number, their position in the list of elected nodes and their reputation score. The image set of the hash function is partitioned into non-overlapping sets and the partitions are assigned to each node. The rules for creating partitions are that the probability of obtaining a hash value in that range is equal to the ratio of the reputation of the node the partition is assigned to over the total amount of reputation or more formally:

$$\begin{aligned} Pr(hash(v\|i) \in Partition(n)) =& \frac{|Partition(n)|}{2^{hash\text{-}size}} = \\ =& \frac{reputation_n}{\sum\limits_{j \in Nodes} reputation_j} \end{aligned} \tag{3.1}$$

Then the hash function is used as a uniform pseudo-random number generator by feeding it the view number and the position of the entity in the election list. Algorithm 1 formalizes the election procedure through the definition of its pseudocode.

The element of randomness is introduced in the leader election process by the hash value of the last block. In fact, if a strong hash function is used, it is impossible for an attacker to know the hash value of a block until its content is broadcast. Even if the previous block is harvested by an attacker, it is computationally impossible to obtain the desired hash value [26], as it would require to find a valid preimage of a hash value. The hash function that was chosen for the leader election process is SHA-256 as it is recognized as a strong enough hash function and for the rest of the work, when referring to the hash function, the SHA-256 function is implied.

It is worth to note that, according to the pseudocode, a node is allowed to be elected more than once during the same election. This does not represent a problem as long as the probability of that node being elected enough times to gain majority by itself is low enough, that is, if the number of times a node gets elected is not greater than f over a total of $3f + 1$ trials. This probability can be calculated by modeling the number of times a node is elected during the election process with a binomial distribution, where the number of trials is $3f + 1$ and the probability of a success is computed through Equation 3.1. Assuming that $f = 10$, which is a realistic value, and the percentage p of the total reputation owned by the node is 10%, the chance of the node gaining control by itself are approximately 0.13%, that is tolerable. If the value of p is reduced to a more realistic value like 2%, the probability drops to the order of one part-per-billion. Equation 3.2 shows how these results are derived.

Algorithm 1 Leader election algorithm.

```
function ELECT(active-nodes, reputations, previous-hash, attempt, n)
    seed = previous-hash || attempt
    intervals = empty list
    total = 0
    for all node $\in$ active-nodes do
        total += reputation[node]
        intervals.add(total)
    end for
    elected = empty list
    for all $0 \leqslant i < n$ do
        hash-value = hash(seed || i)
        index = find-lower-bound(intervals, hash-value)
        elected.add(active-nodes[index])
    end for
    return elected
end function
```

$$
\begin{aligned}
& X_i = \text{“Number of times node } i \text{ is elected as supporter.”} \\
& X_i \sim \mathrm{Binomial}(3f+1, p_i) \\
& \Pr(X_i \geqslant f+1) = 1 - \Pr(X_i \leqslant f) = \\
& \qquad\qquad = 1 - \sum_{k=0}^{f} \binom{3f+1}{k} p_i^k (1-p_i)^{3f+1-k}
\end{aligned}
\tag{3.2}
$$

3.4.3 *Characterization of the Consensus Algorithm*

It is important to consider also the resistance to attacks of the chosen hybrid consensus algorithm before adopting it in the framework. The assumption for the analysis that is carried out in this section is that the application of the hash function behaves like a perfect uniform pseudo-random generator, making the election process is completely fair, that is, the probability that each node is elected is exactly the one shown in Equation 3.1. The number of elected nodes is $n = 3f + 2$ where f is the fault tolerance parameter of Practical Byzantine Fault Tolerance (PBFT). The properties that we analyzed are the probability of a successful malicious block insertion and the probability of the failure of the consensus algorithm in the case of a Sybil attack.

To perform the analysis, two random variables, H and M were introduced. The former indicates whether an attacker has been elected as block harvester and it is a Bernoulli random variable with success probability p, where p is the ratio of the total reputation owned by the attackers. The variable M instead represents the number of nodes controlled by attackers that are elected as supporters and will then participate in the voting process. M is therefore modeled as a binomial random variable with $3f+1$ trials, each having a success probability p, that is the same as in H. A malicious block insertion is successful if the harvester node is malicious and the attackers own at least 34% of the supporters nodes, while a valid block rejection happens whenever the harvester is a malicious node or the attackers own at least 34% of the supporters nodes. Equations 3.3 and 3.4 show how the formulas for the two target probabilities are derived.

$$\begin{aligned}
&H \sim \mathit{Bernoulli}(p)\\
&M \sim \mathit{Binomial}(3f+1, p)\\
&\Pr(\text{“A malicious block is inserted”}) =\\
&\quad = \Pr(H=1 \land M \geqslant f+1) =\\
&\quad = \Pr(H=1)(1-\Pr(M \leqslant f)) =\\
&\quad = p\left(1 - \sum_{k=0}^{f} \binom{3f+1}{k} p^k (1-p)^{3f+1-k}\right)
\end{aligned} \tag{3.3}$$

$$\begin{aligned}
&\Pr(\text{“A valid block is maliciously rejected”}) =\\
&\quad = \Pr(H=1 \lor M \geqslant f+1) =\\
&\quad = \Pr(H=1) + 1 - \Pr(M \leqslant f) - \Pr(H=1 \land M \leqslant f) =\\
&\quad = p + (1-p)(1-\Pr(M \leqslant f)) =\\
&\quad = 1 - (1-p)\sum_{k=0}^{f} \binom{3f+1}{k} p^k (1-p)^{3f+1-k}
\end{aligned} \tag{3.4}$$

Figure 3.3 and 3.4 show the plots of the Equations 3.3 and 3.4 for different configurations and under different points of view. The first graphic in Figure 3.3 shows the probability of a successful malicious block insertion as the amount of reputation owned by attackers grows. In the graphic, it is possible to see how a larger value of the fault tolerance parameter makes the chance of a successful attack smaller in the interval [0; 33%]. The second graphic, instead, displays the probability of a successful valid block rejection attack as the amount of reputation owned by attackers grows. The consensus algorithm exhibits the undesirable property of having a relatively high chance of rejecting a valid block. This is tolerable though, as the probability becomes almost identical to the ratio of malicious nodes' reputation when approaching 0% (which is the condition in which the algorithm is supposed to be executed according to the threat model). Finally, the graphics in Figure 3.4 shows the same functions as in Figure 3.3, but in relation with the fault tolerance parameter. The graphics highlight the same properties as the first two graphics, but under a different perspective. In particular, it's possible to see that the effect of increasing the fault tolerance parameter has diminishing returns in any configuration.

Another interesting observation is that when the attackers own more than the 34% of the total reputation, the consensus algorithm performs worse with a higher number of supporters, but this case is not compatible with the threat model.

3.5 VALIDATION ALGORITHM

The validation algorithm is the component of the consensus mechanism that enforces the consistency of the transactions contained in the blocks and it directly affects the quality of the collected traffic information. A validation algorithm that performs an analysis of the traffic stream as a whole would be ideal, since it would be able to gain more knowledge about the traffic state and consequently detect outliers and anomalies more easily, thus preventing or mitigating attacks more effectively. An algorithm of this kind would be way too computationally intensive to be run on the In-Vehicle Infotainment (IVI) systems, though. It is for this reason that a set of heuristics was used in place of an optimal algorithm, implying that the validation algorithm generates a certain amount of false positives and false negatives. This does not necessarily make the proposed framework unusable as long as the quantities of wrongly classified transaction remains acceptable. An important concept about the validation algorithm to remark is that its objective is not to detect malicious behaviour, but rather to prevent it from taking happening or to mitigate its effects by accepting and rejecting transaction and through meaningful reputation updates.

The validation algorithm is composed by four heuristics:

- **Illegal transactions filtering**: this component blacklists all transactions that are considered illegal and therefore should not be considered by the other heuristics.

- **Transaction validation**: this heuristic evaluates the believability of the transactions and categorize them as *acceptable*, *unacceptable* or *implausible*, while blacklisted transactions are always considered as implausible. It also counts the number of rewards and penalties to give to each node.

- **Misbehaviour penalization**: this step further analyzes the transaction to search for misbehaviour such as selective transaction generation.

- **Reputation update**: this last component uses the results obtained in the previous steps to compute the variation of reputation to apply to each node.

In the following section, we explain and discuss the choices made for each heuristic that composes the validation algorithm, together with the pseudocode of each heuristic.

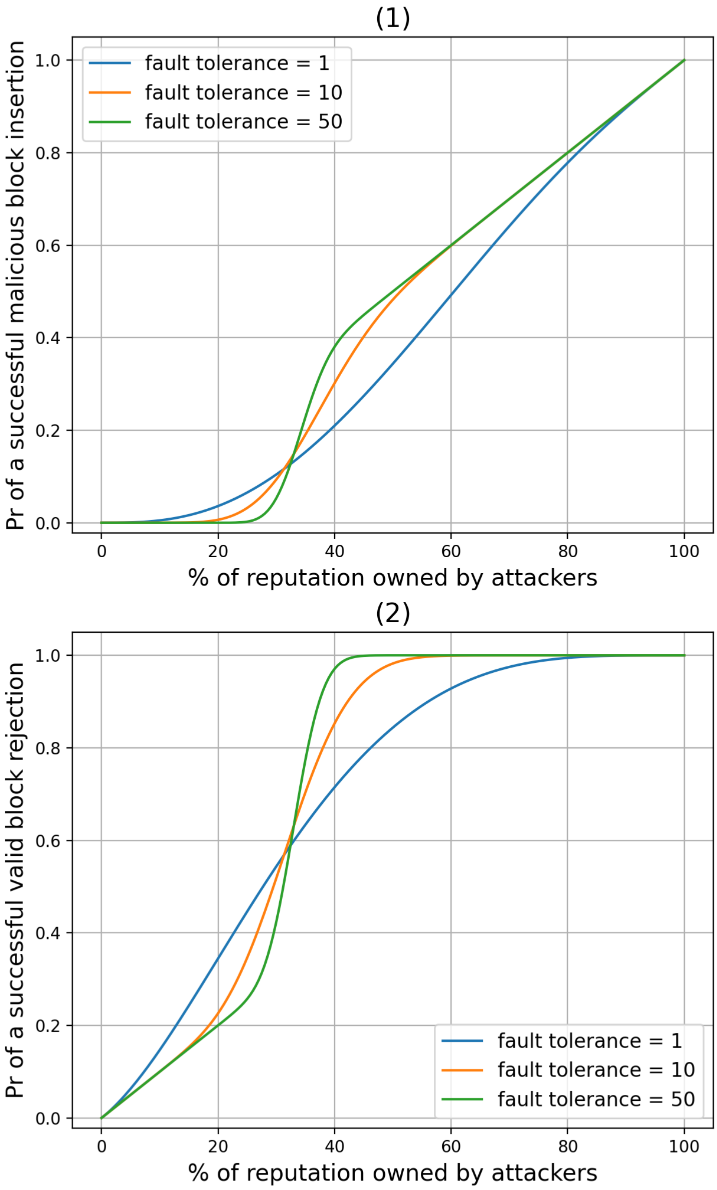


Figure 3.3: Resilience to Sybil attacks under different fault tolerance parameter values. (1) The probability of a successful malicious block insertion attack in relation with the amount of reputation owned by attackers. (2) The probability of a successful valid block rejection attack in relation with the amount of reputation owned by attackers.

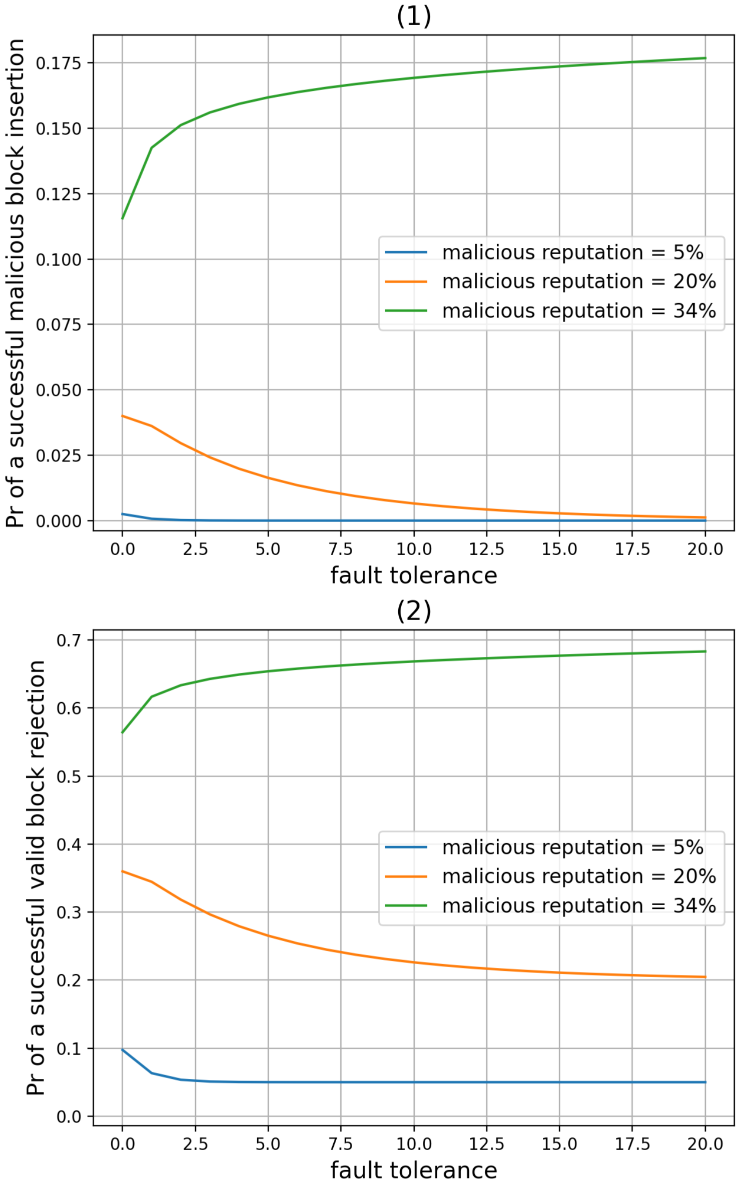


Figure 3.4: Resilience to Sybil attacks depending on the amount of reputation owned by attackers. (1) The probability of a successful malicious block insertion attack in relation with the fault tolerance parameter. (2) The probability of a successful valid block rejection attack in relation with the fault tolerance parameter.

3.5.1 *Illegal Transactions Filtering*

The first operation that is performed on the transactions is filtering. During this step the transactions that are deemed illegal are blacklisted and the senders gain one penalty for each blacklisted transaction they have generated. A first version of the filtering heuristic only considered the case in which a node generates a transaction that targets itself. The advantage of this approach is that no valid transaction is affected and that it successfully prevents nodes from self-boosting their own reputation score. Unfortunately, the other heuristics are unable to deal with attackers that perform a copycat attack (see Table 3.1), so, in order to mitigate this type of attack, we decided to consider as illegal any transaction that is not confirmed by the target. A transaction is considered confirmed if in the same block there exists at least another transaction that has the same sender and target of the transaction under examination, but with the roles inverted. This mean that transaction A → B is considered illegal unless the same block also contains transaction B → A. The final heuristic has a complexity of $O(\mathtt{blocksize}^2)$ and its pseudocode is shown in Heuristic 2.

Heuristic 2 Invalid Transaction Filtering

```
function FILTER(transactions)
    blacklist = empty list
    for all t ∈ transactions do
        if t.sender = t.target then
            blacklist.add(t)
        else
            confirmed = false
            for all t2 ∈ transactions, t2 ≠ t do
                if t.sender = t2.target ∧ t.target = t2.sender then
                    confirmed = true
                end if
            end for
            if ¬confirmed then
                blacklist.add(t)
            end if
        end if
    end for
    return blacklist
end function
```

3.5.2 *Transaction Validation*

The purpose of the transaction validation heuristic is to identify which transaction can be considered reliable and which cannot. The basic idea, which was originally proposed in Legler's thesis [18], is to compare new transactions with the ones in past blocks and to check if some upper bounds related to the travel distance are satisfied. The results obtained by checking the validity of these bounds are then used to compute a reliability score and if the score is higher than a threshold, the transaction can be accepted.

To derive suitable bounds, we considered the geometry of the problem, the physical limit of how far can a vehicle travel in a certain amount of time and the implications of each type of condition. Each transaction is modeled as circle in a plane with the center placed at the sender position and the range equal to the maximum allowed detection range. The real position of the detected vehicle can be anywhere within the circle. While it is impossible to know the real average speed of the vehicles, it is reasonable to assume that there is a maximum speed that is very rarely exceeded. This means that the maximum distance a vehicle can travel has to be lower than $bound = v_{max} \cdot \delta t$, where v_{max} is the maximum feasible speed of a vehicle and δt is the interval of time between two measurements. Figure 3.5 shows two transactions and the distances that can be estimated that are relevant for transaction validation. It is possible to identify four different cases in which a bound can be applied, which are $id(A) = id(C)$, $id(A) = id(D)$, $id(B) = id(C)$ and $id(B) = id(D)$, where $id(X)$ is the identity of node X. The only distance that can be directly measured is AC, which is equal to the euclidean distance between the two positions of the senders. The other three cases, instead, require some computations in order to derive a usable bound. Since the derivation is similar for all the cases, only the one relative to the case in which $id(A) = id(D)$ is shown in detail:

$$\begin{aligned}
&AD \leqslant v_{max} \cdot \Delta t \\
&DC \leqslant range_{max} \\
&\overrightarrow{AC} = \overrightarrow{AD} + \overrightarrow{DC}
\end{aligned}$$

By applying the triangle inequality and applying the bounds:

$$AC \leqslant AD + DC \leqslant v_{max} \cdot \Delta t + range_{max}$$

The conditions of the four cases can thus be defined in terms of measurable quantities in the following way:

$$id(A) = id(C) \implies AC \leqslant v_{max} \cdot \Delta t \tag{3.5}$$

$$id(A) = id(D) \implies AC \leqslant v_{max} \cdot \Delta t + range_{max} \tag{3.6}$$

$$id(B) = id(C) \implies AC \leqslant v_{max} \cdot \Delta t + range_{max} \tag{3.7}$$

$$id(B) = id(D) \implies AC \leqslant v_{max} \cdot \Delta t + 2 \cdot range_{max} \tag{3.8}$$

Ideally, all the four conditions should be enforced on any pair of transactions, but in a realistic scenario, this is not possible. The reason

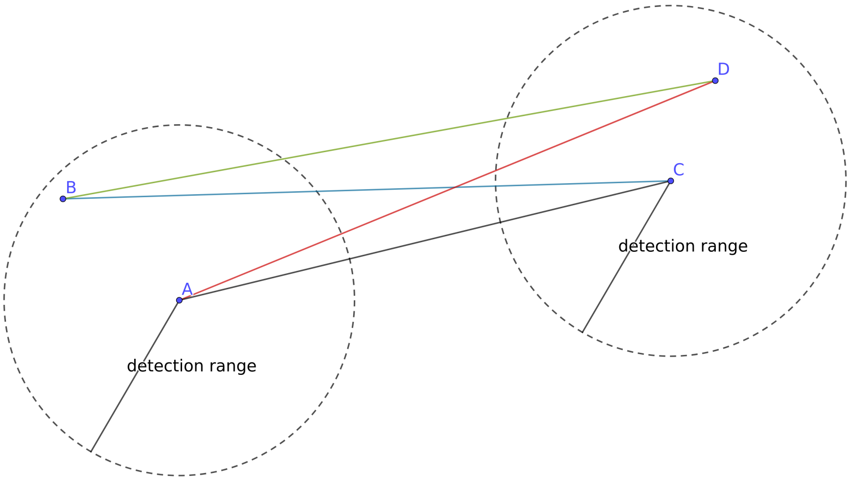


Figure 3.5: Two transactions and the meaningful distances between nodes.

is that a certain number of invalid transactions can be erroneously evaluated as valid, since the validation algorithm contains heuristics. This fact has no relevance on Condition 3.5, because the two transactions have the same sender, so no other node can affect the evaluation of the condition. This is not true for the other three conditions. In fact, if a transaction with a fake position sent by an attacker is wrongly classified as valid, all the legit nodes that detect the attacker at the real position will fail to satisfy Condition 3.7. In this way the attacker gains an amplification factor for which with only one transaction, it is able to affect the evaluation of several other transactions. For this very reason, Condition 3.7 is excluded from the heuristic. The remaining conditions are also affected by wrongly classified transactions, but they cannot be easily exploited to gain an amplification factor, so instead of being strictly enforced, Conditions 3.6 and 3.8 are evaluated though a score. To calculate the value of the score, the lists of concordant and discordant nodes (I_C and I_D respectively) are computed first. The lists are populated by comparing the new transaction with the transaction in a certain number of previous blocks. For each previous transaction that satisfies the one of the two preconditions, the sender is added to I_C when the respective postcondition is also true, otherwise it in added to I_D. The score is then calculated as follows:

$$\text{score} = \frac{\sum\limits_{i \in I_C} \text{reputation}_i - \sum\limits_{j \in I_D} \text{reputation}_j}{|I_C| + |I_D|}$$

If the score is above a certain threshold, the transaction is considered as acceptable and the sender is rewarded, otherwise it is marked as implausible and the sender is penalized. In this way nodes that behaved properly for a long time have a higher weight in the evaluation of the score because it is assumed that they are more likely to advertise correct information. To prevent the framework to collapse in case of low traffic density, the evaluation of a transaction is suspended if $|I_C| + |I_D| = 0$ and no reward or penalty is given to the sender. In this case the transaction is marked as unacceptable rather than as implausible. The complete reward scheme is thus the following:

$$\neg\mathtt{coherent} \implies \text{implausible, penalize sender}$$

$$|I_C| + |I_D| = 0 \implies \text{unacceptable, do not penalize sender}$$

$$\mathtt{score} = \frac{\sum_{i \in I_C} \mathtt{reputation}_i - \sum_{j \in I_D} \mathtt{reputation}_j}{|I_C| + |I_D|}$$

$$\mathtt{score} \geqslant \mathtt{threshold} \implies \text{acceptable, reward sender}$$

$$\text{else} \implies \text{implausible, penalize sender}$$

An early version of the heuristic only used Condition 3.8 to reduce the number of checks and so to improve the performance, but in this way, the sender position is never validated. This allowed attackers to exploit legit nodes to teleport around by simply generating a transaction targeting a legit node in which the attacker advertises a position compatible with the one of the target node. The lack of checks on the position of the sender caused this kind of transactions to pass unnoticed and so to be accepted. The final version of the heuristic, instead, performs all checks but Condition 3.7 and has a complexity of $O(N \cdot \mathtt{blocksize}^2)$, where N is the number of previous blocks. The pseudocode of the final version is presented in Heuristic 3.

3.5.3 *Misbehaviour Penalization*

The main objective of this heuristic is to penalize and mitigate the suspect behaviours that is not dealt with in the transaction validation step, which mainly involves selective transaction generation and self-sustaining malicious groups. The reason why these cases are not handled during transaction validation is because these misbehaviours rely on the *absence* of transactions, while the transaction validation heuristic can only deal with the transactions that are in the block. Since no node has knowledge of the traffic stream, it is difficult to reliably understand what node is misbehaving. The problem is further worsened by the presence of false positives and false negatives and the fact that some transactions can be dropped or lost. Because of the difficulty of this problem, we had to consider and discard several methods before identifying a usable mitigation.

One of the two approaches used to tackle this problem is to identify suspect behaviour at group level. We considered three different:

1. Nodes are treated as vertices and transactions as edges in an undirected graph and the connected subgraphs are identified. After this operation, the relevant properties of the subgraphs

Heuristic 3 Transaction validation algorithm

```
function VALIDATE-BLOCK-DATA(new-trans, old-blocks, reputations)
    count = 0
    score = 0
    coherent = true
    for all b $\in$ old-blocks do
        for all old-trans $\in$ b.transactions do
            distance = euclidean-distance(new-trans, old-trans)
            $\Delta$time = |new-trans.timestamp - old-trans.timestamp|
            bound = MAX-SPEED * $\Delta$time
            check-bound = false
            if new-trans.sender = old-trans.sender then
                coherent = coherent $\wedge$ distance $\leqslant$ bound
            else if new-trans.sender = old-trans.target then
                bound += old-trans.range
                check-bound = true
            else if new-trans.target = old-trans.target then
                bound += old-trans.range + new-trans.range
                check-bound = true
            end if
            if check-bound then
                count += 1
                if distance < bound then
                    score += reputations[old-trans.sender]
                else
                    score -= reputations[old-trans.sender]
                end if
            end if
        end for
    end for
    if coherent $\wedge$ count = 0 then
        return Unacceptable
    end if
    if coherent $\wedge$ score/count $\geqslant$ THRESHOLD then
        return Acceptable
    end if
    return Implausible
end function
```

like the number of vertices, the total reputation and the average reputation are computed. Then all the nodes that belong to a subgraph that does not satisfy a certain predicate are blacklisted and their transactions become implausible.

2. Similarly to the previous method, the connected subgraphs are computed first. This time, the conflicts between subgraphs are used to determine if the nodes are misbehaving. A subgraph has a conflict whenever there is a missing transaction between a node contained in it and another one contained in a different subgraph. Missing transactions are detected every time two nodes are close enough in space and time to generate a transaction, but the block does not contain it. Like in the previous method, the nodes contained in subgraphs with too many conflicts are penalized.

3. This method attempts to approximate the shape of the connected subgraphs by finding the optimal parameters that make a parametric model fit best. Subgraphs are then penalized when there is an overlap with the shape of another subgraph, and they have less total reputation.

Each method has some issues that made them unsuitable to be used in the framework. Method 1 is not able to give consistent results because neither legit subgraphs nor malicious subgraphs have characterizing properties, but rather the range of possible values for the measured properties is very similar for both types of subgraphs, allowing malicious groups to easily blend in. This means that even if a predicate that is so strict that it becomes unfeasible to perform an attack is used, the number of wrongly classified legit subgraphs would grow to the point that the system collapses to a state were almost all nodes have zero reputation. Method 3 instead gives inconsistent results whenever the shape of the subgraph did not belong to the family of shapes that the parametric model can describe. A simple model with few parameters incapable of describing the different shapes a group can take, especially in presence of crossroads and turnings. More complex models could solved this issue, but the number of parameters and the overhead required to find the optimal values of the parameters would become too high. Finally, Method 2 is too fragile against malicious insiders. A group of attackers can in fact decide to selectively generate transactions in such a way that the subgraph they belong to have a lot of conflicts even if the majority of nodes behaves properly. Another vulnerability that is common to all these methods that operate at group level is depicted in Figure 3.6. The attack consists in having one malicious node to properly that correctly generates transactions toward at least one legit node, while the rest of the attackers only generate transactions among themselves. In this way, the effectiveness of group detection is compromised, while only one malicious vehicle

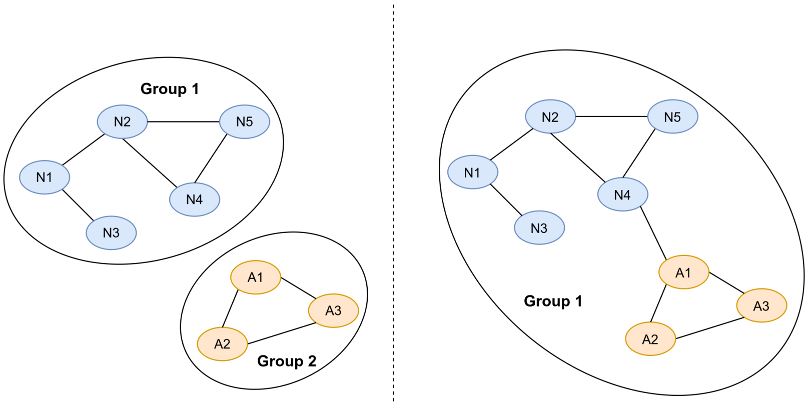


Figure 3.6: On the left, the ideal situation in which legit nodes do not interact with the attackers (A1, A2 and A3). On the right, the simple workaround to decrease the chance to be detected with a single interaction between legit nodes and attackers.

is required to be close to legit nodes, while the others do not even need to be on the road to carry out the attack.

The second type of approaches that we considered operates at node level instead, meaning that the behaviour of each node is evaluated individually, instead of considering the whole group. This approach can lead to situations where only a portion of the attackers are penalized, so the attack can still proceed, at least partially, but this does not represent a problem as long as the remaining attackers are in an isolated area. A first definition of the heuristic used the number of missing transactions to decide whether to blacklist a node, and consequently all its transactions, or not. This method proved itself to be ineffective, as attackers could produce enough acceptable transactions among themselves to compensate for the penalty and lose reputation slower than legit nodes. When legit nodes approach a reputation score of zero, their transactions are rejected more often and attackers eventually take over and start gaining reputation again.

To prevent malicious nodes to take over, we modified the initial heuristic to introduce two modifications. The first one is that when a missing transaction is detected, only the node with less reputation loses the conflict and gains one additional penalty. The second one, instead, is that only lost conflicts count toward the blacklisting phase. Heuristic 4 shows the pseudocode of the final version of the misbehaviour penalization heuristic. Since finding missing transactions requires to compare every transaction, to all the other ones, the computational complexity of the heuristic is $O(\mathtt{blocksize}^2)$.

The assumption used in the design of the misbehaviour penalization heuristic is that misbehaving nodes, on average, have to struggle more than legit nodes to gain reputation, so they are more likely to lose the conflict. This is further enforced by Heuristic 5 that computes a scaling factor for the reputation variation so that nodes that have detected the same targets during the last M blocks are rewarded less. The lack of variance in the detected nodes is a characteristic of malicious groups that selectively generate transactions, as they are much fewer than legit nodes. The complexity of Heuristic 5 depends on what data structure is used to store the adjacency relation. If using a sorted array or a tree structure, the complexity becomes $O(M \cdot \mathtt{blocksize} \cdot \mathtt{log}(\mathtt{blocksize}))$ since the union operation would have a complexity of $O(\mathtt{blocksize} \cdot \mathtt{log}(\mathtt{blocksize}))$. On the other hand, if an hash table is used, the complexity becomes $O(M \cdot \mathtt{blocksize})$ in the average case. In both cases the complexity of this heuristic is lower than the one of the other heuristics.

Heuristic 4 Conflict-driven misbehaviour penaization.

```
function DETECT-MISBEHAVIOUR(reputations, transactions, outcomes)
    conflicts = conflicting-pairs(transactions, outcomes)
    lost-conflicts = empty dictionary
    blacklist = empty list
    for all $n_1$, $n_2$ ∈ conflicts do
        $reputation_1$ = reputations[$n_1$]
        $reputation_2$ = reputations[$n_2$]
        loser = ($reputation_1$ < $reputation_2$) ? $n_1$ : $n_2$
        lost-conflicts[loser] += 1
    end for
    for all node, conflicts ∈ lost-conflicts do
        if conflicts $\geqslant$ MAX-CONFLICTS then
            blacklist.add(node)
        end if
    end for
    return lost-conflicts, blacklist
end function
```

3.5.4 *Reputation Update*

In the reputation update phase, the variation of reputation of each node is computed and it is added to the current reputation score that is retrieved from the previous block. Also, the flags that indicate what nodes have at least one acceptable transaction in the block are set. Thanks to the reputation update phase, the behaviour of a node and its contribution to the blockchain are bound to the reputation score. A node that behaves properly and thus generates more acceptable transactions will have higher reputation and so it is going to be considered more trustworthy. On the other hand, a node that misbehaves,

Heuristic 5 Scale factors computation based on detection variance.

```
function ADJACENCY-RELATION(transactions, outcomes)
    relation = empty set
    for all $t \in$ transactions do
        if outcomes[t] = Acceptable then
            relation.add((t.sender, t.target))
        end if
    end for
    return relation
end function

function SCALE-FACTORS(active-nodes, transactions, outcomes)
    old-blocks = M most recent blocks
    old-relation = empty set
    for all $b \in$ old-blocks do
        relation = adjacency-relation(b.trasactions, b.outcomes)
        old-relation.union(relation)
    end for
    new-relation = adjacency-relation(transactions, outcomes)
    intersection = old-relation $\cap$ new-relation
    scale-factors = empty dictionary
    for all $n \in$ active-nodes do
        count = count pairs like (n, _) in intercesction
        total = count pairs like (n, _) in new-relation
        scale-factors[n] = 1 - count/total * WHEIGHT
    end for
    return scale-factors
end function
```

generates more implausible transactions, so it is going to have a low reputation score. Initially, this part of the validation algorithm was very simple and just computed the activity flags and the reputation variation. The latter was calculated through the following formula:

$$variation_i = reward \cdot num\text{-}rewards_i - penalty \cdot num\text{-}penalties_i \tag{3.9}$$

In the formula, $num\text{-}rewards_i$ and $num\text{-}penalties_i$ are respectively the number of rewards and the number of penalties for node i that were computed in the previous steps, while $reward$ and $penalty$ are parameters representing the base amount of reward and penalty. After applying the reputation score variation, the final reputation is corrected so that it is always in the interval $[0; max\text{-}reputation]$, where $max\text{-}reputation$ is another parameter representing the highest allowed reputation score. The reputation is thus updated in the following way:

$$temp_i = reputation_i + variation_i$$

$$reputation'_i = \begin{cases} 0 & \text{if } temp_i < 0 \\ max\text{-}reputation & \text{if } temp_i > max\text{-}reputation \\ temp_i & \text{otherwise} \end{cases}$$

While this update strategy is simple and has a complexity of $O(\#nodes)$, it has two major drawbacks. The first is that inactive nodes, since they don't generate any transaction, will always have a reputation variation equal to zero. This is undesirable because it allows attackers to gain a large amount of reputation by behaving properly, stop generating transaction for possibly a long period of time and join again when there is a more favorable situation to carry out the attack. Since they have with a high reputation, the attack has a higher chance to succeed. The second issue is that this method is based on the assumption that all transactions should be equally rewarded and penalized. This means that a node that detects the very same target over and over again, will receive the same reward as a node that has a great diversity in the detected nodes. This situation encourages attackers to prefer to selectively generate transactions among themselves, as they can operate in a more controlled way and thus gain reputation faster, while having no repercussions.

The second design of the reputation update step solves both of these problems by introducing the concepts of reputation decay and detection variance. The latter has already been discussed in the previous section and led to the introduction of Heuristic 5 to compute a scale factor for all the active nodes. Reputation decay, on the other hand, implies that a node that is inactive will not have a null reputation variation, but instead, a portion of its reputation score is destroyed, depending on the parameter $\mathtt{reputation\text{-}decay\text{-}rate}$. The updated reputation variation computation is the following:

$$\begin{aligned}
\mathtt{real\text{-}variation}_i =& \mathtt{scale\text{-}factor}_i \cdot \mathtt{reward} \cdot \mathtt{num\text{-}rewards}_i + \\
& -\mathtt{penalty} \cdot \mathtt{num\text{-}penalties}_i \\
\mathtt{reputation\text{-}decay}_i =& -\mathtt{reputation\text{-}decay\text{-}rate} \cdot \mathtt{reputation}_i \\
\mathtt{variation}_i =& \begin{cases} \mathtt{real\text{-}variation}_i & \text{if } i \text{ is active} \\ \mathtt{reputation\text{-}decay}_i & \text{otherwise} \end{cases}
\end{aligned}$$

Since the detection variance has to be computed for each node during the reputation update phase in order to compute the reputation variation, the overall complexity of this step becomes $O(M \cdot \mathtt{blocksize} \cdot \log(\mathtt{blocksize}))$.

3.6 OTHER COMPONENTS

This section deals with the setup and the configuration of the framework accordingly to what was done up to this point. In particular, the topics that are discussed are the initial state of the blockchain and the reward system, since they have a major impact on the functioning of the proposed framework.

3.6.1 *Initial State*

The initial state of the blockchain plays a crucial role in blockchain based systems [19, 15]. The very first block in the blockchain is called genesis block and it is created ad-hoc for each blockchain system depending on how the consensus mechanism works. The purpose of the genesis block is to put the blockchain in a valid state even at the very beginning, so that the consensus mechanism can operate under the correct assumptions when dealing with the first blocks. As an example, the genesis block in Bitcoin contains made-up transactions that create the initial public offer of bitcoins.

In the specific context of the proposed framework, to allow the framework to function properly, the initial state requires a set of elements that have a known position and are trusted. RSUs are a good candidate for this starting phase as they fulfill both these requirements, or anyway it's possible to ensure that RSUs behave correctly at least temporarily. This is the only phase in which a certain amount of trust is required, as once the framework is started, these elements will be treated just like the other regular nodes and will not have any particular advantage or additional permissions. The genesis block contains made-up transactions between the initially trusted nodes and each node will start with a certain reputation. The transactions in the genesis block have a higher range than the regular ones on average because the trusted entities are not likely to be close to each other as it would happen during the normal functioning of the framework. This is not a problem because the transactions in the genesis block are not going to be validated. Also, the initial reputation of trusted nodes is not particularly critical, but it should be high enough to grant that the system does not collapse during the startup phase.

3.6.2 *Reward System*

The reward system is a crucial part of the proposed framework because when it is not configured correctly it can easily lead to the collapse of the framework or to a situation in which misbehaviour is not discouraged. Throughout the research, we considered three reward systems, which differ in how the base reward and penalty are assigned to each node. The three reward systems are:

1. **Constant reward and penalty**: nodes have fixed amounts of base reward and penalty which are defined through some parameters and don't change during the life of the framework.

2. **Constant reward, distrust on mistake**: it is similar to the first reward system, but the reputation is reset to zero if there is even just one penalty.

3. **Constant reward, dynamic penalty**: the reward is treated as in the previous reward systems, but the penalty is computed independently for each node by considering a base value and the trend of the node's reputation in the last blocks. For example, an exponential penalty can be assigned through the formula $penalty_i = base\_penalty \cdot 2^{\#negative\_variations_i}$ where $\#negative\_variations_i$ is the number of times the reputation of node i has decreased in the last blocks and $base\_penalty$ is a parameter.

The last two reward systems would ideally be more effective in preventing malicious behaviour, but the amount of penalty that is assigned is not fully controllable, making the proposed framework likely to collapse to a state where the total reputation is zero or close to zero. The problem is further worsened by the fact that the validation algorithm can produce false positives and false negatives. For this reason we decided to adopt the reward system with constant base reward and penalty, which allows for full control over the penalty, making the framework more controllable and less likely to collapse.

3.6.3 *Transaction Filtering*

Transaction filtering is an important question, especially when considering the fact that the number of transactions that can fit into a block is limited. An unsuitable strategy can easily lead to a significant slowdown of the framework up to the point that it becomes unusable. Also, the transaction filtering strategy interacts with the consensus mechanism in a complex way, as it decides what transactions are going to end up in the block, and thus in the validation algorithm. In fact, if the filtering strategy creates a too big imbalance in the distribution of the transactions, the assumptions required for the heuristics used in the validation algorithm can become too fragile or even incorrect.

A first idea for the design of a transaction filtering strategy is to discard the oldest transaction in the block to make place for the new one. This gives very negative results when attackers spam a lot of made-up transactions. By simply increasing the timestamp by an amount of time that was small so that the made-up transaction is not discarded because it has arrived too early, it is possible for an attacker to produce transactions that are more recent than the others on average. Then, because of the filtering strategy, legit transactions get discarded and legit nodes do not have enough time before block harvesting to produce new transactions to repopulate the block.

A second design relies on statistics computed with transactions as they arrive. In this way, it is possible to decide more carefully what transaction to drop to make place for a new one. For example, a new transaction can only overwrite a transaction from the same sender or from another sender only if the sender of the transaction has too few transactions in the block. Although this strategy seems feasible, the overhead required to update the statistics and to check what transaction to drop is significant and is not responsive enough for the large amount of transactions that each node has to process.

The third strategy is to randomly select a transaction from the block and replace it only if the new transaction is more recent than it. This strategy clearly cannot give the same guarantees as the second one, as it can only provide probabilistic guarantees. However, it is hard for attackers to completely fill up the block with made-up transactions because they would end up overwriting their own transaction with higher probability as more of their transactions are inserted in the block. Moreover, just like attackers, also legit nodes have a higher chance of overwriting made-up transactions when they are already numerous. Another advantage is that this strategy is very fast and only requires very few operations to be completed. In this way, a reasonable trade-off between (D)DoS attack resistance and usability is achieved.

4 IMPLEMENTATION DETAILS

In order to validate and to perform a parameter study on the proposed framework, we developed a simulator. The main requirements that were considered during the design and the implementation of the simulator are simulation realism and simulation efficiency. Since the two requirements are incompatible with each other, in the cases in which they collided we had to evaluate a trade-off. The whole simulator was implemented in C++ 11 and besides the simulation libraries specified in the next section and the standard library, no other external library was used. In the rest of the chapter we present and examine the architecture on which the simulator is based and the main optimizations that we adopted to get an acceptable performance.

4.1 SIMULATOR ARCHITECTURE

It is difficult to achieve a high degree of realism in the simulation of the framework because of the many details to take care of and the overhead they introduce. Traffic mobility simulation is especially important, because it directly affects the nature and the frequency of the interactions between vehicles. For example, a model based on vehicles randomly moving around would be unacceptable, as it is not able to produce realistic mobility and interactions. For this reason, the simulation of traffic mobility is delegated to SUMO [28], that is a microscopic traffic simulator, meaning that it simulates the full behaviour of every vehicle. The network side of the framework is instead simulated with the help of the OMNeT++ [23] library and the Veins [30] framework. The former is a component-based C++ simulation library, that is widely use to simulate distributed systems, while the latter is an inter-vehicular communication simulation framework that relies on OMNeT++ for network simulation and on SUMO for road traffic simulation. The way all these components and the framework simulator interact with each other is shown in Figure 4.1.

The proposed framework is a complex system and the consensus mechanism had to be modified several times as the research went on. Because of this reason, modularity was an important feature of the simulator, and led to a modular component-oriented architecture. Figures 4.2 and 4.3 show the main components in which the framework simulator is divided and their interactions. The design choice to enforce a high degree of modularity and parametrization had a positive impacted on the experiments which the system underwent

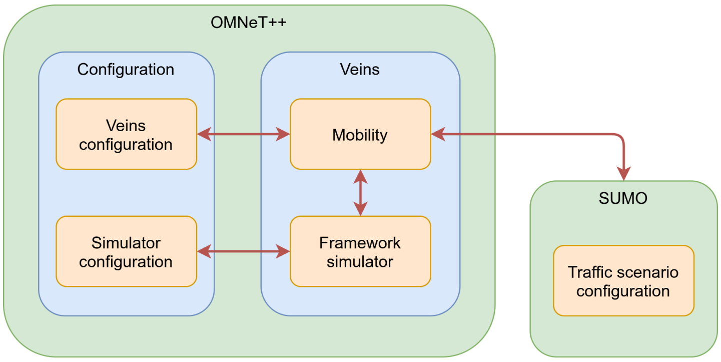
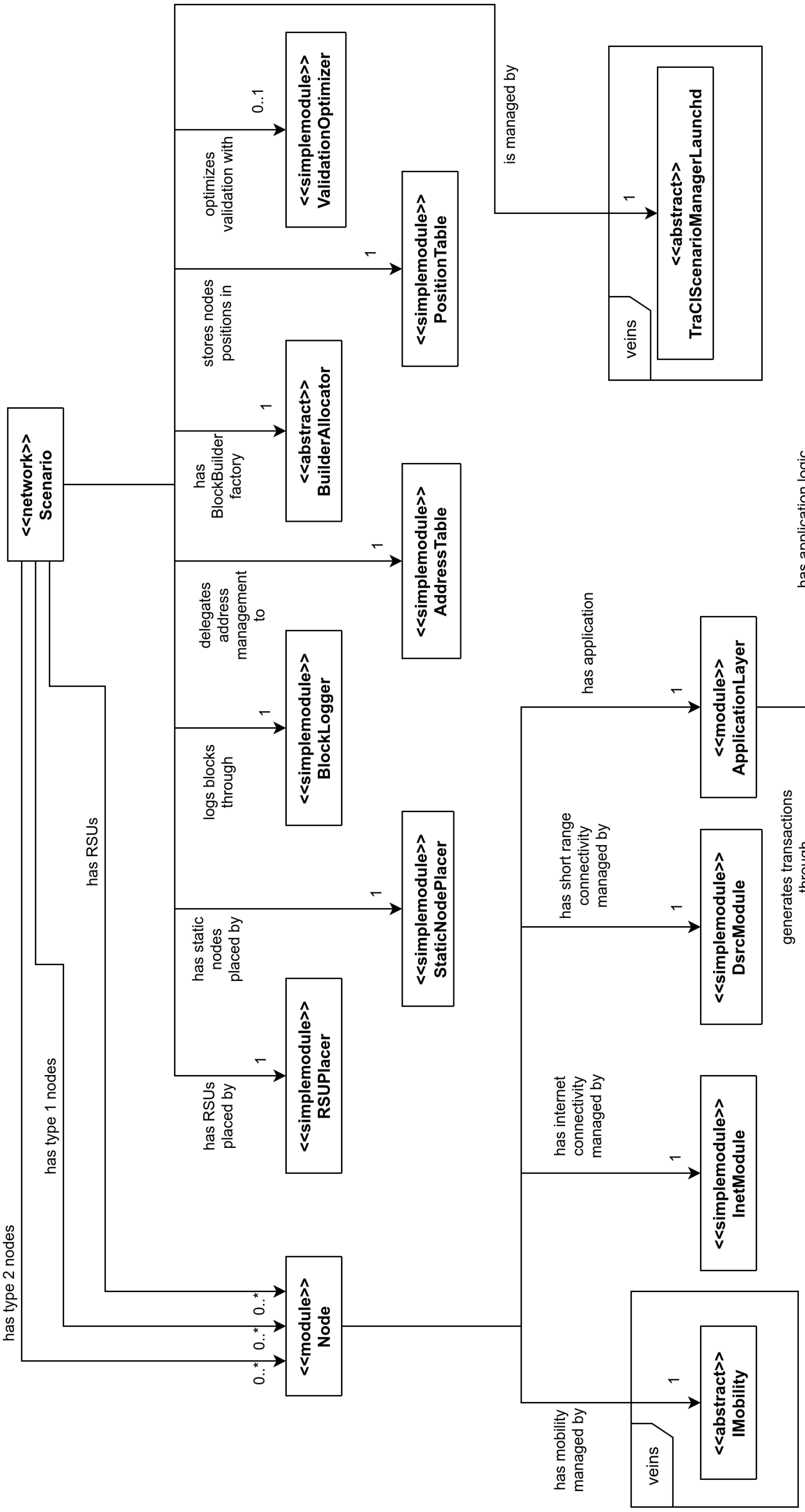


Figure 4.1: The interactions between the various frameworks.

since it allowed to quickly and efficiently try different combinations and configurations without the need to refactor the existing code.

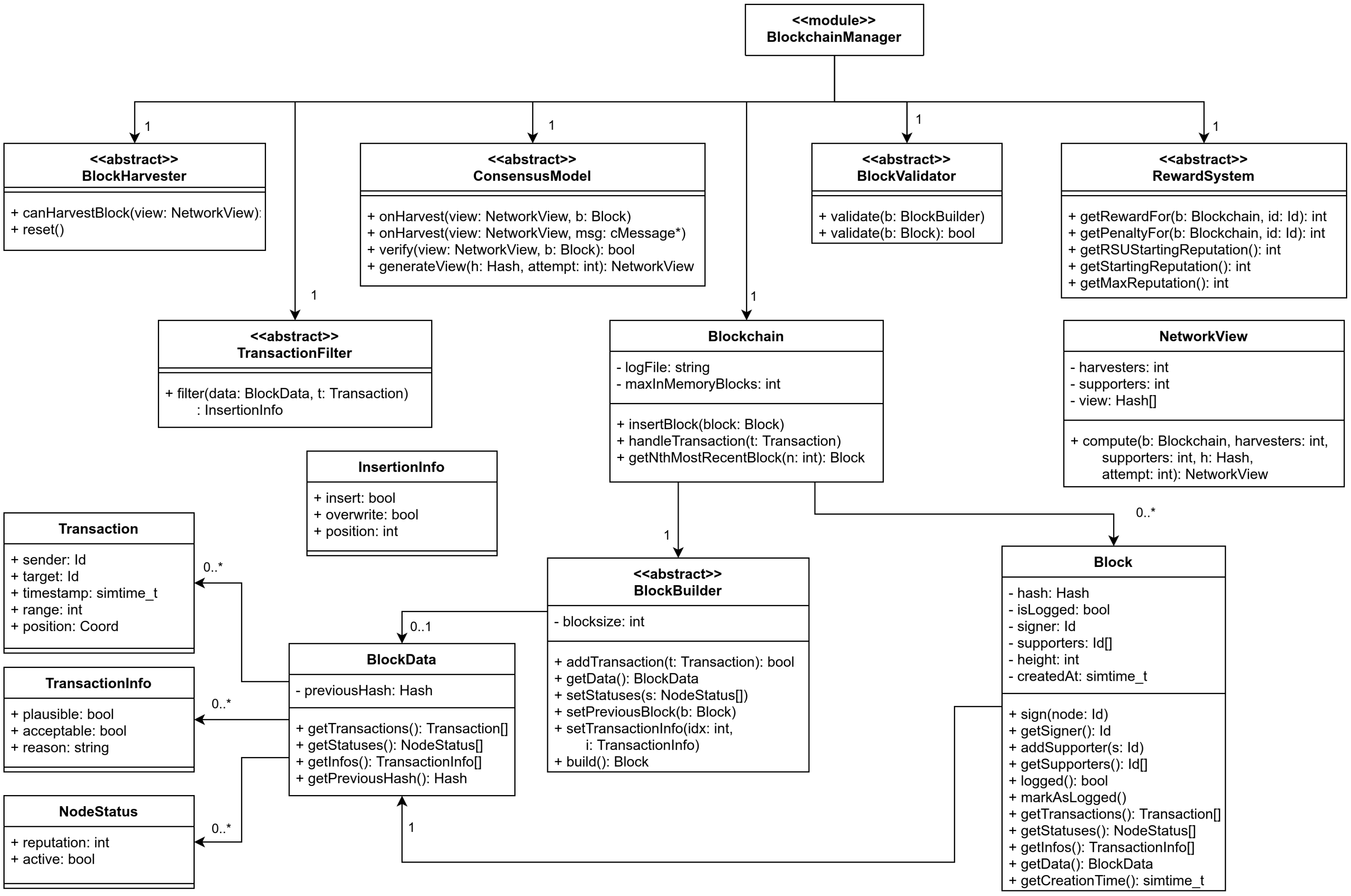
Here is a brief description of the major and most significant components in the framework:

- **RSUPlacer** and **StaticNodePlacer** place RSUs and non-moving nodes at the very beginning of the simulation. RSUPlacer also creates the genesis block.
- **AddressTable** assigns a unique address to each node and can be queried to retrieve all identities and addresses.
- **PositionTable** stores the most recent position of each vehicle.
- **BlockLogger** performs two important operations, which are to log the full blockchain to a log file and to produce the result files containing the statistics about the state and the behaviour of the framework.
- **ValidationOptimizer** caches the most recent validation result. See Section 4.5 for more details.
- **TraCIScenarioManagerLaunchd** is a Veins class that manages vehicle mobility and the communication with SUMO through SUMO's TraCI interface. This class can be specialized to enforce different mobility policies.
- **BuilderAllocator** is an optimization for block creation. See Section 4.4.2 for more details.
- **IMobility** is another Veins class. Its task is to help TraCIScenarioManagerLaunchd to simulate vehicle movement.
- **TransactionGenerator** defines how a node generates the transactions. It can be specialized to simulate different behaviours.
- **Blockchain** contains the logic to handle the blockchain data structure.



<<network>> Scenario
has type 2 nodes
has type 1 nodes
has RSUs
0..*
0..*
0..*
<<module>> Node
has RSUs placed by
1
<<simplemodule>> RSUPlacer
has static nodes placed by
1
<<simplemodule>> StaticNodePlacer
logs blocks through
1
<<simplemodule>> BlockLogger
delegates address management to
1
<<simplemodule>> AddressTable
has BlockBuilder factory
1
<<abstract>> BuilderAllocator
stores nodes positions in
1
<<simplemodule>> PositionTable
optimizes validation with
0..1
<<simplemodule>> ValidationOptimizer
is managed by
1
veins
<<abstract>> TraCIScenarioManagerLaunchd
has mobility managed by
1
veins
<<abstract>> IMobility
has internet connectivity managed by
1
<<simplemodule>> InetModule
has short range connectivity managed by
1
<<simplemodule>> DsrcModule
has application
1
<<module>> ApplicationLayer
generates transactions through
1
<<abstract>> TransactionGenerator
has entry point
1
<<abstract>> Application
has application logic
1
<<module>> BlockchainManager

Figure 4.2: Class diagram representing the components of the simulator at network level.



<<module>>
BlockchainManager
1
<<abstract>>
BlockHarvester
+ canHarvestBlock(view: NetworkView):
+ reset()
1
<<abstract>>
ConsensusModel
+ onHarvest(view: NetworkView, b: Block)
+ onHarvest(view: NetworkView, msg: cMessage*)
+ verify(view: NetworkView, b: Block): bool
+ generateView(h: Hash, attempt: int): NetworkView
1
<<abstract>>
BlockValidator
+ validate(b: BlockBuilder)
+ validate(b: Block): bool
1
<<abstract>>
RewardSystem
+ getRewardFor(b: Blockchain, id: Id): int
+ getPenaltyFor(b: Blockchain, id: Id): int
+ getRSUStartingReputation(): int
+ getStartingReputation(): int
+ getMaxReputation(): int
1
<<abstract>>
TransactionFilter
+ filter(data: BlockData, t: Transaction)
: InsertionInfo
1
Blockchain
- logFile: string
- maxInMemoryBlocks: int
+ insertBlock(block: Block)
+ handleTransaction(t: Transaction)
+ getNthMostRecentBlock(n: int): Block
NetworkView
- harvesters: int
- supporters: int
- view: Hash[]
+ compute(b: Blockchain, harvesters: int,
supporters: int, h: Hash,
attempt: int): NetworkView
InsertionInfo
+ insert: bool
+ overwrite: bool
+ position: int
1
0..*
Transaction
+ sender: Id
+ target: Id
+ timestamp: simtime_t
+ range: int
+ position: Coord
0..*
<<abstract>>
BlockBuilder
- blocksize: int
+ addTransaction(t: Transaction): bool
+ getData(): BlockData
+ setStatuses(s: NodeStatus[])
+ setPreviousBlock(b: Block)
+ setTransactionInfo(idx: int,
i: TransactionInfo)
+ build(): Block
Block
- hash: Hash
- isLogged: bool
- signer: Id
- supporters: Id[]
- height: int
- createdAt: simtime_t
+ sign(node: Id)
+ getSigner(): Id
+ addSupporter(s: Id)
+ getSupporters(): Id[]
+ logged(): bool
+ markAsLogged()
+ getTransactions(): Transaction[]
+ getStatuses(): NodeStatus[]
+ getInfos(): TransactionInfo[]
+ getData(): BlockData
+ getCreationTime(): simtime_t
0..1
BlockData
- previousHash: Hash
+ getTransactions(): Transaction[]
+ getStatuses(): NodeStatus[]
+ getInfos(): TransactionInfo[]
+ getPreviousHash(): Hash
TransactionInfo
+ plausible: bool
+ acceptable: bool
+ reason: string
0..*
NodeStatus
+ reputation: int
+ active: bool
0..*
1

Figure 4.3: Class diagram representing the components of the simulator at blockchain level.

- **BlockHarvester** contains the block harvesting strategy.
- **TransactionFilter** contains the rule to select what transaction to insert and drop.
- **BlockValidator** contains the validation algorithm.
- **ConsensusModel** defines the consensus algorithm.
- **RewardSystem** defines the base values of reward and penalty.

4.2 CONSENSUS ALGORITHM

Practical Byzantine Fault Tolerance (PBFT) is a complex algorithm that requires that the participants update their internal states, exchange messages and manage timeouts in a precise way. The implementation of the Practical Byzantine Fault Tolerance (PBFT) consensus algorithm is based on the specifications given in the paper that first proposed Practical Byzantine Fault Tolerance (PBFT) [6], with some adjustments to integrate it in the framework.

When a new block is broadcast, all nodes check its eligibility by controlling that the harvester has the right to create the block and if the block is eligible, it is stored locally. After this operation the nodes start a timeout and wait for approval signatures from the supporters. Every time an approval signature is received, the nodes check if the sender is among the supporters and if the signature was already received. When there are enough valid signatures bound to the new block, the block is added to the local blockchain and the local state of the node and the timeout are reset.

Supporters have to execute also Practical Byzantine Fault Tolerance (PBFT) alongside the previously defined procedure. The pre-prepare phase begins after the supporters have received the new block and they have checked that the metadata of the new block are correct and that the harvester is eligible. After this, the Practical Byzantine Fault Tolerance (PBFT) algorithm unfolds as it is specified in its paper. The validation algorithm is executed during the commit phase and the approval signature is broadcast only if the block is valid. Because OMNeT++ does not allow time to progress during code execution if not manually managed, the optimization of starting the validation asynchronously during the prepare phase was redundant and would have only generated more complexity. During each phase a timeout is started to avoid deadlocks in case too many nodes stop executing the Practical Byzantine Fault Tolerance (PBFT) algorithm either because they have left the network or because they are misbehaving. The full procedure is shown in the activity diagram in Figure 4.4, while Figure 4.5 highlights how messages are exchanged in the slightly modified Practical Byzantine Fault Tolerance (PBFT) consensus algorithm used in the framework.

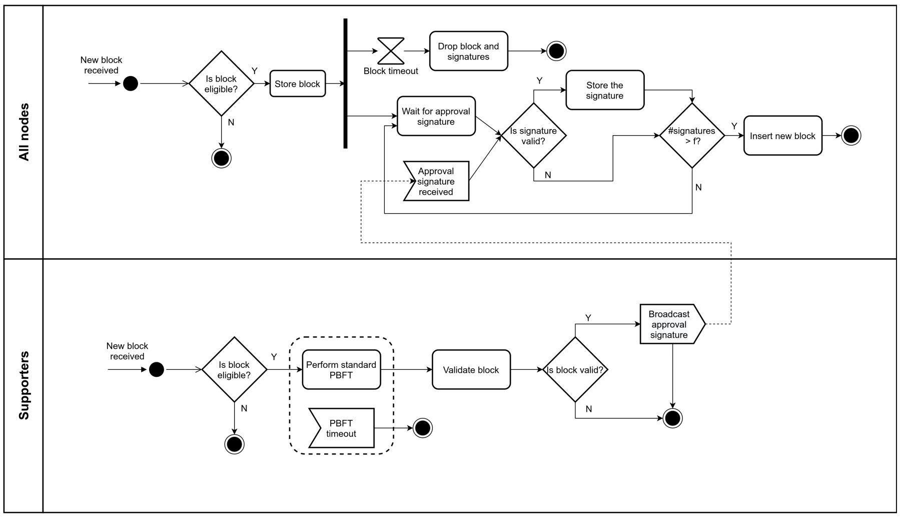


Figure 4.4: The workflow of the implemented consensus algorithm.

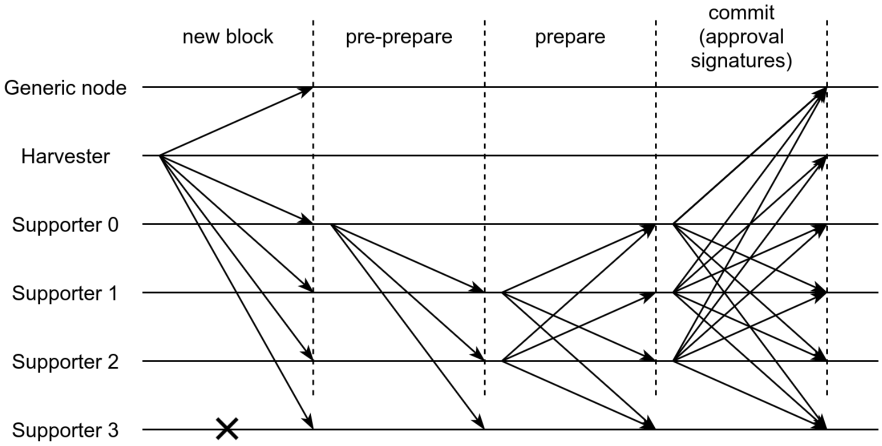


Figure 4.5: The exchanged messages in the slightly modified version of Practical Byzantine Fault Tolerance (PBFT) used in the framework.

4.3 DSRC MODULE

In a real scenario, transactions would be generated through the Dedicated Short Range Communication (DSRC) module or the Cooperative Intelligent Transport Systems (C-ITS) module installed on the vehicle and the weakening of the signal would limit the range of the detection to a certain area. The Veins framework offers a Dedicated Short Range Communication (DSRC) radio module that simulates this situation with realistic signal propagation, but this method does not scale well with the number of Dedicated Short Range Communication (DSRC) modules. For each Dedicated Short Range Communication (DSRC) module, Veins creates an OMNeT++ message and, upon arrival, computes the strength of the signal through a complex procedure. Depending on the receiving antenna and the signal strength, Veins then decides if the receiver module can receive the message and if so, it also decides how many transmission errors to introduce. This is a very realistic approach, but the introduced overhead is significant and causes the simulation to be unacceptably slow when there are over a hundred nodes.

For this reason, the transactions are generated through a custom Dedicated Short Range Communication (DSRC) module. The strategy is to store the positions of all nodes in the position table and to use this information to decide what node is close enough to receive the message *before* sending it. In this way, the overhead caused by the large number of OMNeT++ messages is drastically reduced, and the complex, yet realistic, procedure to determine signal strength is replaced with a simple condition checking if the received is in transmission range. In this way, realism is partially sacrificed in spite of efficiency, but the approximation that is introduced is good enough for the purpose of evaluating the feasibility of the proposed framework.

4.4 BLOCKCHAIN

4.4.1 *Transaction Broadcast*

The major issue with transactions is the large number of them that is generated during runtime. In fact, every node generates a transaction that has to be broadcast to all other nodes every time there is a reply to its Dedicated Short Range Communication (DSRC) probe message. This causes the number of transactions to be in the order of $O(\#nodes^2)$, but with a large coefficient that represents the average number of interactions between vehicles. Moreover, the overhead introduced by the allocation of an OMNeT++ message was about 360% the size of a transaction (64B). In such a situation, during peaks, the consumption of main memory could exceed 8 GB even with just about 1,500 participant nodes (this number is just the limit of participant nodes, not the actual number of active nodes). The number of nodes that can take part in the simulation is thus strongly limited by the amount of available main memory. Also the amount of time required to process every message would take most of the simulation time.

After considering this problem, we decided to emulate transaction broadcast using shared memory. This choice has two major implications, the first being that transaction broadcast is no longer realistic because the transmission time, the propagation time and message omissions are not considered in the simulation anymore. The second implication is that every node, or at least legit nodes, have a perfectly synchronized set of transactions, which is far from realistic. This was though a necessary simplification, so that the simulations can be executed in a reasonable amount of time and main memory. Since the amount of time required to populate, generate and agree on a block is much larger than transmission and propagation latency, this optimization was deemed acceptable nevertheless. In order to implement this optimization, block builders are created though a factory object, the builder allocator, that has the task to manage the shared states. The factory pattern also allows to deactivate this optimization when it is undesired.

4.4.2 *Blockchain Management*

Another memory issue is the management of the blockchain itself. In fact, blockchains require a significant amount of memory to be stored and replicating them for each participant would scale very poorly, even to the point to be simply unfeasible to do so on a single computer. For this reason two optimizations were introduced. The first one is to use read-only shared memory after a block has passed through the consensus mechanism. In this way, only a single copy of the block is kept in main memory and any other copy can simply be dropped. This optimization has no impact on the behaviour and the realism of the simulation because the block has already been digitally signed by the time that the optimization is applied, so even in a real case it would have been impossible for the nodes to alter its content.

The second optimization is block logging. Under the assumption that the framework will not undergo an attack where an attacker tries to fork the blockchain or tries to alter old blocks, only the last few blocks are actually required for the functioning of the framework. The optimization exploit this fact to reduce the amount of blocks to keep in main memory by logging them in the secondary memory and dropping the ones that are older than a certain parameter. Block logging is also important because it allows to analyze the behaviour of the framework after the experiment.

4.5 VALIDATION

The validation algorithm, as explained in Section 3.5, is the most computationally intensive component as it has a quadratic complexity with respect to the number of transactions in a block. Because of this reason, the validation algorithm required special care and had to be optimized significantly in order to allow for a reasonably fast simulation. The algorithm was optimized primarily in two different ways. The first optimization is to reduce the access time and reallocation overhead of the data structure that the algorithm used. The second optimization, instead, relied on the fact that the validation algorithm contains several parts that can be modeled as a computationally intensive reduction operation, meaning that they can be parallelized quite easily.

The data structures used in the implementation of the validation algorithm come from the C++ 11 Standard Template Library (STL), so there are some complexity guarantees for the underlying algorithms as specified by the C++ 11 standard. Because of this, the performance is mainly defined by the choice of the data structure. The factors that affected the choice of a data structure over another one are the data access time and memory reallocation overhead. Because of the optimization of the blockchain that have been previously described, the validation algorithm has plenty of main memory that can be used, so the main concern is about time efficiency. In the original implementation, we used the associative containers `std::map` and `std::set` from STL, which guarantee a logarithmic complexity for both insertion and lookup with no memory reallocation. After some tests, we decided to switch to the use of hash tables under the form of `std::unordered_map` and `std::unordered_set`, as they provided an average speedup of about 200% when enough memory was preallocated. Listing 4.1 shows this optimization on a snippet of the algorithm used to validate transactions and gather statistics about them.

The parallelizable parts followed a simple parallel reduction pattern, so OpenMP was deemed suitable for the task, also because it required minimal code editing. The parallelization process followed an incremental approach, so that it was possible to see where parallelism was best suited and where the introduced overhead countered the benefits. This approach led to the successful parallelization of the most critical loops, namely those that required $O(\mathtt{blocksize}^2)$ iterations, thus achieving a significant speedup. Because the reduction operation in many cases required to work with STL containers, it was necessary to define custom reduction operation, making the implementation incompatible with OpenMP versions prior to 4.0. Listing 4.2 shows a snippet of the procedure used to validate all transactions.

A final optimization which is not related to the implementation of the algorithm itself is validation result caching. In this way, it is possible to avoid executing the validation algorithm once for each

```cpp
ValidationResult validateBlockDataV1(/*...*/) {
    std::map<Address, int> sentTransactions;
    //declarations...
    for(int i = 0; i < transactions.size(); ++i) {
        outcome = validateTransaction(transactions[i], old_blocks);

        sentTransactions.insert({transactions[i]->getSender(), 0});
        //...

        if(outcome.acceptable()) {
            sentTansactions[transactions[i]->getSender()] += 1;
            //...
        }
        //rest of loop body...
    }
    //rest of the function...
}

ValidationResult validateBlockDataV2(/*...*/) {
    std::unordered_map<Address, int> sentTransactions;
    sentTransactions.max_load_factor(0.7);
    sentTransactions.reserve(maxExpectedNumberOfNodes);
    //declarations...
    for(int i = 0; i < transactions.size(); ++i) {
        outcome = validateTransaction(transactions[i], old_blocks);

        sentTransactions.insert({transactions[i]->getSender(), 0});
        //...

        if(outcome.acceptable()) {
            sentTransactions[transactions[i]->getSender()] += 1;
            //...
        }
        //rest of loop body...
    }
    //rest of the function...
}
```

Listing 4.1: Code optimization by replacing tree-based associative containers with hash tables.

```

TransactionInfo validateTransaction(/*parameters...*/) {

    int score = 0;
    int count = 0;
    bool coherent = true;

    for(auto& block: old_blocks) {
#pragma omp parallel for reduction(+:score, count) reduction(&&:coherent)
        for(int i = 0; i < block.transactions().size(); ++i) {
            //loop body...
        }
    }
    //rest of the function...
}

#pragma omp declare reduction(merge: std::unordered_map</*...*/>:/*...*/)

ValidationResult validateBlockData(/*parameteres...*/) {
    //declare containers...
#pragma omp parallel
    {
        //containers memory pre-allocation...
#pragma omp for reduction(merge: /*containers...*/)
        for(int i = 0; i < transactions.size(); ++i) {
            auto outcome = valdateTransaction(/*...*/);
            //rest of loop body...
        }
    }
    //rest of function...
}
```

Listing 4.2: Code optimization by parallelizing critical loops.

supporter node, provided that the input to the validation algorithm is the same, including the locally stored blockchain replica, and the nodes have the same behaviour. This is indeed the case for legit nodes, so the first legit node that executes the validation algorithm, caches the result and the block metadata in the ValidationOptimizer component, while the other nodes have to check if the cached metadata match and if so, simply read the cache. This optimization can also be used by malicious nodes provided that they are not trying to forge a block or perform a Sybil attack, since they still behave properly from the point of view of the consensus mechanism.

5 EXPERIMENTAL PARAMETER CONFIGURATION AND VALIDATION

In this chapter we present the objectives of the experimental validation, together with the experimental setup and the traffic scenario that was used to simulate the framework. We then presents the configurations and the results of the experiments we performed on the framework, which include parameter studies in scenarios without attacks and verification of the resilience of the proposed framework against different types of attacks, as specified in the threat model.

5.1 GOALS

The main goal of the experiments is to perform a feasibility study of the proposed framework. To be feasible, the framework has to be able to function correctly and be usable at least under ideal conditions and under the attack scenarios that have been deemed possible in the threat model in Section 3.3. For this reason, the feasibility study is divided into two main parts:

- **Parameter study**: to be usable, the framework needs to have a stable configuration that allows it to function properly and not to collapse at least in the average case, while still providing some margin on the values of the parameters. The most important parameters are analyzed in order to experimentally find optimal values and estimate their sensitivity. For the experiments relative to the parameter study, we consider the framework as collapsed when the average reputation of nodes gets too close to zero or almost the entirety of the transactions generated by legit nodes are rejected, as in both these situations the proposed framework is incapable of meeting its requirements.

- **Attack resistance**: the other crucial requirement for the feasibility of the proposed framework is the resistance against attacks. The experiments for this part are modeled after the threat model described in Section 3.3. We consider an attack as failed when the average reputation of attackers is significantly lower that the average reputation of legit active nodes and only few malicious transactions are accepted.

The experiments are simulated using the simulator described in Chapter 4 and the base configurations of the test are described in Section 5.4, while the values are changed for the experiments are made clear in each section.

5.2 TRAFFIC SCENARIO

The traffic scenario is the key component to obtain a realistic simulation. In fact, if the vehicles move in an unrealistic way and in an unrealistic road layout, they will generate unrealistic interactions. This prevented to use small and simple traffic scenarios, such as vehicles randomly moving in a square or in grid topologies. Modeling a traffic scenario from scratch requires a large amount of time and effort, as well as the knowledge of the mobility patterns, which requires studying traffic information. Because of this reason, we decided to adopt pre-built SUMO scenarios.

Just like openly available traffic information, also openly available realistic traffic scenarios are scarce and many of them also present substantial simulation issues and simplifications. We considered three real-world scenarios based on actual cities, namely TAPASCologne [1], LuST [8] and MoST [9]. The first one is modeled after the city of Cologne, but it presents so many simulation problems that it is stable enough for tests only when the number of vehicles is limited to the 30% of the total, which is utterly unrealistic. The choice of whether to use LuST, based on the city of Luxembourg, or MoST, based on the Principality of Monaco, was primarily driven by the fact that the latter is more recent and, according to the authors, provides more realistic results.

The MoST traffic scenario spans over a simulated time period of ten hours, during which approximately 46 thousands vehicles are spawned, while about 700 vehicles have to be repositioned during run time because of simulation errors. In the experiments, the number of participants is further restricted to a maximum value and the duration is reduced, so that the experiments remain manageable. Also, public transport, motorcycles and bikes have been excluded as they are not the target of the proposed framework. The RSUs instead are placed so that they lay on the intersections of a fictitious grid over the scenario map.

5.3 EXPERIMENTAL SETUP

In this section, we present the characteristics of the environment which the experiments were performed. The features of the machine used for the experiments are reported in Table 5.1. All the experiments described in this chapter were executed on the very same machine, with no other user application running.

Regarding the traffic scenario simulation, we used the latest available version of MoST (commit 7cee150), together with SUMO version 1.3.1, as suggested by the authors of the MoST scenario. In the network simulation, instead, we used Veins version 5.0 with OMNeT++ version 5.6.1.

COMPONENT	DESCRIPTION
Processor	Intel ®Core ™i7 870 @ 2.93 GHz x8
Main memory	8 GB
Secondary memory	120 GB SSD and 512 GB HDD
Operative system	64-bit Ubuntu 18.04.4 LTS
Virtual machine	No

Table 5.1: Setup of the machine used to run the experiments.

5.4 BASE CONFIGURATION

The simulations required for the experiments need to have a functioning base configuration in order to be meaningful for a parameter study or to confirm that the framework is resistant to certain attacks. To find a suitable base configuration was not an easy task, primarily because there are a lot of parameters which interact with each other in a complex and not easily predictable way. For this reason, the base configuration was found with a trial and error approach, then it was refined by analyzing the effects of the parameters. The final configuration is described in Table 5.2.

The approach adopted to find a functioning base configuration clearly cannot guarantee that the obtained result is a global optimum, but no other suitable configuration was found nevertheless. The high number of RSUs that are inserted in the scenario is needed because the RSUs are not placed according to the road layout, so most of them got placed in unreachable areas. In any case, in all the experiments the total number of RSUs that were actually required never exceeded 30, which means that in the experiments less than the 2% of the nodes are RSUs. The choice of the fault tolerance parameter, instead, is given by the analysis of the consensus algorithm carried out in Section 3.4, and a value of 6 was considered sufficient for the type of experiments that are performed in this chapter.

In the simulations performed for the parameter studies, the framework is always simulated in a trusted environment, that is, in a scenario where there are no malicious nodes and no attack takes place. This means that the simulation is performed under the conditions of normal functioning, so if the framework collapses in such a situation, it is reasonable to assume that it cannot work properly in case of more realistic scenarios where nodes can misbehave. The fact that in the parameters studies there are no attacks also means that all the transactions that are not considered as acceptable are wrongly classified.

COMPONENT	PARAMETER	VALUE(S)
Scenario	Number of vehicles	1500
	Number of RSUs	400
Blockchain	Maximum number of transactions per block	50000
	Time before harvesting a new block	60 s
Transaction validation	Number of previous blocks	3
	Threshold	0.1
Misbehaviour penalization	Detection variance weight	0.67
	Maximum number of tolerable conflicts before blacklisting	2
Consensus algorithm	Fault tolerance	6
Reward system	Maximum reputation	4096
	Initial reputation	64
	Base reward	256
	Base penalty	512

Table 5.2: Base configuration used in the experiments.

5.5 REWARD SYSTEM CONFIGURATION

The reward system has a direct impact on the functioning of the proposed framework and when wrongly configured, it can lead to the collapse of the framework or to a state where even malicious nodes do not lose reputation. In this section, we test different configurations of the reward system to experimentally find bounds and characteristics of the parameters of the reward system. In particular, we analyze how the base amount of penalty and the amount of reputation assigned to newly joined nodes affect the proposed framework.

5.5.1 *Penalty*

This experiment is meant to show how the base amount of penalty that is given for each implausible transaction affects the system. Intuitively, a penalty that is too harsh can easily lead to a collapsed state in which nodes do not have enough reputation to allow legit transactions to be accepted. On the other hand, the penalty should not be too low, because otherwise malicious entities would not be discouraged from carrying out their attack. For these reasons, the penalty should be chosen to be the maximum value that does not lead to a collapse.

Figure 5.1 shows the outcome of the experiment. The first of the two graphics shows the evolution of the average reputation of active nodes as the blockchain used in the framework grows. The results meet the expectations, in fact, harsher reward system lead to a lower average rep, while more bland penalties lead to a higher average reputation. The spike of average reputation at the very beginning is caused by the reputation that is dealt to initially trusted entities in the genesis block. After an initial transient, in all the configurations the average reputation stabilizes and remains constant for the rest of the simulation, although with small fluctuations. The transient is caused by the fact that vehicles are not added all at once, and they also need some time to start generating interactions and to gain reputation. The reward system in which nodes are distrusted after a single implausible transaction, i.e., the configuration with $\mathtt{penalty} = \infty$, eventually leads to a collapse and thus is unusable. The second graphic follows a similar trend, with harsh penalties that lead to a situation where almost all transactions are discarded. It is noteworthy the fact that bland penalties result in a similar number of accepted transactions, implying that a portion of the rejected transactions does not depend on the reputation of the nodes, but it is caused by misclassification.

5.5.2 *Initial Reputation*

The purpose of the experiment is to determine the effects of the initial reputation that is assigned to nodes when they join for the first time

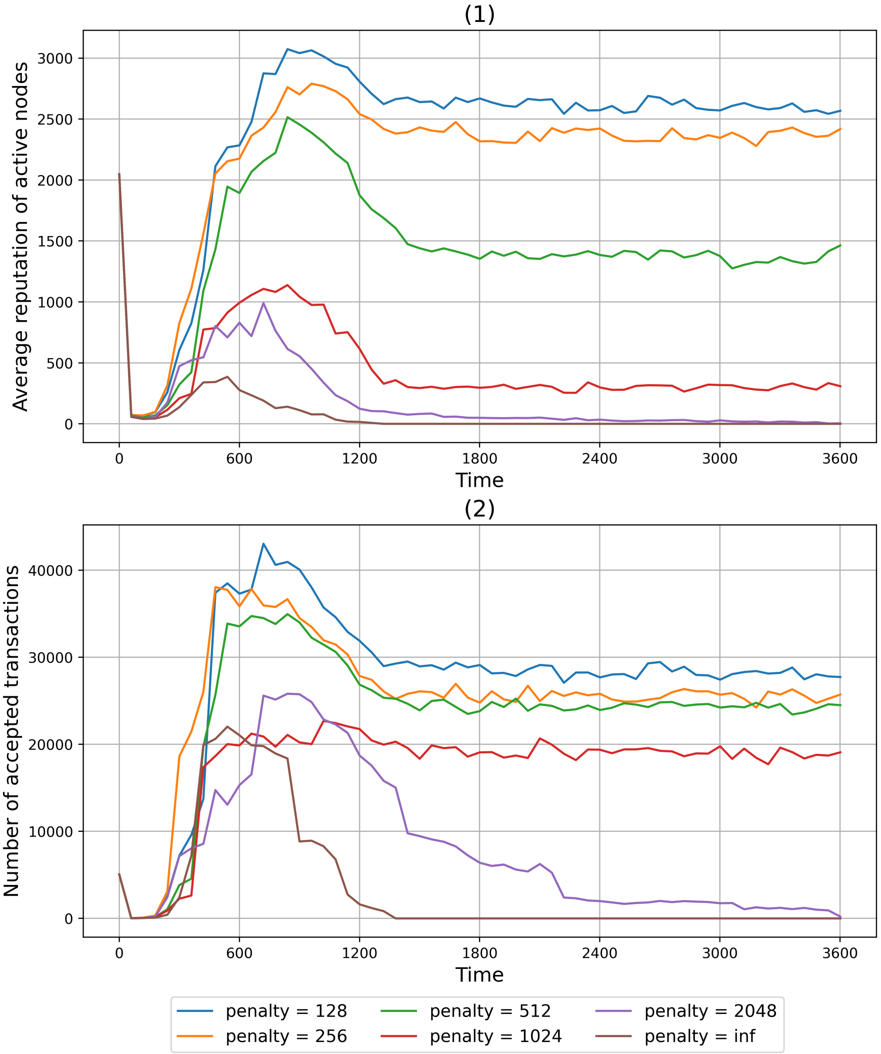


Figure 5.1: The average reputation of active nodes over time (1) and the number of accepted transactions in each block (2) with different base penalty values.

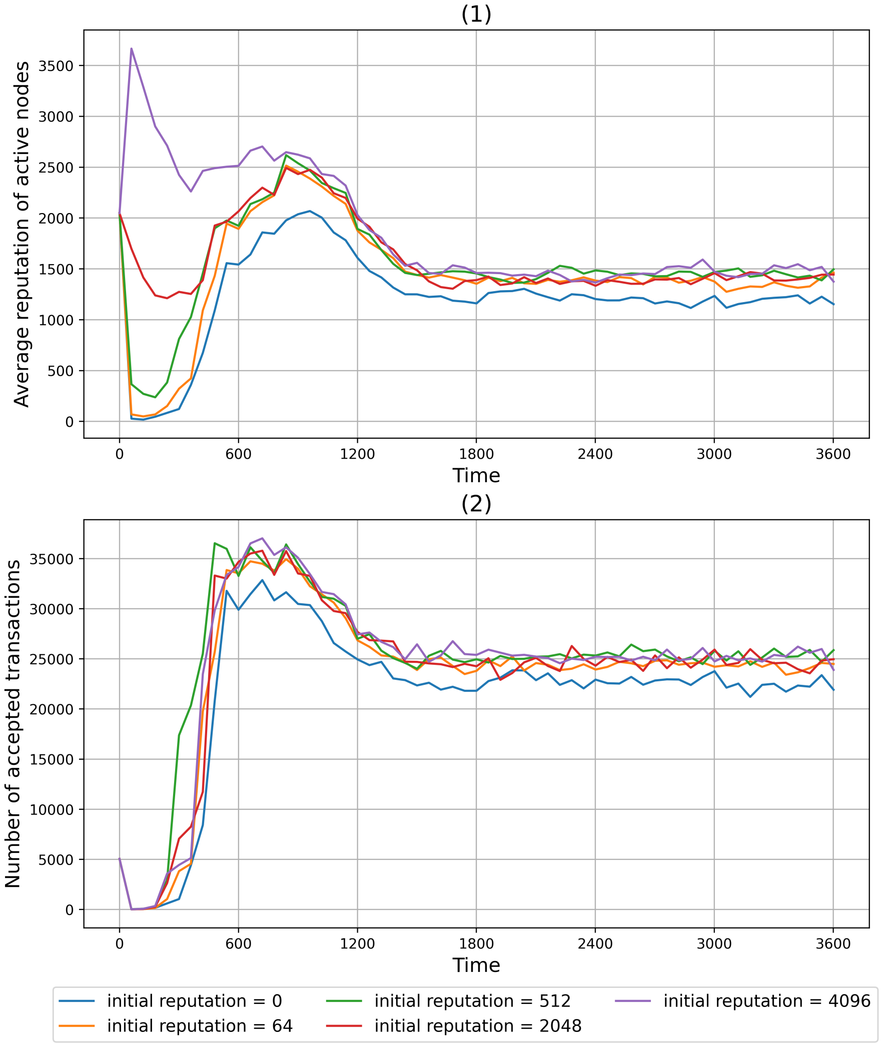


Figure 5.2: The average reputation of active nodes over time (1) and the number of accepted transactions in each block (2) with different amounts of initial reputation.

the blockchain. It is important to notice that the RSUs and the nodes that start as trusted entities are not affected by the choice of the initial reputation, as their reputation is statically assigned before the blockchain is started. Ideally, the framework should be able to work correctly even if the initial reputation is very low, as this situation prevents attackers to join all at once to perform an attack.

From Figure 5.2, it is possible to see that the amount of reputation given to newly joined vehicles does not affect significantly the average reputation of active nodes and the number of accepted transactions when the framework is full operational. The only moment that is affected is the initial transient for the average reputation, but only at the beginning of the simulation. In fact, in all cases the transient has the same duration, while the number of accepted transactions is not significantly affected. The only configuration that performs slightly worse than the other is the one in which the initial reputation is zero. This is caused by the fact that it is slightly harder to have accepted transactions at the very beginning due to having no reputation and the threshold in the transaction validation heuristic, so when vehicles detach from the original group and try to start a new one, it is more likely that they do not have enough reputation to start a new group. These observations, combined with the concept that the initial reputation should be small, lead to the conclusion that a small, but strictly grater than zero initial reputation is the best choice to have a functioning framework that is more resistant to attacks.

5.6 TRANSACTION VALIDATION CONFIGURATION

The transaction validation heuristic has two main parameters, which are the number of previous blocks to consider and the threshold used to determine if a transaction is acceptable. In this section, the two parameters are analyzed independently to experimentally determine their effects on the framework.

5.6.1 *Number of previous blocks*

The transaction validation heuristic uses the transactions contained in previous blocks in order to check the new ones. The number of previous blocks can thus influence the number of new transactions that are labeled as acceptable, meaning that unsuitable values can lead to low acceptance rates even to the point that the framework is unusable. However, if this parameter is allowed to have a too large value, the complexity of the transaction validation heuristic grows, in fact, there is a linear dependency between the complexity and the number of previous blocks, as explained in Section 3.5.2. Consequently, the purpose of this experiment is to study how different values of this parameter influence the performance of the framework and which ones allow the framework to function properly, while being small enough to avoid unnecessary complexity.

The results of this experiment are shown in Figure 5.3. From the graphics, it is possible to notice that, except for the cases in which the number of blocks used in the transaction validation heuristic is 1 and 2, all configurations perform similarly, both in terms of average reputation and number of accepted transactions. This is caused by the fact that the transactions contained in blocks that are too old create very loose bounds, since the interval of time between them and new transactions is large. Because of this reason, the parameter that indicates how many blocks to use in transaction validation has diminishing returns, while its impact on the processor time required to carry out the execution of the transaction validation heuristic is linear.

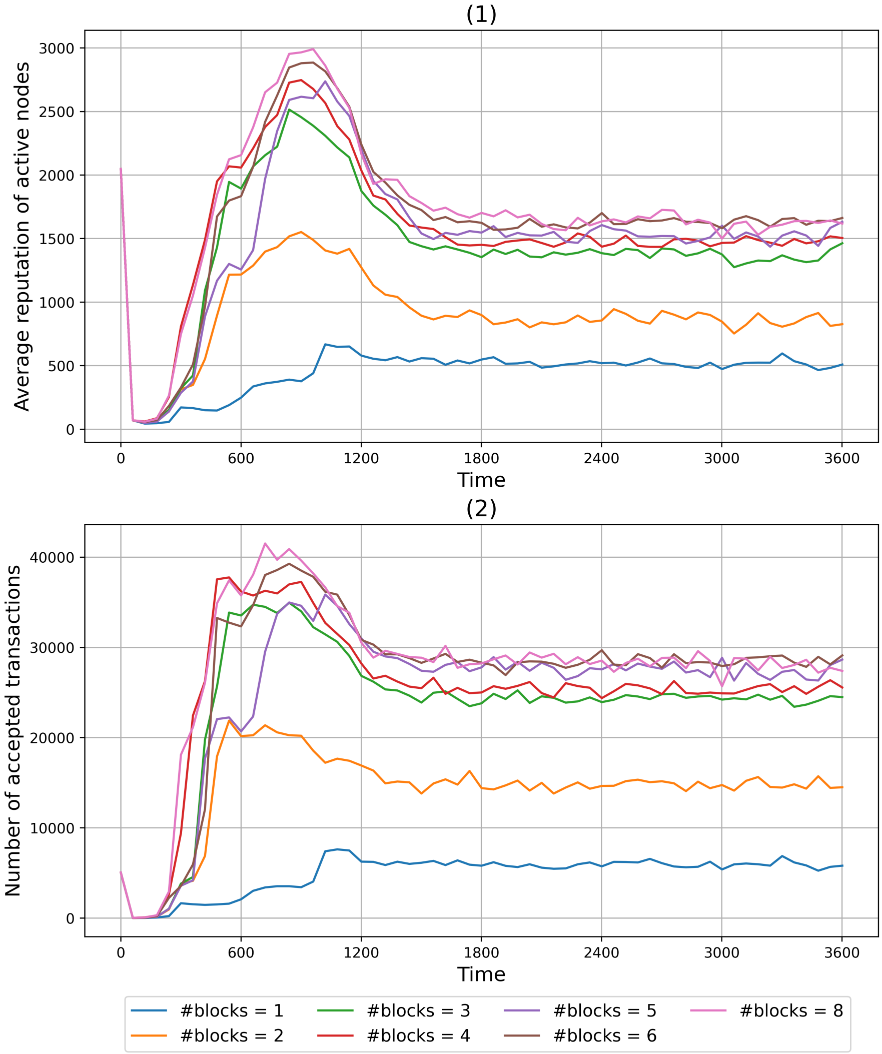


Figure 5.3: The average reputation of active nodes over time (1) and the number of accepted transactions in each block (2) with different numbers of blocks considered in the transaction validation heuristic.

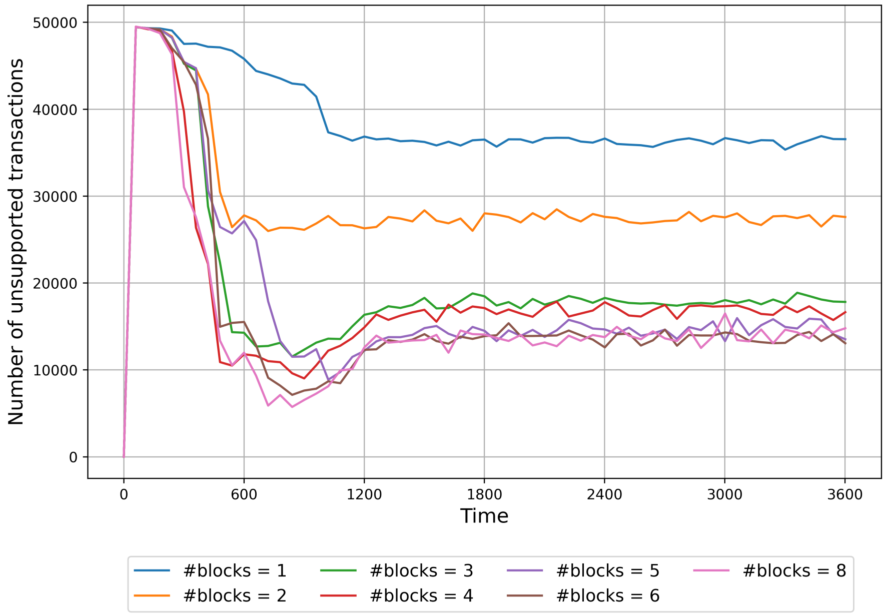


Figure 5.4: The number of transactions in each block that cannot be labeled as acceptable or implausible with different numbers of blocks considered in the transaction validation heuristic.

The cases where $\#\mathtt{blocks} = 1$ and $\#\mathtt{blocks} = 2$, instead, perform significantly worse than the other ones, with the first configuration of the two that is even close to collapsing. The reason behind this behaviour is that the transaction validation heuristic does not have enough information to decide whether to accept or reject transactions in more situations, so the number of times the validation of a transaction has to be suspended increases, as it is shown in Figure 5.4.

5.6.2 *Threshold*

With this experiment, we want to assess if and how the threshold against which the score obtained in the transaction validation heuristic by new transactions is compared affects the stability and the performance of the proposed framework. Ideally, the value of this threshold could be used to tune the flexibility of the framework so that it could become more or less restrictive against fake or imprecise position advertising.

The possible values of the threshold are the same as the possible values of the transaction score, which span over the interval $[-\mathtt{max\text{-}reputation}; \mathtt{max\text{-}reputation}]$. However, this interval is too wide for the threshold and some values are rarely reached or meaningless. For example, if a threshold of $\mathtt{max\text{-}reputation}$ was used, then the score would have to be the maximum possible. In such a situation, the reputation of all the nodes in the concordant set would have to be the maximum, while the discordant set should be empty

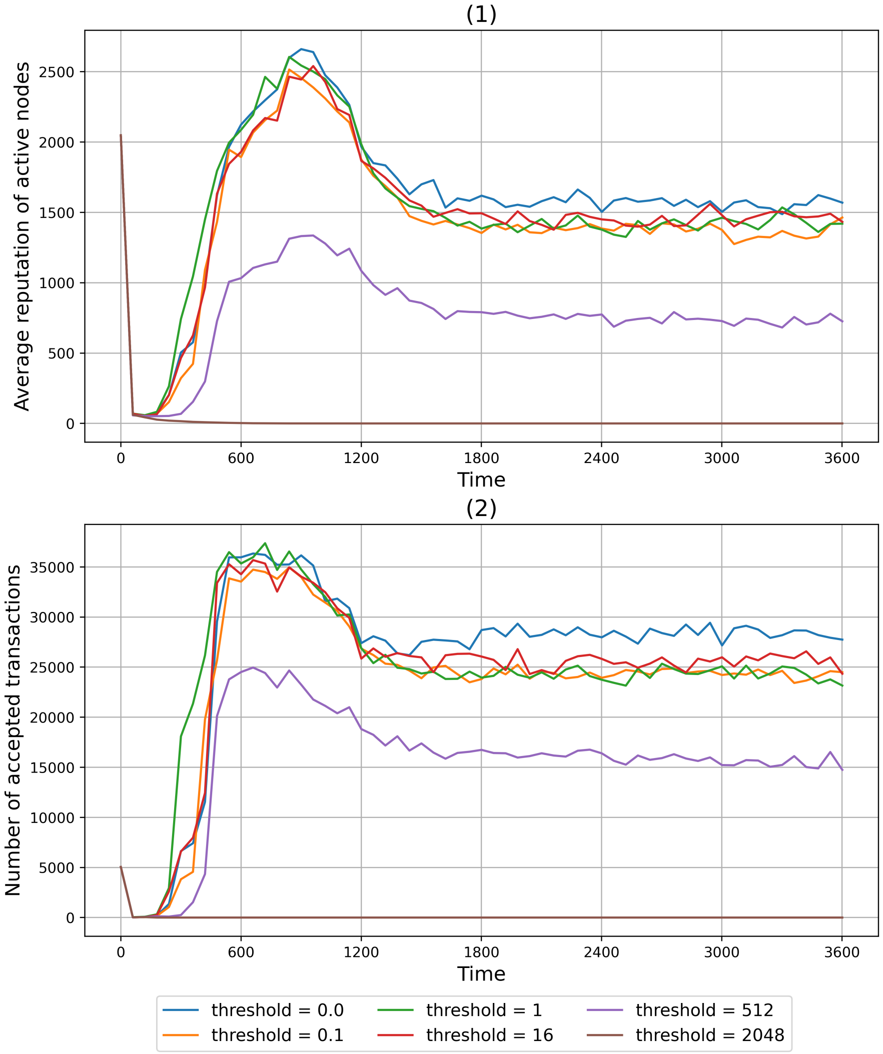


Figure 5.5: The average reputation of active nodes over time (1) and the number of accepted transactions in each block (2) with different thresholds.

or have a total reputation of zero. This in a very unlikely case, so too little transactions would be accepted and the framework would collapse. Negative values, instead, are meaningless, as they imply that the framework would have to accept transactions that are not trusted. For these reasons, the interval of values of the threshold is further restricted to small, non-negative values.

The results of the experiment are shown in Figure 5.5. It is possible to notice that small values for the threshold have a very similar behaviour, except for the case in which the threshold is zero, in which both the average reputation and the number of accepted transactions is higher. Although it may seem that the performance is better with a null threshold, this configuration leads to the undesirable situation in which nodes with zero reputation can make a transaction be accepted. This situation is optimal for attackers, as they would be able to support transactions generated by accomplices even after they have depleted their reputation. Because of this reason, a null threshold should not be used, even though it gives slightly higher performance.

As predicted, large values for the threshold lead to a collapsed state, as it is the case for the configuration in which $\texttt{threshold} = 2048$. In this configuration the framework utterly fails after little time and it becomes unable to provide its services since no transaction can be accepted anymore. When the threshold is 512, instead, the framework is still able to provide its functions, but it does so in a sub-optimal way, where the overall performance, both in terms of average reputation and accepted transactions, is almost halved when the framework reaches a steady state.

5.7 MISBEHAVIOUR PENALIZATION CONFIGURATION

Misbehaviour penalization is the most critical heuristic in the validation algorithm, as it relies on a lot of assumptions and approximations. Its functioning is driven mainly by the weight that is given to the detection variance when computing the scaling factors and the maximum number of tolerable conflicts before blacklisting a node. In the rest of the section, the effects of these two parameters on the performance of the framework are experimentally evaluated.

5.7.1 *Detection Variance Weight*

The weight given to detection variance directly affects the reputation variation, so it is important to understand its effects and what values allow the framework to work as it is supposed to. While it is true that low detection variance is typical of small-sized malicious groups of nodes, legit nodes may detect the same vehicles more than once as well, for example, when they are going the same direction or when they are stuck in a traffic congestion. This means that if the reward is scaled too much because of the detection variance, legit nodes may get a reputation of zero. In fact, if some transactions of a legit node are wrongly classified and the reputation increment generated by correctly classified transactions is nullified or scaled too much, the final reputation variation can be negative. Because of this, the weight given to the detection variance should be the highest value that still allows legit nodes to gain reputation even in the presence of wrongly classified transactions.

The results displayed in Figure 5.6 suggest the fact that the value of weight of the detection variance used in the computation of the scaling factors is not significantly involved in the transaction acceptance process. This is indeed a desirable property because according to the design intentions, the detection variance weight should only scale the reward of suspicious nodes. The effects of the detection variance

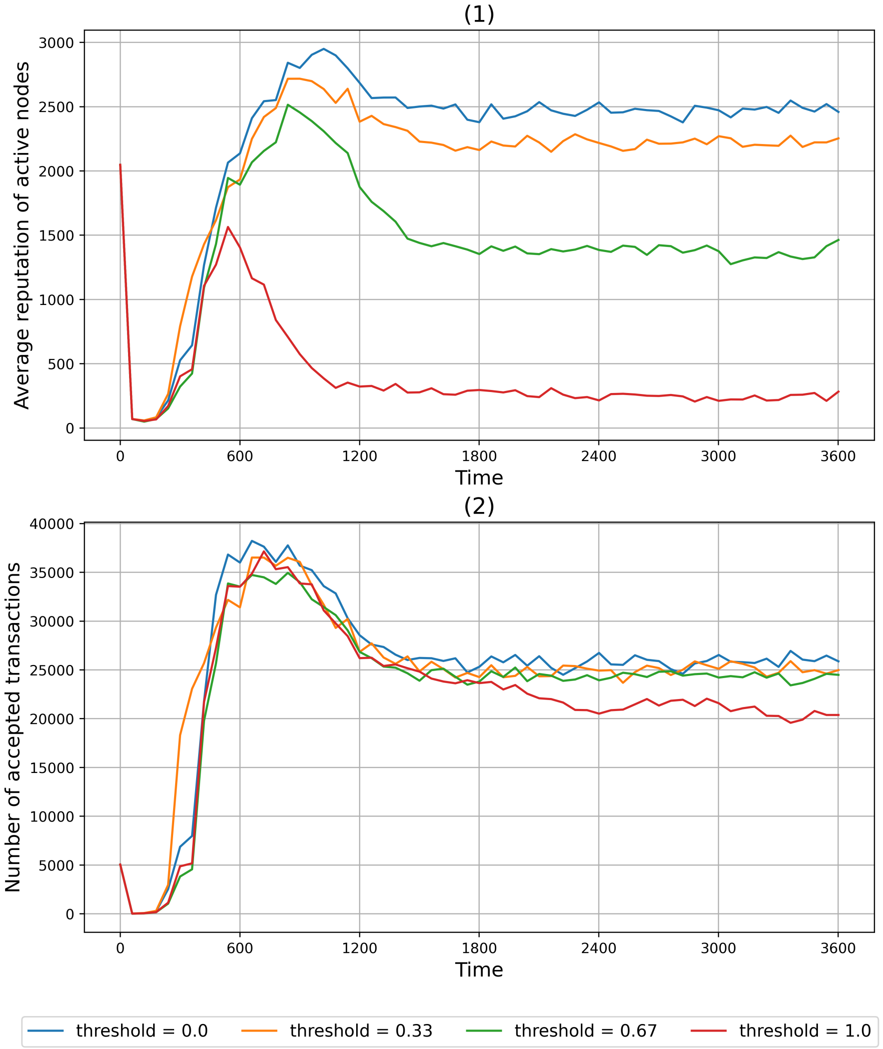


Figure 5.6: The average reputation of active nodes over time (1) and the number of accepted transactions in each block (2) with different detection variance weights used in the scale factor computation.

weight is much more clear in the second graphic, which shows the average reputation of active nodes.

Predictably, higher weights lead to a lower average reputation even if all nodes are behaving properly and there is no malicious node that selectively generates transactions. The variations in the average reputation does not seem to be linearly dependent on the value of the weight. In fact, the same variation of one third has much more effect on the average reputation when it is applied to higher weights. This is likely caused by the fact that the validation algorithm and the interactions between its components are complex, so it is not easy to predict the response to different parameter values. The effects on the average reputation in the case where $weight = 1.0$ are so heavy that the framework is likely to become unstable, since the assumption that legit nodes have higher reputation than misbehaving nodes becomes too fragile. This can easily be exploited by attackers to avoid misbehaviour penalization by gaining just slightly more reputation than the average so that they are rarely or never penalized.

5.7.2 *Maximum Tolerable Conflicts*

The maximum number of tolerable conflicts is the main parameter that drives the misbehaviour penalization heuristic, as it allows to set a bound that separates suspicious behaviour from tolerable behaviour. Since some transactions can be dropped because they are discarded or did not fit in the block, the behaviour of legit nodes may become wrongly considered as suspicious, so this parameter requires to be configured carefully. In fact, too loose values would allow malicious behaviour to simply pass unnoticed, while too strict values would make the framework collapse by penalizing legit nodes too much. The purpose of this experiment is to experimentally find what is the impact of the maximum number of tolerable conflicts on the framework and which values allow the framework to do its operations correctly in a scenario without attacks.

The results of this experiment are displayed in Figure 5.7. The bound determined by the maximum number of tolerable conflicts becomes more bland as the value of the parameter increases, meaning that fewer nodes are blacklisted because of their suspicious behaviour. It is important to remember that the parameter indicating the maximum number of tolerable conflicts is not used to deal the additional penalties since they are given when a conflict is lost, but rather it is used to decide when a node is behaving suspiciously and so has to be blacklisted, thus rejecting all its transactions.

The outcome of this experiment is more difficult to interpret than the previous ones, mainly due to the fact that the misbehaviour penalization heuristic heavily relies on assumptions and simplifications. The first aspect that can be notices is that the framework behaves

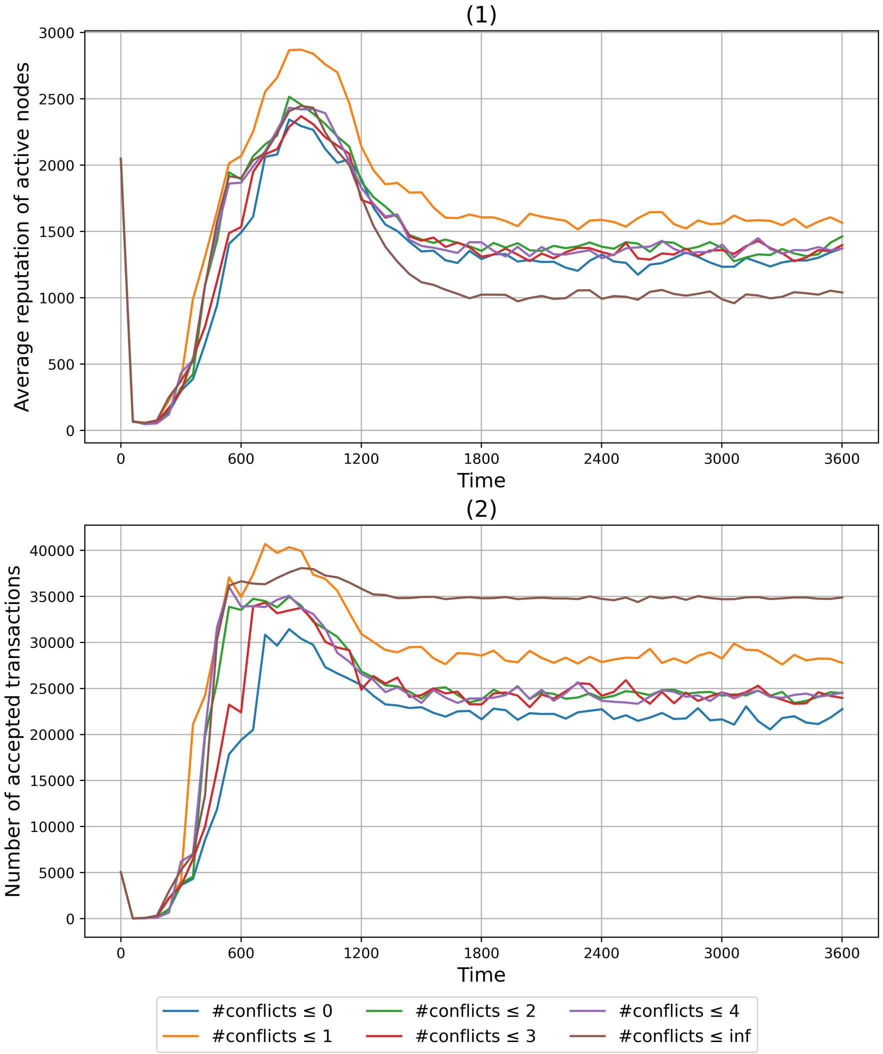


Figure 5.7: The average reputation of active nodes over time (1) and the number of accepted transactions in each block (2) with different upper bounds for the number of tolerable conflicts.

in a counterintuitive way, where the blandest bound produces the lowest average reputation, while stricter bounds have a higher average reputation and behave similarly in all configurations. This behaviour is caused by the fact that when enough nodes are blacklisted, during the validation of the next block more transactions are rejected, causing fewer conflicts to be detected, so the misbehaviour penalization heuristic deals less penalty overall. This is more evident if the second graphic is taken into account. In fact, the second graphic shows that blandest bound produces the highest number of accepted transactions.

5.8 ATTACK SCENARIOS

All the experiments carried out up to this point only considered a trusted environment with no malicious entities or attackers, but in order to satisfy all the objectives that we set for the experimental part of this work, it is important also to consider the cases in which the framework undergoes an attack. Since the security guarantees of the consensus algorithm have already been discussed and characterized in Section 3.4, they are not examined any further.

The attack scenarios considered in this set of experiments are modeled after the threat model presented in Section 3.3. Because the attacks do not interfere with each other, we deemed unnecessary to test a scenario where they all take place at the same time, so each attack scenario only contains one type of attack. This choice also makes possible to better understand the effects and the outcomes of each attack independently. In the next sections, we present the experiments based on attack scenarios and discuss the relationship between the results and the choices made in Chapter 3.

5.8.1 *Random Transactions Generation*

This is a very simple attack where the attackers generate transactions with a random position and a random target to add noise to the transaction validation algorithm and to fill up the block. Although this attack is simple, when this kind of transactions are wrongly categorized as correct they can have and effect on the evaluation of other legit transactions, especially in case the concordant nodes have low total reputation. More importantly, this attack can cause a (D)DoS if the content of the block is filled up by too many random transactions. In this scenario, the attackers spam 8 random transactions per second alongside correct transactions, for a total of 12 transactions per second on average. For comparison, legit nodes generate 4 transactions per second on average.

Figure 5.8 displays the outcome of the experiment. The malicious nodes are 5% of the total nodes bound to vehicles, but they produce approximately three times the normal amount of transactions, up

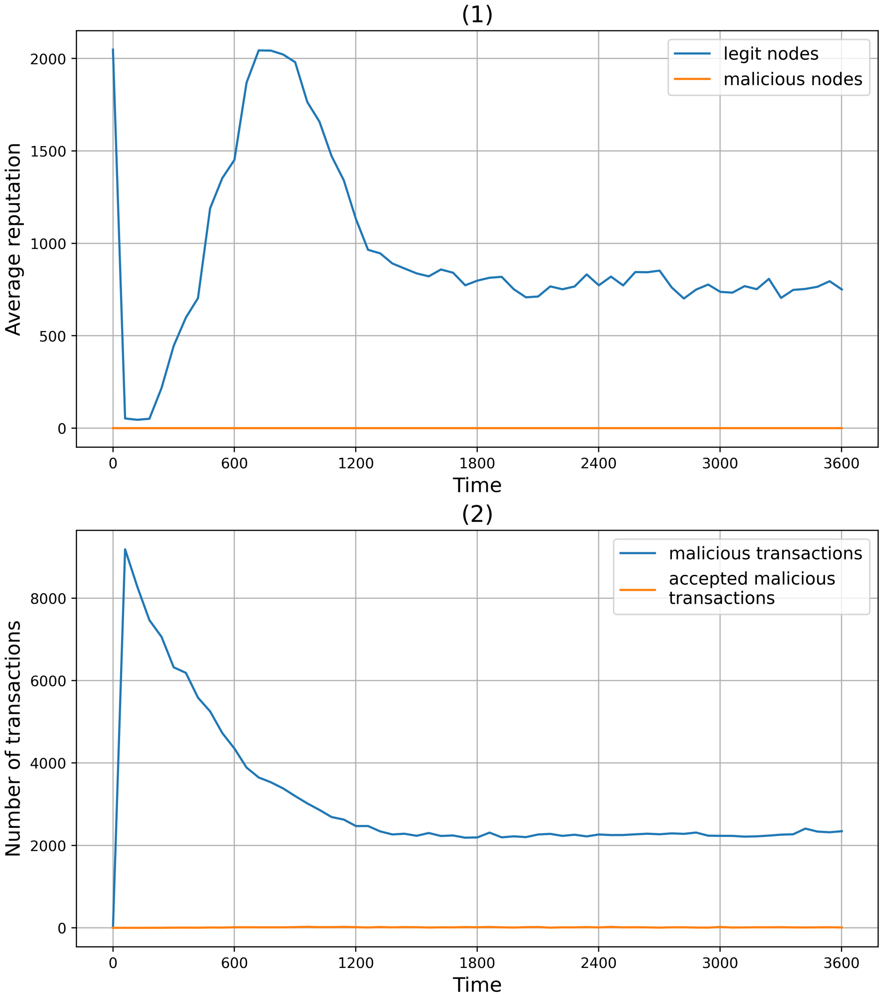


Figure 5.8: The outcome of a random transaction generation attack where 5% of the total nodes are malicious. (1) The average reputation of both attackers and legit nodes over time. (2) The number of transactions that are generated by the attackers and that are accepted by the framework over time. Attackers generate 3 times the number of transaction of legit nodes.

to the point where about 15% of the roughly 457,000 transactions generated before the next block is harvested are from an attacker on average.

The initial peak of malicious transactions in the block displayed at the beginning of the second graphic is simply due to the fact that the framework is still starting and the legit nodes are not yet generating many interactions, meaning that the block gets filled up mainly with spam transactions. As the framework reaches a steady state, the number of spam transactions that are present in the block diminishes, as many of them are dropped, and becomes constant. Both the graphic showing the average reputations and the one comparing the number of accepted malicious transactions with the total number of malicious transactions in the block illustrate how the attack utterly fails. The spam transactions are never accepted, except for a couple of them once in a while, and the reputation of the attackers consistently sticks to zero.

Another important aspect is that although malicious nodes produced about 15% of the total amount of transactions, from the second graphic it is possible to see that only approximately 2,500 transactions generated by malicious nodes ends up in the block when it is harvested, meaning that they are only the 5% of the total number of transactions in the block. This fact is determined by two factors, namely the transaction filtering strategy and the large number of nodes. The reasons why these factors effectively mitigate (Distributed) Denial of Service ((D)DoS) attacks have already been discussed in Section 3.6.3.

5.8.2 *Fake Position Advertisement*

The behaviour of the attackers in this attack scenario is very similar to the behaviour of legit nodes, but when the attackers generate their transaction, they add a large offset to their real position, meaning that they try to appear to be in a different position on the map. Similarly to random transactions, this kind of transactions can lower the overall quality of the traffic information if too many of them are wrongly classified as acceptable. Moreover, they can negatively contribute to the evaluation of the plausibility of the transactions generated by the nodes that have detected the attacker, as the legit nodes can appear to be lying about the attacker's position.

For this experiment, 5% of the total nodes are malicious and the offset used to spoof their position is randomly picked from the range $[4\mathrm{km}; 6\mathrm{km}]$, meaning that they would appear in the other side of a medium-sized city if it were a real scenario. Figure 5.9 shows the outcome of the experiment. Even in this case the attack is unsuccessful, as the average reputation of the attackers reaches zero almost at the beginning and stays constant for the rest of the simulation, except for

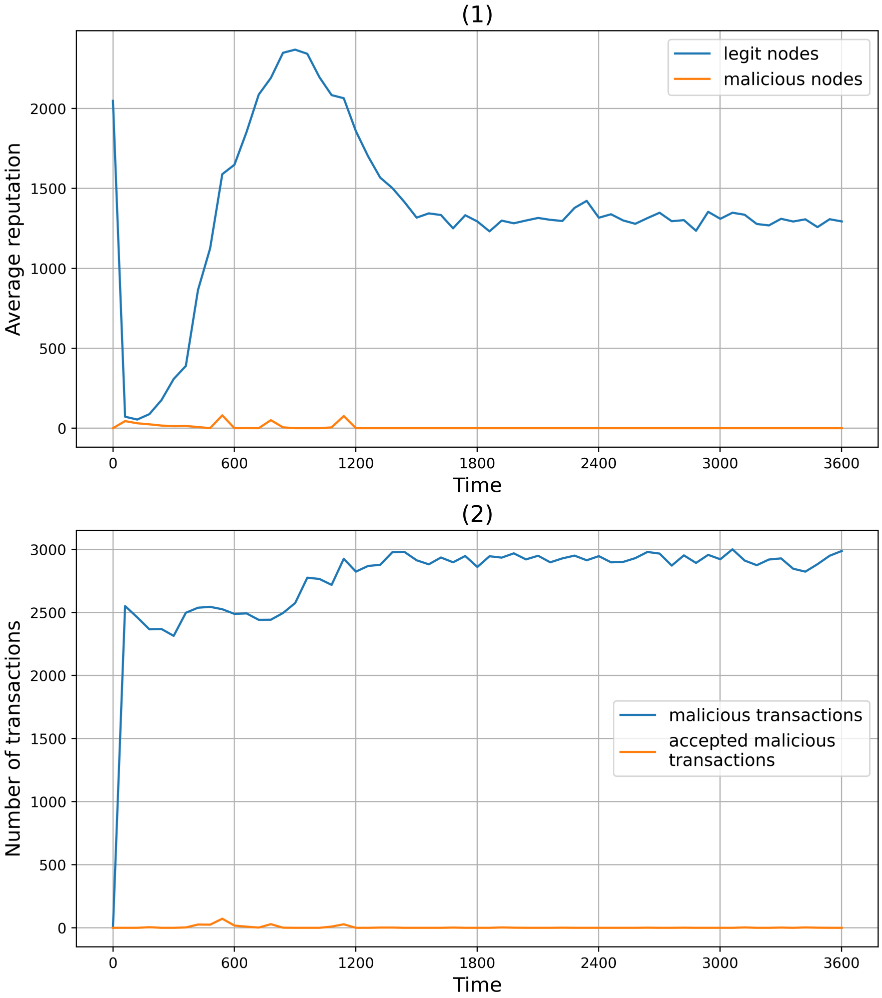


Figure 5.9: The outcome of a fake position advertisement attack where 5% of the total nodes are malicious. (1) The average reputation of both attackers and legit nodes over time. (2) The number of transactions that are generated by the attackers and that are accepted by the framework over time. The offset used by attackers is randomly selected from the range [4km; 6km].

some small spikes at about 600 seconds and 1,200 seconds since the beginning of the simulation. Also, nearly all malicious transactions are systematically rejected and only very few of the get wrongly classified as acceptable. This pattern is due to the transaction validation heuristic, where although the sender position is coherent with the previous ones since the same offset is added, the score computed with the concordant and the discordant nodes does not reach the threshold, so the transactions are simply rejected.

5.8.3 *Teleport Attack*

This attack is very similar to fake position advertisement, but instead of using a fixed offset, the attacker uses a new random position for each transaction it generates. This means that fake position advertisement is a special case of this attack, implying that all the considerations that are valid for fake transaction advertisement are also true for teleport attacks. In this specific scenario, the attackers generate a random offset for each transaction and add it to their real position, in such a way that they can hide their real position and appear to be in distant locations after a very short amount of time.

In this experiment too, the percentage of malicious nodes is 5% of the total. The results shown in Figure 5.10 are very similar to the ones obtained in the attack scenario where attackers try to fake their position by adding an offset to the real position. This is easily explained by the fact that the bounds used in transaction validation that prevent fake position advertisement are closely related with the one that requires nodes to advertise a new position that is coherent with previous ones. This causes the malicious transactions to be rejected according to a similar pattern.

5.8.4 *Copycat Attack*

In this attack scenario as well, the attackers act independently. The attack is performed by selecting a random target and waiting that it broadcasts the transactions it generates. At this point the attacker reads them, copies their content into fake transactions where the attacker itself is the sender and broadcasts the forged transactions. If this attack succeeds, attackers can copycat the behaviour of legit nodes and appear somewhere else in the map, thus successfully spoofing their own position while seemingly behaving properly and gaining reputation.

In this experiment, 5% of the total nodes are malicious and try to carry out the attack. The results of this experiment are shown in Figure 5.11. It is possible to observe that the attack is unsuccessful as the average reputation of the attackers is very close to zero and only a small amount of malicious transactions is accepted. The failure of

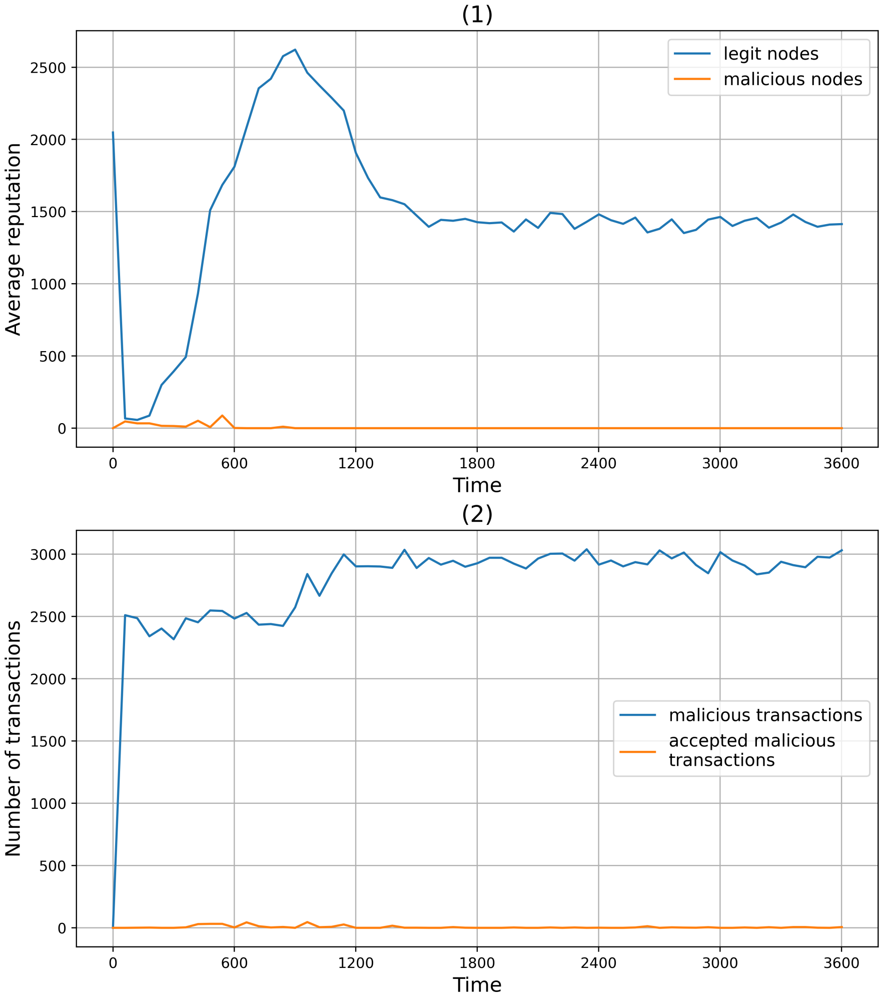


Figure 5.10: The outcome of a teleport attack where 5% of the total nodes are malicious. (1) The average reputation of both attackers and legit nodes over time. (2) The number of transactions that are generated by the attackers and that are accepted by the framework over time. The teleportation distance is randomly selected from the range $[4\text{km}; 6\text{km}]$.

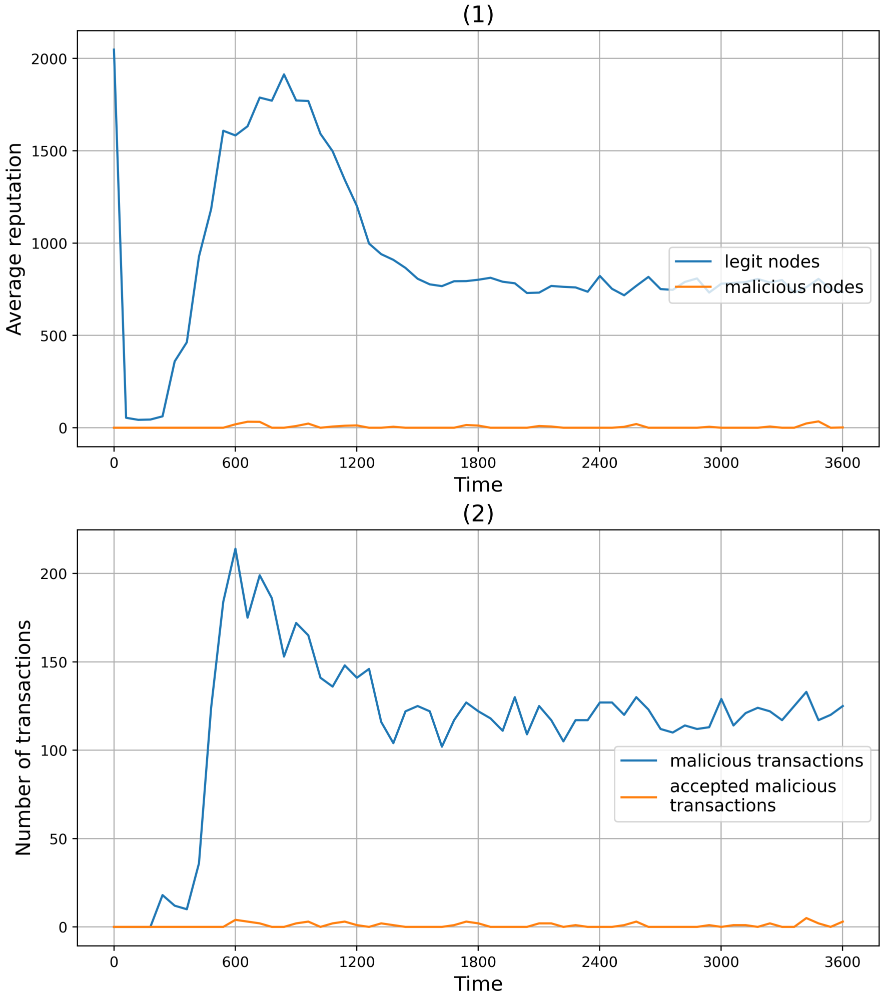


Figure 5.11: The outcome of a copycat attack where 5% of the total nodes are malicious. (1) The average reputation of both attackers and legit nodes over time. (2) The number of transactions that are generated by the attackers and that are accepted by the framework over time.

this attack is to be attributed to the fact that transactions have to be confirmed with another transaction by the detected node. Since the attacker are usually distant from the node they are copycatting, the nodes specified as targets in the forged transactions will not generate the confirmation. In this way, the transactions forged by the attackers will be blacklisted and the attackers will lose reputation consistently. The fact that some malicious transactions are accepted anyway is caused by the fact that some attackers have selected a vehicle that is very close to them as the one to copycat, meaning that the detected nodes can actually detect the attacker, and thus confirm some forged transactions.

5.8.5 *Self-sustaining Malicious Groups*

A self-sustaining group is a set of nodes that exchanges transactions exclusively among themselves. This corresponds to a subgraph in the detection graph generated when the transactions are considered as arcs and nodes as vertices of a graph. A malicious self-sustaining group is a particular type of self-sustaining group where the malicious nodes selectively generates transactions so that legit nodes are excluded, or the malicious nodes are elsewhere, possibly even not on road, and they only fake the interactions with each other in order to lower the quality of traffic information. Legit nodes cannot detect the nodes that form the malicious group, but it is only when the malicious nodes generate fake transactions that the self-sustaining malicious group represents a threat for the framework. In fact, if the attackers are on road and correctly generate transactions, it is not relevant if they are generating transactions selectively, as they are giving valid traffic information anyway. In the case they generate fake transactions, instead, the attackers can, for example, fake a traffic congestion by generating transactions among themselves, while gaining reputation because their transactions seem to be correct as the attackers correctly simulating the interactions. If this attack succeeds consistently, the usability of the framework is compromised, as explained in the threat model described in Section 3.3.

Figure 5.12 shows the attempt of a self-sustaining malicious group of 5 vehicles to fake a traffic congestion. The attack starts after 840 s since the beginning of the simulation with a malicious vehicle that sends fake transactions for three minutes to the members of the group after having behaved properly until that moment. In this way, the vehicles in the malicious group can gain enough reputation to start the attack. After an initial phase where almost all malicious transactions are accepted, the attackers start to lose reputation drastically mainly due to the detection variance based scaling factor and the misbehaviour penalization heuristic, and their average reputation becomes significantly lower than then average reputation of legit nodes.

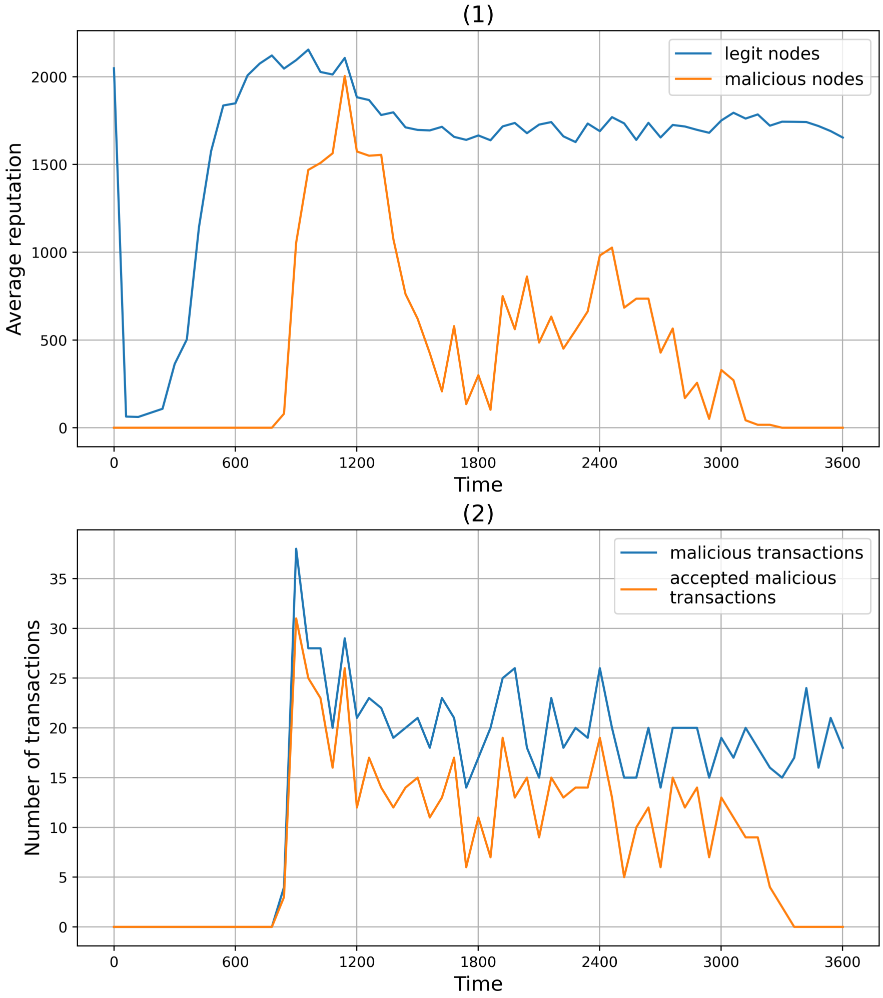


Figure 5.12: The outcome of an attack generated by a self-sustaining malicious group with 5 members that starts at time 840 s. (1) The average reputation of both attackers and legit nodes over time. (2) The number of transactions that are generated by the attackers and that are accepted by the framework over time.

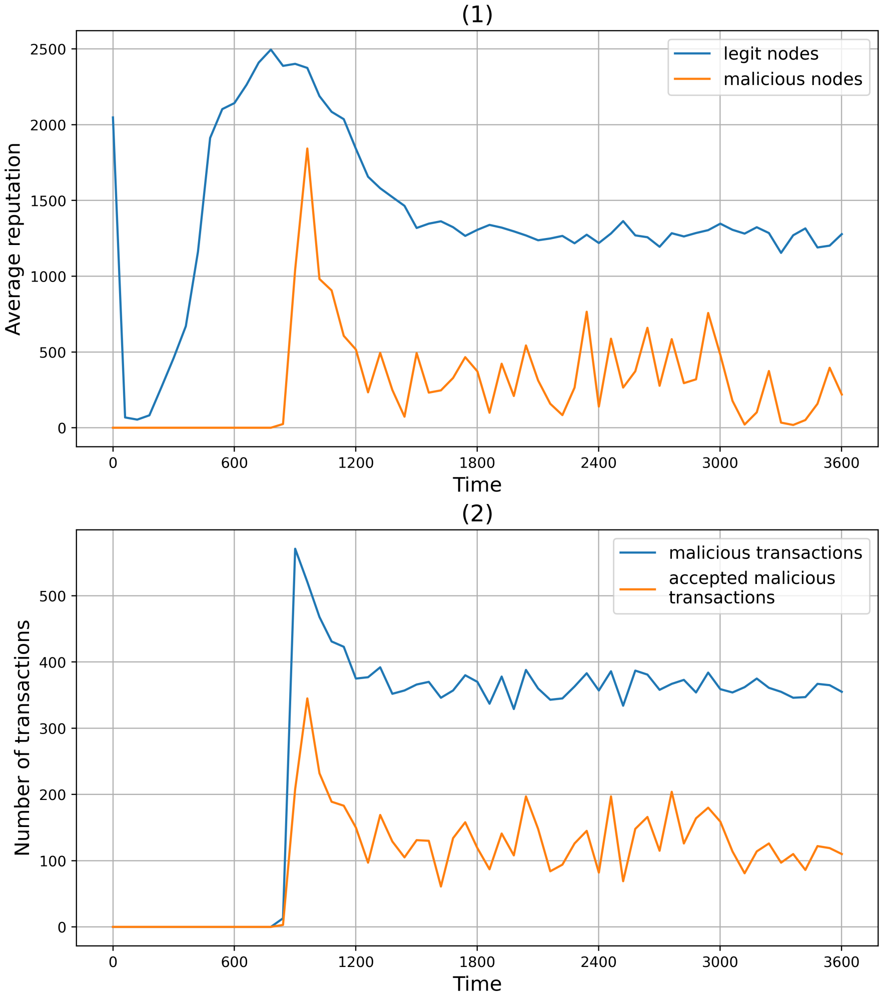


Figure 5.13: The outcome of an attack generated by a self-sustaining malicious group with 20 members that starts at time 840 s. (1) The average reputation of both attackers and legit nodes over time. (2) The number of transactions that are generated by the attackers and that are accepted by the framework over time.

This situation eventually leads to the complete failure of the attack when the attackers lose all their reputation and are thus unable to support each other's' transactions.

A larger scale attack is shown in Figure 5.13, where the size of the group is increased to 20 vehicles. The attack is performed in the same way as in the attack with a group of only 5 vehicles, the only difference being the size of the self-sustaining malicious group. The graphics show a very similar pattern to the smaller scale attack. In fact, the attackers initially gain a lot of reputation and have many transactions that are categorized as acceptable, but as time passes, the attackers lose almost all of their reputation, with moments where their average reputation is almost zero, and the amount of accepted transactions is reduced significantly. This time though, the attack does not completely fail in the time window considered for the simulation, since the larger number of vehicles allows the attackers to better support each other, and thus to counter the mitigations more effectively. Another factor is that the density of legit nodes varies over time and when there are more attackers than legit vehicles in a certain area, the latter generate more transactions and so are more likely to gain reputation and win the conflicts in the misbehaviour penalization heuristic.

6 CONCLUSIONS

6.1 LIMITATIONS AND FUTURE WORK

A major problem encountered during the research carried out for this thesis is the lack of suitable traffic scenarios for the SUMO traffic simulator. Only the MoST scenario was deemed good enough for the experiments, but although the authors set it up carefully, it is not based on a real traffic stream, but it is modeled after mobility data and realistic movement patterns. This fact, however, does not completely invalidate the results obtained in the experiments, since the traffic simulation is realistic anyway. A second and more important limitation related to the traffic scenario is the fact that since only one traffic scenario was available and usable, the configuration found through the experiments may be overfitted. To lower the chance of overfitting, the configuration was tested at different time windows of the MoST scenario, but it is not possible to know how tied to the MoST scenario the configuration is.

The bounds in the transaction validation heuristic used to create the conditions that determine if a previous transaction is concordant or discordant with a new one are rather loose. Because of this, the reliability of the positions of the nodes in the framework is rather low. This means that the sender can still lie about its position if the offset between the advertised position and the real one is small enough. The bounds we adopted in the proposed framework have been chosen because they are intuitive and easy to checks, but it is possible that, with deeper knowledge about transport engineering and the properties of traffic streams, the bounds can be redesigned to give a better yield.

The computational complexity of the validation algorithm has a quadratic dependency on the number of transactions stored in a block. While this fact is not inherently positive or negative, it suggests that the proposed framework may not scale well when the blocks are too big, especially because the proposed framework is executed on IVI systems. The quadratic dependency is determined by the fact that every transaction in the new block has to be compared with all the transactions contained in older blocks or the new blocks itself. Although it is possible to parallelize the critical loops of the validation algorithm, the computational complexity is still quadratic on the size of the block and cannot be reduced without redesigning the validation algorithm completely. An alternative approach for the design of the validation algorithm could be to rely on a statistical approach, where a node only has to check a smaller and fixed portion of the transactions.

This approach could effectively reduce the computational complexity to linear, but it would possibly require the redesign of most of the consensus mechanism and it would also give probabilistic guarantees that need to be carefully proved and enforced.

6.2 SUMMARY

The objective of this thesis was to design a new decentralized framework based on the blockchain data structure that could be used to collect real-time traffic information, and to perform a feasibility study on it. The traffic information is collected in the form of transactions in a blockchain which are generated by vehicles through a GPS module and a short range communication module when they get close enough to other vehicles to generate interactions. The proposed framework aimed at providing a viable alternative to widely used centralized systems that offer similar services like Google Maps and Waze, so it had to be designed considering a realistic environment where malicious entities can try to attack the system to reduce its usability or to lower the quality of the collected traffic information. Due to the impossibility of testing the proposed framework in a real scenario, we also developed a simulator based on OMNeT++, the Veins framework and the SUMO microscopic traffic simulator where the main requirements were to be as realistic as possible and efficient.

The design process of the framework was broken down into few loosely-coupled logical units that were analyzed and modeled individually, while the choices required in the design phase were justified with mathematical arguments or with informal, but precise, explanations. In addition to the trivial functional requirements, the main drivers that regulated the choices made during the design phase were derived from a threat model carefully defined for the specific context of the proposed framework.

While designing the proposed framework, the consensus mechanism was the main focus, as it is the component that directly characterizes the properties of a blockchain based system. The part of the consensus mechanism that defines how distributed consensus is achieved was conceived starting from existing consensus algorithms used in widespread blockchain based systems, while the part relative to the validation algorithm, which enforces the consistency of the content of the blocks, was designed ad-hoc for this application. In order to achieve the desired security properties, we spent most of the effort in analyzing and studying the behaviour of each component of the consensus mechanism that regulates the growth of the blockchain, paying particular attention to group of attackers that try to pass forged blocks off for valid ones or that try to fake traffic congestion though malicious behaviour.

The experimental part of this thesis was further divided in a parameter study and a series of attack scenarios, although all the experiments that we performed relied on the MoST SUMO scenario to have a realistic traffic scenario and realistic interactions between vehicles, and on a base configuration that was found with a trial and error approach. In the parameter study we showed how the parameters affect the performance of the proposed framework by letting one parameter at a time change its value and comparing the results. We then discussed the outcomes, while also comparing them to the expectations and relating them with the components of the framework, and explained why certain values can or cannot be used in the configuration of the proposed framework. With this approach we studied the behaviour of all the main parameters used in the validation algorithm.

Finally, we performed a set of experiments to show that the framework can indeed provide the desired security properties and it can resist to the attacks defined in the threat model. We excluded the attacks for which a mathematical argument about the probability of success was already provided, namely those related to the consensus algorithm, since the feasibility of those attacks was already rigorously discussed. Though the experiments, we showed that the framework is resistant to attacks that try to lower the quality of the traffic information or that try to spoof one's real position, as well as to attacks that try to fake a traffic congestion. We also explained the reasons why the those attacks failed completely or at least partially. With this thesis, we have thus showed that the proposed framework is feasible and since companies are actively investing in the technologies required for its functioning, in a near future it could be used in a real scenario to collect traffic information.

BIBLIOGRAPHY

[1] *"TAPAS Cologne" Scenario*. URL: https://sumo.dlr.de/docs/Data/Scenarios/TAPASCologne.html (visited on 10/09/2020) (cit. on p. 60).

[2] Arati Baliga. "Understanding Blockchain Consensus Models." In: (2017) (cit. on pp. 10, 11).

[3] Dave Barth. *The bright side of sitting in traffic: Crowdsourcing road congestion data*. 2009. URL: https://googleblog.blogspot.com/2009/08/bright-side-of-sitting-in-traffic.html (visited on 10/09/2020) (cit. on p. 7).

[4] Marcin Bernas, Bartłomiej Płaczek, Wojciech Korski, Piotr Loska, Jarosław Smyła, and Piotr Szymała. "A Survey and Comparison of Low-Cost Sensing Technologies for Road Traffic Monitoring." In: *Sensors* 18.10 (Sept. 2018), p. 3243 (cit. on pp. 1, 6).

[5] Stefan Brands and David Chaum. "Distance-Bounding Protocols." In: *Advances in Cryptology — EUROCRYPT '93*. Ed. by Tor Helleseth. Berlin, Heidelberg: Springer Berlin Heidelberg, 1994, pp. 344–359 (cit. on p. 19).

[6] Miguel Castro and Barbara Liskov. "Practical Byzantine Fault Tolerance." In: *Operating Systems Design and Implementation*. 1999 (cit. on pp. 11, 24, 51).

[7] Samsung Electronics Co. *Exynos Auto 8890 Powers IVI System in the New Audi A4*. 2019. URL: https://www.samsung.com/semiconductor/minisite/exynos/newsroom/pressrelease/exynos-auto-8890-powers-ivi-system-in-the-new-audi-a4/ (visited on 10/09/2020) (cit. on pp. 1, 6, 13).

[8] L. Codeca, R. Frank, S. Faye, and T. Engel. "Luxembourg SUMO Traffic (LuST) Scenario: Traffic Demand Evaluation." In: *IEEE Intelligent Transportation Systems Magazine* 9.2 (2017), pp. 52–63 (cit. on p. 60).

[9] L. Codecá and J. Härri. "Towards multimodal mobility simulation of C-ITS: The Monaco SUMO traffic scenario." In: *2017 IEEE Vehicular Networking Conference (VNC)*. 2017, pp. 97–100 (cit. on p. 60).

[10] Phil Daian, Rafael Pass, and Elaine Shi. "Snow White: Robustly Reconfigurable Consensus and Applications to Provably Secure Proof of Stake." In: *Financial Cryptography and Data Security*. Ed. by Ian Goldberg and Tyler Moore. Cham: Springer International Publishing, 2019, pp. 23–41 (cit. on p. 25).

[11] John R. Douceur. “The Sybil Attack.” In: *Peer-to-Peer Systems*. Ed. by Peter Druschel, Frans Kaashoek, and Antony Rowstron. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 251–260 (cit. on p. 23).

[12] Ashutosh Dhar Dwivedi, Gautam Srivastava, Shalini Dhar 4, and Rajani Singh. “A Decentralized Privacy-Preserving Healthcare Blockchain for IoT.” In: *Sensors* 19.1 (Jan. 2019), p. 326 (cit. on p. 8).

[13] *Hyperledger Architecture, Volume 1* (cit. on pp. 8, 12, 24).

[14] J. B. Kenney. “Dedicated Short-Range Communications (DSRC) Standards in the United States.” In: *Proceedings of the IEEE* 99.7 (2011), pp. 1162–1182 (cit. on p. 19).

[15] Sunny King and Scott Nadal. *PPCoin: Peer-to-Peer Crypto-Currency with Proof-of-Stake*. 2012. URL: https://www.peercoin.net/whitepapers/peercoin-paper.pdf (visited on 10/09/2020) (cit. on pp. 11, 24, 43).

[16] Eleftherios Kokoris Kogias, Philipp Jovanovic, Nicolas Gailly, Ismail Khoffi, Linus Gasser, and Bryan Ford. “Enhancing Bitcoin Security and Performance with Strong Consistency via Collective Signing.” In: *25th USENIX Security Symposium*. 2016 (cit. on pp. 12, 24).

[17] Eleftherios Kokoris-Kogias, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Ewa Syta, and Bryan Ford. “OmniLedger: A Secure, Scale-Out, Decentralized Ledger via Sharding.” In: *2018 IEEE Symposium on Security and Privacy (SP)*. 2018 (cit. on p. 20).

[18] Renato Legler. “Analysis of a Distributed Ledger Framework for Automotive Positioning Applications.” Master’s Degree Thesis. Politecnico di Milano, 2019 (cit. on pp. 6, 18, 34).

[19] Satoshi Nakamoto. *Bitcoin: A Peer-to-Peer Electronic Cash System*. 2009. URL: https://bitcoin.org/bitcoin.pdf (visited on 10/09/2020) (cit. on pp. 7, 10, 12, 24, 43).

[20] *NEM Technical Reference*. Feb. 2018 (cit. on pp. 11, 12, 24).

[21] M. Niranjanamurthy1, B. N. Nithya1, and S. Jagannatha. “Analysis of Blockchain technology: pros, cons and SWOT.” In: *Cluster Computing* 22 (2019), pp. 14743–14757 (cit. on p. 9).

[22] Karl O’Dwyer and David Malone. “Bitcoin Mining and its Energy Footprint.” In: *Irish Signals & Systems Conference (ISSC)*. 2014 (cit. on p. 11).

[23] *OMNeT++* (cit. on pp. 2, 47).

[24] Andreas Pfitzmann and Marit Köhntopp. "Anonymity, Unobservability, and Pseudonymity — A Proposal for Terminology." In: *Designing Privacy Enhancing Technologies: International Workshop on Design Issues in Anonymity and Unobservability Berkeley, CA, USA, July 25–26, 2000 Proceedings*. Ed. by Hannes Federrath. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 1–9 (cit. on p. 20).

[25] Deepak Puthal, Nisha Malik, Saraju P. Mohanty, Elias Kougianos, and Gautam Das. "Everything You Wanted to Know About the Blockchain." In: *IEEE Consumer Electronics Magazine* 7.4 (2018) (cit. on p. 9).

[26] Phillip Rogaway and Thomas Shrimpton. "Cryptographic Hash-Function Basics: Definitions, Implications, and Separations for Preimage Resistance, Second-Preimage Resistance, and Collision Resistance." In: *Fast Software Encryption*. Ed. by Bimal Roy and Willi Meier. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 371–388 (cit. on p. 27).

[27] The Internet of Services Foundation. *Internet of Services: The Next-generation, Secure, Highly Scalable Ecosystem for Online Services*. 2017. URL: https://github.com/iost-official/Documents/blob/master/Technical_White_Paper/EN/Tech_white_paper_EN.md (visited on 10/09/2020) (cit. on pp. 11, 20, 24).

[28] *SUMO* (cit. on pp. 2, 47).

[29] Qualcomm Technologies. *Audi of America, Virginia DOT and Qualcomm Announce Initial C-V2X Deployment in Virginia*. 2020. URL: https://www.qualcomm.com/news/releases/2020/01/22/audi-america-virginia-dot-and-qualcomm-announce-initial-c-v2x-deployment (visited on 10/09/2020) (cit. on pp. 1, 6, 14).

[30] *Veins* (cit. on pp. 2, 47).

[31] Volkswagen. *Car2X in the new Golf: A "technological milestone"*. 2020. URL: https://www.volkswagen-newsroom.com/en/stories/car2x-in-the-new-golf-a-technological-milestone-5919 (visited on 10/09/2020) (cit. on pp. 1, 6, 14).

[32] Simon Weckert. *Google Maps Hack*. 2020. URL: http://www.simonweckert.com/googlemapshacks.html (visited on 10/09/2020) (cit. on pp. 7, 20).

[33] Gavin Wood. *Ethereum: a Secure Decentralized Generalized Transaction Ledger*. 2020-09-05. URL: https://ethereum.github.io/yellowpaper/paper.pdf (visited on 10/09/2020) (cit. on pp. 8, 24).