POLITECNICO
DI TORINO

Politecnico di Torino

Master Degree Course in Computer Engineering

Master Degree Thesis

Evaluation of Blockchain Technologies for Vehicular Applications

Supervisors
Carla Fabiana Chiasserini
Paolo Giaccone
Giovanni Malnati

Candidate
Michele Macagno

Academic Year 2017-2018

Abstract

Since the birth of Bitcoin in 2009, the blockchain, an immutable sequence of blocks cryptographically linked to each other, has triggered the interests of people, attracted by this disruptive technology aiming to revolutionize the interactions between people and machines. The blockchain deployment environment is totally distributed and characterized by the absence of trust between the parties and by the absence of a central authority. The use of blockchain technology, originally restricted to the cryptocurrencies, subsequently found a wide range of business applications, including supply chain management, energy market, etc.
This thesis analyses the adoption of a blockchain in the context of ITS (Intelligent Transportation System), as enabling technology to securely and transparently store the messages exchanged between the intelligent vehicles. If the messages exchanged by vehicles are stored, they can be used for many purposes, e.g. to identify the drivers violating traffic rules, to locate stolen vehicles or to simplify the activities of conflict resolution carried out by the insurance companies. The analysis of the vehicular application requirements and blockchain characteristics highlighted some limitations of the blockchain, including the low throughput, high transaction latency and high energy consumption. The demystification of the traditional blockchain technologies, also known as permissionless blockchains, has led to the identification of the permissioned blockchains platform and, particularly of Hyperledger Fabric, as the best candidates for the development of the thesis project. The permissioned blockchains try to overcome some of the limitations of traditional permissionless blockchains but introduce a central authority that constraints the access to network to a limited set of authorized users. The final objective of the thesis is the evaluation of the performance and scalability of the Hyperledger Fabric platform used in the context of a data-intensive vehicular application.

Acknowledgement

I would like to thank my supervisors Carla Chiasserini, Paolo Giaccone and Giovanni Malnati who have believed in me and supported me with patience.
I would also like to thank German Sviridov that helped me during the months of this thesis.
A special thanks to my family that supported me every day of my life.

Contents

Chapter 1

Introduction

1.1 Scenario analysed in the thesis

This thesis deals with a densely populated vehicular environment, characterized by a lot of vehicles exchanging messages each other. These messages are standardized according to the European CAM format (Cooperative Awareness Message) proposed by ETSI (European Telecommunications Standards Institute) and contain much information on the vehicles, such as their geographical coordinates, speed, acceleration and direction. In the next few years, these messages could be regulated by European laws becoming the de facto standard for the vehicular communication and, if stored on common storage units, could be used to track the path of the means of transport, locate stolen vehicles or to identify the drivers violating traffic rules. In the same way, the insurance companies could access the information contained in the CAMs to simplify the conflict resolution process of road accidents. In order to store the messages, every vehicle must be equipped with an USIM 4G (Universal Subscriber Identity Module), so that it can access the Internet by using the services provided by the mobile network operators. The storage of CAMs is however not without problems. By storing the messages on a traditional centralized or distributed database the system administrators could delete or tamper stored information without anyone noticing; on the other hand, if the data were freely accessible, the user privacy might be at risk. In addition, saving all the transmission and reception information of the CAMs, the amount of stored data continuously grows and generates another issue for the management of the system. Fig. 1.1 shows a vehicular environment characterized by vehicles exchanging CAMs each other. The vehicles can receive the CAMs only if they are in the radio range of the transmitter (represented as a circle in the picture). Additionally, each vehicle is connected to the nearest base station of the network operators to store the collected CAMs on a common storage unit.

1.2 Goals of this thesis

This thesis deals with the evaluation of existing blockchain platforms, either permissionless or permissioned, to figure out if this type of technology can be used to solve the above-mentioned problems. The blockchain platforms, in fact, are characterized by the properties of transparency and decentralization that distinguish them from the other data storage solutions. By creating multiple independent blockchains,

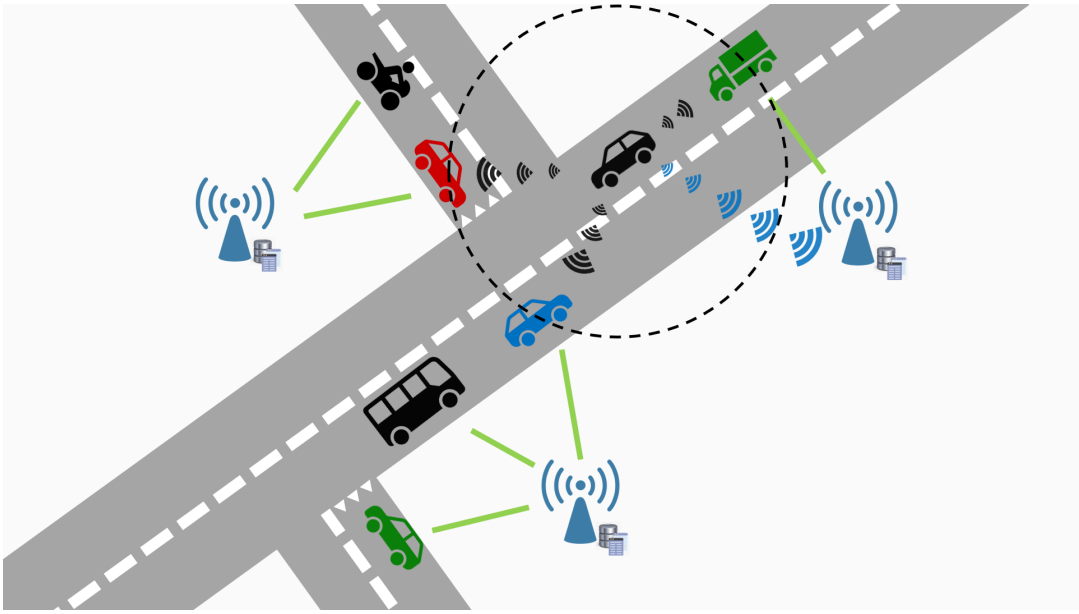


Figure 1.1: Vehicular environment characterized by vehicles exchanging CAMs each other and connected to the base stations to access the Internet.

each of which stores only the messages exchanged between the vehicles travelling in a well-defined geographical area, it is possible to manage to the problem of the huge amount of data that must be stored. The first part of the thesis summarizes the state-of-the-art review, fundamental to evaluate the strengths and weaknesses of the existing blockchain platforms and to identify the one that best suits the requirements of the vehicular application. The next step describes the design decisions of the distributed system, the implementation of the application that must be used to store the CAMs in the blockchain and the vehicular mobility simulator able to generate the message traces exchanged between the vehicles. The final objective of this thesis is the evaluation of performance and scalability of the different blockchain networks used in the context of vehicular applications.

1.3 Content of the thesis

The remainder of this thesis is structured as follows:

- Chapter 2 gives an overview of the blockchain and distributed consensus problem, focusing the analysis on the permissionless blockchain platforms, such as Bitcoin and Ethereum.
- Chapter 3 describes the existing permissioned blockchain platforms, including MultiChain, BigchainDB, Hyperledger Sawtooth and Hyperledger Fabric.
- Chapter 4 concludes the scouting activity with the description of two Distributed Ledger Technologies that are not based on a blockchain, i.e. Iota and Hashgraph.
- Chapter 5 deeply analyses the thesis scenario and describes the architecture of the distributed system designed for the vehicular application.

- Chapter 6 shows some implementation details of the vehicular mobility simulator, explains how to deploy a running Hyperledger Fabric blockchain network and outlines the characteristics of the benchmarking tool used for evaluating the performance of the blockchain.
- Chapter 7 describes the experiments conducted to evaluate the performance and the scalability of the blockchain application.
- Chapter 8 concludes the thesis, summarizing the theoretical and experimental results and describing the future research directions concerning Hyperledger Fabric and the other blockchain platforms.

Chapter 2

Blockchain and Permissionless Blockchains

This chapter introduces the blockchain technology. Sec. 2.1 focuses on the two fundamental categories of blockchain platforms (permissionless and permissioned), comparing them to the already existing storage solutions (centralized and distributed databases); additionally, it analyses the main blockchain data structures (blocks and transactions), the problem of the distributed consensus and the concepts of script and smart contract. Sec. 2.2 describes the users of the Bitcoin network, the transactions and blocks constituting the Bitcoin blockchain and the scripting language. The detailed analysis of the proof-of-work, i.e. the Bitcoin consensus algorithm, highlights the limitations and the possible attacks on the Bitcoin protocol. Sec. 2.3 discusses the key aspects of Ethereum, one of the most widely used smart contract blockchain platform, with particular focus on its proof-of-work consensus algorithm and on the research effort that is necessary to switch to a consensus mechanism based on proof-of-stake. Sec. 2.4 concludes this chapter, outlining the limits and possible countermeasures that can be adopted to address the problems of proof-of-work blockchain platforms.

2.1 Introduction to blockchain

The blockchain is a technology that can be used to safely store information. It is also known as DLT - Distributed Ledger Technology and can be represented as a distributed database composed by a chain of blocks logically linked to each other and containing multiple transactions [1]. The blockchain is an **append-only** data storage solution that is very difficult to tamper and that can be used as an alternative to the traditional databases. Its adoption is recommended when the **integrity of stored data** is threatened by the existence of multiple readers and writers that can write concurrently new data. The most popular application of the blockchain concerns the cryptocurrencies, like Bitcoin. Bitcoin, the first real application based on a blockchain, has been analysed in the context of this thesis to identify the strengths and weaknesses of the blockchain platforms.
The blockchain was born as a totally distributed ledger solution, where users can freely join the network, maintain a full replica of the ledger and execute transactions. No third parties controlling the access to the ledger exist, nobody can prevent the execution of transactions but all participants have the possibility to verify if the

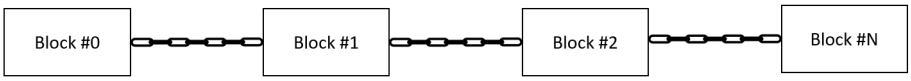


Figure 2.1: A simple representation of a chain of blocks

transactions executed by the other users are reliable or not. These features, very useful in the context of a cryptocurrency, where the total decentralization of the concept of coins permits to transfer money without the interaction with a central authority like a bank, represent a limitation for the applicability of blockchain storage solutions to different scenarios. For this reason, two different types of blockchain, characterized by different network access policies, can be described as follows [2]:

- **Permissionless blockchain**: it is a totally distributed blockchain, where users can join and leave the network without restrictions. No central authorities are involved and no certificates are released to users for their identification [2]. The reliability of stored data is related to the presence of honest users in the network. As long as most of the members is honest, the transactions stored in the blockchain can be considered reliable. Examples of permissionless blockchain solutions are Bitcoin [3] and Ethereum [4].
- **Permissioned blockchain**: it is a totally or partially centralized blockchain characterized by a central authority, or a consortium of authorities that distribute the certificates and the permissions to the users to join the network. The users can have different roles; some of them can only read the data stored in the ledger, others can write new data or interact with the blockchain to execute transactions. Some permissioned blockchain solutions, like Hyperledger Fabric [5], enable cooperation between different companies to create shared ledgers and securely manage data. Examples of permissioned blockchain platforms are Hyperledger Fabric [5] and MultiChain [6]. The permissioned blockchains can be distinguished in two other categories:
  - **Public permissioned blockchain**: users can access stored information without restrictions. The possibility to write new blocks in the blockchain, instead, is reserved to a limited set of trusted participants [2].
  - **Private permissioned blockchain**: only the users authorized by a Trusted Third Party (TTP) can read and write data. In this way, the privacy of users' data is guaranteed. [2]

2.1.1 Alternative storage solutions

It is a good practice to check the characteristics of the relational and non-relational databases before adopting a blockchain in a project because they often best satisfy the application requirements. These alternative storage solutions are described as follows:

- **Relational databases**: they allow to obtain very high throughput in terms of transactions handled per unit of time and provide a full-featured SQL language

for the execution of complex queries on data stored in user-defined tables. A limitation of these systems is the centralization: the storage capabilities and the computational power of the system are concentrated in a single machine [7]. This means that the scalability of the system is obtained by increasing the power of the hardware (vertical scalability) [7].

- **Non-relational** or **NoSQL databases**: they are different from the relational databases because the data is not stored in tables [7]. Every NoSQL database adopts his own storage solution (key-value pairs, document based, graph) and a custom query language [7]. The most important property of these databases is the horizontal scalability that consists in increasing the number of cooperating machines to increase the storage and computational capabilities of the distributed system. Each node of the distributed database maintains a replica of a portion of stored data and only with the cooperation of all the servers it is possible to rebuild the whole dataset [7]

2.1.2 When to use a blockchain

A blockchain is the best alternative to the traditional storage solutions in the event that the applications require the characteristics listed below.

- Disintermediation. A blockchain must be used in a distributed environment, where many users or organizations cooperate to safely store data without the interactions with a central authority [2]. The permissioned blockchain, however, are characterized by the existence of a Trusted Third Party (TTP), an intermediary that does not directly certify the operations but grants the permissions to the authorized blockchain users that cooperates to maintain the state of the ledger [2].

- Append-only log of transactions and data integrity. In a blockchain it is not possible to remove or modify the information previously stored in blocks and it is necessary to store new data at the end of the chain [2].

- Creation of distributed applications. The code of the distributed applications is run by multiple nodes of the network. After the execution of the code associated to every transaction contained in the blocks, each node knows the final state of the ledger [4].

- Transparency. In a permissionless or public permissioned blockchain, everyone can read the data stored in the ledger. The transparency is achieved at the cost of privacy. In the event that the privacy of users' data must be preserved, it is better to use a private permissioned blockchain that restricts the access to data to a limited set of authorized users [2] or, alternatively, a permissionless blockchain based on **zero-knowledge-proof** [2] The zero-knowledge-proof is used in some cryptocurrencies, e.g. ZCash, and permits to preserve the privacy of transactions. The participants of the network, in fact, can verify the validity of the transactions without knowing the addresses and the amount of coins transferred between the parties involved in the transactions [8, 9].

In the case the application requirements include the mentioned characteristics, a blockchain is probably the data storage solution. On the contrary, the use of a blockchain is discouraged by the existence of the drawbacks listed below.

- Low throughput in terms of the number of transactions executed per second. The maximum throughput achieved with a permissioned blockchain is very high if compared to the one of a permissionless blockchain but it is still many orders of magnitude lower than the one of a traditional distributed or centralized database. The throughput values of the blockchain platforms vary from a minimum of 7 transactions per second (tps) managed by Bitcoin [1], to a maximum of about 110000 tps that can be achieved by Hyperledger Fabric with the BFT-SMaRt consensus algorithm [10]. Many other permissioned blockchains, like Ripple and MultiChain, allow to obtain a throughput value of about 1000 tps [11].

- High latency of transactions. When a user executes a transaction, it is necessary to wait some time before it is successfully validated and stored in the ledger. The latency is strictly related to the concept of **finality of transactions** [12]: in a permissionless blockchain, every transaction must be verified, inserted in a new block, the block must be validated and distributed to the other network participants. In addition, before the transaction can be considered as committed, a given number of blocks must be appended to the chain, i.e. the transaction must receive a certain number of confirmations. This number of confirmations is not predefined but, as a rule of thumb, in Bitcoin it is necessary to wait at least 6 confirmations (about 1 hour) before considering as committed a transaction [1]. If this waiting time is not respected, the transactions could be invalidated [12]. The problem of transaction finality is evident in almost every blockchain platform, except for certain permissioned blockchains [12].

- Full data replication. The majority of the blockchain platforms requires every node to maintain a full replica of the stored data, so that the validity of incoming transactions can be determined by simply accessing a local replica of the blockchain, without any further interaction with the other members of the network. However, in this way, when the size of the blockchain grows too much, many users can no longer store the whole replica of the blockchain, i.e. they have to trust the other members of the network, with a subsequent risk of centralization of the decision-making power in the hands of a small subset of users.

2.1.3 Blocks and hash pointers

The term blockchain comes from its base storage unit, the block. The first block of the chain is called *genesis block* [1] and it is the starting point of an always growing linked list of blocks that can be represented as a chain. Each block includes a list of transactions, a timestamp and a hash of the previous block of the chain. The hash of the current block, in fact, is stored as a field of the next block of the chain and works as a hash pointer towards the previous block. This concatenation of blocks provides the tamper-proof property of the blockchain because by modifying the data

stored inside a block, its digest (i.e. hash) varies too. This means that it is sufficient to modify the content of a single block to invalidate all the subsequent blocks of the chain [1]. A graphical representation of a chain of blocks is shown in Fig. 2.2.

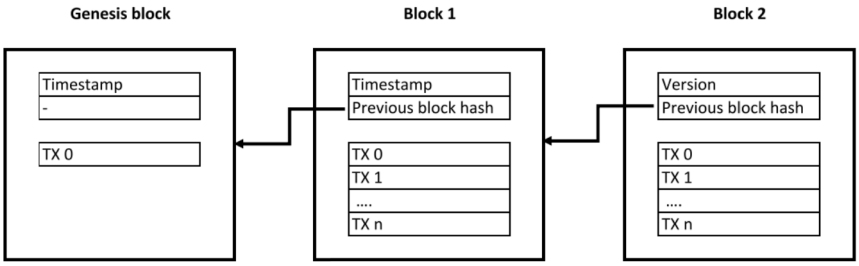


Figure 2.2: Representation of a chain of blocks logically linked by the hash of the previous block.

2.1.4 Transactions

The transactions are combined in blocks and represent the smallest units of data that can be stored in the blockchain. The transactions are not standardized and their content strictly depends on the applications for which the blockchain has been conceived. Many blockchain platforms concern a transaction as a simple transfer of ownership of coins. In this case, the members of the distributed network can verify if a transaction is valid or not verifying if the issuer of the transaction is the current owner of the asset and if he authorized the transfer of the coins by digitally signing the transaction [1]. It is reductive to consider a transaction as a simple mechanism for transferring assets because also general-purpose applications can be built on top of a blockchain storage platform. For this reason, the transactions can be considered as generic read/write operations that triggers the state changes of a shared state database by executing arbitrary pieces of code [2, 11].

Money transfer transactions

As previously mentioned, the simplest transactions are used to transfer the ownership of coins. Each user of the network is associated to a pair of asymmetric keys; the private key is kept secret and is used to sign the transactions in order to authorize the money transfer. If an attacker steals the private key, he obtains full control on the digital money of the victim. The public key, instead, must be spread to the network participants; it is used to verify if the signatures applied by the respective private key are valid, i.e. to verify if the transaction has been authorized by the real owner of the coins. The hash of the public key is computed to derive an **address**, a pseudonym used to receive the payments and to improve the privacy of users. After this brief recall on blockchain terminology, an example showing the transfer of coins from Alice to Bob is given. Fig. 2.3 provides a graphical representation of the process described below.

1. Suppose Alice owns some coins associated to her address. The address was generated by computing the hash of her public key.

2. Bob calculates the hash of his public key to generate an address and shares it with Alice.

3. Alice creates a transaction transferring a given amount of coins to Bob's address. She uses as source address the one mentioned in the first step. She digitally signs the transaction with her private key to demonstrate to be real owner of the transferred coins.

4. Alice distributes her public key to the network participants so that they can verify her digital signature.

5. Bob and all the other peers of the network, using Alice's public key, are now able to verify the correctness of the digital signature calculated by Alice using her private key. This verification demonstrates that Alice intentionally executed the money transfer to Bob. They also verify if the hash of the public key of Alice, the one used to verify the signature, corresponds to the source address of the transaction. If the two matches and the digital signature has been correctly verified, Alice for sure authorized the transfer of her coins.

The transactions are managed differently depending on the blockchain platform that is used. This example only describes the use of asymmetric cryptography for the management of blockchain transactions.

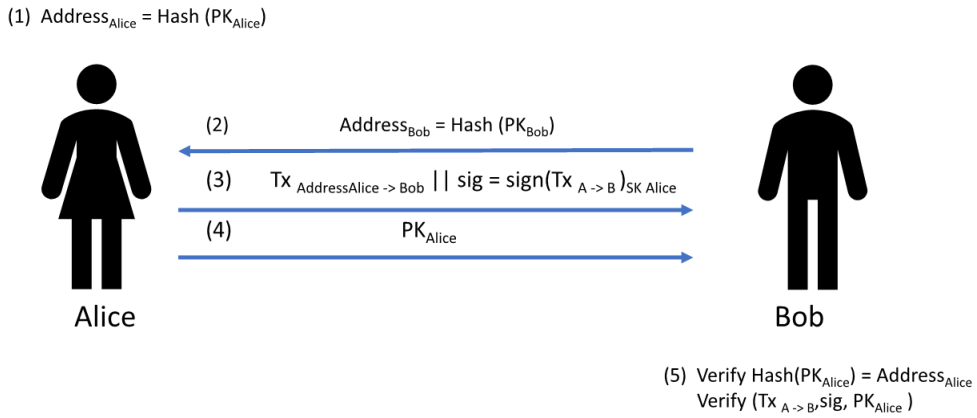


Figure 2.3: Representation of the lifecycle of a transaction.

2.1.5 Consensus algorithms

In a distributed system multiple parties interact each other to maintain an overall coherent state of the ledger. Many users can simultaneously execute transactions or access the ledger for reading and writing. The problem of the **distributed consensus** requires that all the participants agree on the total ordering of the events and maintain a replica of stored data, so that they can verify the validity of the new transactions before appending them to the ledger [11]. The total ordering of transactions is coupled with the **state machine replication**: all the nodes of the network must be aware of the current state of the blockchain (shared state), and of the events occurred over time (state changes) [11]. For this reason, the blockchain

platforms require that all the nodes store a full replica of the ledger. The full data replication is necessary in the permissionless environments because the participants can join or leave the network at any time. By using partial data replication, in fact, if every node storing a replica of the same portion of the dataset goes offline, it would be impossible to recover the final state of the ledger. Partial data replication is instead feasible in a permissioned environment, where the identity of users is well-known. Among all the blockchain platforms analysed in this thesis, only BigchainDB adopts the partial data replication and achieves the storage scalability [13] .
In a distributed system, misbehaving or faulty network nodes have to be managed [11]. The nodes, in fact, can accidentally crash, can be subverted by an attacker or the network communication problems can compromise the interactions between peers [11]. The two main categories of consensus algorithms that have been developed to deal with these faulty nodes can be described as follows:

- **Crash-tolerant** algorithms. They can be used in permissioned environments and manage crashed or unreachable nodes. To reach consensus the subsequent inequality must be respected: $n \geq 2f + 1$, where n is the total number of nodes of the distributed network and f indicates the number of crashed nodes. Examples of crash-tolerant consensus algorithms are Paxos, Raft, Zab [11].

- **Byzantine-fault tolerant** algorithms. They manage crashed, unreachable and intentionally faulty nodes. To reach consensus in a permissioned environment, the inequity $n \geq \frac{2}{3}f + 1$ must be satisfied, where n is the total number of nodes, f the number of misbehaving nodes. Examples of fault-tolerant algorithms are PBFT and BFT-SMaRt [11].

The above-mentioned algorithms can be used in a permissioned environment where the users can be identified. The permissionless blockchain platforms, like Bitcoin and Ethereum, instead, require to use the **Proof-of-Work** (PoW) or **Proof-of-Stake** (PoS) consensus algorithms because they do not require the identification of users. Their main purpose consists in randomly selecting a user to create a new block with a probability that is proportional to the amount of **computational power** or **coins** controlled by the user [1]. Many other permissionless blockchain consensus algorithms exist but all of them are characterized by the same goal of the proof-of-work, avoiding the monopolization of the creation of blocks in the hands of a small subsets of users [1]. To pursue this objective they introduce randomness in the creation of the blocks of the chain, giving the possibility to many network participants to decide which transactions have to be validated and inserted in the blocks [1].

Byzantine Generals Problem

The **Byzantine-fault tolerant** consensus algorithms tolerate faulty nodes. The term Byzantine comes from the paper *The Byzantine Generals Problem* written in 1982 by Lamport Leslie et al. [14]. In this paper, the author describes a war scenario, in which Byzantine soldiers have to decide whether to strike an attack or to retreat. The only way to successfully conclude an assault consists in involving simultaneously the whole army [14]. The existence of multiple generals, some of whom are potentially malicious, requires a mechanism for reaching consensus on the decision

to be taken. The main problem is the existence of traitors between the generals. The paper demonstrates that if at least the 66.6% of the generals are honest, all the soldiers can converge to the same decision [14]. In distributed computing this behaviour can be compared to a group of machines that acts honestly or tries to tamper data. The existence of an asynchronous communication channel requires to work with at least $n \geq 3f + 1$ nodes to find an agreement [12], where n is the total number of nodes of the network and f is the number of faulty nodes. Moreover, the number of messages that must be exchanged between the nodes in order to reach the consensus is proportional to the square of the number of participants of the distributed network ($\Theta(n^2)$, where n is the total number of participants) [14].

2.1.6 Scripting and Smart Contracts

The added value of the blockchain platforms is the possibility to write **scripts**, notably programs whose execution is associated to the validation process of transactions. This means that the scripts are not executed once during the creation of a new block but they are executed every time a blockchain node verifies the transactions before appending the block to the chain. This feature demonstrates that it is possible to write general-purpose applications on top of the blockchain, more complex than the simple cryptocurrencies. The blockchain scripting languages are split in the two categories described as follows:

- **Stack-based** scripting languages. They are characterized by a limited set of instructions and, among other things, do not allow to implement loops; the only mechanism that can be used to repeat a sequence of instructions consists in writing many times the same piece of code. This feature ensures that the scripts are always executed in a finite time interval. The Bitcoin protocol is based on a stack-based scripting language.

- **Turing-complete** scripting languages. They are full-featured programming languages that allow to implement loops. The blockchain platforms based on a Turing-complete scripting language implements a strategy to avoid the infinite loops. The execution of an infinite sequence of instructions overloads the computational capabilities of all the nodes of the blockchain that are running the code of the script. The Turing-complete scripts of Ethereum and of the other blockchain platforms are called **smart contracts**.

2.2 Bitcoin

Bitcoin was created in 2009 by Satoshi Nakamoto and it is the first permissionless blockchain platform ever built [3]. There are many scientific papers covering this topic and, for this reason, the identification of strengths and weaknesses of this blockchain platform is eased. Bitcoin is a totally decentralized cryptocurrency characterized by the absence of a central authority [1] where all the members of the network store a local copy of the ledger containing the list of transactions. Every node of the network, accessing his local copy of the ledger, checks if the users are authorized to spend their coins and verifies that nobody tries to spend twice the same coins [1]. The greatest problem of a totally distributed currency, in fact, is the absence of a central authority acting as guarantor of money transfers. Malicious

users can try to **double spend** the same coins without anyone noticing [3]. Bitcoin was the first distributed ledger technology able to solve this problem in a totally distributed and permissionless environment (Sec. 2.2.7 provides additional details on the double spending attack). The main concepts of Bitcoin are summarized as follows:

- Bitcoin is a peer-to-peer protocol that allows users to exchange coins without the interactions of a trusted third party (TTP).
- Money transfers are represented by transactions. Each transaction indicates who sends the coins, who is the beneficiary and the amount of transferred coins.
- The transactions are contained in the blocks that are cryptographically linked to compose the blockchain.
- Each user of the network stores a local copy of the blockchain. The absence of a central authority requires that the network participants replicate the content of the blockchain so that they can execute read/write operations on the local copy of the ledger.
- The proof-of-work consensus algorithm allows to select randomly a user for creating the new blocks, increasing the reliability of the distributed network.

The remainder of this chapter deeply analyses the Bitcoin protocol by focusing on the transactions, blocks, users and proof-of-work consensus algorithm.

2.2.1 Users of Bitcoin network

Bitcoin is a distributed peer-to-peer network developed to allow users to exchange coins without interacting with a central authority [3]. Everyone can freely access the network by generating a **pair of asymmetric keys** [1]. In particular, each user is identified by means of the following pieces of information:

- **Private key**. Each user secretly stores a private key and uses it to digitally sign the transactions for authorizing the money transfers. The private key must be kept secret because if an attacker controls it he can steal the money of the victim [1].
- **Public key**. It is generated with the private key and is used to verify the correctness of the digital signatures calculated with the relative private key. The signature verification process allows to verify if the money transfer has been executed by the real owner of the coins [1].
- **Address**. It is derived by the public key by sequentially applying two hashing functions (SHA-256 and RIPMED-160) [1]. The Bitcoin protocol uses the addresses to identify the recipients of the money transfers and to avoid the direct exposure of the public keys. In this way the difficulty to track the money transfers is increased. Every address, in fact, creates a sort of pseudonym for the user, making more difficult to date back its identity [1]. To improve the privacy of users, a new pair of asymmetric keys and the relative address are generated for every new transaction.

Each user can control many pairs of asymmetric keys (and addresses) by means of a **wallet**, a piece of software developed to easily manage keys and to execute money transfers on the Bitcoin network [15]. The users are advised to generate a new pair of asymmetric keys and an address every time they need to receive a payment. In this way it is more difficult to date back the identity of the users by reading the history of transactions.

2.2.2 Transactions

The Bitcoin **transactions** allow to transfer the ownership of coins between users. Each transaction is **uniquely identified by its hash** and is composed by many **inputs** and **outputs** [16].

- **Inputs**. The inputs of a transaction refer to the outputs of the other transactions that transferred the ownership of coins to the current owner. [17].
- **Outputs**. The outputs indicate who are the receipts of the transactions (even more than one) and the amount of coins transferred to each of them [17].

Input and output sections of a transaction are composed by many fields.
Each input is composed by:

- **Hash of the previous transaction**. It is used to identify the transaction whose output will be spent in the current transaction.
- **Index**. It is a number that allows to understand which of the outputs of the previous transaction will be spent. Every transaction, in fact, can have multiple outputs [16].
- **scriptSig**. It is a digital signature that proves that who executed the transaction is really the owner of the coins [1, 16].

Each output is composed by:

- **Value**. It indicates the amount of Bitcoin that must be transferred to a given recipient [16].
- **scriptPubKey**. It is a script, i.e. a sequence of instructions that must be executed to certify the money transfer. It contains the address of the recipient of the coins that must be transferred. The widely used script is called P2PKH, Pay-to-PubKeyHash and is used to transfer the coins between different users [1, 16]. More complex behaviours can be obtained using different scripts. [1, 16].

The above-mentioned fields must be replicated for every input and output of the transaction. An output of a transaction that has not yet been referred as input of another transaction is called *Unspent Transaction Output - (UTXO)* [1, 16]. Every UTXO can be spent at most once [1, 16]. Every transaction can combine many inputs and outputs to obtain every possible amount. Three main situations can occur:

1. The sum of inputs is equal to the sum of outputs.

2. The sum of inputs is greater than the sum of outputs. In this case another output is added to the transaction, whose beneficiary is the sender of the coins. This represent the concept of *change* [16].

3. The sum of inputs is lower than the sum of outputs. In this case, other inputs must be added to the transaction [16].

Typically, the sum of inputs is slightly greater than the sum of outputs. The difference is used as reward for miners for their contribution in securing the network (**transaction fee**). For additional details see Sec. 2.2.6. Fig. 2.5 shows three transactions (*TX1, TX2, TX3*). *TX3* has two inputs, the first output of *TX1* and the first output of *TX2*. The *Output 1* of *TX3* is an UTXO because no other transactions refer to it.

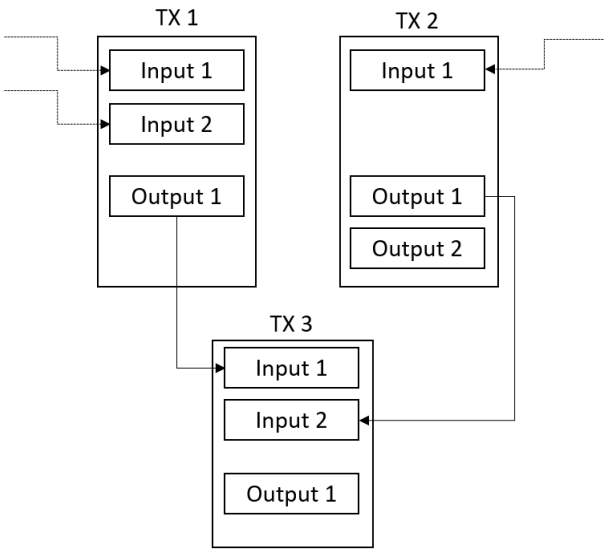


Figure 2.4: Transactions with multiple inputs and outputs.

Dissemination of transactions

After having executed the transactions, the users spread them to the other network participants so that they can create new blocks with these transactions. The transactions, to be effective, must be inserted in a new block, the block must be validated and appended to the blockchain.

2.2.3 Hash and SHA-256

Before describing the blocks, it is necessary to introduce the hash functions that are used to logically link the blocks of the blockchain. A hash function is a function that computes the digest (or hash) of the input data, i.e. generates a fixed-length string that summarizes the content of the input of the function. These functions are non-invertible, i.e. starting from the generated output it is not possible to recover the original text. An ideal hash function generates different outputs for every different input. In reality, calculating the hash of different inputs there is a remote possibility to obtain the same result, i.e. to generate a **collision**. The rate of

occurrence of collisions is very low and is inversely proportional to the length of the digests generated by the hashing function. The fundamental application of the hash functions is the **data integrity** verification. If a user generates and securely stores the hash of a document, he can verify in every moment if the document has been modified. It is sufficient to recompute the digest of the document and to compare it with the original hash. In the event that the two values are different, this means that the document has been modified [18]. SHA-256 is an example of hashing algorithm that generates hashes of 256 bits. It is very important in the context of this thesis because it is used by the Bitcoin protocol [1, 19].

2.2.4 Blocks and blockchain

The block is the main storage unit of a blockchain. It contains and persists data relative to multiple transactions. The blocks create a chain in which every block is related to the previous one by storing the hash of the previous block [1]. This data structure permits to create an append-only, always growing list of blocks and transactions. Blocks are essentially split in two sections, the **header** and the **list of transactions** [20]. The **header** is composed by the following fundamental fields [16, 21]:

- **Version number** of the Bitcoin protocol.
- **Timestamp**. The timestamp refers to the instant in which the block has been created. The list of blocks is chronologically ordered. For this reason, the Bitcoin protocol can be considered a mechanism of distributed timestamping [1].
- **Root of the Merkle tree**. It is a hash (SHA-256) that summarizes the list of transactions of the block. Additional details about Merkle tree are available in Sec. 2.2.5.
- **Previous block hash**. It contains the hash of the header of the previous block of the chain. It acts as a pointer between the current block and the previous block of the chain.
- **Difficulty**. The difficulty, also known as *Bits*, indicates the computational complexity of the cryptographic puzzle necessary to validate the block (additional details on Sec. 2.2.6).
- **Nonce**. It is a number that must be set to satisfy the difficulty of the block (additional details on Sec. 2.2.6).

Every block also contains the **list of validated transactions**, each of them is in turn composed by many fields (Sec. 2.2.2). Each block is uniquely **identified by the hash of its header**, calculated with the SHA-256 algorithm. The hash is not stored inside the block itself but in the next one. Furthermore, it contains a reference to the transactions of the blocks, given by the root of the Merkle tree contained in the header of the block.
The time necessary to read the data stored in the blockchain is proportional to the length of the chain. An indexing mechanism based on LevelDB has been introduced

in Bitcoin to speed up the read operations [1]. Fig. 2.5 shows a graphical representation of a blockchain; the first block of the chain is called genesis block. Except the first block, the other blocks always contain a hash pointer to the previous block of the chain (*Previous block hash*). The blocks do not contain the hash of the current block.

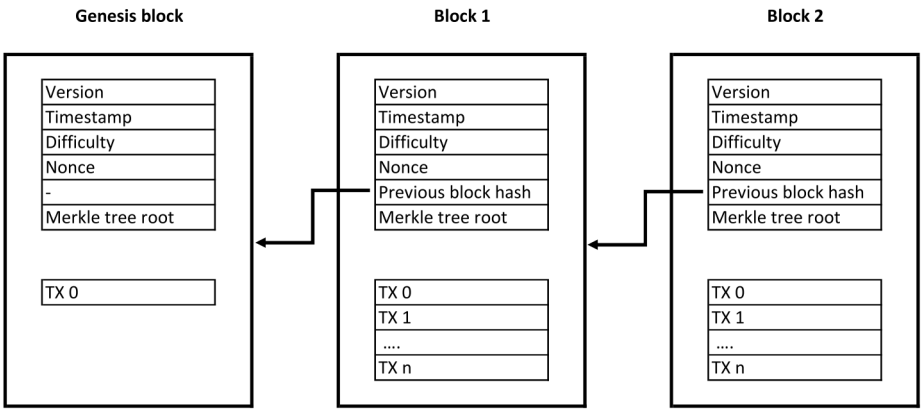


Figure 2.5: Graphical representation of the chain of blocks of Bitcoin.

2.2.5 The Merkle tree

The Bitcoin protocol requires that all peers store a full replica of the ledger, introducing additional problems for all the users' devices characterized by limited storage capabilities. To overcome this problem, the Merkle Tree data structure has been introduced. It allows to summarize the content of multiple transactions inside a single hash value, the Merkle tree root. The generation of the Merkle tree consists in the subsequent applications of the hash function (SHA-256 in the case of Bitcoin). Fig. 2.6 shows an example of Merkle tree of the Bitcoin protocol (SHA-256 algorithm). Once the hashes of all the transactions have been calculated, the hashes must be grouped in pair of two, concatenated and used again as input of the hash function. This behaviour is repeated until only one hash value remains; it is called **Merkle tree root** [1]. The generated data structure is a **binary tree** in which every parent node is simply the hash of the concatenation of the two children.

Fig. 2.6 describes the generation of a Markle tree. Tx0, Tx1, Tx2 and Tx3 are the four transactions of the block. Hash0, Hash1, Hash2, Hash3 are the hashes of the four transactions:

$$Hash0 = hash(Tx0)$$

$$Hash1 = hash(Tx1)$$

$$Hash2 = hash(Tx2)$$

$$Hash3 = hash(Tx3)$$

When the hashes has been computed, the hash of their concatenation must be calculated again:

$$Hash01 = hash(Hash0|Hash1)$$

$$Hash23 = hash(Hash2|Hash3)$$

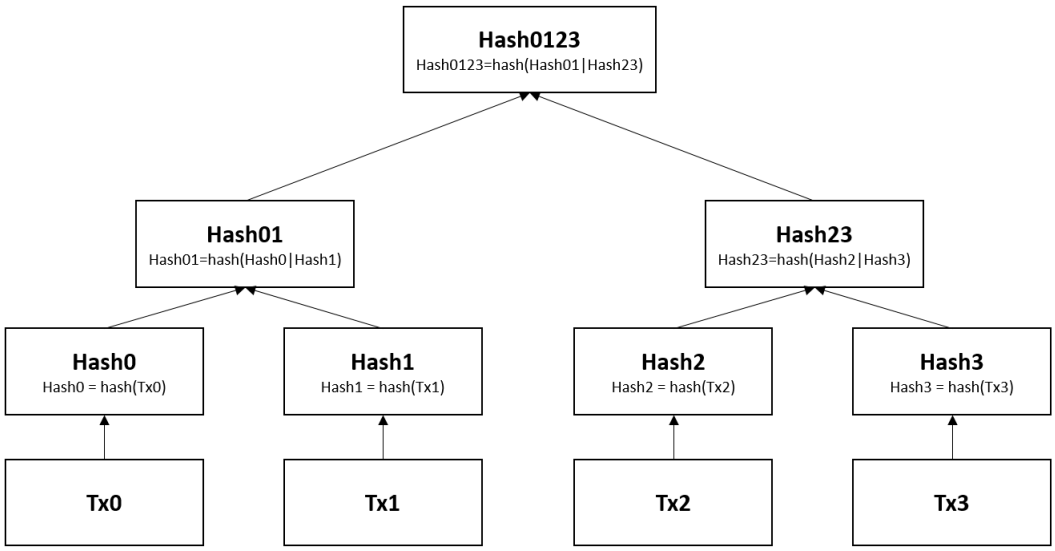


Figure 2.6: Graphical representation of a Merkle tree [22]

The process is concluded when only one value remains (Merkle tree root):

$$Hash0123 = hash(Hash01|Hash23)$$

The Merkle tree root is stored in the block header and represents a snapshot of all the transactions contained in the block. It is very important as it allows to divide the nodes of the network in two categories:

- **Full nodes**: they store a local complete replica of the blockchain. The full nodes can easily verify the reliability of transactions by accessing their own local replica of the ledger [1]. At time of writing, the size of the Bitcoin blockchain is about 178 GB [23] and constantly increases as a result of the creation of new blocks.
- **Lightweight nodes**: they store only the header of the blocks and not the whole set of transactions. To verify the transactions, the lightweight nodes implement the SPV (Simplified Payment Verification). They download a portion of the Merkle tree from the other peers of the network and verify the correctness of the transactions with the help of the Merkle root that is stored in the header of the blocks [1]. The lightweight clients are less secure than the full nodes. For this reason, this type of clients is recommended only for the devices with limited storage capabilities (e.g. mobile devices) and for the validation of transactions transferring a limited amount of coins [1].

2.2.6 Bitcoin consensus algorithm

Bitcoin is a permissionless blockchain whose users can join and leave the network as needed. The validation of blocks and transactions contained in it is executed by the so-called miners without the interaction with a trusted third party. The **miners** are the entities of the distributed network that validates the transactions and blocks. They collect and verify the correctness of the transactions executed by the users of the network. The **validation of transactions** consists in the execution of the

associated scripts and verification of the absence of double spending attempts [16]. In the event that both requirements are satisfied, the transactions can be considered valid and inserted in a new block.

Composition of new blocks

The miners create the new blocks to append to the blockchain. The first transaction of each block is the **coinbase transaction** or **generation transaction** [1]. It is a particular type of transaction that transfers a predefined amount of Bitcoin to the miner of the block as a reward for its job. Unlike other transactions, it does not spend an UTXO but it creates new Bitcoins. The Bitcoin genesis block contains only a coinbase transaction that distributed the first 50 Bitcoins to a predefined user. The reward for the subsequent mined blocks depends on the number of blocks already present in the chain (i.e. block height) [1, 16]. The following formula can be used to compute the reward (r) of the block with block height h_b [16]:

$$r = \frac{50\text{BTC}}{\lceil \frac{h_b}{210000} \rceil}$$

The reward for the mined blocks is halved every 210000 blocks and makes the Bitcoin a deflationary coin, introducing a limit on the maximum circulating supply [16]. The minimum reward for the mined blocks cannot be lower than 10^{-8} Bitcoin (1 Satoshi). When this limit will be exceeded, the coinbase transactions will no longer provide a reward for the mined blocks. For this reason, the transaction fees have been introduced as additional reward mechanism for the miners. As introduced in Sec. 2.2.2, the **transaction fee** is equal to the difference between the sum of Bitcoins referenced by the inputs of the transaction and the sum of the Bitcoins referenced by the outputs of the transaction [1] The miners can freely choose the transactions to insert into a block and they typically choose those transactions providing the highest fees, in order to earn much money. To validate the blocks and to obtain the relative reward, the miners compose the blocks and solve a complex cryptographic problem called **proof-of-work**.

Proof-of-Work consensus algorithm

The proof-of-work (PoW) is the consensus algorithm implemented by the Bitcoin protocol. It tolerates the crashed and unreachable nodes and faulty nodes subverted by an attacker. The Bitcoin proof-of-work is similar to Hashcash algorithm [1] and is executed by miners of the network for the validation of the new blocks. It consists in varying the **nonce** stored in the header of the block, computing the SHA-256 hash of the header of the block and verifying if the result is numerically lower than a given threshold (the *difficulty* of the block) [1]. The procedure must be repeated until the last condition is verified. When a miner finds a valid solution for a block, it spreads the new mined block to other network peers so that they can verify the transactions contained in it and the solution to the proof-of-work. If everything is correct they append the new block to their own replica of the blockchain, otherwise they discard the block. The time required to find a valid solution to the proof of work is known as **block interval** and depends on two parameters:

- the difficulty of the block (i.e. the maximum acceptable hash value) [1]

- the hashing power controlled by the miners of the network [1].

The difficulty of each block is stored in its header and is changed whenever 2016 blocks are mined. The Bitcoin protocol maintains constant at 10 minutes the block interval: a shorter block interval implies a larger number of minted coins and a depreciation of the currency. Since the computational capabilities of the devices used for mining increases over time, to maintain constant the block interval it is necessary to increase the difficulty of the blocks [1]. The proof-of-work introduces a sort of **randomization in the creation of new blocks**: miners of the network must solve a complex computationally expensive cryptographic problem such that their blocks are accepted by the other peers of the network. The probability for miners to create the subsequent block of the chain is proportional to the hashing power they control. The higher the computational power, the higher the probability of becoming the miner of the block [1]. The proof-of-work solves the problem of the Sybil attack in the Bitcoin network. The Sybil attack simply consists in creating false identities trying to obtain the voting majority of a distributed network [1]. In proof-of-work, the voting power of a user is given by its hashing power. The computational power of a user is determined by the characteristics of the devices under its control and not by the number of fake identities he creates [1].

The problem of forks

Mining is an asynchronous operation; different miners work autonomously and compete to find the solution to the proof-of-work. In some cases many miners find the solution to the proof-of-work of different blocks with the same height [1]. This implies that many blocks refer to the same parent block. These two blocks potentially contain conflicting transactions (e.g. transactions spending the same UTXO). This behaviour describes the creation of a bifurcation of the main chain that is called **fork** [1]. Since the bifurcations cannot exist in the blockchain, the miners create the subsequent blocks and decide to which parent block they must be connected to. At a given point in time, the peers will see two different branches with a different length: only the transactions belonging to the longest chain are considered valid. The blocks belonging to the shorter branches are called **orphaned** and their transactions do not longer affect the final state of the ledger [1]. Since the occurrence of forks can invalidate the transactions, it is necessary to wait some time before considering a transaction as committed. The **number of confirmations** is the parameter that must be controlled to quantify the risk of a transaction of being invalidated by a fork. This number is defined as the number of blocks starting from the block containing the transaction up to the end of the chain [1, 16]. As a rule of thumb in Bitcoin it is recommended to wait a number of confirmations not lower than 6 [1]. The recommended number of confirmations is proportional to the amount of coins transferred [1, 16]. This property describes the uncertainty of the validity of transactions, called **absence of finality of transactions** [12]. Many blockchain platforms, especially the permissionless ones, are affected by this characteristic. Fig. 2.7 shows a graphical representation of a chain of blocks in which two forks occurred. Each square represents a block of the chain, the black one is the genesis block, the grey squares compose the main chain, while the orange squares are the orphaned blocks. The blocks belonging to the different branches are characterized by the same block height because they are connected to the same parent block (i.e. same *Previous*

block hash).

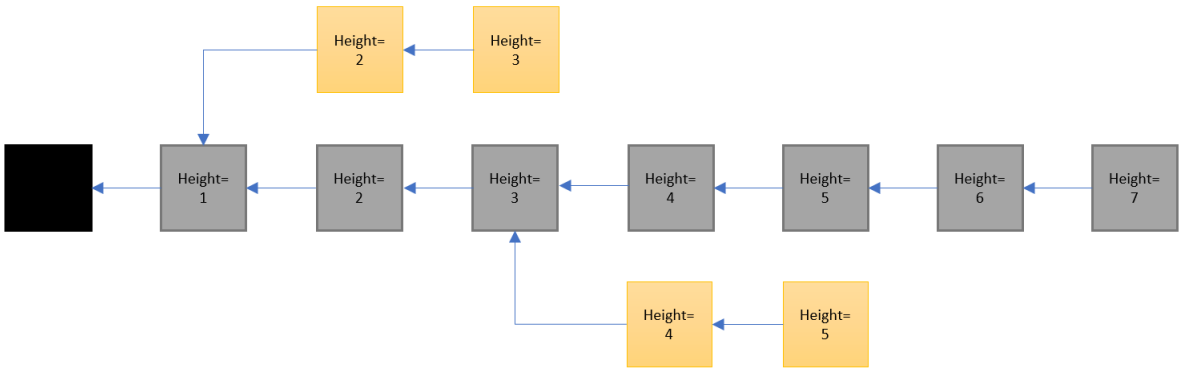


Figure 2.7: Graphical representation of a chain in which two forks occurred.

Solo mining and mining pools

The miners compete to find the solution to the proof-of-work before the others. Only the miner that creates a block that is appended to the main chain obtains the reward. The miners can decide to cooperate or not with the other users to find the solution of the proof-of-work. For this reason, the miners can be subdivided in two categories:

- **Solo mining** is the term used to indicate a device or a set of devices controlled by a single person. This person tries to find the solution to the proof-of-work without cooperating with other users. The probability of mining a block before the other users is very low and is proportional to the computational power controlled by the user [1].
- **Mining pool** is the term used to indicate a group of miners that cooperate to find the solution to the proof-of-work for the new blocks. As soon as one of the miner of the pool finds the solution of a block, the reward is split between all the participants of the pool, accordingly to the amount of computational power provided during this operation [1].

Today, if a user wants to start the mining operations, it is more profitable to join a mining pool; the total computational power of the Bitcoin network is too high if compared to the one that can be provided by a single machine. The risk of solo mining consists in waiting a very long time, even infinite, before earning any amount of money.

Devices and time required to compute the proof of work

The miners are also distinguished by the type of devices used for the mining operations. The time required to compute the proof-of-work for a block depends on the hashing power of the devices. Three main categories of devices are used to compute the proof-of-work:

- CPU

- GPU
- ASIC

The CPU mining is the most traditional technique and consists in exploiting the computational power of the cores of the processors to compute the hashes. The GPU mining allows to obtain better performances than CPU mining because every GPU is composed by many cores. The computation of the hashes, in fact, is an operation that can be easily parallelized and executed by the video cards. Today, the most widely used alternative to the CPUs and GPUs is called ASIC (Application Specific Integrated Circuit). The ASICs are special-purpose devices that are optimized to quickly compute the SHA-256 hashes and to find the solution to the proof-of-work [1]. The massive adoption of ASICs in mining factories makes nearly impossible to earn money with CPU and GPU mining. In addition, the concentration of the mining power jeopardizes the security of the Bitcoin protocol because concentrates the mining power and the capability of creating blocks in the hands of a small subset of miners.

Problems of the proof-of-work

The proof-of-work is an interesting solution to the problem of the distributed consensus but it is affected by some important drawbacks:

- **Low throughput**. The number of transactions per second that can be committed is very low. The Bitcoin throughput varies from 4 to 10 transactions per second, with an average value of **7 transactions per second** [1, 2].
- **High latency**. It is necessary to wait a given number of confirmations before considering a transaction as committed. In Bitcoin it is suggested to wait at least 6 confirmations, i.e. about one hour [1, 12].
- **Absence of finality of transactions**. It indicates that the occurrence of a fork can invalidate the transactions. This characteristic is strictly related to the high latency of transactions [12].
- **Concentration of the mining power in the hands of an attacker**. The attackers can monopolize the creation of blocks if they are able to concentrate more than the 51% of the total computational power of the network [1]. Even with a lower percentage of computing power a successful attack can be conducted (25% could be enough) [12].

Modification to the Bitcoin consensus algorithm

The Bitcoin consensus algorithm can be modified and many improvements can be introduced over time. However, differently from a centralized system in which the administrators can take technical decisions, in Bitcoin and in other permissionless blockchains the majority of the participants must agree on the updates of the consensus algorithm [16]. The **hard fork** is a mechanism that allows to create a bifurcation of the main chain introducing some modifications to the consensus algorithm. For example, with a hard fork it is possible to modify the algorithms used to compute the hashes or the digital signatures. Unlike the traditional forks described in Sec. 2.2.6,

the hard forks generate two coexisting chains that are unable to communicate. Both chains grow independently and do not invalidate the content of the other chain [16]. In addition, the hard forks require to update the software; the users can install a piece of software that is compatible with the older version of the protocol or can install a new version of the program that is able to interact with the new branch of the chain [16]. Depending on the version of the software installed by the users, the latter can interact only with the supported branch. The blocks belonging to the other branches are considered as invalid and automatically discarded. For this reason, in the case an hard fork occurs multiple bifurcations can coexist [16].
A **soft fork** occurs in correspondence of a simple modification to the consensus algorithm that is backward compatible with the older version of the client software [16]. The old and the new versions of the software can coexist and cooperate working on the same chain of blocks. In particular, all the operations executed by the new version of the software are always interpreted and accepted by the older versions of the software. The reverse might not be guaranteed because the soft forks introduce restrictions to the consensus algorithm [16]. The term fork may be inappropriate in case of a soft fork because no bifurcations are created; a soft fork, in fact, is only a slight modification to the consensus algorithm [16].

2.2.7 Possible attacks

Many different threats affect Bitcoin and the other proof-of-work based blockchains. The most important ones can be described as follows:

- **Double spending**. The double spending attack is very popular in the permissionless blockchains and consists in an attempt of spending twice the same coins. The Bitcoin consensus algorithm aims to solve this problem without the need of interacting with a trusted authority that verifies the reliability of the money transfers. The miners, in fact, cannot include transactions that try to spend twice the same amount of coins because the other peers of the network detect the cheating attempt and discard the block [1]. Even if this problem seems to be completely solved, forks can occur and the 51% attack can be put in place.

- **51% attack.** The 51% attack is possible if an attacker controls more than a half of the computational power of the whole network and if it is able to generate a secondary chain, i.e. a fork of the main chain. In this case he can double spend the same coins. The main attack process is explained with an example, where Alice is the attacker and Bob the victim. [1]:

  - Alice, the attacker, buys some products sold online by Bob, the victim.
  - Bob accepts payments in Bitcoin and Alice creates a transaction that transfers the requested amount to Bob.
  - The Alice's transaction addressed to Bob is spread across the Bitcoin network and reaches the honest miners (all the miners that are not directly controlled by the attacker can be considered as honest).
  - Honest miners receive and validate the transaction transferring the coins to the victim. They also insert the transaction in a new block.

  - Bob sees that the transaction has been inserted in a block, he eventually waits a given number of confirmations and ships the product to Alice.
  - Alice, in the meanwhile, creates another conflicting transaction spending the same UTXO of the transaction addressed to Bob. This second transaction is a double spending transaction and is addressed to another destination address that is under the control of Alice. Instead of spreading this transaction to the whole Bitcoin network Alice sends it to the miners under her control.
  - The miners controlled by Alice create a fork of the main chain, they generate a new block containing only the double spending transaction and discard the transaction addressed to Bob. They continue the mining process and append many blocks to the secondary chain, increasing the number of confirmations of the double spending transaction addressed to Alice.
  - When the secondary chain is grown enough, i.e. it is longer than the main chain, Bob ships the product to Alice and the miners under the control of Alice spread the blocks of the secondary chain to the honest peers of the network.
  - Since the chain generated by the attackers is longer than the chain created by the honest miners, the former becomes the main chain and the transaction addressed to Bob is invalidated. Bob loses either the money and the product.

  The only safety measure that the victim can introduce consists in waiting a greater number of confirmations for the transaction before shipping the products. This countermeasure can be insufficient if the attacker controls the majority of the mining power of the whole network [1]. The larger the amount of computational power controlled by the attacker, the higher the probability of successfully conduct an attack. Recent research activities demonstrated that the proof-of-work consensus algorithm is susceptible to attacks when the 25% of the total computational power is concentrated in the hands of a single entity [12].

- **Infinite delay of transactions.** The miners have the possibility to delay, even forever, the validation of some transactions avoiding to insert them in new blocks. This type of attack can be considered as a denial of service attack because prevents transactions from being committed [16]. The delay of transactions is problematic if an attacker controls the majority of the mining power of the network and is able to create all the blocks of the main chain; in such a situation, the honest miners of the network are not able to create the blocks inserting the transactions that the attacker intentionally discarded.

The Bitcoin consensus protocol does not permit to spend the money of other users unless the private key of the owner of the coins is stolen.

2.2.8 Bitcoin scripting

As mentioned in Sec. 2.2.2, the Bitcoin protocol allows to implement scripts. The most popular script is the Pay-to-PubKeyHash that is used to transfer money be-

tween users [1]. The Bitcoin scripting language is **stack-based**: the instructions are managed by a stack and executed in a well-defined order, without the possibility to repeat sequences of instructions (no loops) [1, 16]. The presence of infinite loops could compromise the functioning of the network because the nodes involved in the execution of an infinite loop cannot validate other transactions [1]. Since the Bitcoin scripting language is deterministic, all the users obtain the same result regarding the execution of the scripts. Thus, they can draw the same conclusions on the validity or invalidity of the transactions.

2.3 Ethereum

Ethereum is another example of permissionless blockchain based on proof-of-work consensus algorithm. It presents many differences and similarities from Bitcoin, and it is based on a totally independent blockchain. The main goal of Ethereum consists in giving the possibility to users to write **smart contracts** for the development of **DAOs** - Decentralized Autonomous Organizations [4]. These two interesting concepts guided the research activity to deeply analyse this technology.

2.3.1 Transactions, Blocks, State

Exactly as explained for Bitcoin in Sec. 2.2.2 and Sec. 2.2.4, Ethereum also introduces the concept of transactions and blocks. The transactions execution is triggered by the users with a mechanism similar to the Bitcoin one, based on a digital signature scheme: a user creates a transaction attaching a digital signature to prove his identity [4]. The transactions are then validated by the miners and collected into blocks logically linked to each other to create a chain. The Ethereum whitepaper emphasizes the concept of **state**: at any given point in time, the whole blockchain system is characterized by the current state, that is changed accordingly to previously executed transactions [4]. The current state allows to have a complete view on the amount of coins owned by users, on stored information and on pieces of software that can be run with the execution of transactions. All this information is associated to participants and collected in **accounts**. There exist two types of account [4]:

- **Externally owned account**. It is based on the digital signature scheme and can be used to transfer ownership of coins and to carry some text.
- **Contract account**. It is more complex than externally owned account because it is associated to a piece of code that must be executed, a smart contract. Each contract is associated to a local storage solution based on key-value pairs, allowing the management of the state of the account and a faster data retrieval.

An interesting feature of accounts consists in the possibility to exchange **messages** between them. An intercommunication mechanism for accounts is this way implemented and complex behaviours can be obtained [4].

2.3.2 Ethereum smart contracts

A **smart contract** is defined as a piece of software that is run by the blockchain nodes and that can contain arbitrary instructions. With a smart contract it is also

possible to develop an oracle, an application able to influence its execution depending on data stored outside of the blockchain, that can be obtained, for example, contacting an external data source, like a website [4]. The Ethereum smart contracts are more powerful than Bitcoin scripts because they are based on a **Turing-complete programming language**, giving the possibility to execute any kind of operations, including instructions that modify the execution flow of instructions, e.g. loops [4]. The possibility to repeat portions of code introduces the risk of creating **infinite loops**. Since the code execution is performed by every node of the network, this behaviour can be very dangerous because it can overload the computational capacity of the nodes [4]. In such a situation, other transactions can no longer be executed because the nodes of the network are managing an infinite sequence of instructions. Ethereum introduces the concept of **gas** to overcome this problem. Unlike Bitcoin, every Ethereum transaction presents two additional fields, **STARTGAS** and **GASPRICE**, indicating respectively how many instructions will be potentially executed by the code triggered by the transaction itself, and the price that must be paid for one instruction [4]. Every transaction, including simple payments, transfer of data or the execution of more complex smart contracts, requires the payment of gas, in other words, of fees. Considering the smart contract execution, in the case the number of computational steps exceeds the expected one (i.e. the STARTGAS), the gas is totally consumed. Since the execution of code has not yet been completed, a rollback procedure is started. All the modifications applied to the state during the execution of the code are discarded and the previous state of the contract is restored; no intermediate states are possible. As an additional countermeasure, the issuer of the transaction cannot receive back the amount of coins associated to the spent gas. This characteristic is fundamental to discourage malicious users from executing denial-of-service attacks: to conduct an effective attack overloading the computational capacity of nodes, in fact, a large amount of coins must be spent.

DAO - Decentralized Autonomous Organization

The main purpose of Ethereum platform is the creation of DAOs, Decentralized Autonomous Organizations, controlled by the execution of smart contracts. People may interact to execute complex tasks without the intervention of a central authority [2, 4]. People are grouped in organizations where decisions are taken by the majority of members. The public Ethereum blockchain allows to enforce the trust between unknown participants and preserves the correct execution of the smart contracts. In a DAO, a lot of services can be required by customers and paid with cryptocurrencies. A DAO can also be associated to a mutual fund that can be controlled only by the majority of votes of participants [2]. Many proposals are available in [24] as projects that can be built on top of Ethereum blockchain. One example out of many can be the trading of electricity executed directly between customers producing it (for example using photovoltaic panels), and users interested in buying it. Different policies and costs can be decided by customers without the need of mediation of power companies [24].

2.3.3 Ethereum consensus mechanism

The consensus mechanism underpinning Ethereum is based on proof of work. Similarly to Bitcoin, it requires the solution of cryptographic problems. The algorithm

adopted by Ethereum is called **Ethash** [25, 26] and presents many differences from Bitcoin Hashcash. One of the fundamental differences is the amount of main memory required to run this algorithm. To mine new blocks, in fact, it is necessary to reserve at least **1 GB of main memory** to store a complex graph [25]. The latter is generated starting from a set of information of the whole chain and must be recomputed after a well-known number of blocks is mined (30000 blocks) [4]. This characteristic allows to classify Ethash a **memory-hard** consensus algorithm [25]. The mining algorithm for a block can be summarized with the following steps:

1. Generation of a **seed**. The seed is a number that depends on the headers of previous blocks of the chain [25].
2. Generation of a **cache**. It depends on the value of the seed and is characterized by low storage requirements [25].
3. Generation of a **dataset**. It is derived from the cache and is characterized by high storage requirements [25].
4. Aggregation of random parts of the dataset for the computation of the hashes [25]. These steps must be repeated as soon a valid hash value is found.

To correctly execute mining operations, a miner needs to store the whole *dataset*, i.e. it must be a full node, while the verification of transactions and blocks can be executed only maintaining a replica of the cache [25]. The high memory requirement, coupled with the high storage requirements for mining new blocks, weakens the massive usage of ASIC devices for blocks mining, limiting the concentration of computational power typical of Bitcoin [25]. This can be considered a good improvement for the security of the network, making more difficult to strike a 51% attack.

Ethereum proof of work is different from Bitcoin. In Bitcoin proof of work miners need to know only the header of the previous block and the valid transactions of current block. A miner is not required to be also a full node, especially if it belongs to a mining pool. In this case a Bitcoin miner simply receives an already composed block and varies a nonce to compute sequentially the hashes to find a valid one. This type of operation can be easily distributed among a multitude of nodes, even with low storage and memory capabilities, increasing the possibility to create malicious botnets, a set of devices subverted by an attacker whose job is computing a large number of hashes to mine new blocks.

2.3.4 Scalability

Exactly as many solutions based on proof of work, the scalability of Ethereum blockchain is limited. The maximum reached throughput is about **13 transactions per second** [27] with new blocks generated every 12 seconds (block interval) [26]. Throughput value is very low if compared to other storage solutions like centralized databases. This limit the fields of applicability of Ethereum blockchain.

Talking about storage consumptions, the actual length of Ethereum blockchain is about 315 GB [23]. Considering the Ethereum blockchain is born in the half of 2015 [27], the growth rate is higher with respect to Bitcoin blockchain.

2.3.5 Proof of Stake

The Ethereum community is working on the possibility to switch from proof-of-work to **proof-of-stake (PoS)** consensus algorithm. Proof of stake is an alternative consensus algorithm that can be adopted in byzantine environment like a permissionless blockchain. Actually there exists alternative coins adopting proof of stake algorithm, the most famous one is Peercoin [28]. Other implementations of proof of stake are available, like the Delegated Proof of Stake of Bitshares [27], that promises to reach higher throughput values with respect to proof of work.

The proof of stake implementation proposed by Peercoin is based on the concept of *coin age*. The coin age is a numeric value that indicates how much coins a user owns and from how much time (days) are hold. The higher the coin age, the higher the probability of becoming the next miner of the block [1]. When a miner is chosen to create the new block, he obtains a reward, but at the same time his coin age is reset. In this way, for the next mined block, another miner will be potentially chosen as candidate for the generation of blocks [1]. Over a long period of time, even miners possessing a small amount of coins will be possibly selected as miners. The computation of the coin age is made practicable by the existence of timing information stored in blocks and transactions. The main objective of this technique consists in avoiding long computationally expensive sequences of cryptographic operations. The computation of hashes of the block still remain but the difficulty is significantly reduced depending on the coin age of miners, in order to facilitate the mining operations [28]. Of course, even in this scenario forks can occur. Instead of choosing the longest branch as Bitcoin proof of work does, branches are chosen accordingly to the coin age of blocks, in particular it is selected the branch with the highest coin age [28].

Casper: proof of stake Ethereum proposals

Ethereum foundation is actively working on an innovative proof of stake algorithm for Ethereum platform called **Casper** [29]. It aims to solve security problems of traditional PoS algorithms (like the one introduced by Peercoin) introducing some features typical of Byzantine Fault Tolerant algorithms [29] (for example resilience to 33% of malicious users in a network [12]). The main goal of Casper consists in finding an effective solution to fork problems. For this purpose, it introduces high penalties for cheating miners. To take part to the validation process, each user must bet an amount of coins (called deposit), the highest the amount, the highest the probability of being chosen as block validator. A misbehaving miner is punished losing the gambled amount. Since to be chosen as block miner it is necessary to bet large amounts of coins, notably lower than the possible reward, misbehaving actions are highly discouraged [29]. A detailed description of this algorithm and an analysis on the possible attacks and countermeasure is shown in [29].

2.3.6 Advantages and Disadvantages

The Ethereum platform introduced some important features with respect to Bitcoin. The main advantages are:

- Smart contracts based on Turing-complete programming language.

- Possibility to easily transfer text associated to transactions.
- Possibility to develop Decentralised Autonomous Organizations.
- Possibility to limit the concentration of mining power and introduction of ASIC resistance.

Disadvantages of Ethereum are the same of all the blockchain solutions based on proof of work, as explained in Sec. 2.4.

2.4 Limits of proof-of-work based blockchains

Analysing Bitcoin and Ethereum the main problems deriving from proof of work consensus algorithms arises. The low throughput, high computational power and energy consumption are the three main drawbacks making very difficult the adoption of these solutions for some purposes. The main parameters that can be modified to achieve better performances are:

- **Block size**. Increasing the block size it is improved the number of transactions accepted per second, since the difficulty of proof of work does not change. At the same time, a higher propagation time of blocks over the network increases probability of forks. [30]. According to [30] it is better to avoid block sizes larger than 8 MB to maintain secure the network.
- **Time of generation of blocks**. Each blockchain solution introduces a block generation time (e.g. 10 minutes for Bitcoin, 12 seconds for Ethereum). According to [30], increasing the block interval it is also improved the security of the blockchain, because of the probability of forks decreases. For this reason, the Bitcoin blockchain can be considered more secure than the Ethereum one.
- **Number of confirmations per block**. Since proof-of-work algorithms does not present consensus finality [12] it is necessary to wait a given number of blocks are appended to the chain before transactions contained into an arbitrary block are considered accepted by the network. In this time interval, in fact, there is an high risk of forks that may invalidate the transactions. To speed up transactions approval, it is possible to reduce the number of required confirmations but this is very dangerous because it increases the risk of successful double spending attacks [1].
- **Mechanism of blocks propagation** [30] To reduce the latency of blocks propagation, shrewdness on how new mined blocks are transmitted to neighbours can be taken. An increased latency time, in fact, can increase the risk of forks. Three techniques can be summarized:

  1. When a block is ready, a notification is broadcasted to peers, and they will require the transmission of the whole block if needed. This is the technique adopted by Bitcoin [1, 30]. This solution increases the latency but reduces, when possible, transmission of useless data.
  2. When a block is ready, it is immediately broadcasted to peers. This alternative reduces the latency i.e. the probability of forks [30].

3. Blocks are immediately broadcasted only to a subset of peers, while other peers receive a notification. This trade-off is adopted by the Ethereum platform [30].

Given the above, it is concluded that dealing with fundamental parameters of proof of work based solutions can be very useful to improve performances at the expense of the security of the network. According to [30], in a proof of work platform with a good dimensioning of blockchain properties, it is possible to obtain a throughput of about **66 transactions per seconds**, maintaining a security level similar to the one of Bitcoin network.

Some researches demonstrated it is not feasible to use permissionless blockchain solutions for IoT applications, because of the too large amount of transactions to be stored and managed [31]. This restriction is not only applied to IoT applications but to all possible applications requiring to deal with a large number of transactions per second. To overcome this problem it is necessary to consider alternative solutions based, for example, on Byzantine Fault Tolerant consensus algorithms or substitutable storage solutions, like Iota or Hashgraph.

Chapter 3

Permissioned Blockchains

The analysis of permissionless blockchain solutions revealed some limitations of these platforms, for example the **low throughput** and the **high latency** of transactions. This chapter describes many permissioned blockchain platforms able to overcome the limitations of the permissionless ones but characterized by the existence of a central authority. The survey is focused on the different consensus algorithms, like the Proof of Elapsed Time, Round-Robin and BFT consensus algorithm. The first four sections describe respectively MultiChain, BigchainDB, Hyperledger Sawtooth and Hyperledger Fabric. Sec. 3.5 shows the differences between the BFT and proof-of-work consensus algorithms and Sec. 3.6 concludes the chapter comparing all the blockchain platforms analysed in this thesis.

3.1 MultiChain

MultiChain is one of the most popular open-source platforms that can be used to create private permissioned blockchains, where only the authorized users can transact, create blocks, send messages, etc. Users are identified with a pair of asymmetric keys associated to a public key certificate. A particular type of transaction is used to authorize new users to join the blockchain. When a new blockchain is created, in fact, the genesis block provides all the permissions to only one user. The latter is an administrator of the blockchain and has the possibility to release certificates and to authorize new customers to join and interact with the network [6]. The source code of MultiChain is derived from the Bitcoin Core because the MultiChain developers want to keep similar functionalities to those of Bitcoin, so that future improvements of the Bitcoin protocol can be easily implemented in MultiChain [6]. MultiChain has been developed as a technology that can be used to facilitate the transfer of assets, for example cryptocurrencies, and does not provide the possibility to write smart contracts to be executed on the nodes of the network.

3.1.1 MultiChain consensus protocol

Unlike other private blockchain platforms, MultiChain is not based on a Byzantine Fault Tolerant consensus algorithm. In MultiChain the concept of blocks, transactions and miners still remain as well as a mechanism to validate blocks. Differently from Bitcoin, mining in MultiChain is not based on a computational expensive proof-of-work, but it is based on a **Round-Robin** (RR) scheduling technique to introduce

randomness in the selection of miners of new blocks [6]. When a user is selected to become the miner of a block, for a well-defined number of following blocks he will be automatically excluded from the mining operations. The **mining diversity** ($0 \leq m_d \leq 1$) is the parameter enforcing this rotation of miners [6]. The adoption of mining diversity avoids the concentration of the mining power in the hands of a small subset of users but can compromise the efficiency of the blockchain. To obtain a trade-off between security and efficiency it is necessary to select an appropriate value of mining diversity (typically $m_d = 0.75$) [6]. The mining diversity is used to compute the **spacing** i.e. the number of blocks that must be appended to the chain after a miner has validated one block and before he can create another block [6]. The spacing can be computed with the following formula:

$$s = \lfloor m_d \cdot n_{miners} \rceil - 1$$

where s is the spacing obtained by decrementing by one the result obtained by rounding the multiplication of the mining diversity (m_d) with the total number of authorized miners of the blockchain network (n_{miners}) [6]. The identification of the miner of a block, necessary for the correct application of the Round-Robin scheduling, is made possible by the introduction of a digital signature in every mined block.

Forks

The consensus algorithm based on Round-Robin scheduling allows forks and indicates the absence of finality of transactions. The frequency of forks is increased if some miners remain isolated, for example for some problems on the network [6]. Similarly to Bitcoin, forks are solved considering as valid only the longest generated chain of blocks [6].

3.1.2 Scalability

Data relative to scalability of MultiChain platform is available in [32]. The throughput that can be obtained is assessed to **1000 transactions per second**, while the time required to create a block is indicated of only **2 seconds** [32]. Concerning the number of nodes that can take part to the blockchain, in [32] it is not estimated an upper bound, but it is indicated that the number of nodes can be potentially infinite [32]. Even if optimistic, this consideration can be considered truthful since MultiChain is derived from Bitcoin and the latter is actually managing a very large number of network nodes.

Time required for read-only operations

One of the most important drawbacks of MultiChain, common to many blockchain solutions, is the long time required to retrieve stored information. Data is stored in blocks, and the blocks must be read in sequence in order to find the required data. For this reason, it is difficult to implement and execute complex queries. The applications requiring continuous read-write operations on data are obstructed by this problem. The whitepaper of MultiChain states that the future releases of the platform will be provided with an interesting feature, an additional **centralized relational database** that may be used to execute complex queries using the full

featured SQL query language [6]. This database will be installed in every peer of the network, and will be populated with the data coming from the chain of blocks. The blockchain will preserve the immutability of data, while efficient read operations will be provided by the SQL database [6]. If really implemented, this feature should become an important breakthrough for the adoption of MultiChain platform in complex projects.

3.1.3 Multiple Chains

An important characteristic of MultiChain, emphasized by its name, is the possibility to create **multiple blockchains** [6]. Some users of the network, with adequate permissions, have the possibility to generate secondary chains. A secondary chain is identified by its own genesis block, its miners, its users, its permissions. It is a totally independent blockchain that can work in parallel to other existing chains [6]. Different chains are able to cooperate to allow the exchange of information [6]. For example, some users can simultaneously join multiple chains and transfer assets between them. This feature is very interesting in permissioned environments because it allows to diversify the users according to data they can read and transactions they can execute. In this way, the reliability of blockchain networks can be improved, by promoting the adoption of the blockchain in business environments.

3.1.4 Messaging

Another interesting feature of MultiChain is the possibility to exchange messages between nodes of the blockchain. Messages can be exchanged independently on transferred assets, making easier the propagation of information [6]. MultiChain provides different types of messages, unicast and broadcast messages. At time of writing, unicast messages have not yet been implemented, their goal consists in providing an end-to-end communication mechanism between two nodes of the network. **Broadcast messages**, instead, are already available and are called **streams** [33]. A broadcast message is sent to all the participants of the chain subscribed to a given stream, providing an efficient mechanism of data distribution. Received messages are locally stored by receivers, by adopting the typical data replication pattern of blockchain platforms. The sender of a messages can be easily identified because each message contains a digital signature [6, 33]. The format of messages can be described as follows [6, 33]:

- Key [optional] that can be used for unambiguous identification of the message stored in the form of a key-value pair.
- Timestamp
- Payload
- Hash to summarize the content of the message
- Digital signature applied by the sender. It is fundamental to prevent the transmission of anonymous information.

Data confidentiality in streams

The data shared with streams is written in plain text and anybody can access its content. It is important to introduce the possibility to encrypt the content of messages so that data confidentiality can be preserved. As explained in [33] the best technique to share the content of a message in a distributed environment, by allowing only a limited set of recipients to access contained information, consists in the use of symmetric and asymmetric cryptography [33]. The overall encryption and decryption process can be described as follows [33]:

1. The sender of the message must collect public keys of recipients.
2. The sender encrypts the message with a symmetric key (the same key is used for all potential receivers) and broadcasts the encrypted message to blockchain users.
3. The sender broadcasts another message containing for each recipient of the encrypted message, the symmetric key of step 2 encrypted with the public key of the recipient. In this way, only authorized users will be able to decipher the symmetric key and to access the transferred data.

Messages and Bitcoin

The developers of MultiChain created a particular Bitcoin wallet implementing the message exchange functionality for the Bitcoin blockchain. This project is called CoinSpark [1] and adds the possibility to store text attached to Bitcoin transactions [6]. This feature can be very helpful to create official documents: once transactions are validated by the blockchain, the content of these documents can no longer be manipulated.

3.1.5 Advantages and disadvantages

Advantages

- Permissioned blockchain.
- No concentration of processing power due to Round-Robin scheduling adopted by the consensus algorithm.
- The security of the network can be preserved also with a small number of nodes participating in the blockchain.
- Discrete throughput (1000 tps).
- Low latency.
- No transaction fees and no rewards for miners.
- Multiple chains can be created and can exchange information.
- Messaging and streams feature.

[1] available at `http://coinspark.org/`

- Traceability of information. The messages are timestamped and digitally signed by the sender.
- Consensus algorithm tolerant to maliciously misbehaving nodes.

Disadvantages

- Impossibility to develop smart contracts.
- Absence of finality of transactions (forks can occur).
- Read access time not optimized (future improvements).

3.2 BigchainDB

BigchainDB is an innovative open-source solution that tries to merge the scalability of distributed databases and the security of blockchain platforms [13]. In particular, BigchainDB promises to satisfy the following requirements [13]:

1. High throughput (up 10^6 transactions per second) and low latency of transactions.
2. Finality of transactions.
3. Complete query language.
4. Tamper-proof data storage, with an immutable chain of blocks linked together.
5. Possibility to use the platform to manage the creation and the transfer of any kind of asset.

BigchainDB is characterized by a permissioned environment where a well-known set of authenticated users, called nodes, collaborates to safely store data without the coordination activities of a central authority. Actually, BigchainDB tolerates only **accidental failures of nodes** while Byzantine failures cannot be managed. For this reason, it is considered a crash-tolerant and not a Byzantine-fault tolerant blockchain platform [13].

3.2.1 Data structure

The data structure of BigchainDB is different from the typical chain of blocks of other blockchain platforms. BigchainDB storage solution, in fact, is composed by **two distributed databases** (RethinkDB or MongoDB instances can be used) characterized by the following functions [13]:

- Database **S**, also known as ***transaction set*** or ***backlog***. It collects the transactions generated by the clients and not yet validated. At the beginning, i.e. before the first transaction is generated, it is empty [13].
- Database **C**, also known as ***blockchain***. It contains valid and invalid transactions, grouped into blocks logically linked to each other. At the beginning, i.e. before the first block collecting the transactions is generated, it contains one empty block called *genesis block*.

The interactions occurring between S and C will be described in Sec. 3.2.2. The use of two distributed databases allows to obtain a high transactions throughput. Unlike many other blockchain platforms, in fact, these two databases are not completely replicated in each node of the network. They are split in smaller portions, called shards, replicated a given number of times by a subset of all the participants of the network [13] (replication factor is typically set to 3). In this way, adding nodes to this distributed system implies increasing the amount of data that can be globally stored, i.e. the scalability in terms of storage requirements is obtained [13]. On the other hand, the nodes of the network cannot execute operations based on their own local replica of data, because it is necessary to merge all the shards owned by different participants to obtain an overall view of stored data.

Transactions and blocks

BigchainDB maintains the concepts of blocks and transactions [13]. A transaction can contain generic information and can be used, among other things, to create and transfer assets between customers. Blocks are used to collect transactions and are logically linked with previously generated blocks, similarly to a traditional blockchain. Unlike the other blockchain platforms, blocks do not contain the hash of the previous block of the chain in their header. This information is stored at granularity level of transactions, i.e. for each transaction contained in the block it is stored a reference to the previous block [13]. Each block, in fact, contains a set of transactions that must be voted by the nodes of the network to decide if the content of the transactions can be considered valid or not. The voting nodes attach to every transaction of the block the hash of the previous block of the chain [13]. This is fundamental to demonstrate the transaction has been voted according to the current state of the blockchain, identified by the previous block.

Absence of forks

In BigchainDB forks cannot occur. Each node, in fact, does not maintain a full replica of data and must access to the same distributed databases in order to execute read and write operations. Therefore, the overall state of the ledger is shared among all the participants of the network and forks cannot occur.

3.2.2 BigchainDB consensus mechanism

In BigchainDB, the consensus algorithm is called **BCA** - BigchainDB Consensus Algorithm, and it is executed by a well-known set of nodes of the network that decides if the transactions can be considered valid or not [13]. This classification is executed according to a voting mechanism strictly related to the concept of quorum: when the majority of nodes (50%+1) agrees on the validity or not of a transaction, a decision is taken [13]. The consensus mechanism is triggered by the generation of new transactions executed by clients of the network, and can be outlined as follows [13]:

1. One node of the network (n_1) receives a new transaction and verifies if it can be considered potentially valid (for example it queries S to see if it does not contain an identical transaction) [13]. In the case this preliminary test is

passed, n_1 stores the transaction in the backlog S and notifies another node of the network (n_2) that it will be in charge of managing that transaction [13]. On the contrary, in case the transaction is not valid, it is automatically discarded [13].

2. Node n_2 collects a set of transactions stored in S, and creates a block [13]. This block is persisted into the blockchain database C, and contains only transactions that must be validated. For this reason, the new block is classified as **undecided** [13].

3. To execute the validation of blocks, voting nodes check the validity and express a vote for every transaction contained therein. Each vote is relative to a single transaction and it is composed by the hash of the previous block and by the voting decision. The latter can assume only two values, **valid** or **invalid**. [13].

4. When the majority of voters agrees on the validity or not of a transaction, the latter is classified. It is sufficient that only one transaction of the new block is considered invalid to categorize the whole block as **decided_invalid**, otherwise it is marked as **decided_valid** [13].

5. All the transactions contained in a *decided_valid* block are committed, while all the transactions of a *decided_invalid* block do not affect the state of the ledger. If a *decided_invalid* block contains some valid transactions, they are stored again in S so that they can be inserted in another block and committed. Valid and invalid blocks will never be removed from C and generate an always growing chain of blocks.

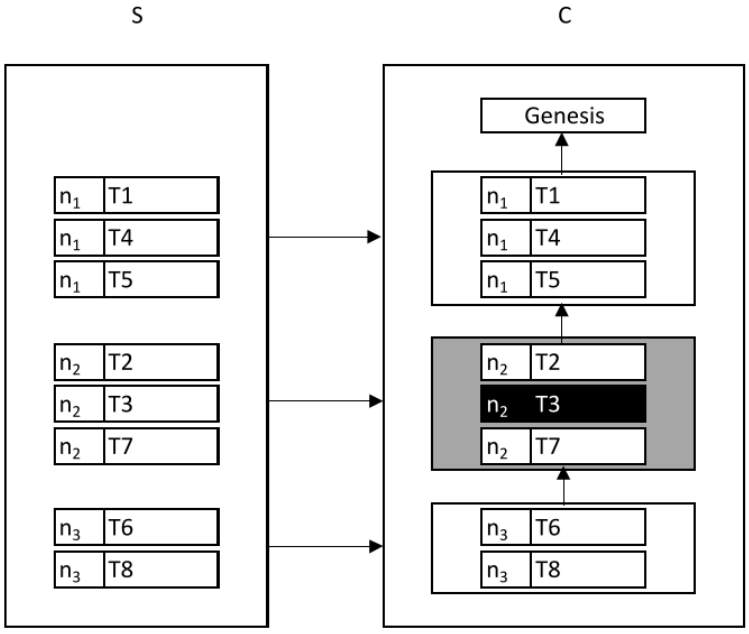


Figure 3.1: Example of backlog (S) and blockchain database (C).

Fig. 3.1 shows an example describing the BigchainDB consensus algorithm. In the picture, S indicates the backlog and C the blockchain database. The blocks drawn

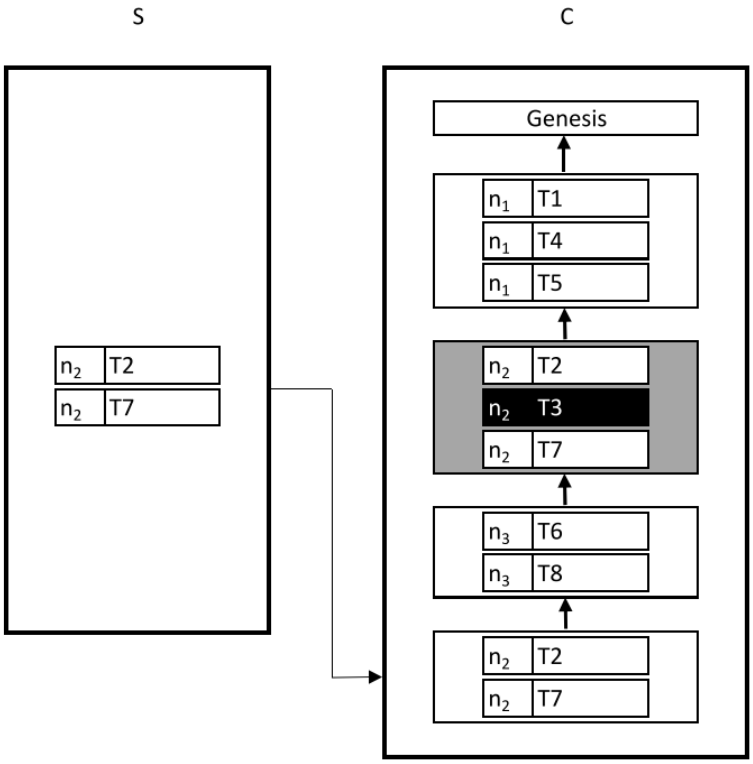


Figure 3.2: Example of backlog and blockchain database containing the transactions *T2, T7* previously stored in an invalid block.

in C with a white background are *decided_valid* blocks, while the grey background indicates the *decided_invalid* blocks. The black background indicates an *invalid* transaction stored in C. In the proposed example the backlog S contains 8 transactions (*T1, T2,..., T8*). The network is composed by three nodes, n_1, n_2, n_3, to which are assigned some transactions that must be validated. The node n_1 creates a block (*block 1*) containing the transactions *T1, T4, T5*. The majority of nodes voted as valid these three transactions, and therefore the *block 1* has been marked as *decided_valid* and committed. The second block (*block 2*) is created by node n_2 and contains the transactions *T2, T3* and *T7*. The transaction *T3* has been considered as *invalid* by the majority of nodes, thus the correspondent block has been marked as *decided_invalid* but it has not been removed from the chain of blocks. For this reason, the third block (*block 3*) contains a set transactions that refer to *block 2* as the previous block of the chain. After the transactions have been inserted in valid blocks, they are removed from the backlog.
Fig. 3.2 shows that the transactions *T2* and *T7* have been reinserted in S because they had been marked as *valid* even if they were stored in a *decided_invalid* block (*block 2*). The two transactions have been subsequently stored in a new block (*block 4*) and validated.

3.2.3 Scalability

The authors of BigchainDB whitepaper [13] conducted multiple experiments to test the scalability of the platform. The results shown in [13] are summarized in this

section.

Throughput

According to experimental results of [13], the throughput of BigchainDB varies from **1000 to 2000 writing transactions per second** (tests conducted on a cluster of 32 and 64 nodes). The performances are strictly related to the number of processes simultaneously writing to the databases. Additional details on the results are available in Table 3.1 and Fig. 3.3.
Table 3.1 summarizes the experimental results conducted by the authors of [13] to evaluate the throughput of BigchainDB. *ID* indicates the identifier of the experiment, *Nodes* indicates the number of nodes of the cluster, *Writing processes* indicates how many processes simultaneously execute writing operations on BigchainDB. th_{theory} and th_{exper} represent respectively the theoretical and experimental throughput values (expressed as thousands of transactions executed per second). The theoretical values refer to the first experiment (data taken from [13]).
Fig. 3.3 summarizes the experiments 1,2,3 shown in Table 3.1. The green line indicates the expected throughput values (theoretical), while the violet line indicates the experimental results (Data taken from [13]).

ID	Nodes	Writing processes	$\mathbf{th_{theory}}$ (k tps)	$\mathbf{th_{exper}}$ (k tps)
1	32	64	-	1.0
2	32	80	1.25	1.2
3	32	96	1.5	1.4
4	64	64	1.0	1.0
5	64	96	2.0	2.0
6	64	128	1.5	1.5

Table 3.1: Scalability tests conducted by the authors of [13]

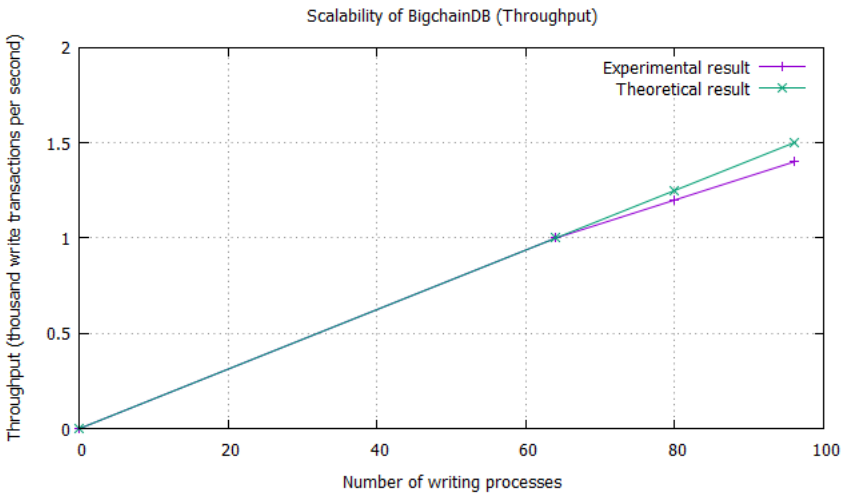


Figure 3.3: Results of the scalability tests conducted by the authors of [13].

Storage capabilities

Unlike other blockchain platforms, BigchainDB adopts sharding instead of full replication of data. This means that data is not completely replicated in each node of the network but each participant stores only a portion of the whole dataset [13]. Each shard must be replicated a defined number of times for fault tolerance. Typically, replication factor is set to 3. In traditional blockchain platforms, the maximum amount of data that can be stored depends on the storage capabilities of each node. In BigchainDB, instead, it is proportional to the number of nodes composing the cluster. Increasing the number of nodes of the cluster, the overall storage capabilities of the distributed system are increased. Fig. 3.4 shows a graphical representation of the storage scalability of BigchainDB. It represents how many terabytes of data that can be stored varying the number of nodes. Each node can store at most 48 TB. The replication factor is set to 1 (data coming from [13])

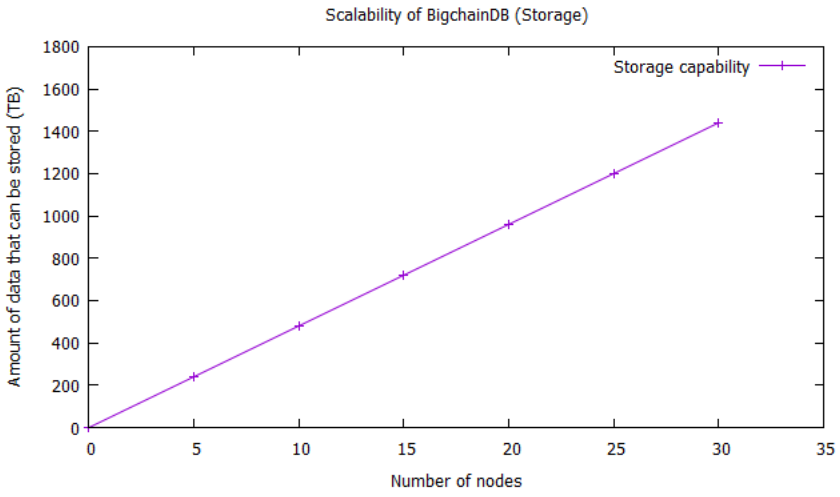


Figure 3.4: Storage scalability of BigchainDB (data coming from [13])

3.2.4 Advantages and disadvantages

Advantages

- High throughput.
- Low latency.
- Finality of transactions.
- High storage capabilities (linearly dependent on the number of nodes).
- Full-featured query language.
- High performances in read-write operations.

Disadvantages

- Impossibility to write smart contracts.
- Absence of Byzantine fault tolerance (crash-tolerant consensus algorithm).

3.3 Hyperledger Sawtooth

Hyperledger Sawtooth is an interesting open-source project that allows people to create permissioned (and permissionless) blockchains using an innovative consensus mechanism called Proof of Elapsed Time (PoET) [34]. The goal of Hyperledger Sawtooth consists in giving the possibility to users to create their own customized blockchain solution to track and manage any kind of assets [35]. One interesting application of a Sawtooth based blockchain is the management of supply chain and traceability of fish, from the fishermen to end consumers [35]. Sawtooth provides the possibility to develop user defined smart contracts, increasing the possible fields applications of this technology. However, this solution is still under development; at time of writing the first stable version 1.0 has been released [34].

3.3.1 Consensus mechanism

The consensus mechanism of Hyperledger Sawtooth is based on a **Proof of Elapsed Time** (POeT). The goals of POeT are shown as follows:

- Avoiding the concentration of mining power in the hands of a small subset of users.
- Identifying users in a secure and verifiable manner, either in permissioned and permissionless environments.

Proof of Elapsed Time can be used to choose one user, among the whole set of users, as the candidate to validate one block (called *batch* in Sawtooth lingo). The selection of the **leader** for block validation, is executed imposing a random waiting time to all users. Therefore, each potential miner of blocks is requested to sleep for a given time interval before validating the batch. The first user of the network that finishes this waiting time, becomes the validator of the block. The problem of this solution consists in verifying if the miner really waited the requested time. To solve this problem, an Intel based solution has been developed. Starting from 2015, some Intel Core and Intel Xeon processors were equipped with the Intel Software Guard Extensions (SGX) [36]. This piece of hardware allows to create a trusted execution environment in which instructions can be executed, proving in a verifiable manner the CPU completed the related operations. Therefore, SGX allows to generate a random time, suspending the execution of instructions of the machine, and proving the miners really waited the declared random time before creating the block [11]. For debugging purposes, even CPUs not equipped with SGX are authorized to participate in the validation process. In real execution environments, especially permissionless ones where identification of participants is complex, it is required that only devices providing an Intel SGX module can participate in the validation of blocks [11].

PoET is a non-final consensus algorithm. Forks can occur and are solved choosing the longest chain as the main chain.

The possibility for attackers to have a greater influence on the network is strictly proportional to his availability of devices equipped with Intel SGX [11].

Scalability

A detailed analysis on this platform has not been conducted. According to [10], throughput level that can be obtained with Sawtooth is about **1000 transactions per second**.

3.3.2 Advantages and disadvantages

Advantages

- Possibility to write smart contracts.
- Possibility to create a permissioned blockchain.
- Discrete throughput.
- Low power consumption required by consensus mechanism.

Disadvantages of this solution

- Specific hardware requirements (Intel SGX).
- Absence of finality of transactions.

3.4 Hyperledger Fabric

Hyperledger Fabric is one of the Hyperledger open-source projects maintained by the Linux Foundation [37]. It is an innovative **private permissioned blockchain** platform [37] developed using the Go programming language [38]. Unlike other blockchain platforms it is **not based on a native cryptocurrency** but permits the execution of **chaincodes**, i.e. smart contracts in the Hyperledger lingo [37]. Furthermore, it allows the cooperation of different members and organizations so that they can maintain different replicas of stored data, execute the code associated to transactions and verify the result of transactions executed by different members. The fundamental characteristic of Hyperledger Fabric is the **modularity** [39]. Some functionalities, like the consensus mechanism or the identification of users are not standardized and can be replaced by the blockchain administrators, depending on the needs of their applications. Hyperledger Fabric version 1.0 provides an integrated **crash-tolerant consensus algorithm** based on a Kakfa cluster ordering service and a ZooKeeper ensemble [37]. The administrators of a Hyperledger Fabric blockchain are free to replace the existing consensus mechanism and to adapt their favourite algorithm, also a Byzantine-fault tolerant one, so that it is possible to fit the requirements of the applications [37]. For example, if the deployment environment of the blockchain is not susceptible to Byzantine failures, it is possible to use a crash-tolerant consensus algorithm to achieve better performances. On the contrary, a BFT-SMaRt consensus algorithm, like the one proposed in [40], can be used to deal with malicious participants.

Hyperledger Fabric is also focused on the management of the **privacy** of information, that can be obtained using a full-featured **authentication and authorization** system to restrict the access to unauthorized users [37]. By default, the

management of user authentication is executed by the **MSP** module - Membership Service Provider module [41]. The privacy can also be improved by creating multiple independent ledgers, called **channels**, characterized by data access policies for the different kinds of users [37, 42].

3.4.1 Data structures

Similarly to other blockchain platforms, the smallest piece of information that can be managed in Hyperledger Fabric is the transaction. A transaction is associated to the execution of a chaincode and to a change of state of the distributed system.

The overall view of stored data is managed by the ledger. The ledger of Hyperledger Fabric is composed by three main elements:

- **Chain**. It is the chain of blocks. The blocks are logically linked to each other and include in their header the hash of the previous block of the chain. Each block contains a list of transactions and a hash summarizing this list [43]. The chain provides the tamper-proof property to the blockchain, by creating an append-only log of blocks and transactions [37].

- **State Database**. It maintains the current state of the ledger, generated by the sequential execution of the transactions, in the same order they appear in the chain of blocks. The entries of the state database are in the form of **key-value pairs** and allow to speed up data retrieval, thanks to the use of queries operating on the keys [37]. Every key-value pair is also associated to a *version number*, useful to understand which was the last operation that modified the value associated to such a key [37]. The state database allows random access to data stored in the blockchain and increases the performances of data retrieval operations. The chaincodes can read, write and delete the values associated to the keys using the functions *GetState, PutState, DelState* respectively [37].

  Hyperledger Fabric version 1.0 allows to adopt two different database management systems [37]:

  - **LevelDB**
  - **CouchDB**

  Both allow to retrieve the value associated to the keys, to execute range queries on keys (indicating a minimum and maximum value that can be assumed by a key) and to execute queries based on *composite keys*, i.e. keys composed by many fields [44]. Furthermore, CouchDB allows to execute complex read-only queries, by accessing the different fields of the value associated to the key. This additional feature is possible provided that the value associated to the key is a JSON document. [44].

- **History index**. The history index is another instance of LevelDB that allows to execute queries to retrieve the full history of the modifications to the value associated to a key [45]. For this purpose, it accesses data stored inside blocks by using a LevelDB index to speed up the operations [45]. The function *GetHistoryForKey* must be invoked in a chaincode to access the history index and to retrieve the history of a key [45].

3.4.2 Architecture of Hyperledger Fabric

The traditional blockchain platforms, like Ethereum or Hyperledger Fabric up to version 0.6, are based on a **order-execute** architecture [37]. Their consensus algorithm generates new blocks by inserting a set of transactions in it. Then, every peer of the network must execute the code associated to the transactions to verify their correctness and to update the state of the ledger [37]. This architecture may impose some limitations in terms of throughput of the system because all the peers of the network must execute the same pieces of code. In addition, it is not compatible with the execution of **non-deterministic** code [37] (the majority of programming languages, like Java, Go, etc. are non-deterministic). In the case many peers execute the same set of non-deterministic operations and obtain different but potentially valid results, the final state of the ledger might not be the same for all the users [37].

Hyperledger Fabric, starting from version 1.0, introduced an innovative architecture, known as **execute-order-validate architecture** [37]. It is based on the differentiation of the roles of different members of the network. The actors of this architecture are divided in three categories:

- **Clients**. The clients are the users of the blockchain. They require the execution of transactions, by submitting transaction execution requests to the endorsing peers [37]. They collect the signed responses generated by endorsing peers, check if enough endorsing peers have executed the transaction (i.e. if the endorsement policy has been satisfied) [37] and broadcast the transaction proposals to the orderers [37].

- **Peers**. Peers are divided in two categories:

  - **Endorsing peers**. They receive transaction execution requests submitted by clients and simulate the execution of transactions according to the current state of the ledger [37].
  - **Committing peers**. They receive blocks created by the ordering service and verify the correctness of contained transactions. They also maintain a replica of the ledger, including the whole chain of blocks, the state database and the history index [37].

  Every endorsing peer plays also the role of committing peer; the vice versa is not guaranteed.

- **Orderers**. They are also known as Ordering Service Nodes (OSN). They receive the transaction results signed and broadcasted by clients, order them chronologically and create the blocks. The latter are then distributed to committing peers for the validation [37]. Orderers execute many other operations in the blockchain, for example they are in charge of applying modifications to the permissions of the channels, by creating blocks containing the *configuration transactions* [37]. The orderers are not aware of the current state of the ledger and do not verify the correctness of the transactions that they insert in the blocks [37].

The separation among different nodes of the previously described roles i.e. clients, peers and orderers, allows the implementation of an execute-order-validate architecture [37]. Every participant of the blockchain plays a specified role and only a subset

of the endorsing peers is in charge of executing chaincode functions. In this way, the operations (i.e. endorsement, ordering and validation) can be parallelized and the performances of the blockchain network can be increased [37]. By default, in Hyperledger Fabric the peers and the orderers are executed inside different **Docker containers**. In this way it is possible to test a working blockchain by using a single hosting machine. However, in real deployment environments it is necessary to user different hosting machines, each with a different role.

Membership Service Provider (MSP)

The management of different nodes and users requires a mechanism for the authentication and the authorization of the participants of the blockchain. For this purpose the **MSP** (Membership Service Provider) has been developed. As well as other components of Hyperledger Fabric, also the MSP is modular. The default MSP implementation, in fact, can be replaced with another module able to fulfil the requirements of the applications [41]. The MSP executes the identification of the participants with a procedure based on digital signature and signature verification scheme [37]. It is based on a PKI - **Public Key Infrastructure** that includes a hierarchy of X.509 certificates to allow members of different organizations (peers and orderers) and clients of the blockchain to be authenticated and authorized for the execution of the required operations [46].

For each organization taking part to the blockchain it is present a **Root CA**, a Root Certification Authority and, possibly, a set of intermediate CAs that are delegated by the root CA for the execution of its same operations [41]. They release and distribute X.509 certificates for the authentication and authorization of peers and orderers of the organization. An associated CRL, Certificate Revocation List, plays a complementary role, by indicating that some certificates can no longer considered valid and shall not be accepted for the authentication of the users [41].

Transactions Flow

The transactions flow of Hyperledger Fabric is divided in three main steps: [37]:

1. **Execution**
2. **Ordering**
3. **Validation**

Execution. The execution phase of a transaction begins with a *transaction request* issued by an authenticated and authorized client that sends a signed request to a set of endorsing peers [37]. This transaction proposal provides all necessary information for the execution of a chaincode function, including the parameters of the invoked function and the channel in which the chaincode must be executed [37]. In addition, a mechanism to avoid replay attacks permits to verify if it is the first time that the same transaction is executed [37]. The proposal is digitally signed by the client so that the receivers, i.e. the endorsing peers, have the possibility to verify if the transaction issuer is authorized for the execution of the operation [37]. The endorsing peers receive the transaction proposal, verify if the digital signature belongs to an authorized user and then start the execution of the code associated

to the transaction [37, 47]. The endorsing peers may read the current state of the ledger by accessing the content of their own local replica of the state database. All the key-value pairs read from the state database constitutes the **read set** of the transaction [37]. For every key, it is also associated the version of the key, i.e. a numeric value that allows to understand how many times the value associated to such a key has been modified; it is fundamental for the validation phase [37]. At the same way, during the simulation of transaction execution, some key-value pairs may be stored in the state database, updating the relative version number of the key. This set of updated keys constitutes the **write set** of the transaction [37]. Every endorsing peer produces a signed response containing the results of the execution of the transaction [37]. The signed responses are forwarded to the client requesting the transaction execution. All the modifications to the state occurred during the transaction execution are not committed to the ledger, i.e. to the state database is not updated. [37].

Ordering. The issuer of the transaction, i.e. the client, collects the signed responses generated by the endorsing peers, verifies if the number and identity of the endorsers satisfy the endorsement policy and then broadcast the result, i.e. the transaction, to the Ordering Service Nodes [37]. The orderers receive the transactions, order them chronologically and insert them inside a new block [37]. The orderers do not care about the content of the block and on the validity of contained transactions, they simply create new blocks. The generated block is then distributed to the committing peers for the validation phase [37].

Validation. The validation phase is executed by all the peers taking part to the channel [37]. For each transaction stored inside the received block, the committing peers check if the endorsement policy has been satisfied, i.e. if a sufficient number of endorsing peers signed the transaction proposal [37]. Typically, the endorsement policy requires that different peers, potentially belonging to different organizations, sign the transaction so that the transaction result can be considered valid. In addition, the committing peers check the read-write set of each transaction [37]: for each element in the read set, it is verified if the version number of the key accessed during transaction simulation is equal to the version of the key of the current state of the ledger. In other words, the transaction is considered valid if there are no other transactions in the block that modified the version number of the key under analysis [37]. Transactions are validated according to the order of appearance inside the block [37]. Only the transactions satisfying the two steps of validation are committed and can modify the state of the ledger. On the contrary, they are marked as invalid and do not affect the state of the ledger.

Fig. 3.5 shows a graphical representation of the transactions flow described as follows:

1. A client submits a transaction proposal to the endorsing peers.

2. The endorsing peers execute the code associated to the transaction, generate the read-write set and sign the result.

3. The signed result is sent back to the client.

4. The client collects a sufficient number of endorsements so that the endorsement policy can be satisfied.

5. The client broadcast the transaction to the orderers.
6. The orderers collect transactions, order them chronologically and create a signed block.
7. The orderers distribute the new block to all the peers (endorsing and committing peers).
8. The endorsing and committing peers execute the validation of the transactions contained in the received block; they verify if the read-write set is coherent with the current state of the ledger and if the endorsement policy has been satisfied.
9. The client is notified of the result of execution of the transaction.

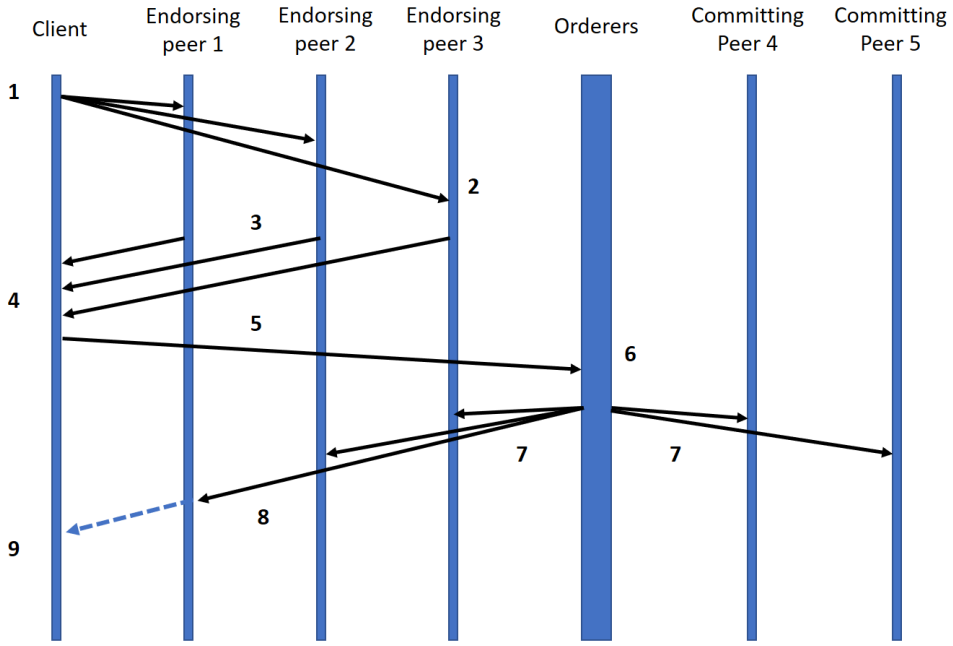


Figure 3.5: Transactions flow of Hyperledger Fabric version 1.0

3.4.3 Hyperledger Fabric consensus mechanism

Hyperledger Fabric version 1.0 consensus mechanism is based on a **crash-tolerant algorithm** based on a Kafka ordering service and a ZooKeeper ensemble [37]. The process to obtain consensus on the order of transactions is the one one described as *transaction flow* in Sec. 3.4.2. Because of its consensus modularity, Hyperledger Fabric allows to replace the consensus algorithm to better fit the requirements of the application. Recent research activities are intensively working on the development of BFT consensus algorithm for Hyperledger Fabric. In [40] it is shown an implementation of the **BFT-SMaRt consensus algorithm** applied to Hyperledger Fabric version 1.0 that allows to obtain throughput values greater than 10000 transactions per second, with a sub second latency [40].

3.4.4 Chaincodes - Smart Contracts

Hyperledger Fabric is a blockchain platform developed for the execution of chaincodes, i.e. smart contracts in the Hyperledger lingo [37]. The chaincodes can be written in any supported programming language, but the languages widely adopted in version 1.0 of Fabric are Go and NodeJS [48]. The code of the chaincode functions can be **non-deterministic** [37]. This characteristic differentiates Hyperledger from all the blockchain platforms analysed so far and influences its performances. Another important feature is the possibility to limit the number of peers executing the code of the chaincode [37]. In other blockchain platforms, like Ethereum, smart contract code was executed by all the peers of the network during the validation phase of the transactions [4]. This imposes some limitations on the scalability of the platform in terms of throughput. In Hyperledger Fabric, only a subset of peers identified as **endorsing peers** must execute the transactions before they can be committed. This means that different endorsing peers can simultaneously simulate the execution of different transactions and the throughput can be notably increased. A set of rules, called **endorsement policies** decides how many endorsing peers must execute the chaincode, imposing additional constraints on the identity of peers: for example a typical endorsement policy requires that the endorsement of the transactions must be executed by at least one peer for each organization. By default, if an endorsement policy is not specified, Hyperledger Fabric requires that at least one peer executes the chaincode, regardless of the organization it belongs to [49]. The endorsement policy is common to all the transactions of a given chaincode and can be decided by the administrator of the network during the instantiation of the chaincode.

The possibility to limit the number of endorsing peers executing the chaincode allows to increase the parallelization of the operations. Different endorsing peers, in fact, can simultaneously run the code associated to different transactions [37]. On the other hand, by limiting the number of endorsers of transactions to one peer only, it is not possible to verify if different peers obtained the same result during the execution of the transaction [37]. This can be a problem if the code associated to the chaincode is not deterministic [37]. To increase the security of the blockchain application it is preferable to require that many peers endorse the same transaction.

The chaincode is available only in the channels in which it has been installed and instantiated. In every channel it is also possible to deploy different chaincodes. Every chaincode can access its current state, by using the *GetState*, *PutState* and *DelState* functions. In version 1.0 of Fabric a chaincode can query the state of another chaincode to retrieve stored information. According to Fabric documentation, in future releases of the platform every chaincode may be able to update the state of another chaincode by executing writing operations [50]. Every chaincode is always executed in a separated Docker container with respect to the container of the peer [37]. In this way, if some errors occur during the chaincode execution, the chaincode container is stopped while the associated peer container can continue working [48].

Another category of chaincodes called **system chaincodes** is used to manage the operations related to the architecture of the blockchain [37]. Some examples of operations that can be executed by a system chaincodes are:

- Retrieval of single blocks of the blockchain [51].
- Execution of the digital signature of the transaction result proposed by an endorsing peer [51].

- Execution of the required operations to validate and commit transactions [51].

The execution of the code of the system chaincodes is not managed by isolated Docker containers [51].

Protection from DOS attacks

Similarly to other blockchain platforms, Hyperledger Fabric adopts a protection mechanism against the execution of infinite loops in the chaincode functions. As shown in Sec. 2.3.2, Ethereum introduced the concept of gas and gas price to avoid denial-of-service attacks, by limiting the number of operations that can be executed by a transaction. Hyperledger Fabric is not based on a cryptocurrency, thus it cannot impose monetary penalties against very long lasting sequences of instructions executed by a chaincode. For this reason, Fabric introduced the concept of **timeout of chaincode execution** by imposing an upper bound on the execution time of the transactions. If the timeout is exceeded, the transactions fail and their modifications to the ledger are not committed.

3.4.5 Channels

In Hyperledger Fabric, every channel represents an independent ledger. Every peer can participate in multiple channels and maintain a replica of different ledgers [42]. The use of channels is important to improve the privacy of the system. Each channel, in fact, limits the access to information to a well-defined set of authorized users [42]. The development of a multi-channel blockchain is particularly useful in scenarios characterized by the participation of different competing organizations. Different channels can be used to separate shared information from confidential information that cannot be disclosed to members not belonging to the organization [37]. Only the peers that joined a channel, in fact, have the possibility to read data and issue transactions in that channel. To join a channel and to maintain a replica of data it is necessary to be authorized and authenticated [42]. A chaincode must be installed and instantiated in every channel in which it must be used. The chaincode that is executed by a peer can access another channel (different from the one in which the chaincode is installed) to read stored data if and only if the peer that is endorsing the transaction is authorized to access both channels (write operations are not permitted). [42]. Fig. 3.6 shows a Hyperledger Fabric blockchain composed by 3 peers and 3 channels. *Peer 1* joined three channels *A, B, C* and maintains a replica of the three chains of blocks, *Peer 2* joined only channel *A*, while *Peer 3* joined channels *B,C*.

3.4.6 Advantages and disadvantages

Advantages

- Permissioned blockchain.
- High throughput and low latency of transactions.
- State database to speed up the execution of queries.
- Consensus modularity.

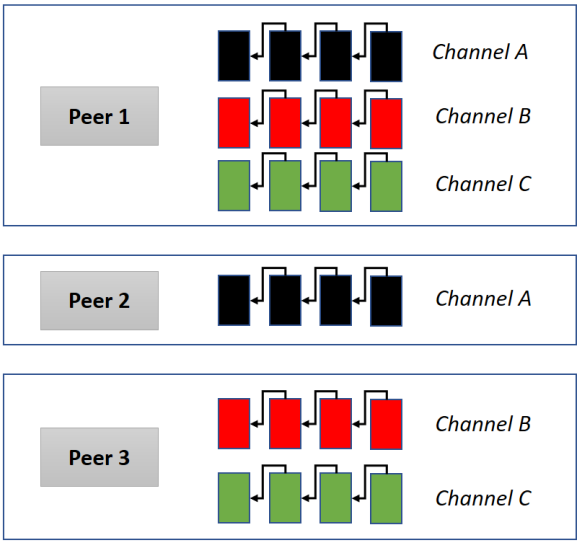


Figure 3.6: Example of a Hyperledger Fabric blockchain composed by 3 peers and 3 channels

- Possibility to create multiple independent ledgers (channels).
- Possibility to execute non-deterministic chaincode functions.
- Possibility to develop chaincodes in different programming languages.

No particular disadvantages have been identified for this platform.

3.5 Byzantine-Fault Tolerant consensus algorithms

The Byzantine fault tolerance is an important characteristic that must be maintained by distributed systems. It indicates that the system must be able to deal with correctly working nodes, accidental crashes and, in particular, misbehaving participants trying to prevent reaching consensus [12].

The content of this section is based on the paper of Vukolić et al [12], that provides a very interesting and detailed analysis of the **BFT - Byzantine Fault Tolerant** - consensus algorithms. In particular, the author focused his analysis on the differences with respect to proof-of-work consensus mechanism, the one adopted by Bitcoin, Ethereum and many other permissionless blockchains.

BFT consensus algorithms are actually implemented in some permissioned blockchains, like Hyperledger Fabric, where the PBFT - Practicable Byzantine Fault Tolerant is available for Fabric v0.6 and BFT-SMaRT available for Fabric v1.0 [11]. Another blockchain platform adopting BFT-SMaRt is R3 Corda [11].

3.5.1 Comparison between Proof of Work and BFT

Identification of participants

BFT consensus algorithms can be applied in a permissioned environment, in which a central authority monitors the access to network and is aware of the identity of

nodes. Before joining the distributed network, every user must be authorized and must receive a valid certificate released from a Trusted Third Party (TTP) [12]. Furthermore, because of this controlled environment, users that want to take part to the consensus protocol must be configured so that they can interact with other peers [12].

In the networks based on proof-of-work consensus algorithm, the users can join and leave the network without receiving any kind of permissions and it is sufficient that users generate a pair of asymmetric keys to interact with other network participants. This total freedom for network participants increases the risk that malicious users try to tamper data [1].

Resistance to malicious nodes

BFT consensus algorithms are either crash tolerant and Byzantine fault tolerant. These two properties indicate respectively that BFT algorithms can deal with accidental failures of nodes (i.e. crashes of nodes or communication problems) and with malicious participants trying to compromise the possibility to obtain the consensus in the network [12]. Obviously, there exists a limit on the maximum number of misbehaving nodes that can be tolerated. In the case more than **33.3% of nodes** maintains a faulty replica of data [52], it is not possible to reach consensus even with the adoption of BFT consensus algorithms [12] [52].

Proof-of-work consensus algorithms can tolerate a maximum of 25% of computational power concentrated in the hands of an attacker before the consensus might be compromised [12]. Therefore, the adoption of BFT algorithms implies also an increased security level for the distributed network.

Throughput

BFT algorithms allow to obtain a very high number of committed transactions per second (more than 10000 transactions per second) [12]. This value is huge if compared to the results that can be obtained by Bitcoin proof-of-work (i.e. less than 10 transactions per second).

Number of nodes of the network

BFT consensus algorithms are not intended to work with a very large number of nodes. An increase in the number of nodes taking part to the network, in fact, implies a decrease of performances of the distributed system. According to [12], experiments involving a large number of nodes have not yet been performed on distributed networks based on BFT consensus algorithms. On the contrary, performances and security levels of the networks based on proof-of-work are proportional to the number of participants [12]. For this reason, proof-of-work algorithms are not suitable to manage distributed networks characterized by a small number of nodes.

It is important to differentiate the concept of **nodes** and **clients** of the network. The nodes participate actively to the consensus protocol by maintaining a replica of the ledger and by taking part to the consensus mechanism. A client of the network, instead, is simply a user, requiring a set of services. Both proof-of-work and BFT consensus algorithms allow to obtain good results in terms of the number of clients of the distributed system [12].

Finality of transactions

The concept of transaction finality is strictly related to forks. In proof-of-work, forks can cause some transactions to be invalidated even after being included into blocks [1], i.e. transactions cannot be considered final [12]. To deal with this problem it is necessary to wait a given number of confirmations before considering the transaction as confirmed and secure [1]. This characteristic can be very problematic, especially if the blockchain is used in a scenario with legal repercussions.

BFT algorithms do not consider the occurrence of forks. As soon as a new transaction is validated by peers, it is automatically committed, without an additional waiting time and without the possibility of being subsequently invalidated. This means finality of transactions is immediately reached. The bottleneck of a distributed system based on BFT consensus algorithms becomes the propagation time of data over the network, and the number of messages that must be exchanged between peers ($\Theta(n^2)$, where n is the number of nodes of the network) [14].

Energy consumption

BFT algorithms are characterized by a very low energy consumption, notably lower with respect to proof-of-work. In BFT consensus algorithms, in fact, it is not necessary to solve complex cryptographical functions [52]. On the contrary, proof-of-work based technologies are famous for the huge amount of wasted energy that is necessary to maintain the system [52]. This introduces money troubles to maintainers of the network (i.e. miners), and, in particular it is dangerous with regard to air pollution.

Table 3.2 summarizes the comparison between proof-of-work and BFT consensus algorithms. Data refers to [12]

	Proof of Work consensus	**BFT consensus**
Access mode to network	Permissionless	Permissioned
Resistance to attacks	25% of computing power	33% of faulty replicas
Throughput	<100 tps	>10000 tps
Maximum number of nodes	Very high	Limited
Maximum number of clients	Very high	Very high
Consensus finality	No	Yes
Energy consumption	High	Low

Table 3.2: Comparison between proof-of-work and BFT consensus algorithms (data coming from [12])

.

3.6 Comparison between the analysed blockchain platforms

Table 3.3 compares the blockchain platforms analysed in this thesis.

Platform Name	Permissioned	Consensus Algorithm [A]	Fault Tolerance [B]	Smart Contracts	Currency	Throughput (tps)	Latency	Energy Consumption
Bitcoin	No	PoW	BFT[1]	Yes[3]	Yes	7	60min	High
Ethereum	No	PoW[2] PoS	BFT[1]	Yes[4]	Yes	13	3min	High
MultiChain	Yes	RR	BFT[1]	No	No	1-2k	-	Low
BigchainDB	Yes	BCA	CFT	No	No	1k	-	Low
Sawtooth	Yes/No	PoET	BFT[1]	Yes[4]	No	1k	-	Low
Fabric	Yes	Kafka BFT-SMaRt	CFT BFT	Yes[4]	No	110k	< 1s	Low

Table 3.3: Comparison of blockchain platforms

[A] Consensus algorithms: Proof-of-Work (PoW), Proof-of-Stake (PoS), Round-Robin (RR), BigchainDB Consensus Algorithm (BCA), Proof of Elapsed Time (PoET), Kafka-based ordering (Kafka), BFT-SMaRt

[B] Fault Tolerance: Byzantine-Fault Tolerance (BFT), Crash-Fault Tolerance (CFT)

[1] Resistance to Byzantine failures, not a strict BFT consensus algorithm implementation

[2] PoW memory hard implementation (ASIC resistance).

[3] Stack-based scripting language.

[4] Turing-complete smart contracts.

Chapter 4

Blockchainless Distributed Ledger Technologies

This chapter describes two blockchainless Distributed Ledger Technologies based on a data structure different from the chain of blocks characterizing the blockchain platforms. The blockchainless platforms aim to overcome the limitations of the permissionless blockchains, like the low throughput and high latency, without necessarily introducing a central authority that manages the access to network. Sec. 4.1 and Sec. 4.2 describes respectively Iota and Hashgraph.

4.1 IOTA

IOTA is a modern Distributed Ledger Technology released in 2016. It has been developed to become the *"cryptocurrency for Internet-of-Things"* [53]. It is considered a blockchainless technology because it is not based on a chain of blocks. However, it creates a tamper-proof log of monetary transactions shared among the participants of the network. [53].

4.1.1 Introduction to Iota

Iota has been developed to allow the interactions of IoT devices [53]. These devices are able to communicate with others, exchanging payments for required services without the interaction of a central authority. One of the most important characteristic of Iota cryptocurrency is the **absence of fees** [53]. Users can freely exchange coins, so that even small amount of money can be transferred without any kind of problems. A multitude of **micro-payments** can be issued by users to exchange small pieces of information [53]. One of the greatest problems of traditional blockchain platforms, in fact, is the need to introduce transaction fees as incentive for miners to secure the ledger, making impossible to adoption of a blockchain to transfer very small amount of coins. The absence of transaction fees of Iota is combined with the **absence of miners**. In Iota, there are no miners validating blocks of transactions and there is not the need of a reward mechanism. This implies the absence of new minted coins and a constant circulating coins supply.
The Iota network is **permissionless**, anybody can download the wallet application [1], generate a seed and start sending and receiving coins. A **seed** is simply a sort of

[1] available at `https://github.com/iotaledger/wallet/releases`

identifier of the wallet, with which it is possible to generate addresses to send and receive coins [54].

4.1.2 Iota ledger data structure

Iota is based on a data structure, called **tangle**, that is used as an alternative to the chain of blocks. The tangle is a Directed Acyclic Graph (DAG), composed by **nodes** and **edges**, indicating respectively the **transactions** and the **verification of transactions**. The first transaction of the DAG is called *genesis transaction* and can be reached by every node of the graph by following the directed edges. Notice that, in Iota, the blocks do not exist, so the validation of data is executed at the granularity of transactions. Each transaction can be connected to other transactions with the edges. By using the example of Fig. 4.1, the Iota terminology is summarized as follows [53]:

- A, B, C are three **transactions**.
- Transaction A is called **genesis transaction** because it is the first transaction of the DAG.
- Transaction B **directly approves** transaction A. In the same way transaction B validates transaction A because exists a directed edge that connects B to A. This property is not revertible, i.e. transaction A does not approve transaction B.
- Transaction C **indirectly approves** transaction A because exists a path from C to A with at least an intermediate node.
- Transaction C is called **tip** because it has not yet been approved by any other transaction.

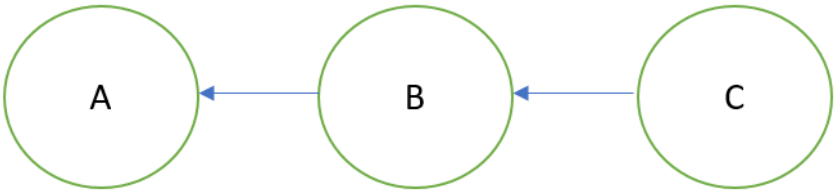


Figure 4.1: A simple DAG composed by three nodes only.

Without going into technical details, the validity of a transaction is strictly related to the number of other transactions that directly or indirectly approve it [53]. Similarly to the blockchains based on proof-of-work, the transactions are not final, i.e. they can be invalidated over time. In case of double spending attempts, characterized by different conflicting transactions, in fact, a transaction should be invalidated: Iota consensus algorithm treats as valid only the transaction with the greatest confidence value, i.e. the one that has been approved more times by the other transactions [53]. The number of confirmations required to consider a transaction as valid is not fixed. It strictly depends on the decisions taken by involved

parties exchanging money. The higher the confidence of the transaction, the lower the probability for the recipient of transferred money of being scammed. In the following paragraph it is shown an example to clarify the described concept.

Double spending attack in Iota

A seller accepts payments with Iotas. A malicious user decides to buy some goods issuing an Iota transaction. After the transaction has obtained a reasonable level of confidence, the seller executes the shipments of the merchandise. At this point the attacker issues a double spending transaction directed to another wallet under his control; if the second transaction obtains a larger number of confirmations, the seller of the product loses his money because other nodes of the tangle consider the original transaction as invalid [53].
In Iota, an attacker can successfully issue a double spending transaction if it is able to create a sub-tangle that must be kept private as long as the double spending transaction obtained a large number of confirmations. At this point the attacker can spread the sub-tangle to other network participants and invalidate the original transaction [53]. This behaviour is similar to the one described as double spending attack of the Bitcoin protocol and shown in Sec. 2.2.7.

4.1.3 Iota consensus mechanism

Iota is totally distributed, each peer of the network needs to maintain a replica of the ledger. For this reason, a distributed consensus mechanism is necessary to allow the verifiability of the transactions. The absence of a chain of blocks and miners requires an alternative solution to deal the problem of distributed consensus.
In Iota, every user that issues a new transaction needs to validate at least two already existing transactions [53]. There are two fundamental operations in this process, the first consists in deciding how to select the transactions to validate, while the second is relative to the validation process of the previously selected transactions.

Tips selection algorithms

As part of the validation process of the transactions, it is necessary to decide which transactions (in particular tips) to validate, in order to reduce the time required to obtain multiple approvals for the transactions. The level of confidence of a transaction does not depend only on the number of direct approvals, but also on the indirect ones. There are many techniques that can be exploited to choose the tips to validate. The simplest one consists in randomly selecting the transactions to validate, but this method takes too long and the time required is not predictable. A user, in fact, can decide to approve not only tips, but also already validated transactions, by increasing the time required to approve new transactions and slowing down the growth of the DAG [53].
Many other possibilities are shown in Iota whitepaper [53] but, actually, the most providing one is based on the **MCMC** (Markov Chain Monte Carlo) algorithm. MCMC consists in randomly choosing an arbitrary number of nodes of the graph, in crossing DAG branches in their reverse order with the purpose of locating the transactions not yet validated. The tips that have been reached with the lowest number of steps become the candidates of the validation process [53].

Validation process of transaction

After having identified the transactions to validate, it is important to follow additional steps to conclude the validation: [55]

1. Computation of a digital signature [55]. Each transaction issuer must prove to be the real owner of the coins he is going to transfer. Exactly as in the Bitcoin protocol, the sender of coins must execute a digital signature of the inputs of the transaction (i.e. the outputs of the previous transactions).

2. Verification of the absence of conflicting transactions [55]. There is the possibility that two conflicting transactions try to spend the same coins. To identify the conflicting transactions, the validation algorithm requires to cross the DAG, starting from the selected tips up to the genesis transaction, and to check if there are double spending attempts in the whole path. In case of cheating attempts, the validator discards the selected tips and restart the procedure looking for other tips to validate [53].

3. Computation of a proof-of-work. It is necessary to prove the real identity of the issuer of the new transaction. To do this, the hashes of the new generated transaction must be computed in sequence by varying a nonce. The process ends when the computed hash obtains a numeric value lower than a given threshold i.e. it must satisfy the so-called difficulty of the proof-of-work. This algorithm is similar to the one implemented by the Bitcoin protocol [55].

At a first glance, this consensus mechanism seems more complex than the Bitcoin proof-of-work: in Iota, in fact, the computational puzzle must be computed for each transaction, and not only for a set of transactions, i.e. a block. As a matter of fact, the difficulty of the Iota proof-of-work is notably lower than the difficulty of the Bitcoin proof-of-work. This implies that also the devices with low computational capabilities are able to compute the results of the cryptographic problems in a reasonable time interval [53]. In this way, the validation of the transactions is distributed among all the users of the network and it is not concentrated in the hands of a reduced number of miners.

4.1.4 Throughput

One of the advantages of Iota is the high transactions throughput. The value confirmed in May 2017 was about **180 transactions per second**, with the objective of reaching the value of 1000 transactions per second [55]. This value is large if compared to other blockchain platforms but cannot be sufficient to deal with the global volume of the transactions that can be generated by millions or billions of IoT devices. The Iota foundation promises that the scalability of the tangle is proportional to the number of users and to the number of issued transactions. In the future, when a large number of devices will adopt Iota to store data, the throughput should be notably boosted [55].

4.1.5 Tangle snapshots

The data stored in the tangle is replicated in every node, similarly to other distributed ledger solutions. In a scenario characterized by a multitude of devices with

low storage and computational capabilities, an ever-growing set of information could introduce an important limitation for the adoption of Iota. For this reason, the Iota developers introduced the **snapshot** functionality. A snapshot can be considered a sort of picture of the current state of the tangle [56] and can be executed because every Iota transaction is simply a transfer of coins. Therefore, every snapshot summarizes in a new genesis transaction the total amount of coins owned by each user at the time of snapshot execution. From this moment on, the new transactions will refer to this new genesis transaction, with the possibility to delete previously stored information [56]. Early Iota snapshots were associated to a claim procedure that should be executed by every user in order to continue to interact with the tangle. In the future, an automated mechanism will be adopted to introduce transparency and simplicity on this process [56]. In this way, many more devices in the world should be able to store and use the Iota tangle, even if equipped with limited storage resources.

4.1.6 Possible attacks

The greatest risk of Iota is the **double spending problem** described in Sec. 4.1.2. In practice, an attacker can create two transactions trying to spend the same coins. One transaction is directed to a victim, and the other to a wallet under the control of the attacker. The absence of finality of transactions allows the attackers to create a private set of transactions validating the fake transaction. As soon as the latter obtains enough approvals, the new portion of the DAG can be gossiped to other nodes of the network. At this point, new incoming transactions will consider as valid the double spending transaction because it has obtained a larger endorsement [53].

A detailed description of other attacks and countermeasures is described in Iota whitepaper [53]. At a first glance, the arguments shown in that document exonerates Iota from potential attacks, including those based on quantum computing [53]. Actually, the academic literature has not yet proved these assumptions, so it is better to wait much time before adopting Iota as an alternative to traditional blockchain technologies. Even in detailed research activities, like the one conducted by Cachin et al [11], the authors do not take a position about the security of this innovative blockchainless solution.

A research of 2017 conducted by MIT Media Lab found some vulnerabilities on the Curl hash algorithm adopted by the Iota proof-of-work [57]. The Curl hash function, in fact, was not subjected to a long-time revision that is necessary to test the reliability for the cryptographic algorithms [57]. Actually, Curl has been replaced by Kerl, a more secure algorithm based on KECCAK-384 [58].

4.1.7 Advantages and disadvantages

Advantages

- Efficient consensus mechanism managed by every transaction issuer.
- No concentration of processing power because of the absence of miners.
- Low energy consumption. The consensus mechanism is not controlled by energy expensive algorithms.

- Scalability increased with the number of transactions. As indicated in [55], Iota throughput should increase with the number of users and issued transactions.
- Snapshots. Iota provides a snapshot mechanism that allows to remove old and no longer useful information from the tangle.
- No transactions fees and support for micro-payments.
- Resistance to quantum computing [53].

Disadvantages

- Permissionless environment.
- Absence of smart contracts.
- Absence of scientific results on the security of Iota consensus mechanism.

4.2 Hashgraph

Hashgraph is an innovative blockchainless Distributed Ledger Technology. Differently from the majority of solutions analysed in this thesis, Hashgraph is not released with an open-source license, but it is a proprietary solution proposed by *Swirlds, Inc.*

Hashgraph is not based on a chain of blocks but its fundamental data structure is a **DAG**, Directed Acyclic Graph [59]. This alternative data structure allows to obtain throughput values greater than the traditional blockchain platforms. Differently from previously analysed technologies, Hashgraph can work either in **permissionless** and **permissioned** environment, where only authorized users can join the network and maintain a replica of the ledger [60]. The latter is the most common scenario in which Hashgraph is adopted, and tolerates up to the **33% of malicious nodes** to reach consensus on stored data [59].
Hashgraph, actually, is not associated to a cryptocurrency, no coins have been distributed and general-purpose applications can be developed on top of it.

4.2.1 Data structure

The hashgraph data structure is a DAG - Directed Acyclic Graph, an always growing graph in which new nodes, called events, can be appended to already existing ones, without the risk of generating loops in the data structure [59]. Fig. 4.2 shows a graphical representation of a DAG composed by many chronologically ordered events.
Similarly to Iota, in Hashgraph are not presents the blocks, but it is maintained the concept of transaction. The transactions are stored inside nodes of the graph and can contain any kind of information, depending on the requirements of the applications.
The nodes of the graph are called **events**. Every event generated by a user must be connected exactly to two previous events, the first generated by the same issuer of current event, while the second created by another user [59]. Every event is composed by many fields, graphically shown in Fig. 4.3, and described as follows:

- Timestamp. Indicates when the event has been generated [61].
- Transactions. This field is optional and can contain some transactions associated to the current event [61].
- Hashes. They act as pointers toward two previous events [61] (in Fig. 4.3 Hash E1 refers to an event generated by the same user of the current event, while Hash E2 refers to a gossiped event).
- Digital Signature. It is used to identify the user that generated the event [61].

The interconnections between different events are represented as edges in Fig. 4.2, but, in reality, they are implemented as hash pointers stored inside the events.

4.2.2 Hashgraph consensus mechanism

The Hashgraph consensus mechanism is articulated and will not be deeply analysed in this thesis. If the reader is interested in obtaining additional details on the topic

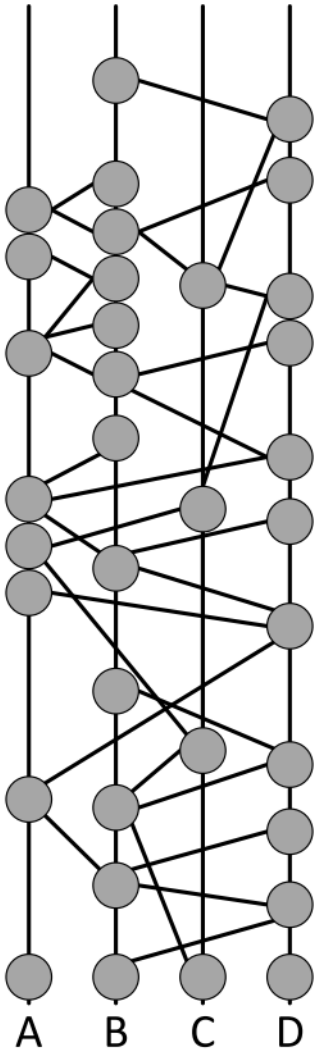


Figure 4.2: Hashgraph DAG composed by many events (image taken from [61]).

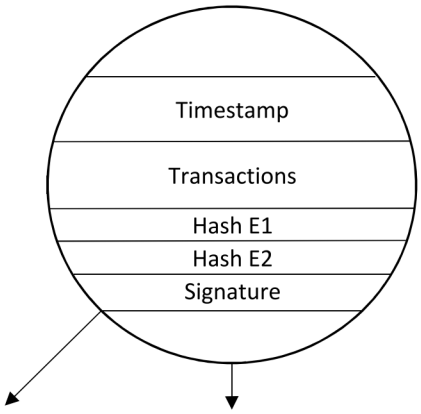


Figure 4.3: Graphical representation of an event.

he can refer to [59] and [61]. This section investigates the mechanism that stands behind the consensus mechanism, the concept of **gossip**.

Gossip

The gossip is the technique adopted to spread data across all the nodes of the network. Every time a node creates a new event it must notify the other peers of this information. Differently from traditional blockchains in which the data is broadcasted to all the interested users, Hashgraph adopts an innovative solution called gossip, that consists in sharing all the known information to just one user. This process [61] is explained with an example that involves four users *A, B, C, D* (graphical representation in Fig. 4.4).

1. The user A must store some information and generates a new event containing one transaction. The current timestamp of A's device is written in the corresponding field of the event. For simplicity this event is called *A1*.

2. When this new event is generated, the node of the network randomly chooses another user (B) and sends (gossips) him the new event. In addition, he also attaches all the information received by other peers of which the recipient is not yet aware of. This operation is called **synchronization** [59] because, after its completion, user B knows exactly the same information of A.

3. The receiver, user B, creates a new event (*B2*) to notify the completion of the synchronization. In case user B has some pieces of information to store, this new event can contain an additional payload in the transaction field. The fundamental characteristic of this event is the presence of the current timestamp of the machine and of two pointers (hash values), one to the event that triggered the synchronization (*A1*), and one to the previously generated event of user B (*B1*).

4. Once the synchronization has been completed and the relative event correctly created, user B must gossip this data to another randomly selected user, for example user D. The latter creates a synchronization event *D2*, etc. and the process goes on forever.

This consensus procedure is characterized by some interesting characteristics:

- It is **fast** [59]. Differently from the proof-of-work algorithm of traditional blockchains it permits to obtain very high throughput values.

- Each **event is timestamped** [59]. When a new event is gossiped, receivers must create a new event to record that the received information has been stored. As soon as all the members receive the gossip of an event it is possible to date the transaction time (computed as the average between the timestamps declared by the event issuer and by all the users that received gossiped information). With reference to the example of Fig. 4.4 the timestamp of the event is computed as the average of the timestamp declared by the events *A1, B2, D2, C2*. With this technique, the **global ordering of transactions** is always decided [59, 61].

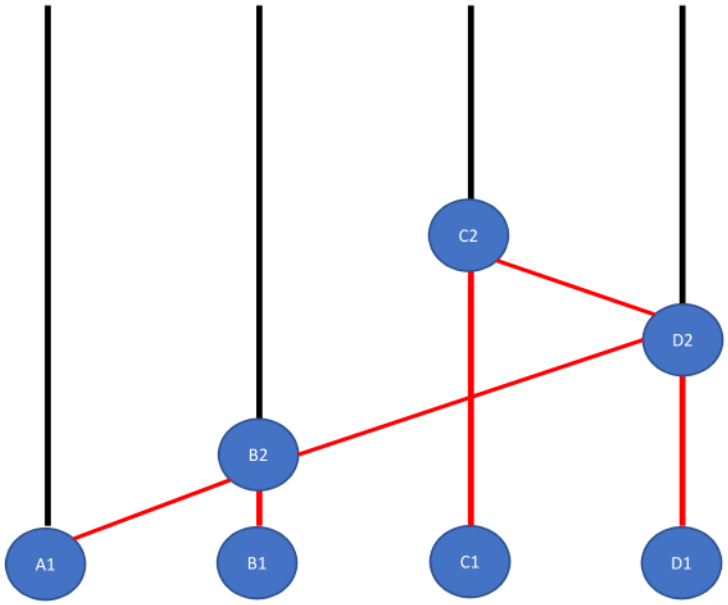


Figure 4.4: Example of DAG generated as a result of event *A1* created by user *A*.

Voting

Another important step of the consensus algorithm is the voting process [61]. Its purpose consists in deciding if an event is **famous** (i.e. well-known by the majority of members of the network) or not. If an event is famous, its content is considered valid and is accepted by the network participants. The whole non-trivial voting procedure is shown in [61]. What emerges from this document is that the voting procedure shall be carried out by visiting the local copy of the DAG and verifying which nodes (and users) received the gossip of the events (and users) [61]. This process can be executed **offline**, i.e. each member of the network reads its local replica of the Hashgraph and decides if an event can be considered famous or not, without the need of interacting with the other members of the network [59]. This is fundamental for performances reasons, because no additional data is spread across the network [59].

Resistance to attacks

Offline voting introduces more resilience to attacks. As long as more than 66% of members are honest and agree on ballot results, it is not possible to manipulate the outcome of the voting procedure [59].
Furthermore, in Hashgraph, there is not the concentration of computational power in the hands of miners. Blocks and miners, in fact, do not exist, while voting is executed locally by every member. In traditional blockchain platforms, miners could decide to delay (even forever) a transaction avoiding to insert it in new blocks [1]. In Hashgraph, each member collaborates with the others to gossip the events and to share known information with the other peers [59].

4.2.3 Signed state

In Hashgraph it is possible to discard the information belonging to old transactions without losing the immutability property of data [59]. This behaviour can be obtained by applying the concept of **signed state**. When a set of events are considered famous, the previously existing nodes of the Hashgraph can be removed because they are no longer required by the consensus algorithm. The information contained in it can be stored in an alternative data storage solution, like a database [59]. While updating the database, the same order of events generated by the Hashgraph consensus mechanism must be preserved. This is fundamental to obtain a final state coherent for all the members of the network. This final state, obtained by feeding the database, is called **consensus state** and must be digitally signed by the creator, inserted as payload of a new event and gossiped to the peers [61]. The receivers of this gossip event can recompute the consensus state and verify the correctness of the digital signature. From this moment on, they have the possibility to remove previous data from the Hashgraph to streamline it [59].

4.2.4 Advantages and disadvantages

Advantages

- High throughput (transactions per second).
- Consensus ordering. If an attacker tries to issue an event containing a wrong timestamp, the real consensus time is obtained as average of the timestamps declared by all the members of the network that received the gossip [61]. It is impossible to issue transactions as they were generated in the past.
- No miners. There is not the risk some transactions are delayed or not included in blocks.
- Possibility to remove old information from the Hashgraph after computing the consensus state (signed state). It requires reduced storage capabilities.

Disadvantages

- Modern technology. The literature has not yet deeply analysed the possible attacks and countermeasures.
- Closed-source project. The source code cannot be reviewed by research community to identify potential vulnerabilities.
- Impossibility to develop smart contracts.

Chapter 5

Distributed Ledger Technologies for Vehicular Applications

This chapter analyses the use-case of this thesis, a urban mobility scenario composed by vehicles able to exchange messages each other, and describes a blockchain based architecture used to store and manage these messages.
Sec. 5.1 illustrates the scenario of the thesis; Sec. 5.2 describes the format and the content of the messages exchanged between the vehicles while Sec. 5.3 explains how to collect the messages into reports to subsequently store them in the blockchain. The main advantages of the blockchain platform used to implement the proof-of-concept applications are outlined in Sec. 5.4 and finally, Sec. 5.5 describes the the overall architecture of the distributed system.

5.1 Description of the use-case

This thesis deals with the communication within a densely populated vehicular environment. Each vehicle can exchange messages with their neighbours. These messages, called CAMs - Cooperative Awareness Messages, are standardized by ETSI - European Telecommunications Standards Institute and carry information like the position, the speed, the acceleration, the direction of the vehicle and the timestamp of generation of the message. This information, if stored, can be used to track the history of the vehicles and to describe past events. In particular, insurance companies could access this data to identify those responsible of road accidents and to simplify the resolution of conflicts. The same information could be used for other purposes, for example to identify drivers not respecting traffic laws, or to easily locate stolen vehicles. The fundamental problem of the applications based on CAMs exchanged between vehicles is the reliability of information. Vehicles can distribute false information because of malfunctions or because they have been tampered to avoid the disclosure of confidential data. For this reason, information carried by CAMs must be verified before it can be used for further applications. Literature proposes different papers on the topic **position verifications algorithms**. The latter are able to classify if the position declared by a vehicle can be considered reliable or not, depending on mutual position of transmitters and of receivers of the messages. To correctly apply these algorithms, receivers of the messages must store additional information on their state, like the geographical coordinates, the speed, the acceleration and the timestamp relative to the instant of reception of the

CAM. The messages transmitted and received by vehicles, together with reception information, must be logged in a common data storage so that the position verification algorithms can retrieve and estimate the reliability of information contained in the messages. To store data, each vehicle is equipped with an USIM (Universal Subscriber Identity Module) 4G, and can access to the Internet using the services provided by mobile network operators. Vehicles create reports containing transmitted and received CAMs and upload data to be logged.

5.2 CAM messages

CAM is the acronym for Cooperative Awareness Message. CAM messages are standardized by ETSI (European Telecommunications Standards Institute) [62] in the context of ITS, Intelligent Transport Systems, and are used for the communication between vehicles and between vehicles and infrastructure devices called RSUs (Road Side Units). For this reason, they are split in two categories, V2V (Vehicle to Vehicle) and V2I (Vehicle to Infrastructure) messages. I2V messages are also available. These messages are intended for real time applications for cooperative awareness, vehicles are able to learn information about the state of nearby vehicles.

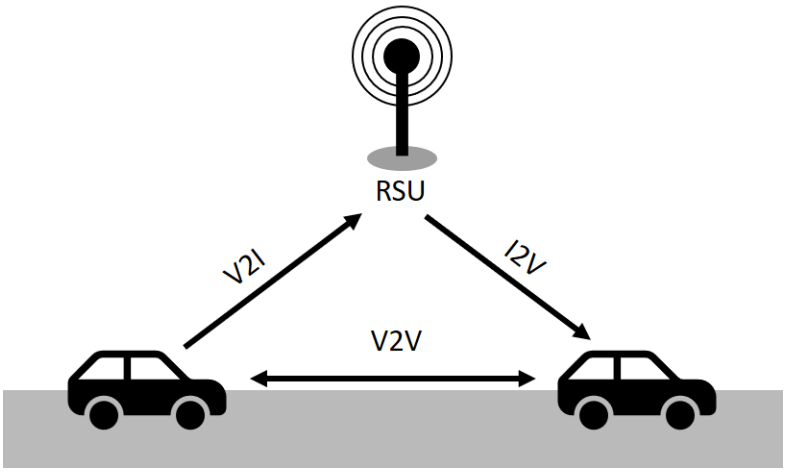


Figure 5.1: V2V, V2I, I2V CAMs.

CAM format is a European standard that will be probably adopted in the future in next-generation vehicles. If that is the case, the applications based on this type of messages will increase their importance and applicability.

5.2.1 CAMs transmission

The transmission of CAMs is a **broadcast** transmission, all the vehicles in the transmitter radio range are able to receive generated CAMs. The transmitter can modify the range varying the transmission power. According to [62], the receiver of a CAM cannot forward the same message to other vehicles. Delivery of CAMs is not guaranteed. Some messages can be lost for a variety of reasons, for example for the congestion of the transmission channel. For this reason, the fact that some messages are not received cannot be considered a cheating attempt of the receiver. The transmission of CAMs is **periodic**, but their generation frequency can change

over time depending on certain factors, for example upon the occurrence of dangerous events, in case the vehicle breaks suddenly or depending on the congestion of the transmission channel [62]. In accordance with the specifications of [62] the CAMs generation frequency varies from a minimum of 1 to a maximum of 10 CAMs transmitted per second. The transmission and reception of CAMs is managed by the so-called, **CA basic service facility layer** [62].

5.2.2 The format of CAM messages

The format of CAM messages is predefined. Developers of applications cannot add or remove fields as needed, they need to use the pieces of information described by the standard. However, an enormous set of information is available in each CAM. Not every field of a message is mandatory; for example certain fields are specific for some types of means of transport. There are fields relative to vehicles for public transports, special transports, emergency vehicles, etc.[62]. All these categories of vehicles transmit CAMs carrying some specific additional information.

Description of most important fields

The application shown in this thesis is based on a set of fundamental information contained in CAMs generated by all types of vehicles. A list of the most relevant fields is shown in Table 5.1.

Field	Description
Header	It contains the identifier of the vehicle that generated the CAM [62]. This value is not fixed but changes randomly over time for privacy reasons [62].
Generation-DeltaTime	It contains timing information, in particular it is relative to the instant in which latitude and longitude of the vehicle have been computed. It is indicated in milliseconds but it is not an absolute value like the unix timestamp. It can assume the maximum value of 65536 (about one minute) [62]. Unit of measurement: 1ms. [62].
Position	It is composed by latitude and longitude values. Unit of measurement: 0.1 microdegree. The accuracy of measurement is provided [63]. This information can be calculated using the GPS receiver.
Speed	It indicates the speed of the vehicle [62]. Unit of measurement: 0.01 m/s [63]. The accuracy of measurement is provided [62].
Heading	It indicates which is the direction of the vehicle with respect to North. Unit of measurement: 0.1 degree. The accuracy of measurement is provided [63].

Acceleration	It indicates the acceleration of the vehicle according to the coordinates (latitude and longitude). Unit of measurement: 0.1 m/s^2. The accuracy of measurement is provided [62].

Table 5.1: Format of a CAM

5.2.3 CAM security envelope

CAMs do not provide security features:

- Peer authentication is not guaranteed. If an attacker is able to send a message stealing the identity of another user, for example using the identifier of another vehicle, the receiver of the message cannot understand the message was sent by a third party.
- Message integrity is not guaranteed. It is not possible to understand if the content of a CAM has been forged by an attacker.

Security features are fundamental if CAMs are used for applications with legal repercussions. Fortunately, the specification [64] provides the standardization of a **security envelope** for CAMs. The most important fields of the security envelope are shown in Table 5.2.

Field	**Description**
Digest of a Public Key Certificate	It allows the identification of the vehicle that transmitted the CAM. It is used to verify of the digital signature.
Timestamp	It is expressed as an absolute value in ms. (different from GenerationDeltaTime of CAM format that can assume 65536 as maximum value).
CAM message	It contains a CAM message, whose content is conform to the standard described in Sec. 5.2.2.
Digital Signature	It is computed with algorithm ECDSA-NISTP-256. The signature introduces the following security features: peer authentication, authentication and integrity of the message, non-repudiation.

Table 5.2: CAM security envelope

Summarizing, the security envelope increases the reliability of transmitted information and provides the following security features:

- It allows the identification of the vehicle that transmitted the CAM.
- It does not permit to attackers to forge the data contained inside a CAM.
- It makes the transmitter legally responsible of the information contained into in the CAM.

5.2.4 CAMs and position verification algorithms

The information transported by a CAM, like position, speed, acceleration, heading and timestamp is fundamental for the application of position verification algorithms. These algorithms allow to classify the position declared by a vehicle as valid or not, considering mutual position of sender and receivers of the messages. Unfortunately, CAMs are application level messages. The position and the timestamp contained in a CAM are computed at application level, when the location of the vehicle has been determined using the information provided by the GPS receiver (Global Positioning System receiver). Position verification algorithms requires to estimate the time of propagation of the signal through the transmission medium. The unspecified delay introduced for the processing of the messages from the application layer to the physical layer prevents the use of information carried by CAMs for the application of the algorithms. For this reason it is necessary to embed more accurate information at physical layer concerning the location and the timestamp of the vehicles relative to the instant the message is broadcasted. Receivers of the messages add the same information at physical layer relatively to the reception instant, so that position verification algorithms considering can compute the estimated distance between vehicles based on the time of propagation of the signal.

Receivers of CAMs can also store additional information that could be useful to implement position verification algorithms. For example the following information can be stored:

- Identifier of the receiver
- Timestamp in which the message was received
- Position (latitude and longitude) of the receiver at reception time
- Speed
- Acceleration
- Heading

Sec. 5.3 explains how to combine transmission and reception information to be persisted inside the blockchain, while Sec. 5.5.6 indicates how to use this data to execute the validation.

5.3 Reports of CAM messages

Vehicles transmit and receive a very large number of CAMs. To simplify the application of position verification algorithms, a new data structure called **report** has been implemented to collect many CAMs. Report data structure **is not standardized** but it has been introduced in the context of this thesis.

A **report** is a data structure that collects generated and received CAMs. A report is created by a vehicle in correspondence of the transmission of a CAM. Every report contains:

- The last generated CAM.

- All the CAMs received which were not present in previously generated reports of the vehicle.

For each received CAM it is also stored a set of reception information, like the position, speed, acceleration, direction and timestamp of receiver, computed at the instant of reception. A report is the data structure that must be persisted for the subsequent application of position verification algorithms.

5.3.1 Structure of a Report

The data structure of a report must be well-defined because reports must used by position verification algorithms. The structure of a report can be outlined as follows:

- $Header$. It indicates that the content of the message is a report of CAMs.
- $ReportId$. Identifier of the report. Each report is uniquely identified by a string, obtained as the concatenation of the *timestamp of transmitted CAM* (considered at physical layer) and the *identifier of the vehicle* generating it (contained in the *Header* of CAM format). The $ReportId$ can be identified also as $GeneratedCamId$ because it depends only on information of the $GeneratedCam$.
- $GeneratedCam$. It contains the last CAM message generated by the vehicle that created the report. Format of CAM message is the one described in Sec. 5.2.
- $GeneratedCamHash$. Hash of $GeneratedCam$ computed using SHA-256 algorithm.
- $ListReceivedCams$. List of CAMs received which were not present in the report previously generated by the vehicle.
- $PreviousReportId$. Identifier of previously generated report.
- $PreviousReportHash$. Hash of previously generated report computed using SHA-256 algorithm.

The $ListReceivedCams$ is a complex field containing different pieces of information replicated for each received CAM:

- $ReceivedCamId$. It is an identifier of received CAM, calculated as the concatenation of timestamp of transmission of the CAM (considered at physical layer) and identifier of transmitter vehicle (contained in the *Header* of CAM format).
- $ReceivedCamHash$. It is the hash of the received CAM.
- $ReceptionInfo$. It contains information relative to the reception of CAM.

The $RepectionInfo$ includes the many fields relative to the instant of reception of the message at physical layer. It can be outlined as follows:

- *ReceptionLatitude*: it it the latitude of the receiver.

- *ReceptionLongitude*: it is the longitude of the receiver.
- *ReceptionTimestamp*: it is the absolute reception timestamp in milliseconds.
- *ReceptionSpeed*: it is the speed of the vehicle.
- *ReceptionAcceleration*: it is the acceleration of the vehicle.
- *ReceptionDirection*: it is the direction of the vehicle with respect to the North.

The format of reports allows to store all the necessary information to execute position verification algorithms and introduces the **tamper-proof property** of reports.

5.3.2 Tamper-proof characteristic of reports

The presence of hash values inside the reports allows to create a **hash chain of reports**.

Hash chain of reports

As shown in previous paragraph, every report contains one field called *PreviousReportHash*. It is simply the hash of the previous report generated by the same vehicle. It allows to generate a sort of chain of all the reports generated by a vehicle. Similarly to many blockchain solutions, the hash of the current report (i.e. block in blockchain context) is not stored but must be recomputed on the fly the verify the correctness of the chain. The first report of the chain is called *genesis report* and it is relative to the first CAM generated by a vehicle. Fig. 5.2 shows a chain of reports logically linked to each other. The following rules must be satisfied so that the chain of reports can be considered valid:

$$\begin{cases} PreviousReportId_n = ReportId_{n-1} \\ PreviousReportHash_n = Hash(Report_{n-1}) \end{cases}$$

where $Report_n$ indicates the whole content of the n^{th} report, and $Hash(x)$ indicates the result of the application of the hash function to x.

The hash chain of reports makes very difficult to tamper data stored inside reports. Every time a vehicle adds a new false report to the already existing chain of reports, a *"fork"* occurs. The cheating attempt can be identified by the existence of a bifurcation in the chain of reports.

Hash of received CAMs

The *ListReceivedCam* contains a set of information for each CAM received by a vehicle. Among other things it contains the *ReceivedCamId* and the *ReceivedCamHash* that allow to logically link the reports generated by nearby vehicles.

- *ReceivedCamId* is the identifier of received CAM and corresponds to the *ReportId* stored in the report of transmitter.
- *ReceivedCamHash* is the hash of received CAM and corresponds to the *GeneratedCamHash* stored in the report of the transmitter.

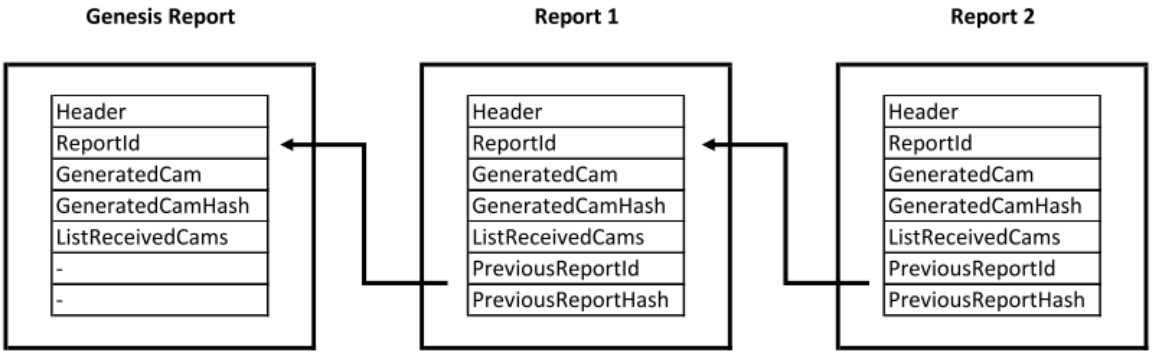


Figure 5.2: Chain of reports logically linked to each other.

In this way a report contains **hashes of all received CAMs**. Vehicles cannot store in a report a different CAM from the one broadcasted to nearby vehicles.
It is enough to check the correspondence of hashes to identify a cheating attempt.
The following example (graphically represented in Fig. 5.3) is used to clarify the concept.
Two nearby vehicles identified as x and y are able to exchange CAMs. The following stream of CAMs is generated:

1. x sends a CAM and generates a report with $ReportId = 01x$
2. y receives the CAM identified by $ReceivedCamId = 01x$.
   y computes the hash of the received CAM $ReceivedCamHash = Hash(Cam_{01x})$
3. y sends a CAM and generates a report with $ReportId = 01y$
   (including $ReceivedCamId = 01x$ and $ReceivedCamHash = Hash(Cam_{01x})$
   in the list of received CAMs)
4. x receives the CAM identified by $ReceivedCamId = 01y$
   x computes the hash of the received CAM $ReceivedCamHash = Hash(Cam_{01y})$
5. x sends a CAM and generates a report with $ReportId = 02x$
   (including $ReceivedCamId = 01y$ and $ReceivedCamHash = Hash(Cam_{01y})$
   in the list of received CAMs)
6. This process is repeated for all the CAMs exchanged by all the vehicles.

This process is represented in Fig. 5.3 where *R01x, R02x, R03x, R...x* indicate the reports generated by the vehicle x and *R01y, R02y, R...y* indicate the reports generated by the vehicle y. The black arrows indicate the *hashchain of reports* generated by the same vehicle. The blue arrows, instead, indicate how the reports generated by different vehicles are connected, i.e. the concatenation that is obtained with the *hashes of received CAMs*.

Summary of tamper-proof property of reports

Each reports contains two types of hash pointers:

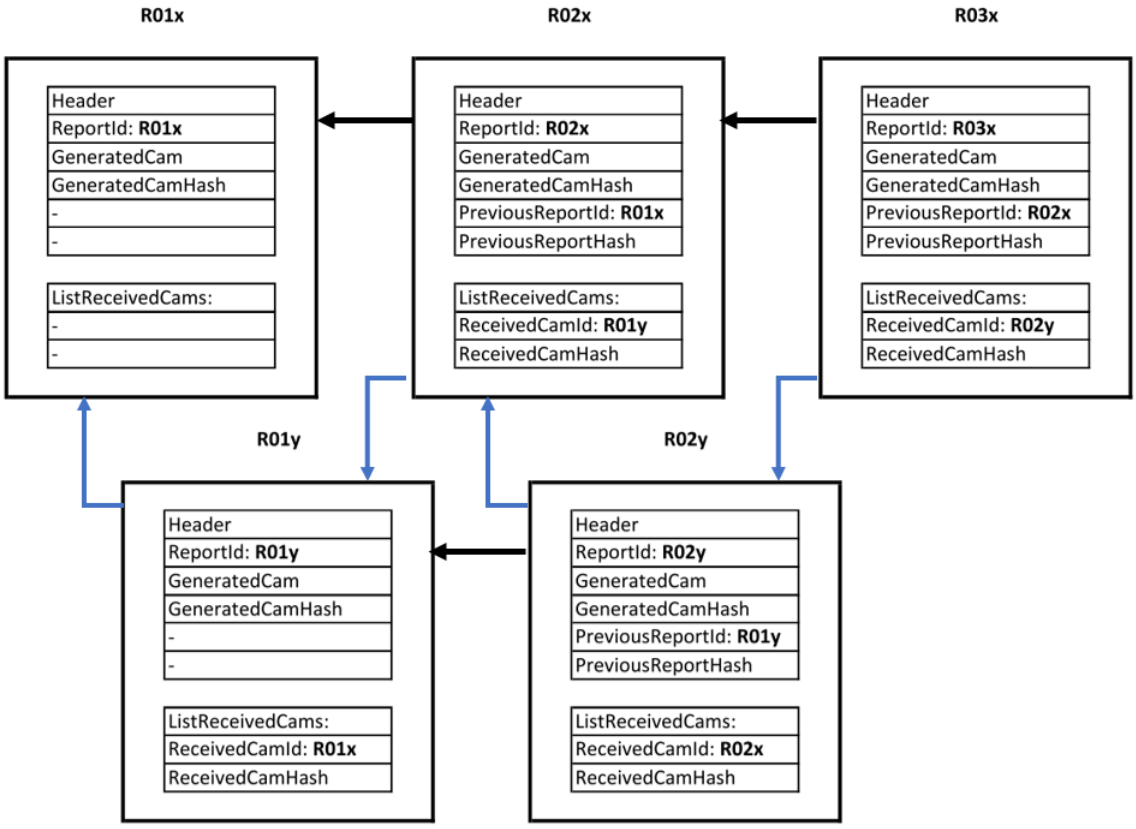


Figure 5.3: Tamper-proof reports generated by the vehicles.

- An hash to connect the current report to the previous report generated by the same vehicle.
- A set of hashes to connect the current report to the reports generated by nearby vehicles.

This configuration makes very difficult to tamper data stored inside reports by inserting fake information.

5.4 The blockchain platform

The blockchain provides a tamper-proof data storage solution that makes possible to store reports. Position verification algorithms for the validation of CAMs can be implemented in the form of smart contracts.
The detailed analysis of available Distributed Ledger Technologies was completed with the identification of **Hyperledger Fabric** as the best candidate for the development of a blockchain based architecture for vehicular applications.

5.4.1 Why Hyperledger Fabric?

Hyperledger Fabric is the blockchain platform that fits better to the requirements of the project under analysis. The main features are shown as follows:

- **High throughput**. Hyperledger Fabric allows to manage a number of transactions per second much higher than that of the other blockchain platforms. The high number of CAMs exchanged between vehicles, in fact, requires to execute many transactions per second to store generated reports.
- **Low latency**. Blocks are committed to the ledger as soon as they are created by ordering service nodes and validated by peers. Sub-second latency may be achieved.
- **Transactions finality**. It is not possible to create forks in the ledger. Valid transactions contained into blocks cannot be invalidated over time.
- **Permissioned environment**. Members of the blockchain are identified by public key certificates. Only authorized users may access the blockchain issuing transactions and accessing contained information.
- **Privacy**. Unauthorized users cannot access data stored into the blockchain. Sensitive information of users are not disclosed.
- **Transparency**. The members of the blockchain can verify executed transactions by playing the roles of committing peers and endorsing peers.
- **Possible cooperation of different organizations**. Different mobile phone network operators and other companies, e.g. insurances, can interact maintaining the state of the ledger and can trust stored information.
- **Disintermediation**. Different companies can interact and manage the ledger without the intervention of a trusted authority controlling the whole process. The role of the trusted authority is limited to the issuing of certificates to authorized members of the blockchain.
- **Chaincodes**. Smart contracts allow to implement position verification algorithms to check the reliability of messages exchanged by the vehicles.
- **Parallel execution of chaincodes**. Different endorsing peers can concurrently execute transactions to store data collected by different base stations.
- **Channels**. It is possible to create multiple independent ledgers, maintained by different subset of peers. This feature can be exploited to implement ledgers containing data relative to limited geographical areas.
- **Pluggable consensus**. It is possible to implement the preferred consensus algorithm, either crash-tolerant or Byzantine-fault tolerant to better fit the application requirements.

5.5 Architecture overview

This section describes the overall architecture of the distributed system, including all the design choices and the roles of different participants of the blockchain.

5.5.1 Vehicles

Vehicles are the fundamental users of the system. They travel, exchange CAMs with nearby vehicles and create reports to be stored in the blockchain. Vehicles do not interact directly with the blockchain, i.e. they are neither clients nor peers. They generate the reports and upload them to base stations using the internet connection provided by their mobile phone network operators. In the event that a vehicle is located in an area characterized by the absence of telephone coverage, reports can be locally stored and must be uploaded as soon as the connectivity is restored.

5.5.2 Base station - eNodeB

What is an eNodeB

The eNodeB, often indicated with eNB, is the acronym for Evolved NodeB and can be considered the evolution of the base station for LTE networks. The eNodeB is an element of LTE network which role is to create a bridge, using wireless communication, between user devices, called UE - User Equipment (in this case the vehicle adopting a 4G USIM), and the mobile network [65]. The access of UE to the mobile network is authenticated thanks to the usage of a USIM [66], in this way it is difficult for an attacker to forge the identity of another user.

Role of the eNodeB in the architecture

The eNodeBs play a fundamental role in the architecture of the distributed system. They provide internet connectivity and authenticate vehicles thanks to the usage of an USIM. Authenticated vehicles can upload reports and are discouraged from providing false information because they can be easily identified.
The base stations acts as **clients** and **peers** of the Hyperledger Fabric blockchain.

- Clients. The base stations execute the transactions to store the reports uploaded by vehicles in the blockchain. Furthermore, they can execute transactions for the validation of stored CAMs. It is a good rule they issue transactions for the validation of the CAMs associated to the reports they requested to store.
- Endorsing peers. The base stations execute the chaincode functions necessary to store the reports and to validate CAMs logged in the blockchain.
- Committing peers. The base stations verify the transactions executed by endorsing peers and update their local copy of the ledger. They also maintain a full replica of the ledger.

5.5.3 Architectural design choices

In Hyperledger Fabric blockchain it is possible to identify different participants, with different roles:

1. Clients
2. Endorsing peers

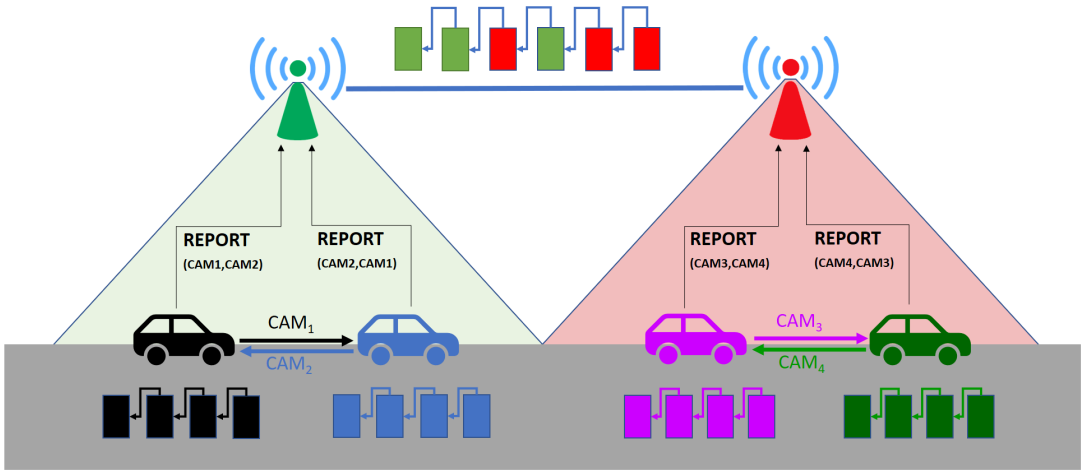


Figure 5.4: Overall architecture of the system.

3. Committing peers
4. Orderers

Base stations act as clients, endorsing peers and committing peers of the blockchain. The role of Ordering Service Nodes is executed by a cluster of Kafka and ZooKeeper nodes. This cluster is set up by different network operators and allows to separate the endorsement and the validation of transactions from the block creation, increasing the efficiency of the blockchain.
The vehicles do not interact directly with the blockchain for performance reasons. The number of vehicles is large and Hyperledger Fabric would require vehicles to be authenticated and authorized to issue blockchain transactions. This model would generate an increase in the number of transactions to be managed and a higher computational overhead.
Vehicles cannot play the role of peers or orderers of the blockchain. They cannot be considered trusted members, their availability is not guaranteed, and they are characterized by low computational and storage capabilities.
On the contrary, the strategy adopted in this thesis allows vehicles to upload their reports to the eNodeBs. After a limited number of reports has been collected, each eNodeB can issue a transaction to store multiple reports simultaneously into the blockchain. This strategy permits to reduce the computational overhead required by the endorsement, ordering and validation of transactions. In addition, the authentication of users is delegated to mobile phone operators, blockchain is unaware of the real identity of vehicles.
Fig. 5.4 shows the architecture of the system and the data stored by the participants. Each vehicle store the hash chain of reports generated over time. The green and red chain of blocks shows that the base stations are interconnected and locally store a complete replica of the ledger containing all the reports generated by the vehicles.
Fig. 5.5 describes the interactions between the actors of the system. In *step 1* the vehicle uploads the report to the base stations. In *step 2* and *3* the base station invokes and endorses the transaction to store the collected reports in the blockchain. The transaction is then transmitted to the ordering service nodes, based on Kafka cluster, that order the transactions and create the block (*step 4*). Finally, the created block is distributed to the committing peers so that it can be validated and

committed (*step 5*).

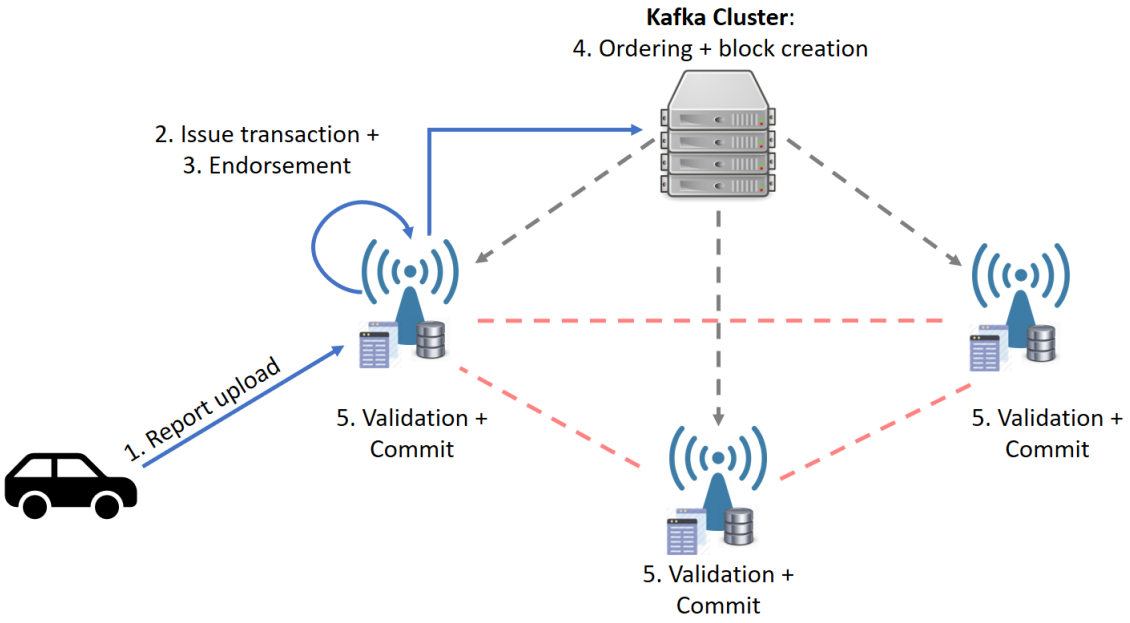


Figure 5.5: Interactions between the actors of the distributed system.

5.5.4 The role of network operators

Hyperledger Fabric allows the cooperation of different organizations, identified by their trusted root certification authority and by their members. Each organization can manage different members like peers, orderers and clients and interacts with other organizations to maintain a replica of the ledger.
In the described architecture, different organizations correspond to different **mobile phone operators**. Each mobile operator manages its base stations, that correspond to clients and peers of the blockchain. Base stations of different mobile phone operators interacts to manage to store reports and to validate contained CAMs.

5.5.5 Ledgers for the management of geographical areas

Hyperledger Fabric allows to create different **channels**, namely separated ledgers that allow to increase the privacy of stored information. Only peers belonging to a channel can access and retrieve stored data. Peers can access simultaneously to different channels.
In the described architecture, the channels can be used to create different ledgers relative to well-defined geographical areas of competence. Fig. 5.6 shows many base stations that, depending on their geographical position, store data of different channels. The letters *A, B, C, D, E, F, G* identify the channels and the relative geographical areas. A base station must join the channels corresponding to the areas covered by its own cell site and by those of neighbouring base stations (potentially of different network operators). Reports collected by a base station are stored only in one of the channels of the blockchain. Instead, the validation of reports requires

to retrieve data stored in different channels, so that it is possible to obtain an overall view of all CAMs stored in the distributed system. This architecture offers many advantages:

- Each peer of the network maintains a replica of a reduced portion of data, correspondent to joined channels. The validation and endorsement of transactions are executed only by peers belonging to the channel.
- The possibility to interact with different channels allows to have a broader view of neighbouring geographical areas, enabling to safely apply algorithms of position verification.

On the other hand, the complexity of the system is increased because of the management of different separated ledgers. In fact, to apply position verification algorithms it is necessary to retrieve CAMs stored in different channels, increasing the number of interactions with the blockchain.

Possible attacks to a multi-channel architecture

Another problem relating to the management of different channels is the identification of cheating vehicles. When an attacker steals the identity of another user, it can send CAMs as if they came from the victim's vehicle. In a scenario characterized by one channel only, it is possible to execute a query to understand that the victim's vehicle is simultaneously located in different geographical regions. The management of different channels makes it difficult to identify this anomaly. In fact, it is to execute different queries to different channels to identify the cheating attempt. This problem is mitigated by the use of the CAM security envelope for CAMs transmission (see Sec. 5.2.3) and by the adoption of the tamper-proof data structure of reports (see Sec. 5.3.2).

5.5.6 Validation of CAM messages

The application of position verification algorithms to CAMs contained in reports is possible only after the data has been committed to the ledger.

Hyperledger Fabric requires that the blocks containing the transactions are committed by peers before the state of the ledger is updated. Two subsequent transactions, the first used to store a CAM and the second used to apply a position verification algorithm to the same CAM could be incompatible. If the first transaction has not yet been validated by committing peers, i.e. the block containing the transaction has not yet been committed, the transaction used to verify the position of the vehicle cannot access the CAM previously stored. Furthermore, position verification algorithms require to access the reports potentially generated by different transactions and peers. In practice, it is necessary to introduce a waiting time between the the storing and validation procedures. This characteristic does not allow to apply position verification algorithms in real-time.

Overall validation process

As mentioned in previous paragraphs, the execution of position verification algorithms requires that the reports are stored in the blockchain. The overall validation process is shown in Fig. 5.7 and can be summarized as follows:

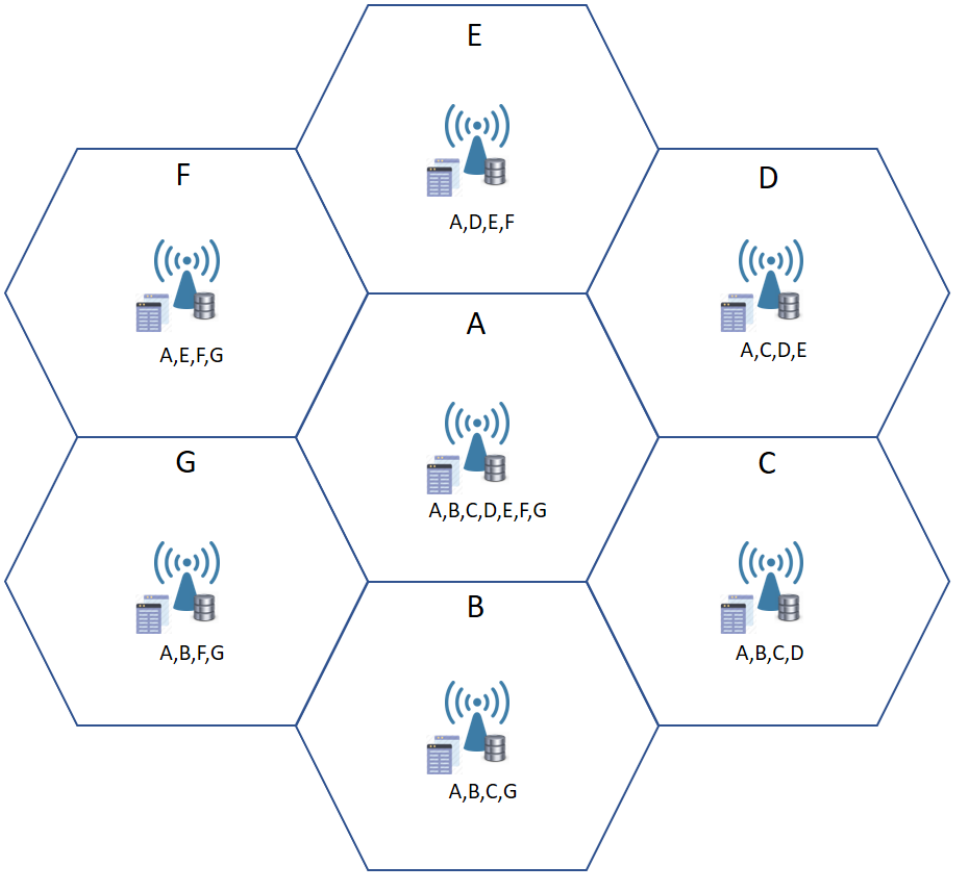


Figure 5.6: Architecture composed by many channels storing data of different geographical areas.

1. Some vehicles exchange CAMs, generate reports, and upload the reports to the base station to which they are connected.
2. The base station, playing the role of client of the blockchain, issues a transaction to store in the blockchain the reports received by vehicles.
3. After the transaction has been committed, the base station introduces an additional waiting time to ensure that neighbouring eNodeBs have stored the previously collected reports in the blockchain.
4. The base station issues one or more transactions to apply position verification algorithms to CAMs previously stored in the blockchain.
5. The base station collects and analyses the result of the validation as soon as the transaction has been committed.

Position verification algorithms

The literature proposes many papers about the possible implementations of position verification algorithms, like [67] and [68]. In this thesis, a simple algorithm has been implemented in the form of a Fabric chaincode to test the performances of the

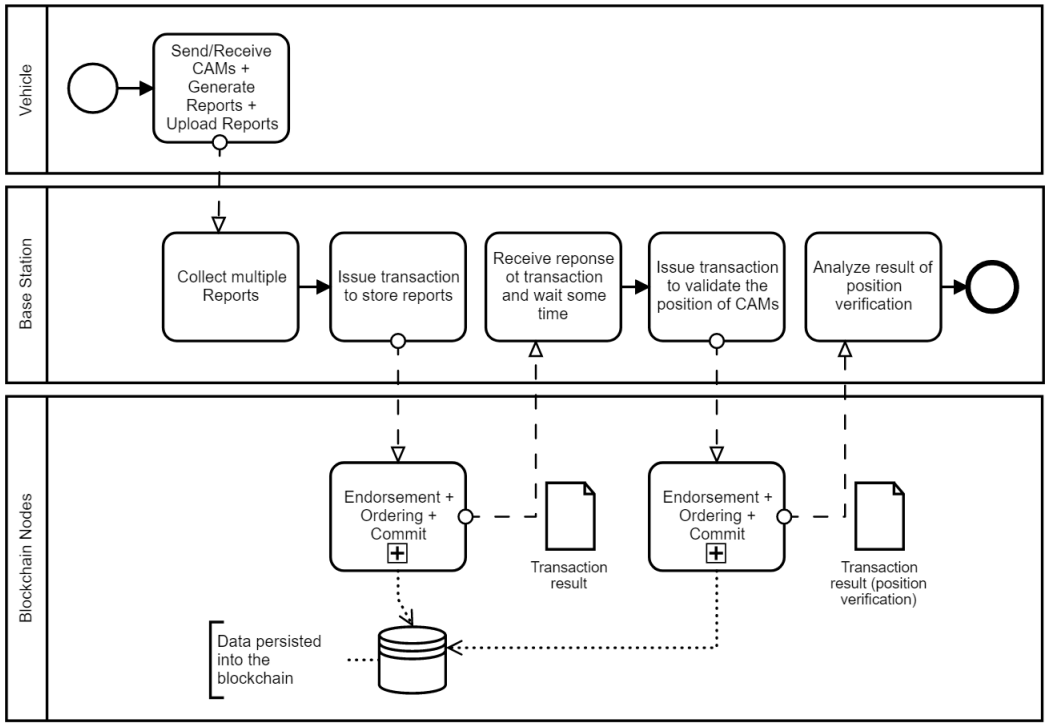


Figure 5.7: Process of storage and validation of the CAMs

proposed architecture. This algorithm is **executed by the endorsing peers** of the blockchain as a result of a transaction request issued by a client to require the validation of the CAMs. The algorithm can be outlined as follows:

1. The chaincode function reads the identifier of the CAM that must be verified.
2. A query is sent to the state database of the blockchain to retrieve the report containing the transmitted CAM and all the reports of the nearby vehicles that received the CAM under analysis.
3. Two variables are initialized to zero. They are used to count how many vehicles consider the transmitted CAM as potentially valid or not valid.
4. For all the reports of the vehicles that received the CAM, it is estimated if the position declared by the CAM under analysis is compatible with the reception information of the CAM. Depending on the result, the counter indicating if the position is potentially valid or not valid is incremented. The estimate depends on the instant of transmission and reception, the propagation time and the mutual position of transmitter and receiver of the CAM.
5. The algorithm is concluded comparing the values assumed by the two counters. The greatest value is used to classify the transmitted CAM as valid or not. The blockchain is updated to store the result of the position verification.

This procedure must be repeated for all the CAMs passed as parameter to the chaincode function. A problem derived by the implementation of position verification algorithms consists in the update of the ledger. The append-only property of the blockchain does not permit to overwrite already stored data and implies that

information of the validated CAMs is duplicated. This problem can be mitigated by updating only the information of the CAMs marked as not valid.

Importance of the validation

Validation of CAMs is fundamental to use data stored into reports for legal purposes. For example, in case of a road accident, insurance companies must start a procedure to resolve the conflict, identifying the faulty car for the attribution of damage. The information collected via exchanged CAMs can be very useful but it must be validated, it is necessary to understand if data provided by both vehicles can be considered reliable or not. There is the possibility some vehicles provide wrong information because of malfunctions or because they have been tampered.
Validation of CAMs stored into the blockchain can be executed periodically to identify vehicles systematically providing wrong information. Results of validation process can be analysed and further inspections can be executed to understand the reasons of these faults.

5.5.7 Limits of proposed architecture

The proposed architecture faces an important problem: a **very large amount of data must be stored in the blockchain nodes**. Storing the reports generated by vehicles requires to manage not only data relative to generated CAMs, but also the overhead relative to reception information (see *ReceptionInfo* in Sec. 5.3.1). Hyperledger Fabric requires that the committing peers of the blockchain maintain a complete replica of the ledger. No other alternatives like *sharding* are possible. The blockchain platforms, in fact, produce an always growing record of transactions and blocks. If intermediate blocks are removed it is violated the tamper-proof property of the blockchain. Hyperledger Fabric adopts a **pruning** functionality to remove invalid transactions from the local replica of the ledger [50] (at time of writing this functionality has not yet been implemented). In view of the fact that the majority of transactions is valid, the benefits derived from pruning can be considered negligible. The multi-channel architecture explained in Sec. 5.5.5 allows to mitigate the problem, by reducing the amount of data that must be stored by the peers of the blockchain.

5.5.8 Comparison with other existing architectures

The literature proposes vehicular applications based on blockchain storage solution characterized by an architecture notably different from the one proposed in this thesis. Some examples are [69] and [70].

Architecture proposed by Leiding et al.

The solution proposed in [69] is based on the Ethereum smart contracts.
Each vehicle directly interacts with the blockchain using its pair of asymmetric keys. The vehicles require the execution of a set of applications and pay transactions fees (gas) for this purpose [69]. The payment of transaction fees may seem inadequate, especially for the services that are required by law. On the other hand, fees can be used to cover the infrastructure maintenance costs, including the RSUs and

the electricity required by miners [69]. This architecture, in fact, is based on a permissionless blockchain and requires the existence of miners. The role of miner can be covered by some infrastructure devices like RSUs and base stations.
The architecture proposed in this thesis is probably more efficient than the one proposed in [69]. Table 5.5.8 shows some differences of the two architectures.

Architecture of [69]	**Architecture of this thesis**
It is based on permissionless Ethereum blockchain platform.	It is based on permissioned Hyperledger Fabric blockchain platform.
The vehicles interact directly with the blockchain	The vehicles do not interact with the blockchain, they only upload data to base stations.
The vehicles pay transaction fees for all the required services. The fees depend on the number and type of transactions executed.	The vehicles do not pay transactions fees.
High number of transactions due to the high number of clients of the blockchain (vehicles).	The number of transactions can be tuned according to the number of reports stored with a single transaction (decided by the base stations).
Creation of a global vehicular blockchain infrastructure (very high storage requirements for nodes).	Possibility to create channels relative to limited geographical areas (lower storage requirements for nodes).

Table 5.3: Comparison of the architecture proposed in [69] and the one proposed in this thesis.

Architecture proposed by Sharma et al.

Authors of [70] proposes a totally different blockchain architecture for vehicular applications focused on scalability and availability.
In this architecture, vehicles are divided in two categories, *miner nodes* and *ordinary nodes*, corresponding to different roles. These roles are assigned to the vehicles by an authority [70].
Controller nodes identify the infrastructure devices, e.g. base stations. They interact with other nodes of the network by sharing data. Ordinary nodes can only access data provided by miner nodes and by controller nodes [70]. Miner nodes manage data collected by different sensors, process and store it locally so that this information can be distributed to other network participants [70]. Controller nodes operate similarly to miner nodes but provide higher computational capabilities. They create a network to allow the exchange of information between vehicles [70].
Drivers and vehicles are represented by means of smart contracts to provide a multitude of services, for example the management of a wallet for payments, the identification of fuel stations, car-pooling service, etc. This architecture faces with an important problem, the availability. Vehicles are not guaranteed to be available and

to be able to provide sufficient computational capabilities.
Comparing this architecture with the one described in this thesis it is possible to highlight some important differences. Table 5.5.8 shows some differences of the two architectures.

Architecture of [70]	**Architecture of this thesis**
Not based on a previously existing blockchain platform. Ad hoc blockchain architecture.	Based on permissioned Hyperledger Fabric blockchain platform.
The vehicles play different roles (miners or ordinary) and interact with the blockchain nodes.	All the vehicles play the same role, they upload data to the base stations and do not interact directly with the blockchain.
Miner vehicles store collected data locally.	Vehicles store collected data only temporarily before the submission to the base stations.

Table 5.4: Comparison of the architecture proposed in [70] and the one proposed in this thesis.

Chapter 6

Applications Developed to Test Hyperledger Fabric

This chapter describes the programs developed to test the performance of Hyperledger Fabric when it used to implement the vehicular application. Sec. 6.1 describes the vehicular simulator that creates the traces of reports containing the CAMs transmitted and received by the vehicles. Sec. 6.2 shows how to deploy a working Hyperledger Fabric blockchain and some implementation details of the chaincode functions used to manage the CAMs. Finally, the chapter describes the benchmarking tool that has been developed to test the performance of Hyperledger Fabric (Sec. 6.3).

6.1 Vehicular Mobility Simulation

The vehicular mobility simulator is an application used to simulate the movements and interactions of vehicles in a urban area. A simple implementation of a vehicular simulator has been implemented using the Python programming language. The simulator is able to simulate the exchange of CAMs between vehicles and the generation of reports.

6.1.1 General description of the simulator

The simulator implements a simplification of **Random Waypoint Mobility Model** [71]. In a Random Waypoint Model, the vehicles execute arbitrary paths in a predefined area, without constraints on the parameters characterizing the movement. Vehicles randomly generate the places of departure and destination and travel with a constant speed [71]. Speed of vehicles is uniformly distributed between a minimum and a maximum value [71]. When a vehicle reaches a destination, it waits a random pause time, it generates a new destination and a speed and it restarts travelling [71].

Simplifications of the proposed implementation

The developed simulator is characterized by a set of simplifications described as follows:

- Speed of vehicles is fixed to 10.0 m/s.
- Acceleration of vehicles is fixed to 0.0 m/s^2

- Vehicles start a new journey as soon as they reach their destination.
- The algorithm is executed on a squared simulation area.
- Collisions between vehicles are not considered.

Input parameters of the simulator

The proposed implementation allows to customize some simulation parameters:

1. The number of travelling vehicles.
2. The number and the location of RSUs (expressed in a Cartesian landmark).
3. Width of the squared simulation area (in meters)
4. CAMs transmission radio range (radius in meters of a circular area).
5. CAMs generation frequency (expressed as number of seconds elapsed between two consecutive CAMs transmissions)
6. Simulation time (seconds)
7. Cheating probability (probability that a vehicle transmits wrong information inside a CAM. It is expressed as relative frequency).

Output of the simulator

The output of the simulator is in a text file containing the list of all reports generated by vehicles in the form of JSON documents. Optionally, it is possible to generate a chart describing the paths of the vehicles. Fig. 6.1 represents the result of a simulation executed with 2 vehicles that travel in a squared area of $1km^2$ in a time interval of 200s. The black circles represent the starting position of the vehicles while the red and green dotted lines represent the paths of the first and second vehicle respectively.

Management of CAMs and reports

The simulator manages the exchange of CAMs between nearby vehicles according to an ideal model. All the vehicles of the simulator transmit and receive CAMs. Every vehicle receives a CAM transmitted by another vehicle if the following inequality is respected :

$$d(tx, rx) < r_r$$

where $d(tx, rx)$ indicates the Euclidean distance between the transmitter and receiver computed at transmission time, and r_r is the radio range passed as parameter to the simulator. The time of generation of CAMs at application layer coincides with the time of transmission of the CAM at physical layer. Each CAM is propagated at the speed of light. As a consequence, the time of transmission of a CAM is different from the reception time. Every time a vehicle receives a CAM, the reception information (position, speed, acceleration, direction, timestamp) is stored. The reports are generated just after the transmission of CAMs. They contain all the CAMs that the vehicles received since the previously generated report.

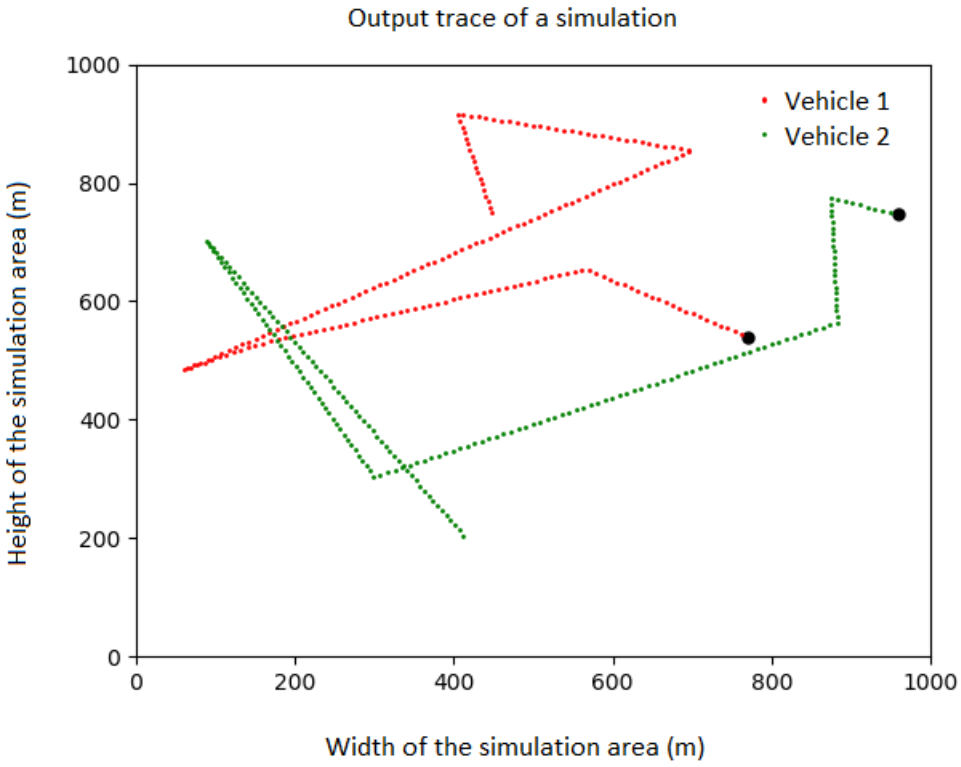


Figure 6.1: Results of a simulation conducted with 2 vehicles.

The example of Fig. 6.2 shows a vehicle ($Tx1$) transmitting a CAM. The vehicle $Rx1$ receives the CAM because $d(Tx1, Rx1) < r_r$, while $Rx2$ does not receive the CAM.

Cheating vehicles

It is possible some vehicles broadcast CAMs containing false location information. The probability that a vehicle sends false information is related to the cheating probability. If a vehicle is identified as a cheater, it always broadcasts wrong location information. On the contrary, if a vehicle is honest, it always provides correct information.

Versions of the simulator

Two versions of the simulator has been implemented:

1. Version without tamper-proof reports. It has been used for the majority of experiments of this thesis. Reports are not logically linked to previously generated one.

2. Version with tamper-proof reports. Each report contains the hash of previously generated report and the hashes of all received CAMs.

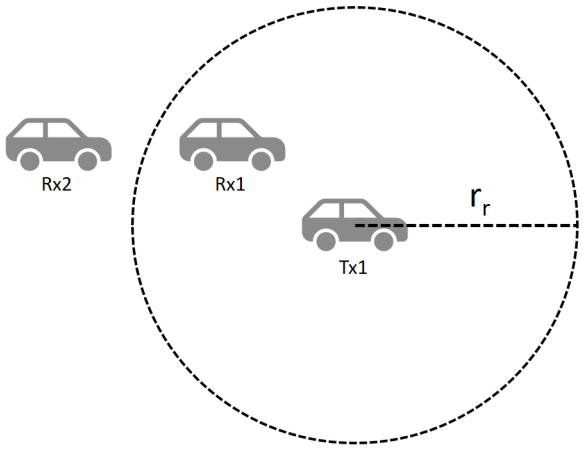


Figure 6.2: Transmission and reception of a CAM

6.1.2 Description of the algorithm

The developed simulator is event-based. It is based on a chronologically ordered list of events. All the operations executed in the system are modelled as an event, characterized by a timestamp that is used to define the handling order of events. The event with the lowest timestamp value is the first to be removed from the list in order to be executed.

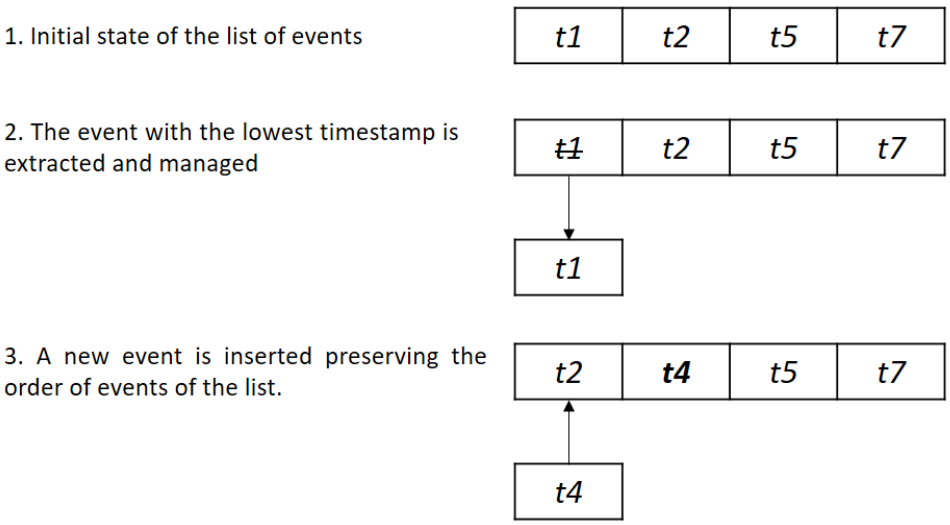


Figure 6.3: Management of the events of the simulator.

6.1.3 Management of events

The simulator manages three types of events:

1. Event for the management of a vehicle reaching its destination.

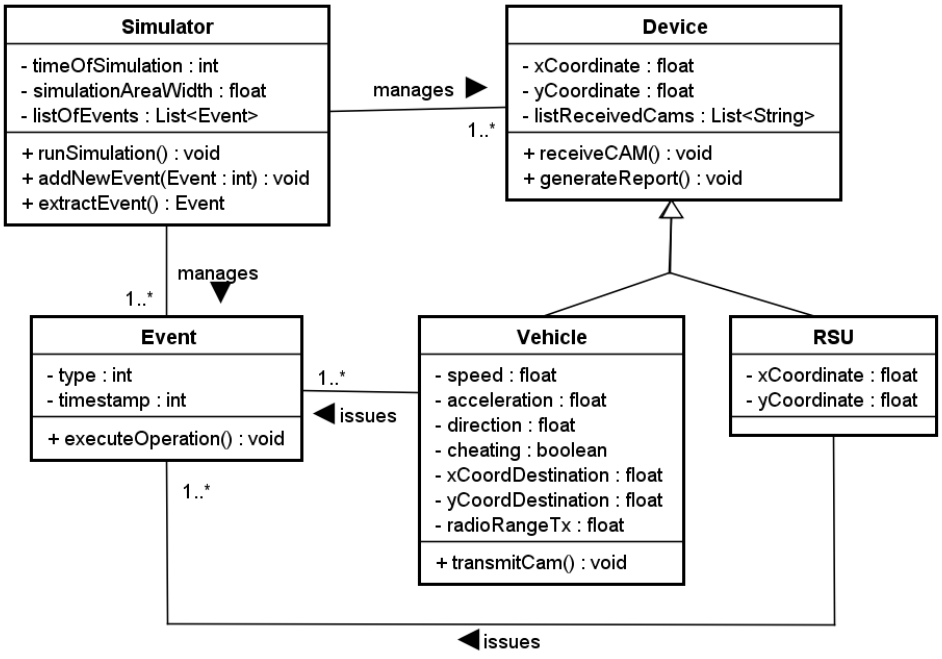


Figure 6.4: UML diagram describing the main classes of the simulator

2. Event for the transmission of a CAM and for the creation of a report (vehicles)
3. Event for the generation of a report (RSUs)

Set up

The algorithm starts configuring the simulation according to parameters provided by the user and initializing an empty list of events.
For each vehicle the departure and destination coordinates are randomly generated together with the timestamp of the departure. The events correspondent to the transmission of the first CAM and to the arrival of the vehicle to its destination are inserted into the list of events of the simulator (arrival time can be computed because the speed of vehicles is considered constant).
For each RSU it is created an event to manage the generation of a report. The event is inserted into the list of events of the simulator.
The simulator starts and infinite loop extracting and managing events in sequence.

Event for the transmission of a CAM and the creation of a report (vehicles)

Every time a vehicle transmits a CAM it updates its geographical coordinates, it creates a report containing the new CAM and the list of received CAMs. The report is appended to an output file and the list of received CAMs is cleared.
The nearby vehicles of a transmitter update their location, add reception information of the CAM and append the result to their local list of received CAMs.
The transmitter schedules the next event of transmission a CAM. The latter is then inserted into the list of events of the simulator.

Event for the generation of a report (RSUs)

Every time a RSU generates a report, it clears the list of received CAMs and appends the report on an output file. The RSU schedules another event for the generation of a report. The latter is inserted into the list of events of the simulator.

Event for the management of a vehicle reaching its destination.

Every time a vehicle reaches its destination, it updates its location, generates new destination coordinates and computes the travel time. A new event for the management of the vehicle reaching the new destination is scheduled and inserted into the event list.

6.2 Hyperledger Fabric project

This section analyses the sequences of operations that must be executed in a single hosting machine to set up a working Hyperledger Fabric blockchain, starting from the installation of fundamental software components, up to the deployment of the chaincode. This section is not intended to be a replacement of the documentation, but it is used to explain the fundamental concepts behind the deployment of an Hyperledger Fabric blockchain. For the technical details and instructions necessary to install Fabric, the reader must refer to the official documentation written by the community and continuously updated[1].

6.2.1 Set up of the working environment

Hyperledger Fabric is compatible with Linux, Mac OSX, and Microsoft Windows and the set of tools and components is common to all the supported operating systems. The operating system used for the development of this thesis project is **Ubuntu 16.04**, thus all the commands shown in the following sections refer to it. The Hyperledger Fabric version that has been installed and used for the whole thesis project is **version 1.1.0-preview**.

Software prerequisites

Hyperledger Fabric and all the required software components are released with an open-source license and can be freely downloaded from the Internet. The list of software that must be installed to run Hyperledger Fabric is shown as follows:

- cURL (latest version)
- Docker (version 17.06.2-ce or greater)
- Docker-compose (version 1.14.0 or greater)
- Go programming language (version 1.9.x)
- Node.js (version 8.9.x - version 9.x is not supported)

[1]Documentation available at: `http://hyperledger-fabric.readthedocs.io/en/latest/`

- Npm (version 5.6.0)
- Python (version 2.7)
- Hyperledger Fabric Examples [2]
- Hyperledger Fabric Binaries (in a fabric-samples folder). They include
  - cryptogen
  - configtxgen:
  - configtxlator
  - peer

The Hyperledger Fabric examples folder contains a set of useful network configurations that can be used to test the installation of Hyperledger Fabric.

6.2.2 Set up of a complete network

Hyperledger Fabric requires to execute multiple operations to create a working blockchain network and to interact by means of the execution of transactions. The list of operations can be described as follows:

1. Generation of the cryptographic material.
2. Generation of the channels configurations.
3. Configuration of the network participants and deployment of Docker containers.
4. Creation and participation to channels.
5. Set up of chaincodes and execution of chaincode functions

All these steps are analysed in detail in this section and refer to the deployment of a Hyperledger Fabric network on a single hosting machine.

Generation of the cryptographic material

As shown in Sec. 3.4.2, Hyperledger Fabric is composed by multiple entities, i.e. peers, orderers and clients, that must be provided with valid X.509 certificates to interact with the network.
The **cryptogen** tool must be used to generate the cryptographic material required by the network participants. This program reads a configuration file (named **crypto-config.yaml** [3]), indicating how many organizations, peers and orderers cooperate in the system [44]. This set of information is fundamental because *cryptogen* creates a folder (called *crypto-config*) containing the set of public key certificates, public and private keys necessary for the authentication and authorization of all the members of the blockchain [44].

[2] Examples available at `https://github.com/hyperledger/fabric-samples.git`

[3] Example available at: `https://github.com/hyperledger/fabric-samples/blob/release/first-network/crypto-config.yaml`

Generation of channels configuration

The different peers and orderers must interact with different channels, maintaining a copy of the ledger. In addition, multiple organizations can simultaneously cooperate and exchange data.
The **configtxgen** tool can be used to set up the different channels and organizations, and to define the so-called *anchor peers* that permits the interaction with the peers belonging to other organizations [44]. These configurations must be specified in a appropriate file, named **configtx.yaml**[4]. In this file it is possible to identify four main sections:

- *Profiles* section. It is used to define the consortium of organizations that can interact, the name of the channels and the set of organizations authorized to participate in the channels.
- *Organizations* section. It is used to characterize the different organizations that will take part to the system, including their name, their unique identifiers and the set of anchor peers.
- *Orderer* section. It allows to specify some information about the orderers of the system, including their hostname and port used for the communication. In addition it is possible to define some parameters about the blocks that will be created in the blockchain, like its maximum size, the maximum number of transactions that can be contained, etc.
- *Application* section. It is used to configure the organizations to which the clients of the blockchain belong to.

The *confitgxgen* tool generates three different outputs, described as follows [44]

- The genesis block used by orderers for the creation of the blockchains.
- The configuration transactions of the different channels characterizing the blockchain.
- The anchor peer transactions that are used to configure the inter-organization communication of peers.

Network configuration and deployment of Docker containers

The different members of the blockchain are deployed inside isolated Docker containers. The network configuration can be defined in the **docker-compose.yaml** file[5], that includes the definitions of the containers of all the different participants of the system (with the exception of clients). In particular, it defines the configuration of the following types of containers:

[4]Example available at `https://github.com/hyperledger/fabric-samples/blob/release/first-network/configtx.yaml`

[5]Example available at `https://github.com/hyperledger/fabric-samples/blob/release/first-network/` or `https://github.com/hyperledger/fabric-samples/blob/release/basic-network/docker-compose.yml`

- *Peer* container. For every peer it is defined its hostname, IP address and ports, the timeout of execution of the chaincode, the addresses of the anchor peers it must refer to for communicating with the members of other organizations, the IP address and port of the CouchDB state database (optional), etc.
- *Orderer* container. For every orderer it is defined the IP address, ports, hostname, etc.
- *Certification Authority* container. It is the CA that releases the certificates to enable secure TLS communication between peers, orderers and clients of the blockchain.
- *CLI* container. The CLI containers allow to ease the execution of command line instructions. The commands issued inside a CLI container, in fact, are forwarded to the other containers as if the were executed directly by the peers or by the orderers. A single CLI container can be used to communicate with different peers by simply configuring the appropriate environment variables.
- *CouchDB* container. It is the container that hosts the CouchDB state database. It allows to configure the IP and ports to allow the interaction of the peers with the CouchDB state database. This container is deployed only if CouchDB is used as state database. The adoption of LevelDB as state database, in fact, does not require an ad-hoc container because its process is executed by the peer container.

The *docker-compose.yaml* file is passed as argument to the **docker-compose** application to automatically deploy all the containers defined in the configuration.

Creation and participation to channels

When the containers of the different members of the blockchain has been started, it is necessary that at least one channel is created and that all the authorized peers join the channel so that they can interact with each other. For this purpose, one of the peers of the organizations must interact with the orderers to require the creation of a new channel by issuing the following command:

```
peer channel create [flags]
```

This command is used to submit a request to one of the orderers of the blockchain, so that it can create the **channel genesis block**. The `create` command requires to specify as parameters the name of the new channel and the *configuration transaction* file previously generated with the *configtxgen* tool. As a result of this operation, the orderers create the **genesis block** for the new channel and send it to the peer. The latter stores this file locally so that it can be used to allow all the authorized peers of the blockchain to join that channel.

All the peers of the blockchain must join the channel to be able to interact with the other members. This can be done with the following command:

```
peer channel join [flags]
```

specifying which is the genesis block of the channel they need to join.

The overall procedure is then concluded with the definition of the **anchor peers** of the different organizations taking part to the channel. This step is fundamental to allow peers of different organizations to interact with each other. This operation can be executed with the command:

```
peer channel update [flags]
```

Set up of chaincodes

When all the peers of the blockchain have been successfully configured to communicate in a channel, they can interact by using the same chaincodes. In this paragraph it is supposed that the chaincode has already been developed (as shown in Sec. 6.2.3) and it is only necessary to configure it for the interactions of the different peers. Many chaincodes can be **installed** in the peers of the blockchain so that they can endorse and validate the transactions. The following command can be used to install the chaincode in a peer:

```
peer chaincode install [flags]
```

Before it can be used, the chaincode must be **initialized** by using:

```
peer chaincode instantiate [flags]
```

During the initialization it is possible to define the **endorsement policy** of the chaincode.

Endorsement Policy definition In version 1.0 of Fabric the endorsement policy is defined at chaincode granularity, i.e. all the functions associated to the chaincode must satisfy the same endorsement policy. The endorsement policy can be specified by as an option of the `instantiate` command [44]. The policy is defined by using boolean operators (*and, or*) and specifying the number and the identity of peers of the different organizations that must endorse the transactions so that they can be considered valid by committing peers [44]. If the endorsement policy is not specified, the default one is used, considering as valid every transaction endorsed by at least one peer of one organization. Some examples of valid endorsement policies are described as follows:

1. `-P OR (org1.member, org2.member)` : it requires that at least one member of the two organizations endorse the transactions.
2. `-P AND(org1.member, org2.member)` : it requires that at least one member of each of the two organizations endorse the transactions

The endorsement policy is fundamental for characterizing the performances of a Fabric blockchain, because it is strictly related to the number of transactions that can be simultaneously endorsed in the blockchain.

Chaincode functions invocation

Two different commands can be used to execute read or write operations:

```
peer chaincode invoke [flags]
```

This command can be used to execute write operations. It requires the endorsement, ordering and validation of the transactions. Every executed transaction is inserted in a block and recorded in the blockchain.

```
peer chaincode query [flags]
```

This command can be used to execute read-only operations. If the chaincode functions contains operations that updates the state of the ledger, the execution of the query does not fail but the modifications to the ledger are not persisted, i.e. no new blocks are created to record the execution of the transaction. The execution of queries, in fact, do not require the execution of ordering and validation phases of transactions; the result of the queries is passed to the client application as soon as the endorsement has been completed.

Chaincode upgrade

It is possible to modify to the code of existing chaincodes and to upgrade the chaincode so that the modifications become effective [44].
The following commands can be used to update an already existing chaincode [44]

```
peer chaincode install [flags]
peer chaincode upgrade [flags]
```

Before upgrading the chaincode it is necessary to install the new version in all the peers of the blockchain [44]). Then, one of the peers issues the `upgrade` command to switch to the new version of the chaincode. In this way the initialization of the chaincode is executed again [44].

Comments on the set up of the networks

In this thesis project different blockchain networks has been set up. Because of the complexity of the overall procedure described in previous paragraphs, a bash script has been written to speed up the deployment of the Fabric networks. The Hyperledger Fabric example folder contains some scripts to deploy basic network configurations [6]. The configuration of the networks composed by many peers can be problematic, especially if they are characterized by different organizations, because a lot of configuration files must be edited. For this reason, do deploy some complex network configurations, it has been used the **netcomposer** tools [7] that allows to simplify and speed up the set of operations.

[6] Example of a script for configuring the network available at `https://github.com/hyperledger/fabric-samples/blob/release/first-network/byfn.sh`

[7] netcomposer available at `https://github.com/ibm-silvergate/netcomposer`

6.2.3 Chaincode for the management of vehicles reports

In Hyperledger Fabric it is possible to write the chaincodes with Go or NodeJS programming languages. The application of this thesis for the management of the CAMs exchanged by the vehicles, has been developed in Go because it is the first language officially supported by Fabric. The functions that can be execute with the thesis chaincode can be described as follows:

1. Storage of the reports generated by the vehicles (write only operation).
2. Verification of the position declared by transmitted CAMs through the application of position verification algorithms (read-write operation).
3. Query to retrieve stored CAMs (read-only operation).
4. Query to retrieve the history of modifications of the stored CAMs (read-only of the history index)

The implementation of these operations is fundamental to test the performances read and write performance of Hyperledger Fabric.

1. Storage of the reports generated by the vehicles

As shown in Sec. 5.3.1 every report is composed by different fields and, among other things, it contains:

- The last CAM transmitted by the vehicle.
- The list of CAMs received by the vehicle (paired with the reception information).

This application manages the reports formatted as JSON documents, exactly as they have been generated by the vehicular simulator. Storing the reports in the form of JSON documents is not a good solution because it requires to execute complex queries operating on the fields of the JSON document to retrieve transmitted and received CAMs. As mentioned in Sec. 3.4.1, the execution of complex queries is possible for read-only operations, and if only if CouchDB is used as state database [44]. In reality, the complex queries can be used also for read-write transactions but their adoption is discouraged because the key-value pairs accessed with a complex query are not considered part of the read-set of the associated transaction [44]. Therefore, the committing peers, during the validation phase of transactions, might accept some transactions whose read-set is not complete and potentially in conflict with respect to the current state of the ledger [44]. In the light of these considerations, it has been decided to manipulate the content of the reports to create **many key-value pairs** for each report, one for each transmitted and received CAM. This processing is executed directly by the chaincode function that is in charge of storing the reports. In particular, every **key** of the key-value pairs is a string formatted as follows:

`<timestamp>_<transmitterID>_<receiverID>`

where *timestamp* is the absolute timestamp (in milliseconds) of the transmission of the CAM, while *trasmitterID* and *receiverID* are the identifiers of the transmitter and of the receiver of the CAM (these identifiers are the ones contained into the header of the last CAM that each vehicle has transmitted).
In every report, it is always contained a transmitted CAM; the key-value pair associated to this CAM is characterized by the value of *receiverID* equal to the value of *trasmitterID*.
The **value** associated to a key, instead, is formatted as a JSON document and includes the transmission and reception information of the CAM, i.e. all the pieces of information that have been exchanged by the vehicles. This particular key-value pairs format allows to easily retrieve the transmission and reception information of the CAMs, that are necessary to apply the position verification algorithms.
Note. In Hyperledger Fabric it is possible to create the so-called *Composite Keys*, i.e. keys composed by different fields that can be exploited to execute range queries based on different fields and information [46]. For simplicity, the keys used in this application are not composite keys but they have been implemented as a unique strings, obtained as the concatenation of the different fields.

Design choices of the storage function The storage function has been implemented to store multiple reports with a single transaction. The computational overhead introduced by the execution of the transactions increases the time required to store the data in the blockchain. For this reason, the storage function has been designed to read a local file containing many reports collected by the base station. With a single blockchain transaction is this way possible to store many reports and to speed up the storage operation of the CAMs.

Alternative reports storage solution An alternative storage function has been implemented to store the reports without processing their content. This alternative implementation is fundamental to test the storage overhead introduced by the blocks and by the transactions, but it has never been used to apply position verification algorithms.

2. Query of the chaincode

A chaincode function was implemented to execute the *range queries* on the CAMs stored in the blockchain. For this purpose, it is necessary to provide two parameters, indicating respectively the lower and upper bound of the keys to retrieve.

3. Validation of the position of CAMs

The position verification algorithm implemented as a chaincode function is a simplification of the algorithm proposed in [67]. The purpose of this simplification consists in testing the read and write performance of the blockchain but it is not conceived for real applications. The classification of the cheating vehicles, in fact, is possible only if the geographical coordinates declared by the cheaters are notably different from their real location. To allow the correct classification of the CAMs, the vehicles traces generated by the simulator take into account this simplified implementation of the position verification algorithm, i.e. the cheating vehicles always declare a false position which is at least 250 meters away from their real location.

The problem of amount stored data The application of the position verification algorithm requires that all the transmitted and received CAMs have already been stored in the blockchain. The verification of the location declared by the vehicles cannot be executed while storing the CAMs in the blockchain because the reports generated by the receivers of the CAM that must be verified may not yet be stored in the blockchain, i.e. they cannot be used for the validation. The validation estimates the reliability of information contained in the CAM and ends by storing in the blockchain the result of the validation of the CAM. Because of the append-only property of the blockchain, upgrading the value of a key requires to write a new key-value pair in the chain of blocks by doubling the storage consumption. For this reason it was decided to update only the key-value pairs associated to the CAMs marked as invalid by the position verification algorithm. By assuming that the majority of transmitted CAMs provide valid information, the growth rate of the blockchain due to the validation of CAMs is this way reduced.

Position verification algorithm implementation The implementation details of the position verification algorithm are available in the form of pseudo-code in Algorithm 1.

4. Query of the history of keys

This function receives as parameter a key and uses the *GetHistoryForKey* function to retrieve the number and the entity of modifications occurred to the value of a key. For example it could be useful to understand how many times a position verification algorithm has been applied to a CAM or to identify the attempts of overwriting the value associated to a key.

Algorithm 1 Algorithm to verify if a vehicle is cheating.

Input: C Set of the keys of the transmitted CAMs that must be verified
Output: Verification Information

$r \leftarrow 300.0m$ ▷ Maximum radio range of vehicles
$\epsilon_r \leftarrow 6.8m$ ▷ Ranging Error - constant[67]
$\epsilon_p \leftarrow 10.0m$ ▷ Position Error - constant [67]
$c \leftarrow 299792548m/s$ ▷ speed of light
for all $c \in C$ **do** ▷ For each identifier of the CAMs that must be verified
    $s \leftarrow$ queryCAM(c) ▷ Query the blockchain to retrieve the transmitted CAM
    $f \leftarrow 0$ ▷ Number of non-valid votes
    $v \leftarrow 0$ ▷ Number of valid votes
    $N_s \leftarrow$ queryReceiversCAM(c) ▷ Query the blockchain to get the reception information of the vehicles that received s
    for all $x \in N_s$ **do** ▷ For all the vehicles that received s
        $d_{sx} \leftarrow distance(s, x)$ ▷ Distance between the position declared in s and x at transmission and reception instants
        $t_{prop} \leftarrow x_{t_{rx}} - s_{t_{tx}}$ ▷ Time of propagation of the message from transmitter to receiver
        if $d_{sx} > r$ **or** $|d_{sx} - t_{prop} \cdot c| > 2\epsilon_p + \epsilon_r$ **then**
            $f \leftarrow f + 1$
        else
            $v \leftarrow v + 1$
        end if
    end for
    if $v = 0$ **and** $f = 0$ **then** ▷ No vehicles received the CAM s. Its position is not classifiable.
        $c_{validity} \leftarrow undefined$
    else if $v > f$ **then** ▷ Majority of receivers of s consider its declared position as valid
        $c_{validity} \leftarrow valid$
    else ▷ Majority of receivers of s consider its declared position as not valid
        $c_{validity} \leftarrow invalid$
        updateBlockchain(c) ▷ Store the CAM s in the blockchain notifying that the declared position is not valid
    end if
end for

At the end of this procedure, one of the states *undefined*, *valid*, *invalid* is assigned to the verification information of the CAM c that has been verified.

6.3 The Benchmarking Tool

An important milestone of the project consists in the development of a benchmarking tool, a bash script able to execute the performance evaluation of the blockchain platform. The benchmark is focused on the analysis of the parameters described in Table 6.1. All the parameters have been evaluated by using some bash commands as source of data. Unless otherwise specified, all the experiments shown in Chapter 7 have been evaluated using the commands of table 6.1.

Parameter evaluated [U/M]	**Bash command**	**Implementation details**
Time [ms]	`date`	The time is computed as the difference between the timestamp values read after and before the execution of an operation.
CPU usage [%]	`top`	A background process has been used to continuously read the values of CPU utilization. The CPU usage is defined as the average of these values.
RAM [MB]	`free -m`	The RAM utilization is computed as the difference between the values of available memory evaluated before and after the execution of an operation.
HDD [MB]	`df`	The storage utilization is computed as the difference between the values of used storage memory after and before the execution of an operation.
RAM of containers [MB]	`docker stats`	RAM consumed only by the Docker containers. It is computed as the difference between the amounts of used RAM after and before the execution of a command.
HDD of containers [MB]	`docker system df -v`	HDD consumed only by the Docker containers. It is computed as the difference between the values of storage memory consumed after and before the execution of a command.

Table 6.1: Parameters evaluated by the benchmarking tool

6.3.1 Operations executed by the tool

The tool is conceived to execute a lot of configuration and deployment operations as described as follows:

- Removal of all the Docker containers, images and volumes from the system before the execution of the benchmarks.
- Creation of the configuration files of different blockchain networks by using the *netcomposer* tool.
- Deployment of the complete blockchain network by using a single hosting machine and different Docker containers for each participant.

In addition, the tool is able to execute transactions to manage the following operations:

- To store the reports in the blockchain, either by processing their content or not (see Sec. 6.2.3). The tool reads the files generated by the vehicular simulator, copies these files in the peer containers (to simulate the base stations collecting reports) and issues a sequence of transactions to store the reports in the blockchain. Every transaction stores all the reports contained in a single input file. Multiple input files reports can be stored during a benchmark.
- To query stored CAMs. The tool executes an exhaustive sequence of range queries in order to retrieve all the CAMs stored in the blockchain. It can be customized to define how many CAMs can be queried with a single transaction.
- To validate the stored CAMs. The tool generates transactions for the validation of all the CAMs stored inside the blockchain. It can be configured to define how many CAMs can be validated with a single transaction.
- To query the history of CAMs. The tool allows to retrieve the history of modifications of stored CAMs.

The tool allows to repeat every experiment several times and to compute some statistics on the results, including the **average**, the **variance** and the **confidence interval at 95%**.

6.3.2 Parameters of the benchmarks

The execution of the benchmarks can be customized by editing an appropriate configuration file. The parameters that can be configured are described as follows:

- Name of the experiment.
- Number of repetitions of each experiment.
- Operations executed during the experiment:
  - Storage of report with processing (see Sec. 6.2.3)
  - Storage of reports without processing (see Sec. 6.2.3)
  - Query of CAMs
  - Validation of CAMs
  - Query of the history of CAMs.
- Folder containing the files with the reports that must be stored.
- Endorsement policy.
- Limit on the percentage of CPU assigned to the peers and orderers processes.
- Number of parallel processes that can simultaneously endorse different transactions.

- Configuration of the network. It is expressed as the name of a configuration file of *netcomposer*. It allows to customize the following parameters:
  - Number of organizations
  - Number of peers per organization
  - Type of ordering service (*Solo* or *Kafka*)
  - Number of orderers (only for Kafka ordering service)
  - State Database (CouchDB or LevelDB)
  - TLS secured communication between peers, orderers and clients, or insecure communication.

6.3.3 Limits of the proposed implementation

The benchmarking tool is characterized by some limitations. First of all it does not test the performance of a blockchain composed by different channels able to interact with each other, as described in the architecture of Sec. 5.5.5.
All the operations performed by the benchmark are executed with the **command line interface**. The real applications are typically developed by using one of the SDKs available for the Go, Java and NodeJS programming languages. The transactions invoked with the command line interface, in fact, can be endorsed only by **one peer**, while the number of peers validating and committing the transactions can be customized.
In addition, the use of the command line interface does **not allow to interact with external clients** and to evaluate the latency introduced by these interactions. All the transactions, in fact, are invoked directly by the peers of the network, by means of the `peer chaincode invoke` command.

Chapter 7

Experimental Results

This chapter analyses the results of the experiments performed to test the scalability and the performance of Hyperledger Fabric. Sec. 7.1 briefly describes the testing environment while Sec. 7.2 shows the reports traces generated by the vehicular mobility simulator and used in the experiments of this chapter. The first experiments are relative to the time of access to the state database and to the history index (Sec. 7.3). Sec. 7.4 inspects how the number and the size of blocks and transactions impact on the storage memory utilization of Fabric and on the time of storage of data.
The second part of the chapter starts with Sec. 7.5 and describes the scalability tests performed with Hyperledger Fabric. Many experiments have been executed to evaluate the effects of the number of peers and organizations (Sec. 7.8), the number of orderers (Sec. 7.7) and the number of CAMs validated or queried by every transaction (Sec. 7.8). The analysis compares the performance obtained for the validation of CAMs obtained with read-only or read-write transactions (Sec. 7.9). Sec. 7.10 evaluates the performance obtained if different endorsing peers concurrently endorse the transactions. Finally, Sec. 7.11 highlights the most important results the scalability tests shown in this chapter.

7.1 Configuration of the test environment

All the experiments conducted on this thesis have been executed on a Virtual Machine with the following characteristics:

- 4 virtual CPUs
- 6 GB of RAM
- 60 GB of storage memory
- Ubuntu 16.04 (64 bit) operating system
- Hyperledger Fabric version 1.1.0-preview

7.2 Simulation traces used during the experiments

The experiments shown in this chapter have been executed by using the reports generated with the Python simulator described in Sec. 6.1. In particular, Table 7.1

shows a set of parameters used for all the simulations.

$\mathbf{N_v}$	$\mathbf{P_c}$	$\mathbf{N_{RSU}}$	$\mathbf{w_{area}}$	$\mathbf{h_{area}}$	$\mathbf{R_r}$
10	60%	5	1000m	1000m	300m

Table 7.1: Configuration of the simulator

Definition of the different fields of Table 7.1.
$\mathbf{N_v}$ indicates the number of vehicles in the simulation, $\mathbf{P_c}$ the probability that a vehicle is cheating, $\mathbf{N_{RSU}}$ the number of RSUs in the area of the simulation, $\mathbf{w_{area}}$ and $\mathbf{h_{area}}$ indicates the width and height of the simulation area, $\mathbf{R_r}$ is the radio range of transmission of the CAMs.

The traces of the reports generated with the simulator configured as described in Table 7.1 are listed in Table 7.2.

ID	Total Size	Total Size Real	N Files	Size Single File	Time Simulation	N Reports	N CAMs
7.1.1	200kB	192kB	1	200kB	10s	139	90
7.1.2	400kB	388kB	1	400kB	18s	256	170
7.1.3	600kB	588kB	1	600kB	26s	374	249
7.1.4	1MB	980kB	1	1MB	42s	627	417
7.1.5	1MB	980kB	10	100kB	42s	627	417
7.1.6	2MB	1.96MB	2	1MB	84s	1255	835
7.1.7	5MB	4.90MB	5	1MB	177s	2649	1764
7.1.8	10MB	9.80MB	10	1MB	401s	5999	3999
7.1.9	20MB	19.60MB	20	1MB	797s	11938	7958
7.1.10	100MB	98.03MB	100	1MB	3900s	58489	38999

Table 7.2: Traces of reports generated with the simulator

Definition of the different fields of Table 7.2.
ID indicates a unique identifier of the trace, so that it is possible to identify the trace used during the different experiments. **Total Size** is the rounding of the **Total Size Real**, i.e. the effective storage consumption of the generated trace. The output of each simulation can be by composed by different files storing a portion of the results; **N Files** indicates how many files has been created during the simulation (the concept of file is important because every single file will be successively stored in the blockchain by using a single transaction). Every generated file has an average size of **Size Single File**, expressed in kB or MB. **Time Simulation** indicates the total duration of the simulation, expressed in seconds. **N Reports** indicates total number of reports generated by the vehicles and RSUs during the simulation. **N CAMs** shows the number of CAMs transmitted by the vehicles (it does not coincide with *N Reports* because the latter considers also the reports generated by the RSUs, while the CAMs are transmitted only by the vehicles).

7.3 Evaluation of time of access to data

The objective of the first experiments executed on Hyperledger Fabric is the evaluation of the time required to read from and write to the blockchain.

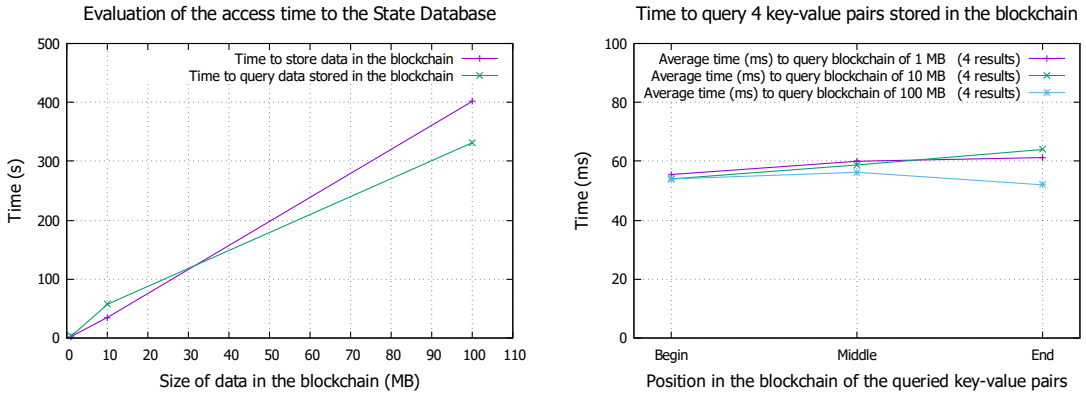
7.3.1 Time of access to the State Database

As mentioned in previous chapters, the blockchain is characterized by an always growing chain of blocks. For this reason, the time required to read data is potentially proportional to the amount of information stored in the blockchain ($\Theta(n)$, where n is equal to the number of elements or transactions stored in the blockchain). The **state database** has been introduced in Hyperledger Fabric to offer random access to stored data, by making the read access time independent on the blockchain size ($\Theta(1)$).

Objective of the experiment: to verify if the time required to store and to query data in the blockchain does not depend on the blockchain size, i.e. on the number of key-value pairs stored.
Implementation details of the experiment: To deploy different blockchains characterized by the same network configuration. For each blockchain it is necessary to store a different amount of data (1MB, 10MB, 100MB). Exhaustive range queries must be executed to retrieve all the key-value pairs stored in the blockchain (every range query is able to retrieve 1000 key-value pairs).
Configuration of the network: see Table 7.3

# Organizations	1
# Peers per org.	2
# Peers total	2
# Orderers	1
CPU peers	100%
CPU orderers	100%
State Database	LevelDB
Endorsing Policy	Default
Input trace	7.1.4, 7.1.8, 7.1.10

Table 7.3: Configuration of the network



(a) Access time to the blockchain (b) Query of 4 key-value pairs

Figure 7.1: Effect of blockchain size on the time of storage and querying

Results. As shown in Fig. 7.1a, the time required to store 1MB, 10MB, 100MB of data in the blockchain and to execute exhaustive queries to retrieve all the stored

key-value pairs is linearly proportional to the blockchain size. In other words, the random access time to the blockchain is independent on the blockchain size ($\Theta(1)$). To confirm this assertion, Fig. 7.1b shows the results of another experiment conducted by executing range queries able to extract 4 key-value pairs stored exactly in the first, in the last and in the intermediate block of the blockchain. Four experiments have been executed for each test, the plotted values indicate the average values of the experiments.
Based on the results of the experiments, it is possible to conclude that the read-write access time of Fabric is independent from the size of the blockchain.

7.3.2 Time of access to the History Index

Objective of the experiment: To verify if the access time to the History Index is independent on the number of modifications made to the key-value pairs.
Implementation details of the experiment: To overwrite 1MB of data stored in the blockchain from 1 to 100 times. Every time the data is overwritten it must be executed a query to the History Index to retrieve the history of one of the keys stored in the blockchain.
Configuration of the network: same as 7.3 (different input trace, only 7.1.4)

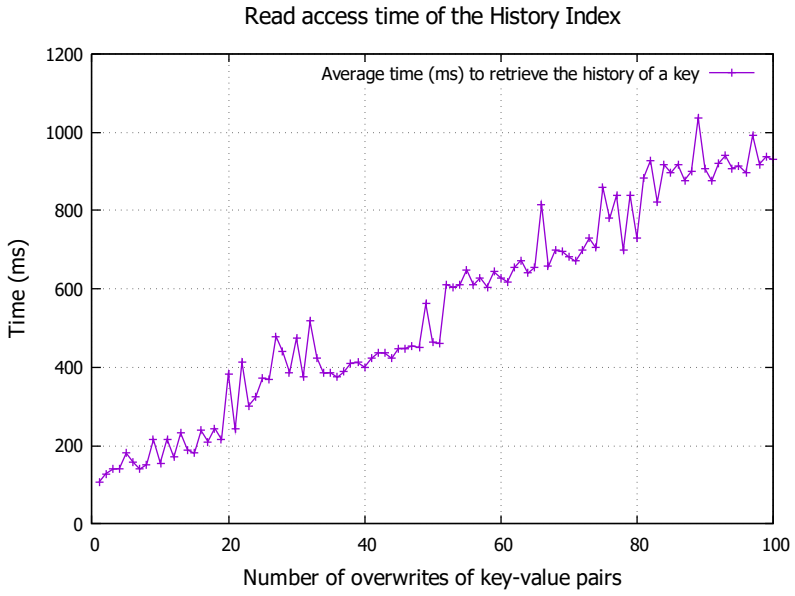


Figure 7.2: Read access time to the History Index

Results. Fig. 7.2 shows the result of the experiment. The time necessary for accessing the history index to retrieve the set of modifications made to the key-value pairs is proportional to the number of times that the data has been overwritten. When the size of the blockchain is increased by a factor of 100 (i.e. when the data has been overwritten 100 times), the time required to access the history of modifications of a key is increased by a factor of 10. This result can be demonstrated by considering that the time required to access the history index is not proportional to the length (i.e. number of blocks) of the blockchain. However, to retrieve the data relative to the history of a key it is necessary to access to many blocks containing the different versions of the key-value pairs (multiple access to data) and to create a result given

by the concatenation of strings. The latter determine an increase of time that is shown in Fig. 7.2.
The glitches that is possible to see in the chart are probably due to the context switches performed by the operating system that may affect the amount of time required for the execution of the queries.
The time (t_{hi}) required to retrieve the history of modifications of a key, can be expressed by the following empirical formula:

$$t_{hi} = 106ms + 8.25ms \cdot n_{ovw} \tag{7.1}$$

where n_{ovw} indicates the number of times that the key has been overwritten and ms is the unit of measurement (milliseconds).

7.4 Effect of the blocks and transactions on the storage memory requirements

This section describes how the creation of blocks and the execution of transactions affect the **storage memory overhead** and the **performance** of the blockchain.
The storage overhead is defined as the difference between the size (in bytes) of the chain of blocks and the size (in bytes) of the trace of reports generated by the vehicular simulator, i.e. before they are stored in the blockchain.
The performance, instead, can be evaluated by analysing the time required to store the reports in the blockchain.

7.4.1 Configuration of blocks

In Hyperledger Fabric it is possible to configure the generation of blocks (i.e. **batches** in Hyperledger lingo) so that it is possible to satisfy the requirements of the different applications. The characteristics of the blocks can be configured by editing the file **configtx.yaml**[1] and by configuring the 4 parameters described as follows:

1. `BatchTimeout`: it indicates the maximum number of seconds that can elapse between the creation of two subsequent blocks. When the timeout elapses and there is at least one already endorsed transaction, the orderers create a block, independently on the number of already endorsed transactions.

2. `MaxMessageCount`: it indicates the maximum number of transactions that can be contained inside a block. When the orderers receive this number of endorsed transactions, they immediately create a block.

3. `AbsoluteMaxBytes`: it indicates the maximum size of each block, expressed in bytes or its multiples.

4. `PreferredMaxBytes`: it indicates the preferred maximum size of each block, expressed in bytes or its multiples.

[1]Example available at `https://github.com/hyperledger/fabric-samples/blob/release/first-network/configtx.yaml`

7.4.2 Storage overhead of blocks

Objective of the experiment: To analyse the amount of storage overhead introduced by blocks with different sizes.
Implementation details of the experiment: To deploy different blockchains characterized by the same network configuration, and to store different amounts of data in the blockchain. The BatchTimeout must be set to 2 seconds to enforce the creation of blocks with one transaction only (alternatively it is possible to set MaxMessageCount to 1). The reports must not be processed by the chaincode and must be stored as JSON documents exactly as they have been generated by the vehicular simulator (see *Alternative reports storage solution* of Sec. 6.2.3). The processing of the content of reports, in fact, introduces an additional computational and storage overhead that would invalidate the analysis; for each transmitted and received CAM, in fact, the processing creates a different key-value pair.
Configuration of the network: same as Table 7.3 with different input traces (from 7.1.1 to 7.1.10, except 7.1.5).
Configuration of blocks generation: See Table 7.4 (note: each block contains exactly 1 transaction).

`BatchTimeout`	2s
`MaxMessageCount`	10
`AbsoluteMaxBytes`	99MB
`PreferredMaxBytes`	512kB

Table 7.4: Configuration of blocks generation

.

Results. The analysis of the results of the experiment demonstrated that the blockchain contains exactly 3 blocks that are created before storing data. These blocks are described as follows:

1. **Genesis block**
2. **Configuration Transaction block**
3. **Block generated by the `instantiate` transaction**

These blocks, whose generation is described by the operations of set up of the blockchain described in Sec. 6.2.2, are fundamental to allow the execution of transactions in a given channel of the blockchain.

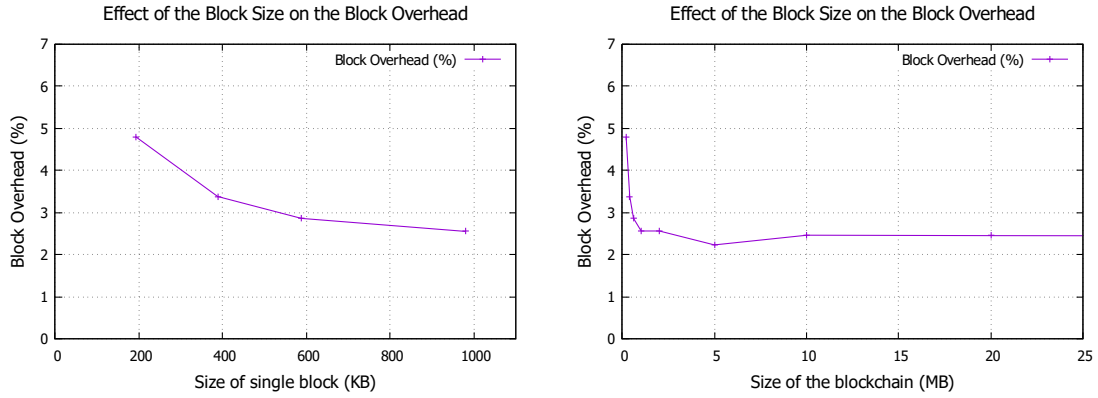
ID Trace	Trace Size	N Blocks	Avg Block Size	Total Blockchain Size	Blocks Overhead
7.1.1	192kB	1	200.87kB	200.87kB	4.80%
7.1.2	388kB	1	397.13kB	397.13kB	3.38%
7.1.3	588kB	1	602.31kB	602.31kB	2.86%
7.1.4	980kB	1	1001.50kB	1001.50kB	2.55%
7.1.6	1.96MB	2	1.00MB	2.00MB	2.55%
7.1.7	4.90MB	5	1.00MB	4.99MB	2.23%
7.1.8	9.80MB	10	1.00MB	10.02MB	2.46%
7.1.9	19.60MB	20	0.98MB	20.00MB	2.45%
7.1.10	98.03MB	100	0.98MB	100.13MB	2.41%

Table 7.5: Numerical results of the experiment

The Table 7.5 shows the numerical results of the experiment; its fields are described as follows:

- **ID Trace** identifies the traces of Table 7.2.
- **Trace Size** indicates the size of the traces generated by the simulator before they are stored in the blockchain.
- **N Blocks** indicates how many blocks have been created in the blockchain (excluding the first 3 blocks).
- **Avg Block Size** is the average size of the blocks of the chain.
- **Total Blockchain Size** indicates the amount of secondary memory used to store the whole chain of blocks (the blocks have been retrieved with the `peer channel fetch` command).
- **Blocks Overhead** indicates the storage overhead in percentage introduced by the chain of blocks.

Note. This experiment considers only the size of the chain of blocks and not the storage utilization of the Docker containers and the hosting machine. The latter have been deeply analysed in Sec. 7.4.6.



(a) Block overhead and block size

(b) Block overhead and blockchain size

Figure 7.3: Effect of the size of blocks and size of the blockchain on storage memory overhead of blocks.

Fig. 7.3a shows that the percentage of storage overhead, introduced by the creation of blocks, is reduced if the size of blocks is increased. This trend can be proven by considering that each block contains in addition the following information[2]:

- The block header (containing the hash of the previous block, the hash summarizing the transactions contained in the block, the digital signature generated by the orderer that created the block)
- The overhead of each transaction (containing the digital signature and the public key certificate of the peers that endorsed of the transactions)

[2]Additional information on the structure of blocks is available at: `https://blockchain-fabric.blogspot.it/2017/04/hyperledger-fabric-v10-block-structure.html`

- The keys that are necessary for the identification of the key-value pairs associated to every stored report. In fact, the keys are computed by the chaincode function because they are not generated by the vehicular simulator.

Each block can be inspected in binary format (by downloading it with `peer channel fetch` command), and it is also possible to use the **configtxlator** tool to transform it in human readable JSON format.
Fig. 7.3b shows that the blockchain overhead in percentage is almost constant if the blockchain is created by blocks with the same size (*Avg Block Size*). The experiments executed with the traces from 7.1.4 to 7.1.10, in fact, are characterized by blocks with a size of about 1MB and by a storage overhead of about 2.5%.

The results of this experiment demonstrates that the size of blocks is relevant for the definition of the blockchain storage overhead. The latter can be not negligible with respect to the amount of stored data if the size of the blocks is very small. Thus, the size of blocks must be taken in consideration especially if the amount of stored data can be critical in the blockchain deployment scenario.

7.4.3 Configuration of the transactions contained in blocks

As introduced in the previous experiment the generation of transactions introduce a storage overhead; for example, each transaction contains the digital signatures necessary for the identification of the peers that executed the endorsement of the transaction. For this reason, a deeper analysis was conducted to evaluate the storage overhead introduced by each transaction. Before analysing in detail the next experiment, it is necessary to introduce some considerations about the correct configuration of the number of transactions per block.

The problem of the BatchTimeout. The BatchTimeout represents an important parameter in the definition of the block size because it forces the creation of blocks independently from the number of endorsed transactions that are ready to be inserted into the new block. The timeout cannot be disabled because by removing the line relative to the BatchTimeout configuration from the configtxgen.yaml file, the timeout is automatically set to a predefined value of 2 seconds (this behaviour has been tested with additional experiments). To simulate an infinite timeout, a very large BatchTimeout value has been introduced, but this choice has caused many drawbacks. With a large BatchTimeout value and a MaxMessageCount greater than 1, in fact, the block containing the `instantiate` transaction is not immediately committed to the ledger. This implies that all the transactions executed before the correct instantiation of the chaincode failed with an error code indicating that it is not possible to execute a chaincode function as long the chaincode has not installed in the peer and instantiated in the channel (this behaviour affects either the `invoke` and `query` transactions). For this reason it is necessary to introduce, between the execution of the instantiate transaction and the execution of any other transactions, a delay longer than the duration of BatchTimeout.
A long BatchTimeout can cause another problem; if the transaction rate of the blockchain is low, it is necessary to wait a long time before that the transactions are validated and committed to the ledger. During the execution of some experiments for this thesis, this problem has been detected and it made necessary to introduce

a waiting time between the storage and validation of CAMs, to allow the correct execution of the position verification algorithm.

Other considerations on the creation of blocks. Three other parameters can be used to configure the generation of blocks, i.e. MaxMessageCount, PreferredMaxBytes and AbsoluteMaxBytes. The MaxMessageCount can be used to create blocks with a well-defined maximum number of transactions. This parameter can be paired with a long BatchTimeout value to create blocks with the desired number of transactions in it. To enforce the creation of blocks with a given number of transactions it is important to be able to respect the AbsoluteMaxBytes parameter that introduces an upper bound on the size of each block: if the size of the transactions exceeds the AbsoluteMaxBytes size, in fact, they cannot be inserted into the same block. Moreover, it is highly discouraged to set a very small value of AbsoluteMaxBytes to limit the size of blocks, because if the size of a single transaction is larger than the AbsoluteMaxBytes, the transaction cannot be validated (an experiment demonstrated this behaviour). In a scenario like the one described in this thesis, characterized by clients requiring to store a large number of reports with a single transaction, a small value of AbsoluteMaxBytes could compromise the correct execution of the operations.

7.4.4 Effect of transaction size on storage overhead

Objective of the experiment: to verify if the number of transactions issued to store the same amount of data in the blockchain is proportional to the storage overhead of the blockchain.

Implementation details of the experiment: to deploy many blockchains with different MaxMessageCount value (the BatchTimeout value has been chosen to satisfy the requirements explained in Sec. 7.4.3).
Configuration of blocks generation: see Table 7.6.
Configuration of transactions: see Table 7.7.
Configuration of the network: same as Table 7.3 (with different input trace)

`BatchTimeout`	8s
`MaxMessageCount`	1, 2, 5, 10, 15
`AbsoluteMaxBytes`	99MB
`PreferredMaxBytes`	3MB

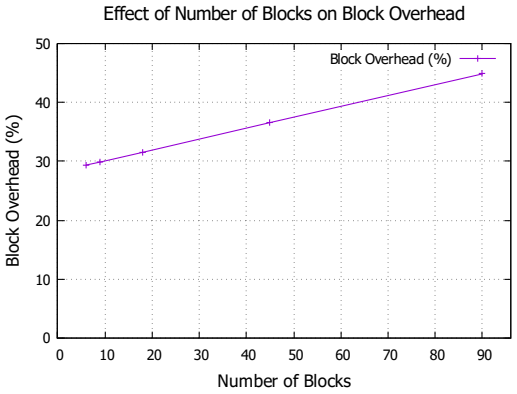
Table 7.6: Configuration of blocks generation

Transactions number	90
Transaction payload	10kB
Trace Size	986.2793kB

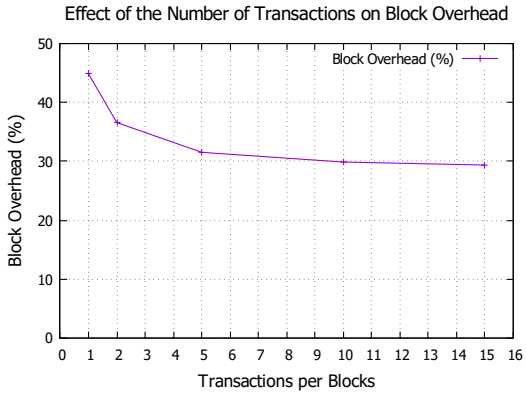
Table 7.7: Details of the experiment

N Blocks (n_{bk})	**Txs per Block** (n_{txbk})	**Block Overhead** (O_{bck})[%]	**Block Overhead** (O_{bck})[**kB**]	**Txs Overhead** (O_{tx})[%]	**Time to Store Data (s)**
90	1	44.83%	442.19kB	4.98%	13.34s
45	2	36.57%	360.72kB	4.06%	13.10s
18	5	31.52%	310.83kB	3.50%	12.76s
9	10	29.87%	294.56kB	3.32%	12.75s
6	15	29.36%	286.41kB	3.26%	12.65s

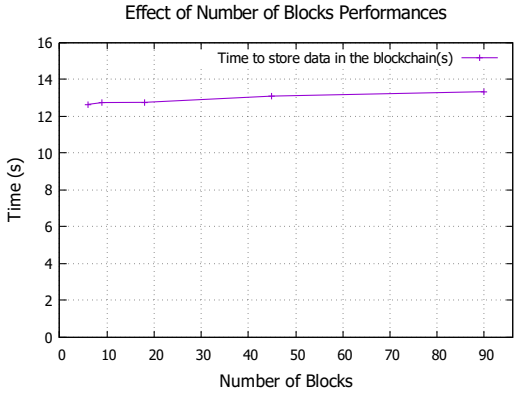
Table 7.8: Results of the experiment



(a) Number of blocks and block overhead



(b) Transactions per block and block overhead



(c) Time to store data in the blockchain

Figure 7.4: Storage memory overhead of blocks and transactions

Result. This experiment analyses the storage overhead introduced by the creation of blocks containing a different number of small sized transactions (10kB). Fig. 7.4a shows that the block overhead in percentage is linearly dependent on the number of generated blocks. At the same way, Fig. 7.4b demonstrates that by increasing the number of transactions per block the total storage overhead is decreased. The numerical results of table 7.11 has been used to derive the following empirical formulas:

$$O_{bk} = O_{bk_{fix}} + O_{tx} \cdot n_{tx} \tag{7.2}$$

$$O_{bk} = 1.8226kB + 3.0906kB \cdot n_{tx}$$

where O_{bk} is the overhead introduced by each block (by including the block and transaction overhead expressed in kB), $O_{bk_{fix}}$ is the fixed overhead introduced by the creation of a block, O_{tx} indicates the overhead introduced by every transaction of the block and n_{tx} is the number of transactions contained in the block. The total storage overhead introduced by the blockchain can be computed as follows:

$$O_{bck} = \sum_{i=0}^{n} O_{bk_i} \tag{7.3}$$

where O_{bck} is the total storage overhead introduced by the chain of blocks, O_{bk_i} is the storage overhead introduced by the i^{th} block of the chain (from this computation are excluded the first three blocks which size is about 19kB).

Proof. To demonstrate the previous formulas it is necessary to compute the fixed overhead introduced by each single transaction and by each single block.
The first step consists in computing the average overhead introduced by each block by using one of the empirical results of Table 7.11 (in particular it has been chosen the experiment characterized by 10 transactions per block).

$$O_{bk_{10}} = \frac{O_{bck}}{n_{bk}} = \frac{294.5566kB}{9} = 32.7285kB$$

where $O_{bk_{10}}$ is the average overhead of each block generated during the experiment, O_{bck} is the total storage overhead of the blockchain, n_{bk} is the number of blocks which constitute the blockchain. It is assumed that every block of the blockchain has the same size, thus:

$$O_{bk_i} = O_{bk_j}, \forall(i,j)$$

Compute the difference ($O_{bk_{dif}}$) between the overhead relative to the blocks containing 10 transactions ($O_{bk_{10}}$) and the overhead relative to blocks containing only 1 transaction (O_{bk_1}).

$$O_{bk_{dif}} = O_{bk_{10}} - O_{bk_1} = 32.7285kB - 4.9132kB = 27.8153kB$$

To find the overhead of each transaction contained in a block, it is necessary to divide $O_{bk_{dif}}$ by the difference between the numbers of transactions contained in one block of the two experiments ($n_{tx_{10}}$ and n_{tx_1} respectively):

$$O_{tx} = \frac{O_{bk_{dif}}}{n_{tx_{10}} - n_{tx_1}} = \frac{27.8153kB}{10-1} = 3.0906kB$$

The result, O_{tx}, indicates the overhead introduced by every transaction contained in a block. At this point it is possible to conclude the proof by calculating the fixed overhead introduced by every block:

$$O_{bk_{fix}} = O_{bk_1} - O_{tx} = 4.9132kB - 3.0906kB = 1.8226kB$$

$O_{bk_{fix}}$ is the fixed overhead introduced by each block without considering the contained transactions. It is computed as the difference between the total overhead of the block containing exactly one transaction (O_{bk_1}) and the overhead introduced by a generic transaction (O_{tx}).

The validity of the numerical parameters can be proved by computing the total blockchain overhead of each experiment and comparing the result with the values O_{bck} of table 7.11.

- Case with 1 transaction per block.
  $O_{bck} = (1.8226kB + 3.0906kB \cdot 1) \cdot 90 = 442.188kB$
- Case with 2 transactions per block.
  $O_{bck} = (1.8226kB + 3.0906kB \cdot 2) \cdot 45 = 360.171kB$
- Case with 5 transactions per block.
  $O_{bck} = (1.8226kB + 3.0906kB \cdot 5) \cdot 18 = 310.961kB$
- Case with 10 transactions per block.
  $O_{bck} = (1.8226kB + 3.0906kB \cdot 10) \cdot 9 = 294.557kB$
- Case with 15 transactions per block.
  $O_{bck} = (1.8226kB + 3.0906kB \cdot 15) \cdot 6 = 289.089kB$

Furthermore, to confirm the results has been considered also the block containing only the `instantiate` transaction. Its size is about 4.63kB and its payload is practically empty, because no key-value pairs are stored during the instantiation of the chaincode. The overhead introduced by a block containing only one transaction is about 4.91kB according to our formula, but this overhead is relative to blocks containing many key-value pairs; as mentioned in Sec. 7.4.2, the keys introduce an additional overhead in the block. For this reason, the empirical formula derived in this section can be considered acceptable to describe the overhead of blocks and transactions.

Blocks and performances of the blockchain. Fig. 7.4c shows that the time required to store data in the blockchain, slightly depends on the number of blocks of chain (this consideration is valid if the number of transactions is kept constant in all the experiments). The empirical formula that describes this phenomenon is:

$$t_{feed} = 5.9s + 0.00825 \cdot n_{bk} \tag{7.4}$$

where t_{feed} represents the time required to store data in the blockchain, n_{bk} indicates the number of blocks of the blockchain, s indicates the seconds.
Notes. The measured time considers only the time required for the endorsement of the transactions (i.e. the time of execution of the `peer chaincode invoke` functions) and not the time necessary for the ordering and committing phases.

7.4.5 Effects of Transactions size

Objective of the experiment: to evaluate how the transactions size affects the storage overhead and the performance of the blockchain.

Implementation details of the experiment: to deploy different blockchains with the same configuration, and execute transactions with a different payload size.
Configuration of blocks generation: see Table 7.9.
Configuration of transactions: see Table 7.10.
Configuration of the network: same as 7.3 (trace 7.1.8 split in smaller files)

`BatchTimeout`	100s
`MaxMessageCount`	10
`AbsoluteMaxBytes`	99MB
`PreferredMaxBytes`	12MB

Table 7.9: Configuration of blocks generation

Transactions per block	10
Transaction payload	Variable
Trace size	10MB

Table 7.10: Details of the experiment

Txs Size [kB]	**N Blocks (n_{bk})**	**Txs per Block**	**N Txs**	**Blockchain Overhead (%)**	**Time to Store Data (s)**
10.88	90	10	894	30.02	122.94
98.74	10	10	99	4.99	35.34
244.95	5	10	40	3.33	31.83
489.75	2	10	20	2.42	29.10
977.65	1	10	10	2.29	29.10

Table 7.11: Results of the experiment

Results. Fig. 7.5a shows that the overhead of the blockchain is linearly proportional to the number of executed transactions to store data in the blockchain. The empirical formula describing this phenomena is:

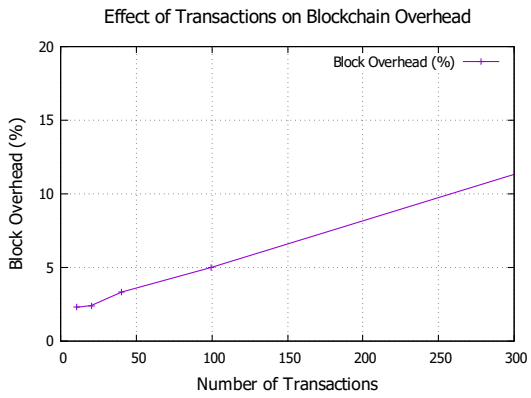
$$O_{bck} = 193.46kB + 3.05 \cdot n_{tx} \tag{7.5}$$

where O_{bck} indicates the total blockchain overhead, n_{tx} indicates the number of transactions of the blockchain, kB indicates the kilobytes. At the same way, Fig. 7.5b shows that the size of transactions is inversely proportional to the blockchain overhead.

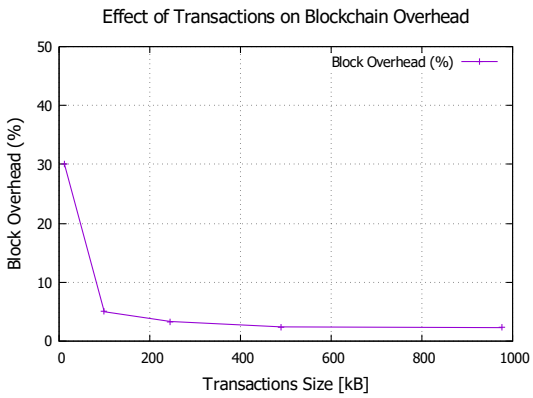
Transactions and performances of the blockchain. Fig. 7.5c shows that the time required to store the same amount data in the blockchain is linearly proportional to the number of executed transactions. The empirical formula describing the phenomena is:

$$t_{feed} = 28.04s + 0.106 \cdot n_{tx} \tag{7.6}$$

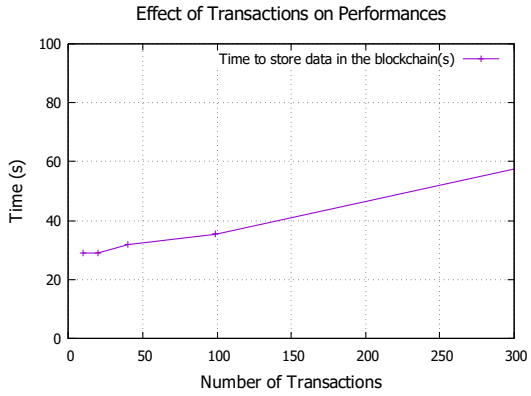
where t_{feed} describes the time required to store data in the blockchain, n_{tx} indicates the number of transactions, s indicates the seconds.



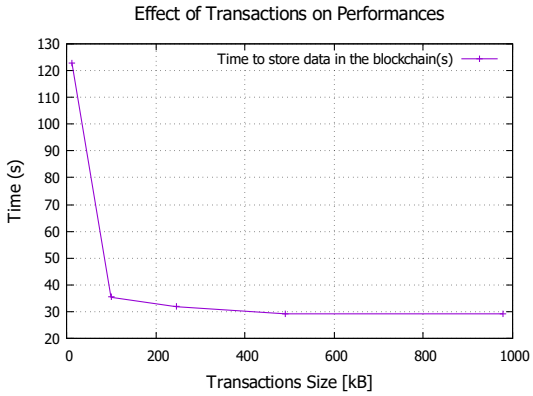
(a) Number of transactions and block overhead



(b) Transactions size and block overhead



(c) Number of transactions and time to store data



(d) Transactions size and time to store data

Figure 7.5: Storage memory overhead introduced by blocks and transactions

This result can be justified by considering that the increase of the number of endorsed transactions requires a larger computational overhead, due to the an increase of the number of calculations of digital signatures and of interactions between the different members of the blockchain.

Comments on the result. The time requires to store data in the blockchain significantly decreases by varying the payload of the transactions from 10kB to 100kB, while it is established when it reaches a payload of about 500kB. The explanation of this phenomena is simple: when the payload of each transaction is very large, the computational overhead required by the endorsement of the transactions becomes negligible with respect to the time required for the execution of the chaincode function. At the same way by increasing the number of transactions the storage overhead of the blockchain is increased because each transaction contributes to the total overhead.

7.4.6 Storage overhead of the hosting machine and containers

In this section the storage memory requirements of the hosting machine and of the different Docker containers used for the deployment of a Fabric blockchain are analysed. All the experiments outlined above, in fact, refer to the storage overhead generated by the **chain of blocks** (retrieved with the `peer channel fetch` com-

mand) and not to the total storage overhead introduced by Hyperledger Fabric (e.g. by the state database, history index, etc.).

LevelDB or CouchDB The first important consideration is relative to the adoption of LevelDB or CouchDB as state database. LevelDB allows to simplify the analysis of the storage utilization because it is installed directly on the containers of the peers and it is therefore sufficient to inspect the different containers to analyse the amount of secondary memory required by the state database. With CouchDB, instead, the analysis is more complex because it involves an ad-hoc CouchDB Docker container and a set of Docker volumes used to persist the data. For this reason, in this section, all the results are relative to network configuration characterized by the adoption of LevelDB as state database.

Objective of the experiment: to understand if the storage consumption of the Docker containers and of the hosting machine are proportional to the size of the blockchain.
Implementation details of the experiment: to deploy different blockchains by storing different amounts of data and using the following commands to analyse the storage consumption:

- `df -v`: for the total storage consumption of the hosting machine.
- `docker system df -v`: for the storage consumption of the Docker containers and volumes.

Configuration of the network and blocks creation: same as the experiment of Sec. 7.4.2.

Results

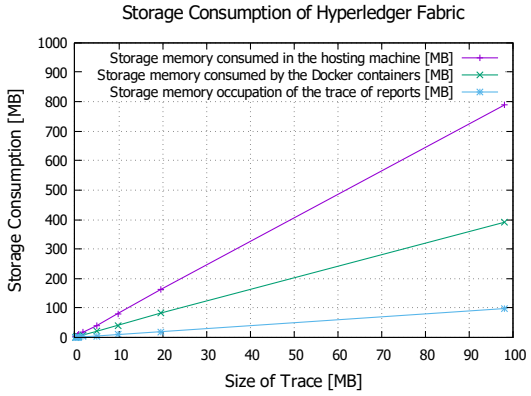
Analysis of Docker containers. The Table 7.12 shows the results of the evaluation of the storage consumption of the Docker containers before and after having stored the data in the blockchain (the information contained in the table is relative to the experiment conducted with the trace 7.1.1 - 192kB). This analysis does not require to evaluate Docker volumes because LevelDB does not create any volume.

Container Name	Size Before Storing Data [kB]	Size After Storing Data [kB]	Difference [kB]
dev.peer0.org1.example.com	0	0	0
cli	0	0	0
peer0.org1.example.com	39.1	457.0	417.9
peer1.org1.example.com	29.9	448.0	418.1
orderer	38.7	244.0	205.3
ca.example.com	38.0	38.8	0

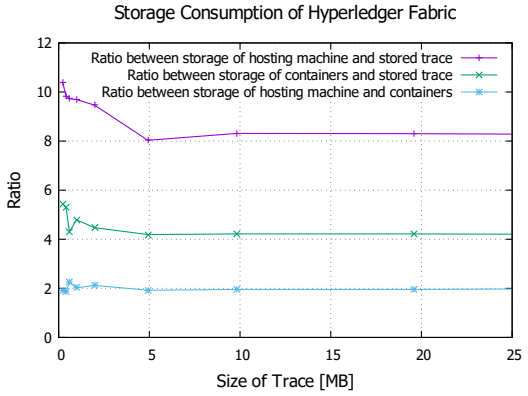
Table 7.12: Analysis of Docker containers

The Table 7.12 shows 4 columns, **Container Name** indicates the name of the containers used, **Size Before** and **After Storing Data** indicate the storage consumption of each single container before and after the reports has been stored in

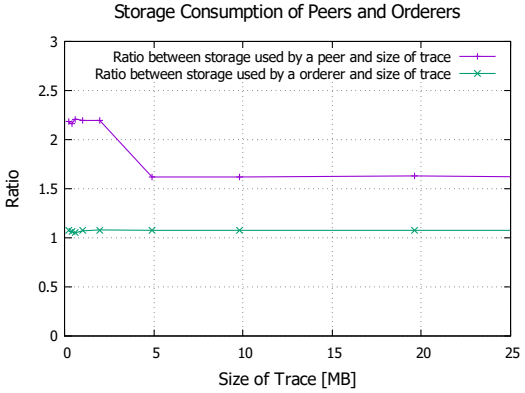
the blockchain, while the **Difference** indicates the amount of secondary memory that has been used to execute the storage of data in the blockchain (i.e. it is the difference between the values of the previous two columns).
The *orderer* consumes exactly 205.3kB of data because it stores locally the complete chain of blocks. The peers, instead, consume about 418kB each, more than the double that the amount of data stored by the orderer. This value is reasonable because each peer stores the chain of blocks, the state database (containing exactly the same key-value pairs stored in the chain of blocks) and the history index. These assertions have been verified by inspecting the file system of the peers and of the orderer to locate the relative files. The *dev.peer0.org1.example.com* is a container created only for the execution of chaincode associated to *peer0.org1.example.com* container and, for this reason, it does not store data. At the same way, the *cli* container and the CA container (*ca.example.com*) do not require to store data during the execution of operations.



(a) Storage consumption of hosting machine, Docker containers and stored trace of reports



(b) Ratio between the storage consumption of hosting machine, containers and stored trace



(c) Ratio between the storage consumption of a peer (and a orderer) and the size of stored trace

Figure 7.6: Storage overhead of Docker containers and hosting machine

Storage utilization of Docker containers and hosting machine. Fig. 7.6a shows the size of the traces containing the reports, the storage utilization of the

Docker containers, and the overall storage utilization of the hosting machine. It is evident that the amount of storage memory consumed by the Docker containers is notably lower than the total amount of storage memory consumed by the hosting machine. Fig. 7.6b allows to quantify these values: in the hosting machine it is consumed about the double of the storage memory used by the containers. The reasons can be numerous but the ones that have been identified are the existence of a storage overhead necessary for the execution of the containers, and the creation of log files that are saved in the hosting machine to track the operations executed by the different containers. These assertion has been demonstrated by executing some tests that reduced the amount of information stored in the logs, e.g. by removing all the instructions of the chaincode writing on standard output, and by changing the *Log Level* of each container so that they store a reduced amount of log data (the Log Level can be edited in the *docker-compose.yaml* file). These precautions made possible to reduce the storage utilization of the hosting machine (numerical results are not reported).
Fig. 7.6b shows that the ratio between the storage consumption of the all the Docker containers and the size of the stored traces is about 4 times larger. The expected value was 5 times larger because the orderer stores the chain of blocks, and both peers store either the chain of blocks and the state database. The correspondence between the theoretical values and the experimental values is verified only with traces with a size lower than 2MB, while starting from 5MB onward, the storage utilization of containers is reduced. The reason is the reduction of the amount of storage memory consumed by the state database. Fig. 7.6c demonstrates that amount of storage consumed by the orderer with respect to the size of the stored trace is almost constant and predictable (a little bit larger than the size of stored trace, due to the overhead introduced by the creation of blocks). On the contrary, the storage utilization of each peer is reduced when the size of the stored trace becomes larger than 2MB. This unusual result has been verified multiple times by introducing a long delay between executing each transaction storing the reports in the ledger, so that it was possible to verify that the ordering and committing phases of the transactions have been concluded before reading the size of containers. It was also verified that the size of the chain of blocks stored in the peers is equivalent to the one stored in the orderer.
This behaviour, even if apparently wrong, is acceptable because LevelDB uses an algorithm for data compression called Snappy [72], that notably reduces the amount of secondary memory required by the state database.

7.4.7 Final considerations on storage overhead

The management of the storage utilization is very important in Hyperledger Fabric, because every peer stores two copies of all the key-value pairs, one copy for the chain of blocks and one copy for the state database. All the experiments conducted in Sec. 7.4 demonstrated that the creation of blocks and transactions is fundamental for the definition of the storage overhead introduced by the blockchain. The configuration of blocks is not an easy task, and requires to know some configuration parameters of the blockchain. The creation of very large blocks can reduce the storage overhead of the blockchain but increases the amount of time necessary for committing data to the ledger, in other words it increases the latency. A trade-off

between efficiency and storage overhead must be chosen by the system administrators, depending on the requirements of the blockchain applications. In addition to the blocks, also the number and the size of transactions affect the performance and the storage requirements of the blockchain. The execution of a multitude of transactions with a small-sized payload introduces a large storage and computational overhead that can degrade the performance of the blockchain. Unfortunately, in some applications it is not possible to execute transactions with a large payload, for example the management of transfer of assets is characterized by transactions with a small-sized payload. In the context of this thesis, it is possible to collect many reports and to store them simultaneously with a single transaction. However, this solution has a significant drawback: it is necessary to postpone the validation of CAMs stored in blockchain so that all the base stations have the possibility to store the reports by executing a reduced number of transactions. The same problem arises if the blockchain is composed by large blocks, because it is necessary that all the blocks are committed before correctly applying the position verification algorithms.

7.5 Scalability tests of Hyperledger Fabric

In this section the results of the experiments on the scalability of Hyperledger Fabric are illustrated. The scalability tests have been executed on a single hosting machine and, for this reason, their results do not constitute a reliable assessment of the Hyperledger Fabric performance. To overcome the limitations of the limited deployment environment, for some experiments, the percentage of CPU time assigned to the peer and orderer processes has been limited. The maximum amount of main memory installed in the hosting machine has never been exceeded in order to avoid that the operation of swapping conducted by the operating system might compromise the result of the experiments. Another major problem relates to the execution of concurrent processes that tries to simultaneously access to the secondary memory, by making the access to disk a performance bottleneck. The following sections describe the experiments relative to the scalability of the blockchain, by highlighting the limits of the deployment environment and the countermeasures adopted to manage them.

7.6 Effect of the number of peers of the blockchain

Objective of the experiments: to verify if the number of peers participating in the blockchain affects the performance of the operations of storage, verification and querying.

Implementation details of the experiments: to deploy different blockchains characterized by a different number of peers. To test for each blockchain the performance obtained by storing the reports, validating all the CAMs and executing exhaustive queries to retrieve all the stored key-value pairs. All the experiments conducted in this section refers to the block configuration of Table 7.13.

`BatchTimeout`	2s
`MaxMessageCount`	10
`AbsoluteMaxBytes`	99MB
`PreferredMaxBytes`	512kB

Table 7.13: Configuration of blocks generation

7.6.1 Experiments with 1 organization and many peers

# Organizations	1
# Peers per org.	2,4,6,8,10,12,14,16
# Peers total	2,4,6,8,10,12,14,16
# Orderers	1
Type of orderer	Solo
State Database	CouchDB
Endorsing Policy	Default

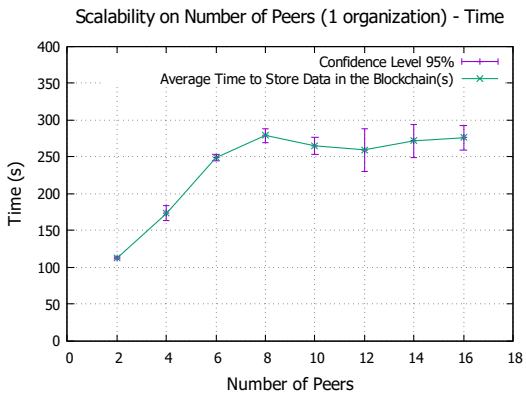
Table 7.14: Configuration of the network

Storage of reports in the blockchain

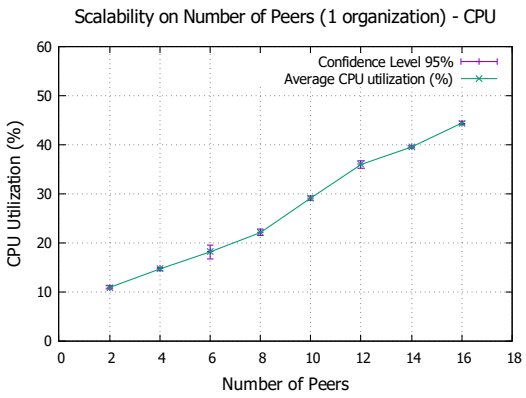
This test performs the storage of 10MB of reports (trace 7.1.8) in raw format (i.e. without processing the content of reports) and evaluates the time required to execute the operation, the percentage of CPU used, the amount of RAM consumed and the storage memory utilization.

CPU peers	5%
CPU orderers	100%
Timeout Peer	300s
Trace Stored	7.1.8 (10MB)
Type of storage	Raw Format

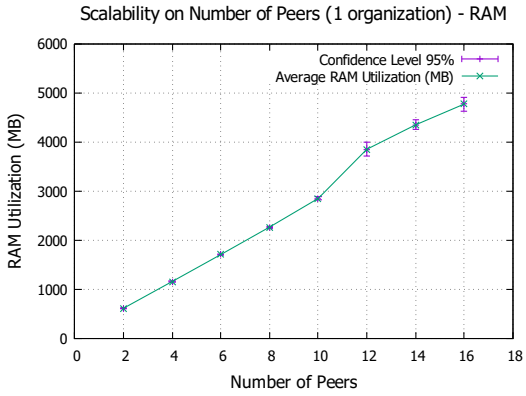
Table 7.15: Additional details on the experiment



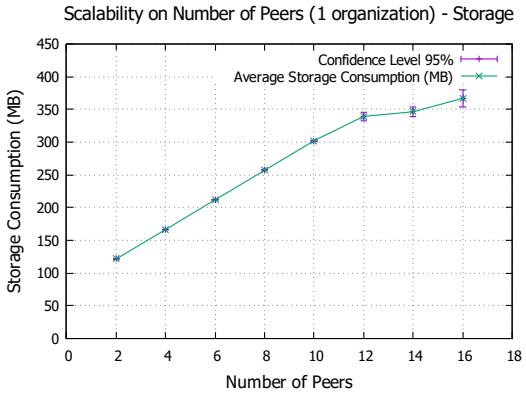
(a) Time required to store data in the blockchain.



(b) Average CPU consumption while storing data in the blockchain



(c) RAM consumption required to store data in the blockchain



(d) Storage consumption required to store data in the blockchain

Figure 7.7: Storage of reports in the blockchain

Results Time. Fig. 7.7a shows that time required to store the reports in the blockchain is linearly proportional to the number of peers, when the latter is lower then 8. With a number of peers greater then 8 the time is established to a value of about 250-300 seconds.

$$t_{store} = 59.62s + 27.46 \cdot n_{peers}$$

CPU utilization. Fig. 7.7b confirms that the CPU utilization of the hosting machine is linearly proportional to the number of peers. This trend can be justified by considering that every peer of the network executes the commit phase of all the transactions. The endorsement of transactions, instead, does not affect the CPU utilization because only one peer sequentially endorses all the transactions.

$$CPU_{store} = 7.19\% + 1.86 \cdot n_{peers}$$

RAM utilization. Fig. 7.7c shows that the deployment of many peer containers requires more main memory (linear dependence).

$$RAM_{store} = 66.57MB + 275.467 \cdot n_{peers}$$

Storage utilization. Fig. 7.7d proves that every peer stores its own replica of the chain of blocks and the state database because the storage utilization is linearly proportional to the number of peers. The experiments characterized by 12, 14 and 16 peers have not been properly executed, i.e. some peers have not received the complete sequence of blocks (probably the storage utilization has been read before all the transactions have been committed) and, for this reason, the last two experiments cannot be considered trustworthy. This problem emerges from Fig. 7.7d, in which the slope of line is reduced when the number of peers exceeds the value of 12.

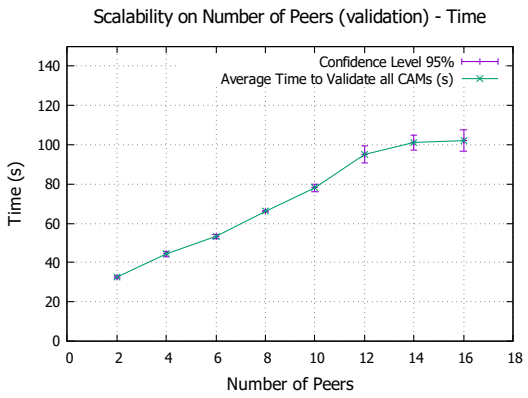
$$HDD_{store} = 76.22MB + 22.59 \cdot n_{peers}$$

Validation of CAMs stored in the blockchain

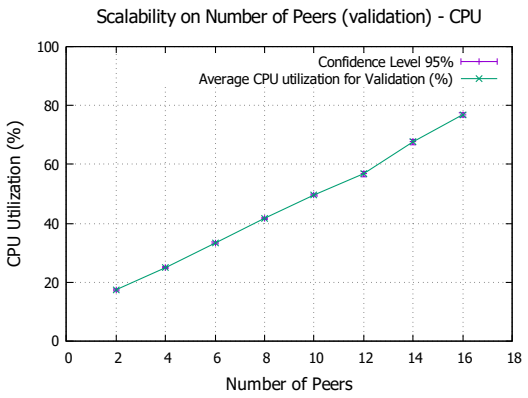
This section analyses the validation of the CAMs stored in the blockchain. The trace used for validation is 7.1.5 (1MB of data split in 10 files of about 100kB each). The reports are not stored in raw format but they have been processed by the chaincode to create many key-value pairs, one for each CAM transmitted and received. The experiment consists in the validation of all the CAMs stored in the blockchain through the execution of a sequence of transactions which validate 20 CAMs at a time. Table 7.16 summarizes the details of the experiment.

CPU peers	10%
CPU orderers	100%
Timeout Peer	300s
Trace Stored	7.1.5 (1MB)
Type of storage	With processing
CAMs validated per transaction	20

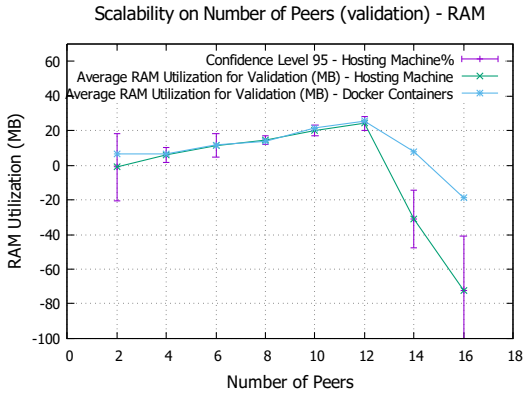
Table 7.16: Additional details of the experiment



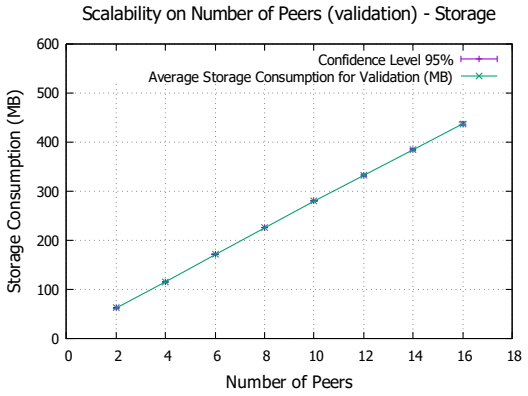
(a) Time required to validate all the CAMs



(b) Average CPU utilization during validation of CAMs



(c) RAM utilization for the validation of CAMs (host and containers)



(d) Storage consumption for the validation of the CAMs

Figure 7.8: Validation of all the CAMs stored in the blockchain

Results

Fig. 7.8a, 7.8b, 7.8d show that time required for the validation of the CAMs, the CPU utilization during the validation and the storage memory consumption are linearly proportional to the number of peers of the network. By increasing the number of peers, in fact, the computational and storage requirements necessary to endorse and commit transactions is also increased. Since the CPU time of each process is restricted to 10% , also the time required to execute the operations is increased as a consequence of the numerous context switches executed by the operating system and by the sequential accesses to the disk that are necessary to read and store the validation information. The increase of the storage memory utilization is determined by the execution of the position verification algorithm that updates the information of the non-valid CAMs by appending new blocks to the blockchain.
The result shown in Fig. 7.8c, instead, is unexpected. The amount of main memory used during the validation of CAMs is slightly increased up to the number of peers is lower than or equal to 12, and then drastically drops below the amount of memory used to store the CAMs. The operating system, in fact, starts swapping the content of the secondary memory to the swap partition. The main memory utilization of the Docker containers was also inspected by using the `docker stats` command. This analysis confirms that the trend of the main memory of the hosting machine

is equivalent to the amount of main memory used by the containers (Fig. 7.8c).
The trend of time, CPU, RAM, and storage memory can be described with the following empirical formulas:

$$t_{val} = 21.27\text{s} + 5.68 \cdot n_{peers}$$
$$CPU_{val} = 9.40\% + 4.02 \cdot n_{peers}$$
$$RAM_{val} = -5\text{MB} + 2.5 \cdot n_{peers}$$
$$HDD_{val} = 7.47\text{MB} + 27.23 \cdot n_{peers}$$

Querying of all the CAMs stored in the blockchain

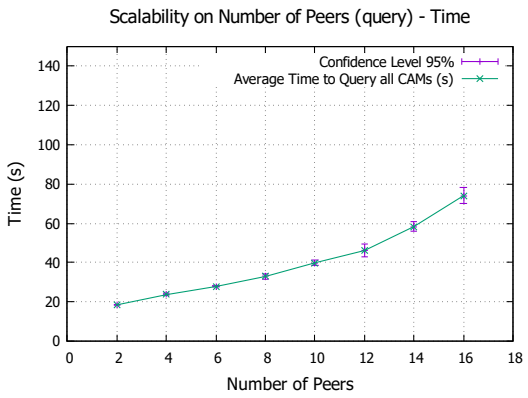
CPU peers	10%
CPU orderers	100%
Timeout Peer	300s
Trace Stored	7.1.5 (1MB)
Type of storage	With processing
CAMs queried per transaction	20

Table 7.17: Additional details of the experiment

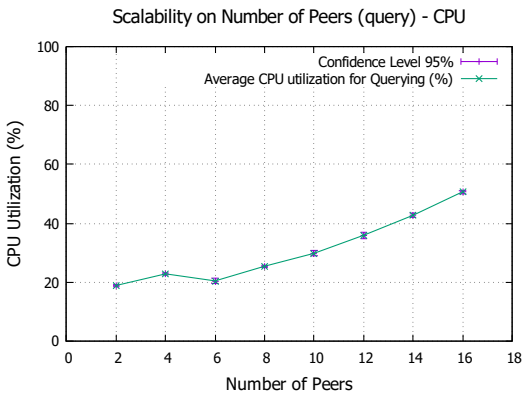
Results

Fig. 7.9a and Fig. 7.9b show that the time and the total CPU utilization required to execute the queries, are linearly dependent on the number of peers of the blockchain. These results are totally unexpected because the transaction flow of query operations does not involve the ordering and committing phases, i.e. only the peer that endorses the transaction performs useful operations. It is evident that every peer, even if it is not involved in the endorsement or ordering of transactions, uses a small percentage of the total computational resources of the hosting machine. Therefore, the context switches performed by the operating system determine the increase of the time required for the execution of query operations.
Finally, the comparison between Fig. 7.8b and Fig. 7.9b shows that the the application of the position verification algorithm is more CPU intensive with respect to the execution of queries.
Fig. 7.9c and Fig. 7.9d show that the main memory used by the querying process is practically absent, while the total storage consumption is established to about 6MB. The latter is independent on the number of peers because no blocks are created and the peers do not replicate new data.
The trend of time, CPU, RAM and storage utilization of the querying process can be described with the following empirical formulas:

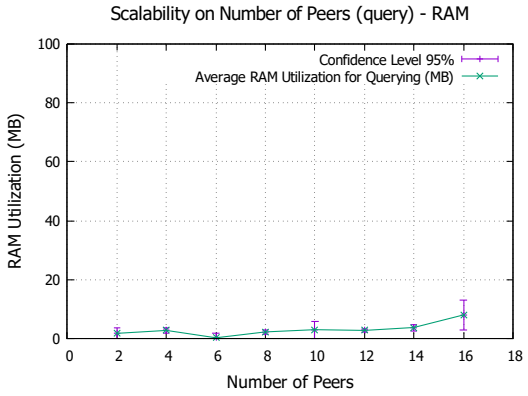
$$t_{query} = 13.2\text{s} + 2.65 \cdot n_{peers}$$
$$CPU_{query} = 16.18\% + 1.36 \cdot n_{peers}$$
$$RAM_{query} = 1.44\text{MB} + 0.15 \cdot n_{peers}$$
$$HDD_{query} = 6.33\text{MB}$$



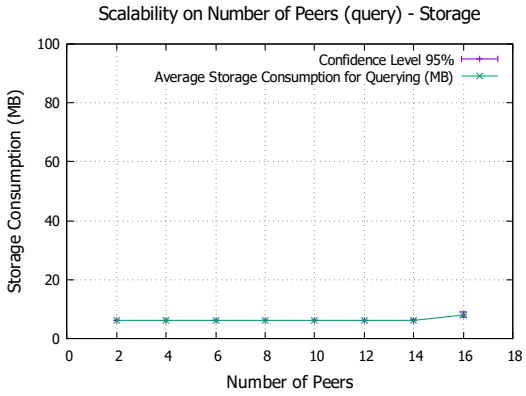
(a) Time required to query all the stored CAMs



(b) Average CPU utilization to query all the stored CAMs



(c) RAM utilization used to query all the stored CAMs



(d) Storage memory utilization to query all the stored CAMs

Figure 7.9: Querying of all the CAMs stored in the blockchain

7.6.2 Experiments with many organizations and 2 peers per organization

This experiment analyses the performance of a blockchain composed by many peers belonging to different organizations, each consisting of two peers. The total number of peers of the different blockchains is equivalent to the one of the experiments shown in Sec. 7.6.1, thus the expected outcome is the same. Table 7.18 shows the configuration of the blockchain networks used in this experiment.

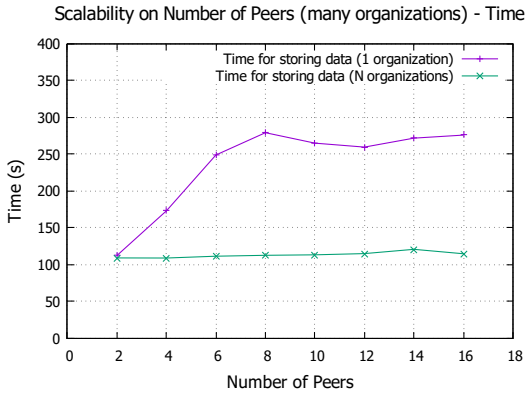
# Organizations	1,2,3,4,5,6,7,8
# Peers per org.	2
# Peers total	2,4,6,8,10,12,14,16
# Orderers	1
Type of orderer	Solo
State Database	CouchDB
Endorsing Policy	Default

Table 7.18: Configuration of the network

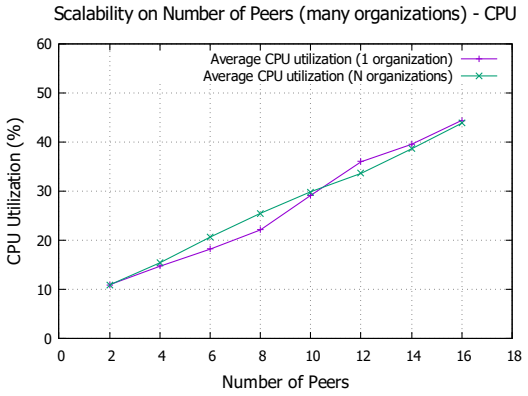
Storage of reports in the blockchain

CPU peers	5%
CPU orderers	100%
Timeout Peer	300s
Trace Stored	7.1.8 (10MB)
Type of storage	Raw Format

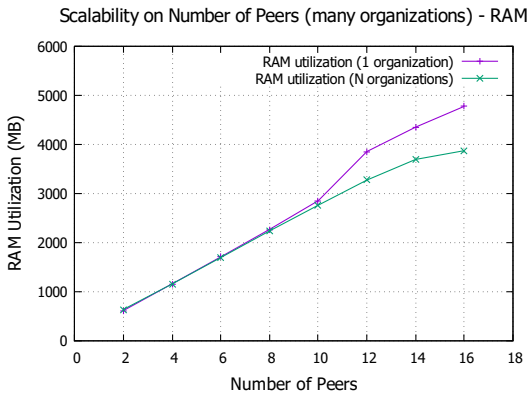
Table 7.19: Additional details of the experiment



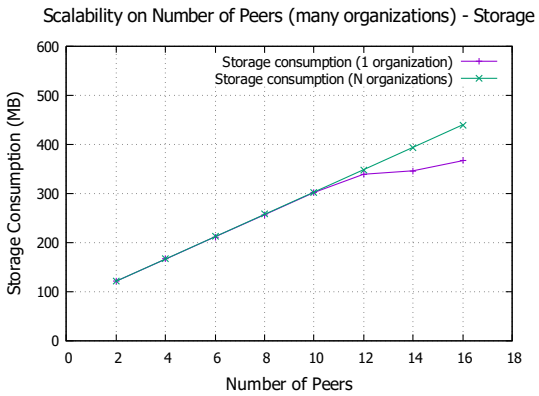
(a) Time required to store data in the blockchain.



(b) Average CPU consumption while storing data in the blockchain



(c) RAM utilization required to store data in the blockchain



(d) Storage memory consumption required to store data in the blockchain

Figure 7.10: Storage of data in the blockchain - N organizations

Results

Fig. 7.10 compares the results of the experiment conducted in Sec. 7.6.1 (characterized by one organization and a variable number of peers) and the experiment of Sec. 7.6.2 (characterized by many organizations consisting of 2 peers each). This comparison is relative to the storage of raw reports in the blockchain. Fig. 7.10a shows that the time required for the endorsement of the transactions is almost constant in the scenarios characterized by many organizations, while grows linearly if the blockchains are composed by one organization only. This behaviour has not

been demonstrated. According to the Hyperledger Fabric documentation, in fact, with the `peer chaincode invoke` command, only one peer of the network endorses the transactions [73]. The default endorsing policy is always satisfied if at least one peer of any organization endorses the transaction, independently on the organization it belongs to. The list of running Docker containers proves that in every experiment only one peer of the network endorses the transactions, because only a chaincode container is created for the execution of the transactions code. For this reason, neither the number of peers or organizations may affect the endorsement of transactions.

Eventually, an in depth analysis of the interactions between peers can be useful to understand this strange behaviour.
The analysis of the CPU, RAM and storage memory utilisation, instead, demonstrated that the two experiments are almost equivalent.
Fig. 7.10d shows that, unlike the networks composed by one organization only, the blockchains consisting of many organizations maintains the linear dependence between the number of peers and the total storage memory utilisation, even if the number of peers is greater than or equal to 12. This is another confirmation of the fact that the interactions of peers belonging to the same organization is different from the inter-organization communication.
The trend of time, CPU, RAM, and storage memory utilisation, necessary to store data in the blockchains composed by many organizations consisting of 2 peers each, can be described with the following empirical formulas:

$$t_{store} = 108.25\text{s} + 0.53 \cdot n_{peers}$$

$$CPU_{store} = 6.14\% + 2.37 \cdot n_{peers}$$

$$RAM_{store} = 110.7\text{MB} + 265.1 \cdot n_{peers}$$

$$HDD_{store} = 76\text{MB} + 22.7 \cdot n_{peers}$$

To prove these results, additional experiments have been executed by creating many blockchains, each one characterized by a different endorsement policy with the format `AND (org1.member, org2.member,...,orgN.member)`, where N indicates the number of organizations taking part to the blockchain. The transactions executed in these experiments were not validated by the committing peers because the `peer chaincode invoke` does not allow different peers endorsing the same transactions; thus, in every experiment composed by many organizations, the endorsement policy of the chaincode has never been satisfied. However, the outcome of these tests demonstrated that, even if the transactions are not validated, the chain of blocks replicated by every peer contains the whole sequences of valid and invalid transactions. At the same way, all the modifications applied to the state database are not committed and do not modify the final state of the ledger.

7.6.3 Experiments with 2 organizations and many peers per organization

The goal of this experiment consists in verifying the results of the tests executed in Sec. 7.6.2. Many blockchains composed exactly by two organizations and a variable number of peers per organization have been deployed.

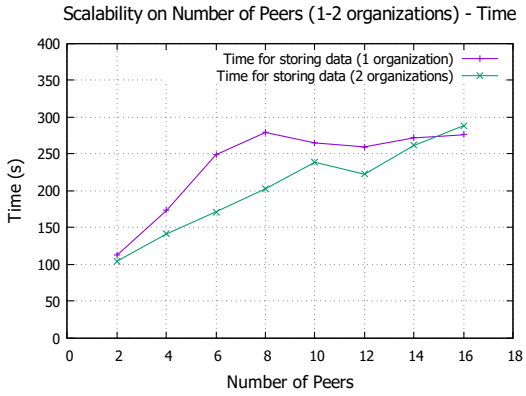
# Organizations	2
# Peers per org.	1,2,3,4,5,6,7,8
# Peers total	2,4,6,8,10,12,14,16
# Orderers	1
Type of orderer	Solo
State Database	CouchDB
Endorsing Policy	Default

Table 7.20: Configuration of the network

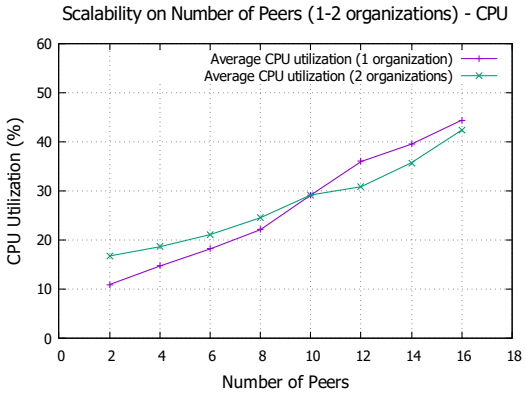
Storage of reports in the blockchain

CPU peers	5%
CPU orderers	100%
Timeout Peer	300s
Trace Stored	7.1.8 (10MB)
Type of storage	Raw Format

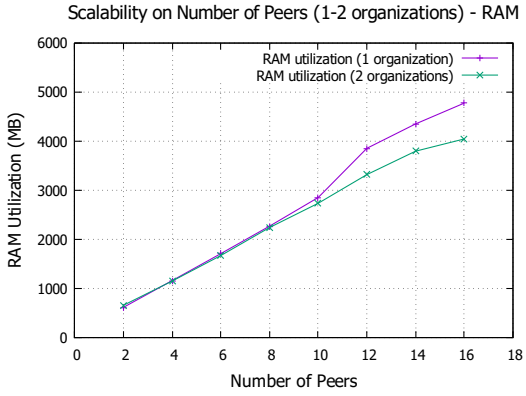
Table 7.21: Additional details of the experiment



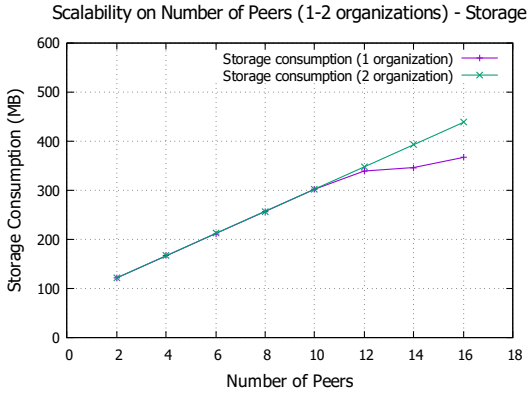
(a) Time required to store data in the blockchain.



(b) Average CPU consumption while storing data in the blockchain



(c) RAM consumption required to store data in the blockchain



(d) Storage consumption required to store data in the blockchain

Figure 7.11: Storage of data in the blockchain - 2 organizations

Results

Fig. 7.11a confirms the results of the experiment of Sec. 7.6.1: the number of organizations affects the time required to execute the endorsement of transactions, despite the overall number of peers of all the organizations is kept constant. The CPU, RAM and storage memory utilisation, instead, are not affected by this parameter and evolve linearly with the number of peers (Fig. 7.11b, Fig. 7.11c, Fig. 7.11d). The trend of time, CPU, RAM, and storage memory necessary to store data in the blockchains composed by 2 organizations, consisting of $n_{peers_{org}}$ peers each, can be described with the following empirical formulas:

$$t_{store} = 70.9\text{s} + 33.6 \cdot n_{peers_{org}}$$
$$CPU_{store} = 13.6\% + 3.1 \cdot n_{peers_{org}}$$
$$RAM_{store} = 140.1\text{MB} + 519.2 \cdot n_{peers_{org}}$$
$$HDD_{store} = 76.5\text{MB} + 45.2 \cdot n_{peers_{org}}$$

7.7 Effect of the number of orderers of the blockchain

The experiments of Sec. 7.7.1, Sec. 7.7.2 and Sec. 7.7.3 analyse how the performance of the blockchain varies by increasing the number of the ordering nodes (Kafka orderers). The experiments relative to the execution of read-only queries have been performed even if the queries do not involve the ordering and committing phases, i.e. the number of orderers should not affect the execution of queries. The configuration of the network used in all the experiments is shown in Table 7.7. The results of the three experiments are shown at the end of Sec. 7.7.3.

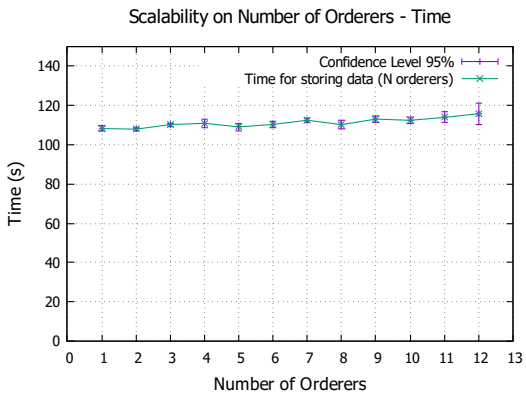
# Organizations	1
# Peers per org.	2
# Peers total	2
# Orderers	1,2,3,4,5,6,7,8,9,10,11,12
Type of orderer	Kafka
State Database	CouchDB
Endorsing Policy	Default

Table 7.22: Configuration of the network

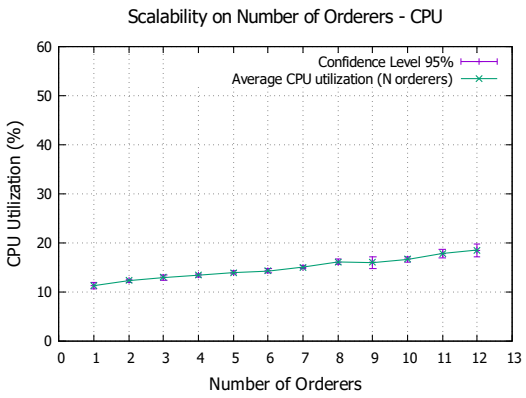
7.7.1 Storage of reports in the blockchain

CPU peers	5%
CPU orderers	5%
Timeout Peer	300s
Trace Stored	7.1.8 (10MB)
Type of storage	Raw Format

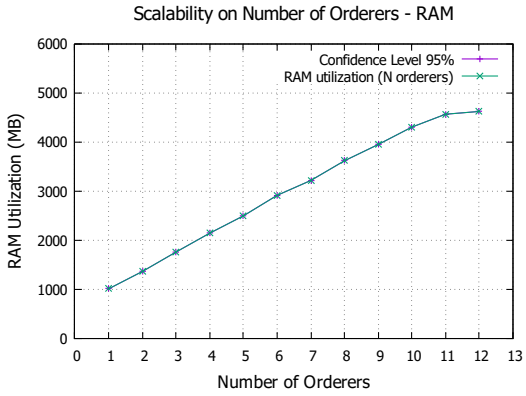
Table 7.23: Additional details of the experiment



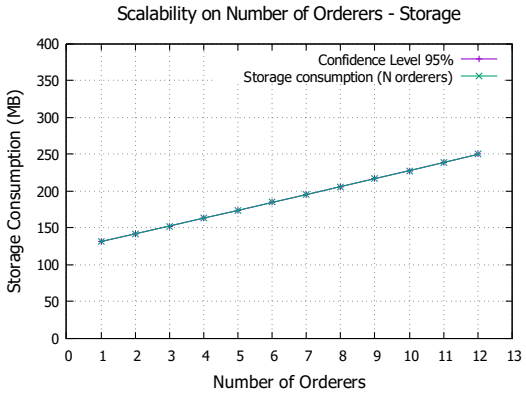
(a) Time required to store data in the blockchain.



(b) Average CPU consumption while storing data in the blockchain



(c) RAM consumption required to store data in the blockchain



(d) Storage consumption required to store data in the blockchain

Figure 7.12: Storage of reports in the blockchain

The trend of time, CPU, RAM and storage memory used to store the CAMs in the blockchain can be described by the following empirical formulas:

$$t_{store} = 107.8\text{s} + 0.46 \cdot n_{orderers}$$

$$CPU_{store} = 10.71\% + 0.59 \cdot n_{orderers}$$

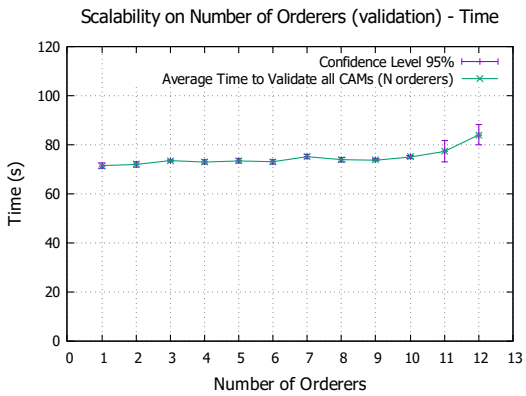
$$RAM_{store} = 646.2\text{MB} + 365.8 \cdot n_{orderers}$$

$$HDD_{store} = 154.2\text{MB} + 7.4 \cdot n_{orderers}$$

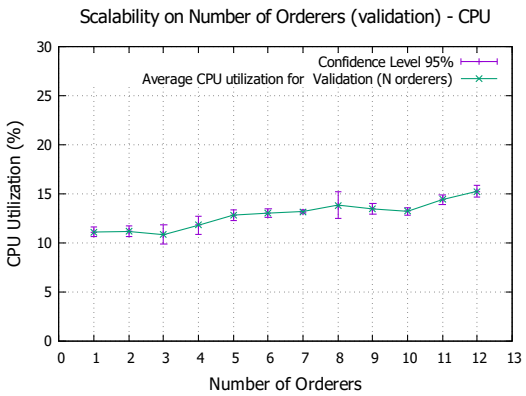
7.7.2 Validation of all the CAMs stored in the blockchain

CPU peers	5%
CPU orderers	5%
Timeout Peer	300s
Trace Stored	7.1.5
Type of storage	With processing
CAMs validated per transaction	20

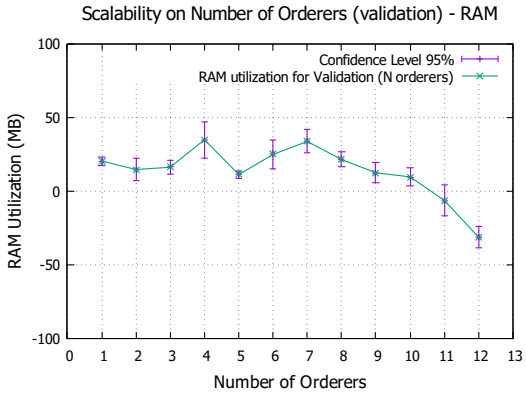
Table 7.24: Additional details of the experiment



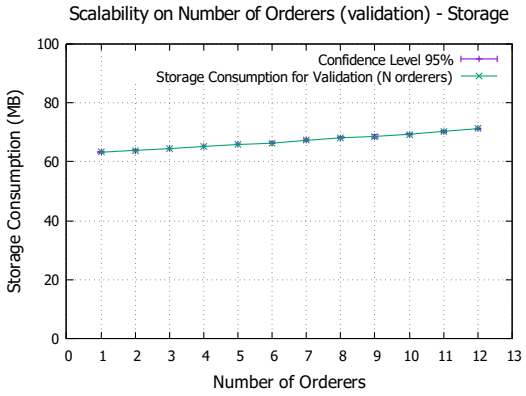
(a) Time required to validate all the CAMs



(b) Average CPU utilization during validation of CAMs



(c) RAM utilization for the validation of CAMs



(d) Storage consumption for the validation of the CAMs

Figure 7.13: Validation of all the CAMs stored in the blockchain

The trend of time, CPU and storage memory used to validate all the CAMs stored in the blockchain can be described by the following empirical formulas:

$$t_{validation} = 71.08\text{s} + 0.39 \cdot n_{orderers}$$

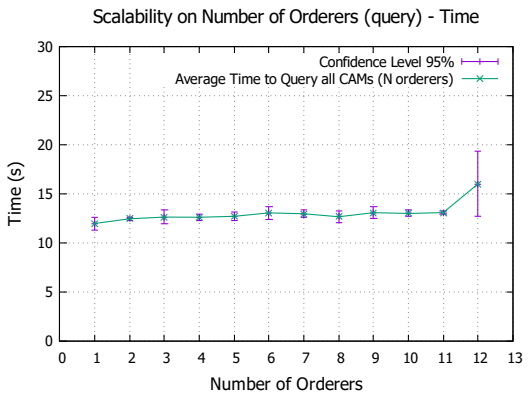
$$CPU_{validation} = 10.88\% + 0.23 \cdot n_{orderers}$$

$$HDD_{validation} = 62.54\text{MB} + 0.69 \cdot n_{orderers}$$

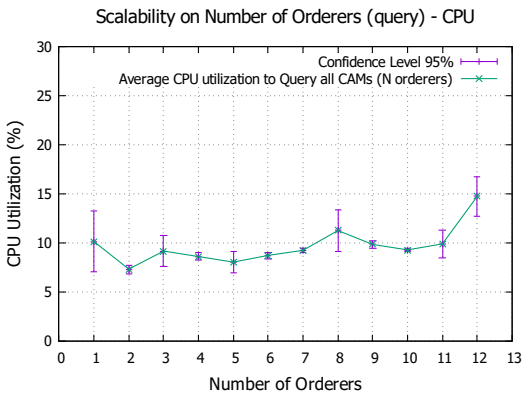
7.7.3 Querying of all the CAMs stored in the blockchain

CPU peers	5%
CPU orderers	5%
Timeout Peer	300s
Trace Stored	7.1.5
Type of storage	With processing
CAMs validated per transaction	100

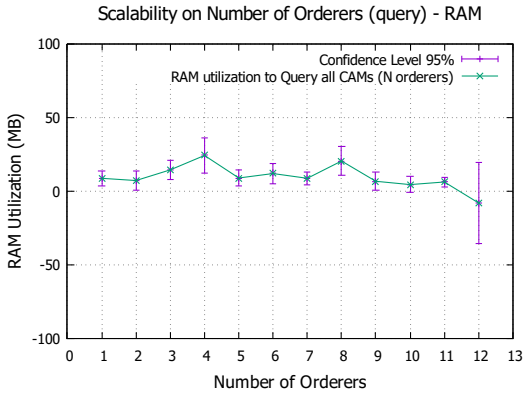
Table 7.25: Additional details of the experiment



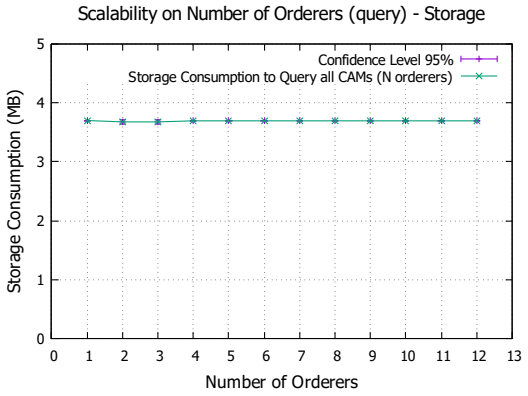
(a) Time required to query all the stored CAMs



(b) Average CPU utilization to query all the stored CAMs



(c) RAM utilization used to query all the stored CAMs



(d) Storage memory utilization to query all the stored CAMs

Figure 7.14: Querying of all the CAMs stored in the blockchain

The trend of time, CPU and storage memory used to query the CAMs stored in the blockchain can be described by the following empirical formulas:

$$t_{query} = 11.87\text{s} + 0.11 \cdot n_{orderers}$$

$$RAM_{query} = 0\text{MB}$$

$$CPU_{query} = 10.23\% - 0.09 \cdot n_{orderers}$$

$$HDD_{query} = 3.70\text{MB}$$

Results

The time required for the endorsement of the transactions used for the validation and storage of CAMs, slightly increases with the number of orderers, as shown in Fig. 7.12a and Fig. 7.13a. The increase of time that is necessary to execute the operations, is related to the growth of the computational requirements of the orderers, as shown in Fig. 7.12b and Fig. 7.13b. The increase in the time and computational requirements is most pronounced in the networks composed by many peers (Fig. 7.7) with respect to the blockchains composed by many orderers (Fig. 7.12). The operations executed by peers (endorsement, signing, verification, committing), in fact, are more computationally expensive then the ones performed by the orderers (ordering

of transactions and signing of blocks). The time measured during the experiments is relative to the time of endorsement of transactions, thus it is more affected by the overhead of the peer processes, with respect to the overhead of the ordering nodes. Concerning the amount of main memory used by the hosting machine, instead, a network with many orderers (Fig. 7.12c) consumes a larger amount of RAM with respect to a network composed by many peers (Fig. 7.7c). Fig. 7.12c shows that during the storage operations, the operating system starts swapping when 11 ordereres are created (the slope of the line describing the main memory utilization is reduced). As usual, the trend of the RAM utilisation during the operations of validation and querying of the blockchain is not predictable and it is typically reduced by increasing the number of orderers (Fig. 7.13c and Fig. 7.14c).
The storage memory utilisation is linearly dependent on the number of orderers, either while storing or validating the CAMs (Fig. 7.12d and Fig. 7.13d). An additional observation about the storage requirements of the network is made by comparing the experiments relative to a blockchain network composed by many orderers (Fig. 7.12d) and a blockchain composed by many peers (Fig. 7.7d). It is evident that the storage memory utilisation is higher in the blockchains composed by many peers with respect to the blockchains characterized by many orderers. Every peer, in fact, stores either the state database and the chain of blocks, while every orderer stores only the chain of blocks.

7.8 Effect of the number of CAMs validated or queried by every transaction

Sec. 7.8.1 and Sec. 7.8.2 aim to identify the best number of CAMs that can be validated or queried with a single transaction, in order to obtain a good validation and querying performance, without the risk of generating very long lasting transactions. Table 7.26 and Table 7.27 show the configuration of both experiments.

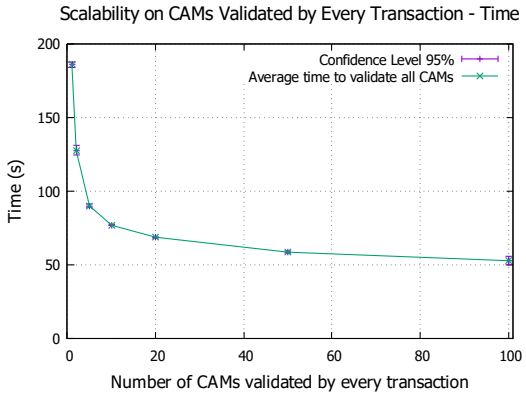
# Organizations	1
# Peers per org.	2
# Peers total	2
# Orderers	1
Type of orderer	Solo
State Database	CouchDB
Endorsing Policy	Default

Table 7.26: Configuration of the network

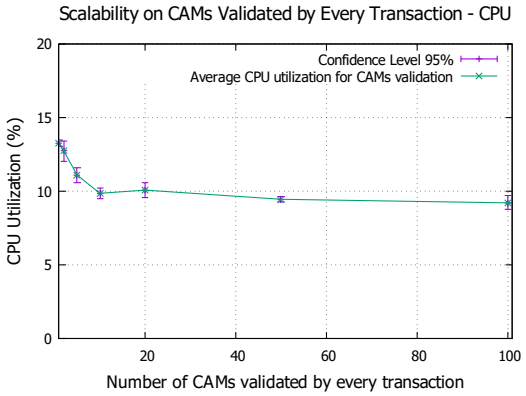
CPU peers	5%
CPU orderers	5%
Timeout Peer	300s
Trace Stored	7.1.5
Type of storage	With processing
CAMs validated or queried per transaction	1, 2, 5, 10, 20, 100, 200

Table 7.27: Additional details of the experiment

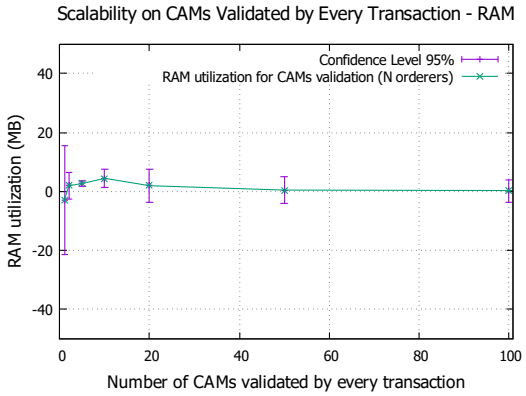
7.8.1 Validation of CAMs stored in the blockchain



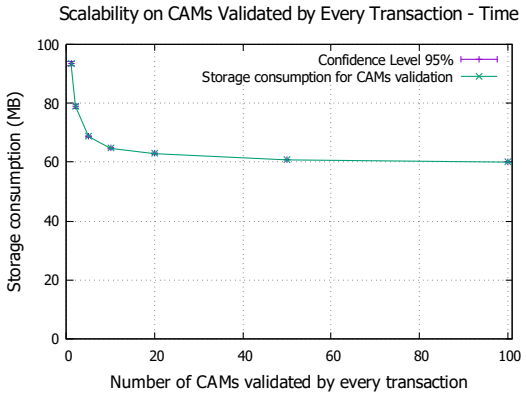
(a) Time required to validate all the CAMs



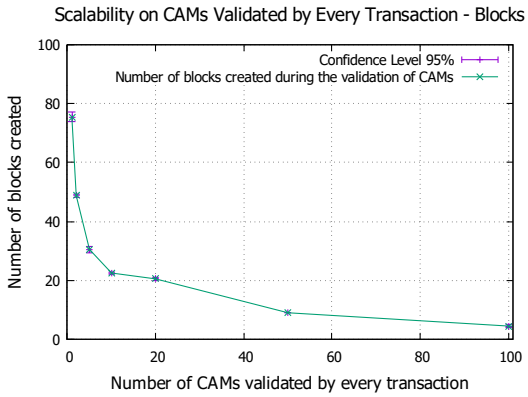
(b) Average CPU utilization during validation of CAMs



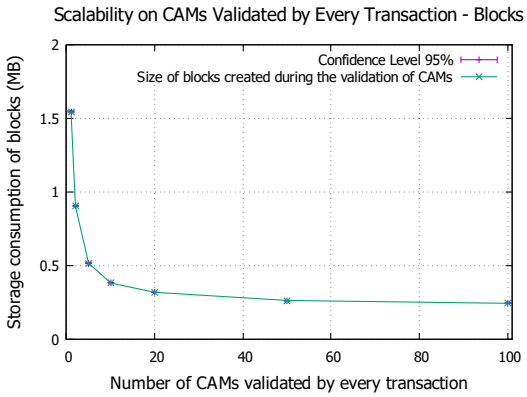
(c) RAM utilization for the validation of CAMs



(d) Storage consumption for the validation of the CAMs



(e) Number of blocks created for the validation of the stored CAMs



(f) Storage utilization of blocks created for the validation of the stored CAMs

Figure 7.15: Validation of all the CAMs stored in the blockchain

The trend of time, CPU, RAM, storage memory utilisation, number and size of the blocks created for validating the CAMs stored in the blockchain, can be described with the following empirical formulas (n_{cams} indicates the number of CAMs validated

by every transaction)

$$t_{validation} = 52.11906 + \frac{286182500 - 52.11906}{1 + (\frac{n_{cams}}{7.787183 \cdot 10^{-9}})^{0.7809806}}$$

$$CPU_{validation} = 9.028278 + \frac{14.92664 - 9.028278}{1 + (\frac{n_{cams}}{3.012399})^{0.9153901}}$$

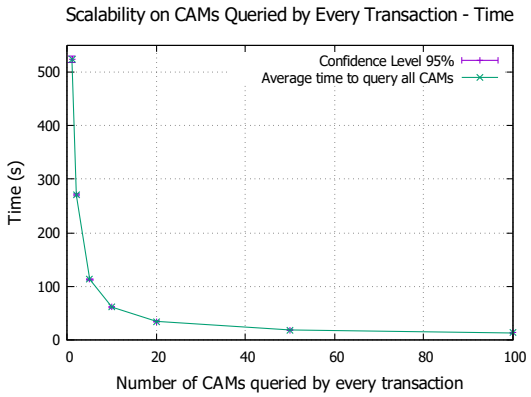
$$RAM_{validation} = 0\text{MB}$$

$$HDD_{validation} = 59.26925 + \frac{19230680 - 59.26925}{1 + (\frac{n_{cams}}{5.644042 \cdot 10^{-8}})^{0.7933975}}$$

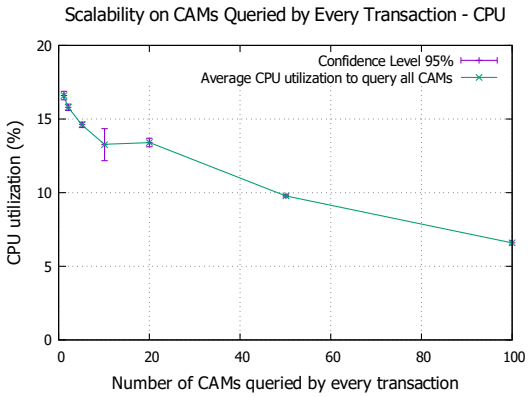
$$NBlocks_{validation} = 0.7139051 + \frac{143972100 - 0.7139051}{1 + (\frac{n_{cams}}{1.810446 \cdot 10^{-12}})^{0.5359125}}$$

$$SizeBlocks_{validation} = 0.2346004 + \frac{2849813 - 0.2346004}{1 + (\frac{n_{cams}}{2.28744 \cdot 10^{-7}})^{0.9544648}}$$

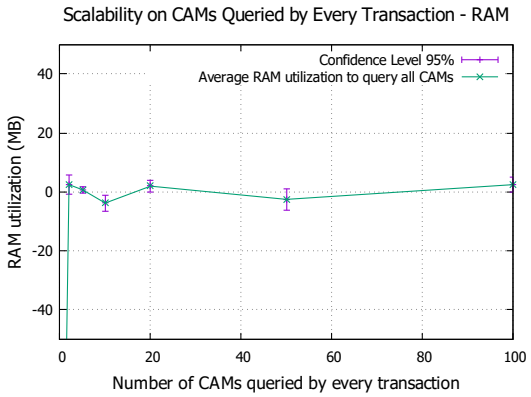
7.8.2 Querying of all CAMs stored in the blockchain



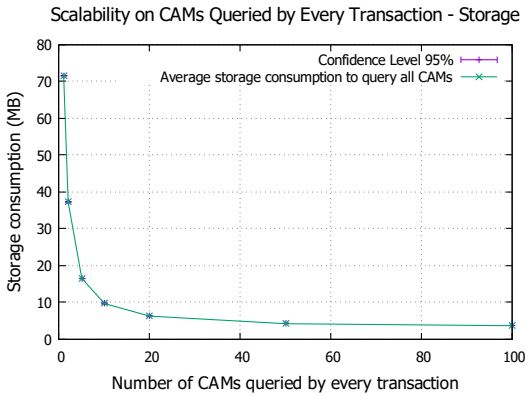
(a) Time required to query all the stored CAMs



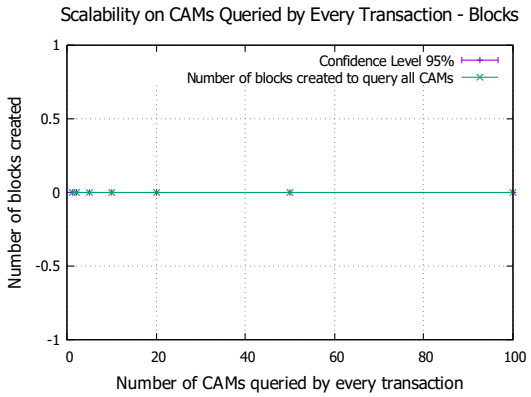
(b) Average CPU utilization to query all the stored CAMs



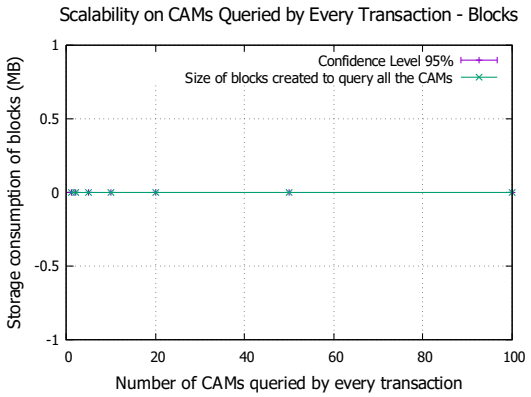
(c) RAM utilization used to query all the stored CAMs



(d) Storage memory utilization to query all the stored CAMs



(e) Number of blocks created to query all the stored CAMs



(f) Storage utilization of blocks created to query all the stored CAMs

Figure 7.16: Querying of all the CAMs stored in the blockchain

The trend of time, CPU, RAM, storage memory utilisation, number and size of the blocks created for querying the CAMs stored in the blockchain can be described with the following empirical formulas (n_{cams} indicates the number of CAMs queried with a single transaction):

$$t_{query} = 7.97985 + \frac{15035.94 - 7.97985}{1 + (\frac{n_{cams}}{0.03598113})^{1.003272}}$$

$$CPU_{query} = -63339.76 + \frac{17.2982 + 63339.76}{1 + (\frac{n_{cams}}{3254528000})^{0.5025865}}$$

$$RAM_{query} = 0\text{MB}$$

$$HDD_{query} = 3.003054 + \frac{2832.753 - 3.003054}{1 + (\frac{n_{cams}}{0.02662319})^{1.019187}}$$

$$NBlock_{query} = 0$$

$$SizeBlocks_{query} = 0\text{MB}$$

Results

Fig. 7.15 shows that the performance of the blockchain varies by modifying the number of transactions validated by every transaction. By validating many CAMs with a single transaction it is possible to reduce the time necessary to validate all the CAMs stored in the blockchain (Fig. 7.15a). In a real deployment environment, the validation of many CAMs at a time requires to wait some time between storing the CAMs in the blockchain and validating them. This implies that the time saved during the execution of the transactions is spent to wait that enough CAMs are stored (and committed) in the blockchain. This problem does not affect the densely populated vehicular environments, but it can be problematic in the case the base stations receive a very limited number of reports to be stored in the blockchain. To deal with this problem, it is necessary to impose a maximum number of CAMs that can be stored and a maximum timeout within which the base stations must generate a transaction to validate the previously stored CAMs. This waiting time, apparently problematic for the performance of the platform, can be exploited to create larger blocks (e.g. characterized by large values of BatchTimeout and MaxMessageCount) to reduce the computational and storage overhead introduced by the creation of blocks. With larger blocks, the base stations should wait a time interval greater than the BatchTimeout before they can safely execute the transactions for the validation of the CAMs.
The amount of CPU utilization (Fig. 7.15b) is reduced by validating many CAMs with a single transaction, because the computational overhead, necessary to digitally sign and to verify the signatures, is drastically reduced.
The amount of main memory used during the validation process (Fig. 7.15c) is independent on the number of CAMs validated by every transaction and assumes a nil value.
The storage memory overhead (Fig. 7.15d), the number of created blocks (Fig. 7.15e) and the storage utilization of blocks (Fig. 7.15f) show that the storage overhead introduced by the validation is inversely proportional to the number of blocks, thus it is suggested to validate many CAMs at a time to limit the total storage requirements of the application. Similarly to the validation of CAMs, also the operations of querying are affected by the number of CAMs retrieved by every transaction

(Fig. 7.16).
The most interesting result of the querying process is the total storage memory utilization: even if no blocks are created (Fig. 7.16e and Fig. 7.16f), the hosting machine storage utilization moves from circa 70MB to 4MB (7.16d); this proves that the overhead introduced by the transactions is enormous, and that is necessary to write few data as possible in the logs, especially in the real deployment environments.

The whole analysis demonstrated that, by validating or querying 20 CAMs per transaction, a good trade-off between the time of execution of transactions, and the total storage and computational overhead is obtained. It is better to avoid very long-lasting transactions but, at the same time, it would be preferable to reduce the computational and storage overhead of transactions. It is discouraged to manage too many CAMs with a single transaction, because, if the time of execution of a transaction increases too much, the transaction execution timeout may elapse and invalidate the transaction itself.

7.9 Comparison of validation with query and invoke

The commands `peer chaincode invoke` and `peer chaincode query` can be used to execute read-write and read-only transactions respectively. By using the query command to execute chaincode functions to update the state of the ledger, the modifications are not committed because the clients do not forward the transaction requests to the ordering service nodes, i.e. only the endorsing phase is executed. This section describes the effects of the validation of CAMs executed as query or invoke transactions. The configuration of the network and the experiment are described in Table 7.28 and Table 7.29. To better understand the results, it is important to remember the configuration of the parameters necessary for the creation of blocks (Table 7.30).

# Organizations	1
# Peers per org.	2
# Peers total	2
# Orderers	1
Type of orderer	Solo
State Database	CouchDB
Endorsing Policy	Default

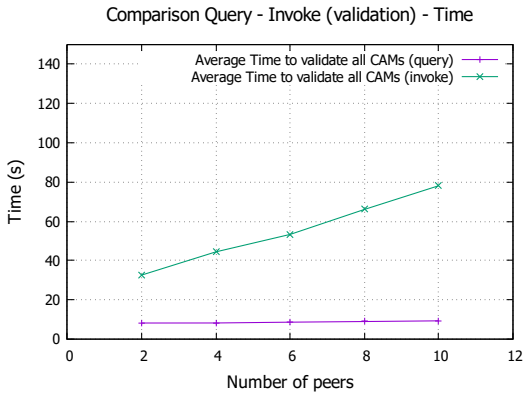
Table 7.28: Configuration of the network

CPU peers	10%
CPU orderers	100%
Timeout Peer	300s
Trace Stored	7.1.5 (1MB)
Type of storage	With processing
CAMs validated per transaction	20

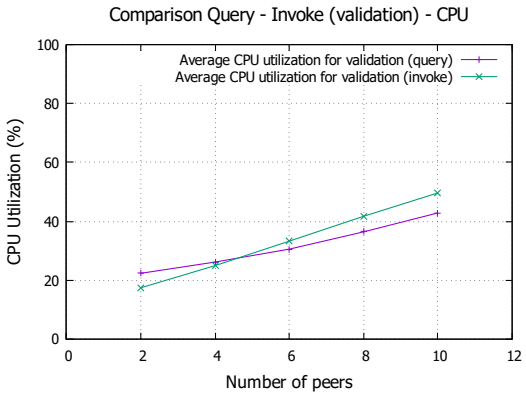
Table 7.29: Additional details of the experiment

`BatchTimeout`	2s
`MaxMessageCount`	10
`AbsoluteMaxBytes`	99MB
`PreferredMaxBytes`	512kB

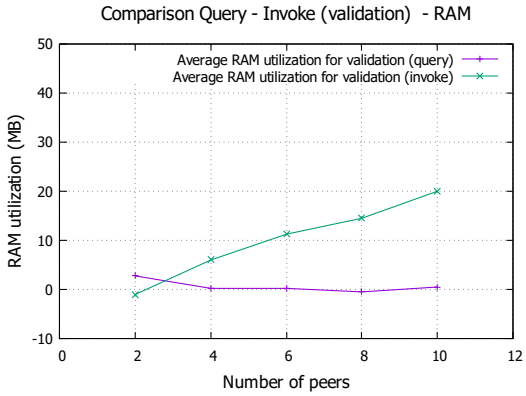
Table 7.30: Configuration of blocks generation



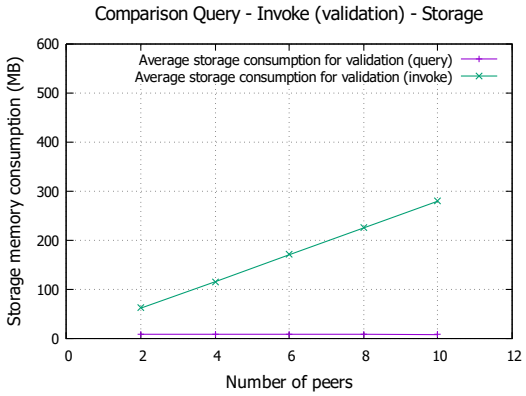
(a) Time required to validate all the stored CAMs



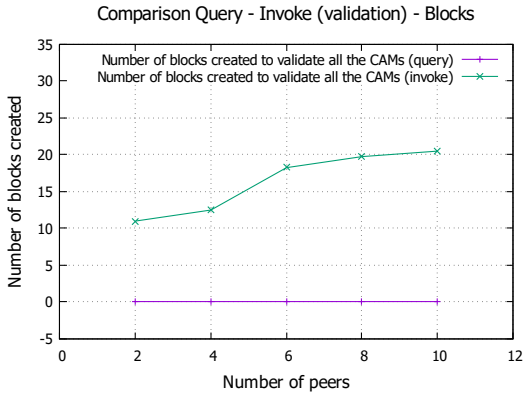
(b) Average CPU utilization to validate all the stored CAMs



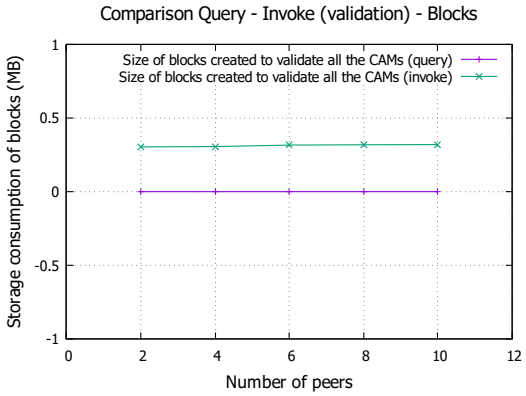
(c) RAM utilization used to validate all the stored CAMs



(d) Storage memory utilization to validate all the stored CAMs



(e) Number of blocks created to validate all the stored CAMs



(f) Storage utilization of blocks created to validate all the stored CAMs

Figure 7.17: Validation of all stored CAMs with `query` and `invoke`

The following empirical formulas describe the trend of time, CPU, RAM, storage memory utilization, number and size of blocks created for validating the CAMs, when the `peer chaincode query` command is used (n_{cams} indicates the number of CAMs validated with a single transaction, vq indicates the validation executed with `query` command).

$$t_{vq} = 7.92\text{s} + 0.14 \cdot n_{cams}$$

$$CPU_{vq} = 17.23\% + 2.56 \cdot n_{cams}$$

$$RAM_{vq} = 0\text{MB}$$

$$HDD_{vq} = 7.85\text{MB}$$

$$NBlocks_{vq} = 0$$

$$SizeBlocks_{vq} = 0\text{MB}$$

The mathematical formulas describing the validation of CAMs with the `invoke` command are the same shown in Sec. 7.6.1.

Results

The most interesting results of this experiment are relative to the time and the amount of computational power required by the validation of CAMs (Fig. 7.17a and Fig. 7.17b). The CPU utilisation of the two experiments, invoke and query, is almost equivalent, while the time required by the transactions executed with the invoke command is notably larger. This behaviour can be explained because the CPU time assigned to every peer has been limited to the 10% of the total computational capabilities of the hosting machine. Furthermore, the numerous context switches occurred when the invoke command is used, due to the execution of the ordering and committing phases of transactions, may impact on the execution time, with an under-utilization of the computational power of the hosting machine. Another possible reason for the increase of time required by the invoke transactions, might be the sequential execution of read-write operations on the secondary memory. The query command does not require to update the ledger, while, using the invoke command, data is read and written to the blockchain during the commit phase. If many read-write requests of access to disk are enqueued, the time required by the operations may increase.
Fig. 7.17d, Fig. 7.17e and Fig. 7.17f confirm the previous assertion, by proving that the invoke transactions write a considerable amount of data to the secondary memory, while the information stored with the query transactions is practically nil (no blocks are created). Fig. 7.17d suggests that the large storage overhead is generated only by the invoke transactions, thus it is probably introduced by the ordering and committing phases of the transactions, and not by the endorsing phase.

7.10 Experiments conducted in multi-process environment

This section deals with the evaluation of performance of Hyperledger Fabric in a multi-process environment. Many parallel processes invoke transactions on different

endorsing peers, to store, validate and query the CAMs. The peculiarity distinguishing Hyperledger Fabric from the other blockchain platforms, in fact, is the possibility to increase the throughput by limiting the execution of the code of transactions to a subset of the peers of the blockchain (the validation and committing phases are instead executed by all the peers). Many network configurations have been used and, for each of them, different experiments have been conducted by modifying the maximum number of concurrent processes (n) invoking the transactions on the different endorsing peers. The number of concurrent processes is limited by the number of peers of the network, thus, the benchmarking tool waits for the completion of the endorsement of all the transactions previously executed before issuing other transactions, i.e. it waits for the completion of all the background processes (`wait` bash command). The time required by the endorsement of all the transactions is measured as the time necessary for the completion of all the background processes. Table 7.31 and Table 7.32 describe respectively the configuration of the network and the additional details of the experiment. This section shows only the experiments conducted with 6 peers because the experiments executed with another number of peers show the same trend.

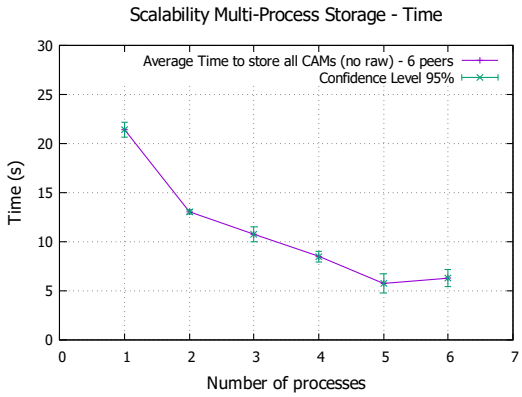
# Organizations	1
# Peers per org.	6
# Peers total	6
# Orderers	1
Type of orderer	Solo
State Database	CouchDB (unless otherwise specified)
Endorsing Policy	Default

Table 7.31: Configuration of the network

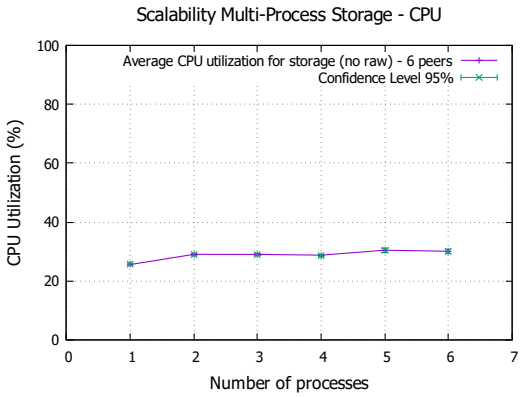
CPU peers	10%
CPU orderers	100%
Timeout Peer	300s
Trace Stored	7.1.5 (1MB split in 100kB files)
Type of storage	With processing
CAMs validated or queried per transaction	20
Number of processes	Variable from 1 to 6

Table 7.32: Additional details of the experiment

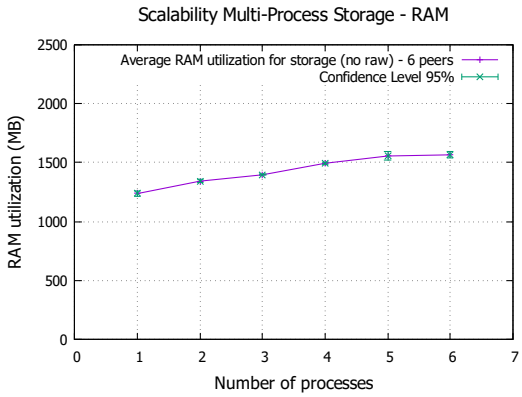
7.10.1 Storage of CAMs with concurrent processes



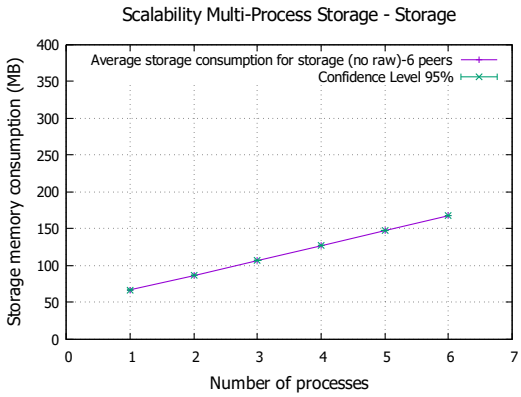
(a) Time required to validate all the stored CAMs



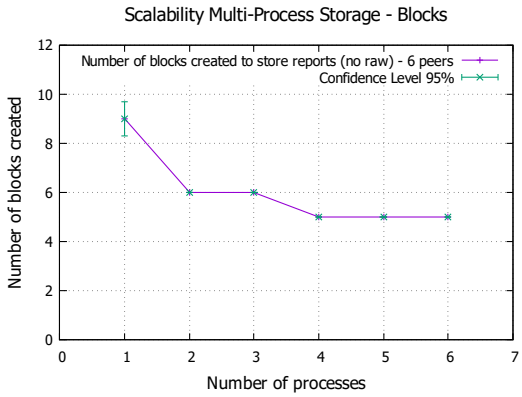
(b) Average CPU utilization to validate all the stored CAMs



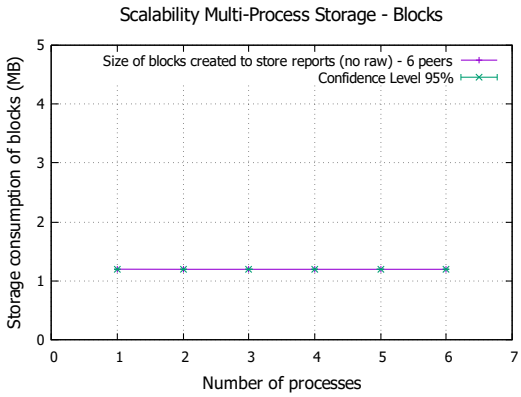
(c) RAM utilization used to validate all the stored CAMs



(d) Storage memory utilization to validate all the stored CAMs



(e) Number of blocks created to validate all the stored CAMs



(f) Storage utilization of blocks created to validate all the stored CAMs

Figure 7.18: Storage of CAMs in multi-process environment (6 peers)

The trend of time, CPU, RAM, storage memory utilisation, number and size of blocks created for storing the CAMs in the blockchain can be described with the following formulas ($n_{process}$ indicates the number of endorsing peers that simultane-

ously endorse the transactions for storing data):

$$t_{validation} = -3.534676 + \frac{3326442 + 3.534676}{1 + (\frac{n_{process}}{3.929898 \cdot 10^{-10}})^{0.5450249}}$$

$$CPU_{validation} = 24.69\% + 0.90 \cdot n_{process}$$

$$RAM_{validation} = 1171.25\text{MB} + 65.75 \cdot n_{process}$$

$$HDD_{validation} = 47.17\text{MB} + 20.11 \cdot n_{process}$$

$$NBlocks_{validation} = 4.669227 + \frac{5939120 - 4.669227}{1 + (\frac{n_{process}}{0.00006843098})^{1.474069}}$$

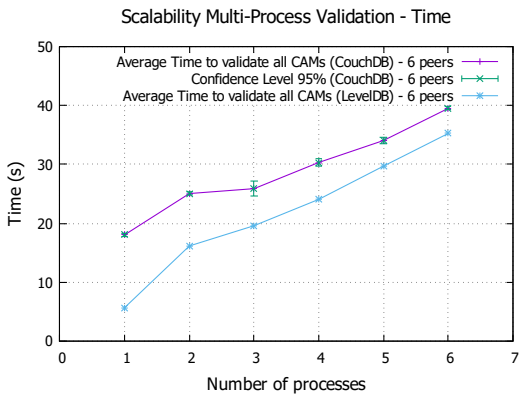
$$SizeBlocks_{validation} = 1.19\text{MB}$$

Results

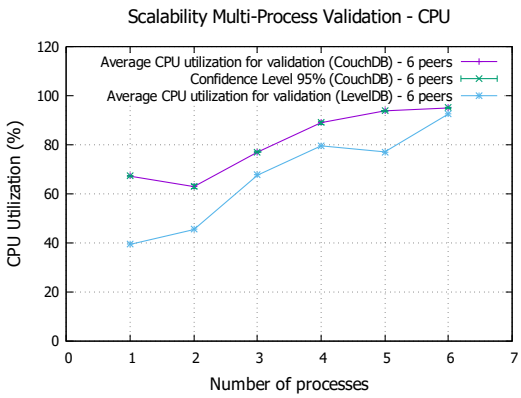
Fig. 7.18 shows that the concurrent endorsement of transactions improve the performance of Hyperledger Fabric. In particular, Fig. 7.18a proves that the time necessary to endorse the transactions is inversely proportional to the number of concurrent processes. The rate of endorsement of transactions also affects the generation of blocks (Fig. 7.18e): the faster the endorsement of transactions, the lower the number of blocks created (the block configuration parameters that determines the creation of new blocks in this experiment are the BatchTimeout and MaxMessageCount). The size of the chain of blocks (Fig. 7.18f), instead, remains basically constant because the difference in the number of generated blocks is negligible, i.e. the overhead introduced by the additional blocks is very low. The CPU utilization (Fig. 7.18f) proves that the endorsement of transactions storing data in the blockchain is not a computationally expensive operation, thus the CPU utilisation has a limited linear growth, proportional to the number of processes. The increase of the number of processes, i.e. the number of Docker containers created for the execution of the chaincode functions, determines a larger main memory utilization (Fig. 7.18c). Finally, the secondary memory utilisation is linearly dependent on the number of processes because, even if the number of peers is kept constant, many Docker containers have been created for the execution of the chaincode.

7.10.2 Validation of CAMs with concurrent processes

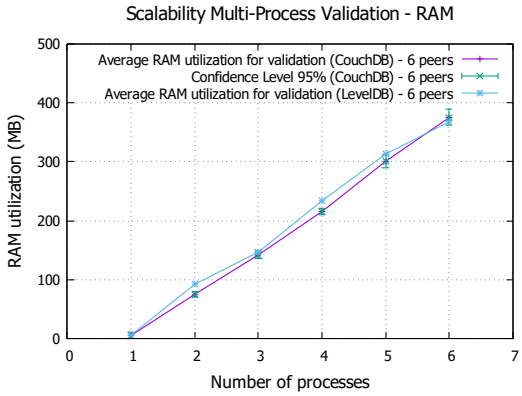
This section shows the results of the experiments of validation of CAMs conducted with many concurrent parallel processes. The discussion of the results is shown at the end of Sec. 7.10.3. The charts of Fig. 7.19 show the result of the experiments that have been conducted with CouchDB and LevelDB state database.



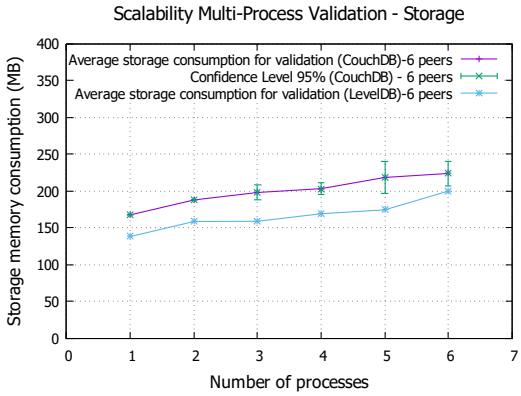
(a) Time required to validate all the stored CAMs



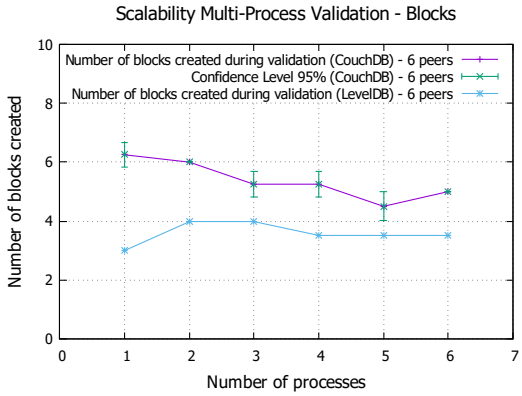
(b) Average CPU utilization to validate all the stored CAMs



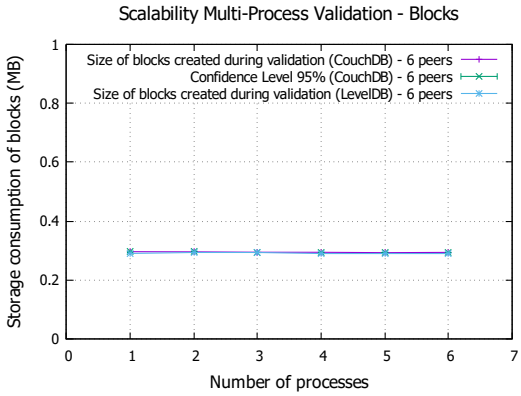
(c) RAM utilization used to validate all the stored CAMs



(d) Storage memory utilization to validate all the stored CAMs



(e) Number of blocks created to validate all the stored CAMs

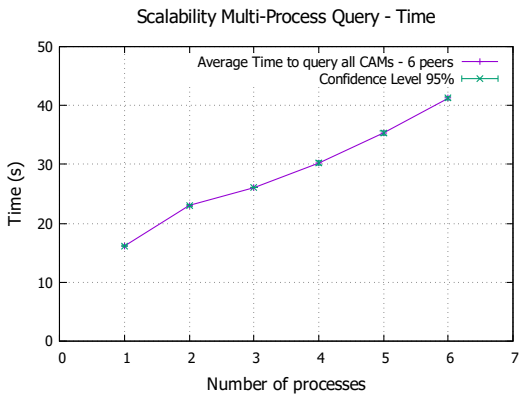


(f) Storage utilization of blocks created to validate all the stored CAMs

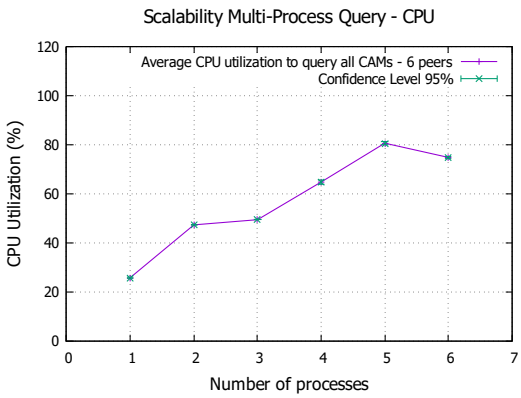
Figure 7.19: Validation of all the stored CAMs in multi-process environment (6 peers)

7.10.3 Querying of CAMs with concurrent processes

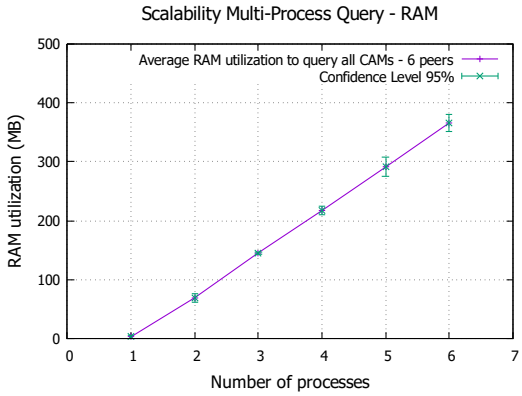
This section shows the results of the experiments of querying of CAMs conducted with many concurrent parallel processes.



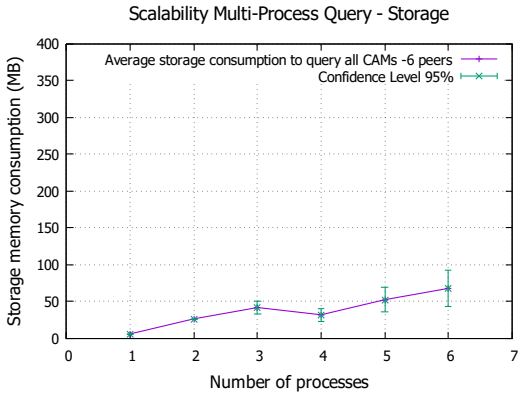
(a) Time required to query all the stored CAMs



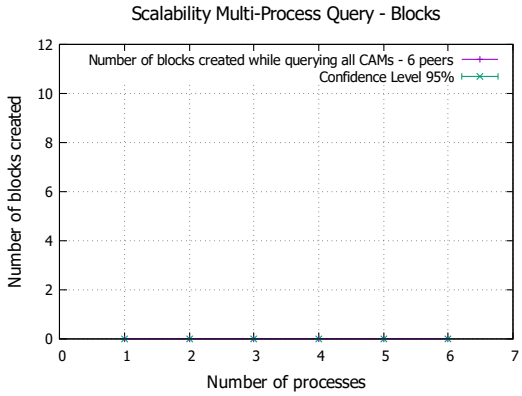
(b) Average CPU utilization to query all the stored CAMs



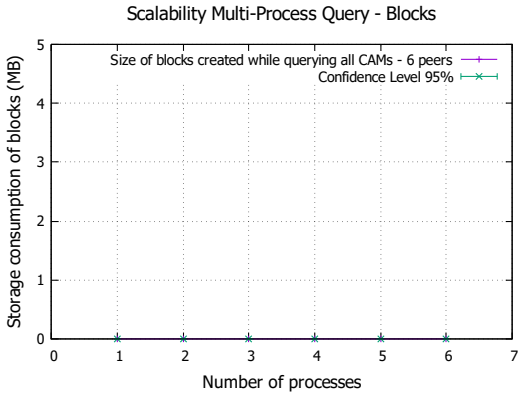
(c) RAM utilization used to query all the stored CAMs



(d) Storage memory utilization to query all the stored CAMs



(e) Number of blocks created to query all the stored CAMs



(f) Storage utilization of blocks query to validate all the stored CAMs

Figure 7.20: Querying of stored CAMs in multi-process environment (6 peers)

The trend of time, CPU, RAM, storage memory utilisation, number and size of blocks created for the validation of the CAMs with the concurrent endorsement of transactions can be described with the following formulas (data is relative to the adoption of CouchDB state database):

$$t_{validation} = 14.74\text{s} + 4.158 \cdot n_{process}$$

$$CPU_{validation} = 61.38\% + 5.45 \cdot n_{process}$$

$$RAM_{validation} = -68.10\text{MB} + 70.35 \cdot n_{process}$$

$$HDD_{validation} = 156.95\text{MB} + 11.23 \cdot n_{process}$$

$$NBlocks_{validation} = 7.65 - 0.40 \cdot n_{process}$$

$$SizeBlocks_{validation} = 0.30\text{MB}$$

The trend of time, CPU, RAM, storage memory utilisation, number and size of blocks created for querying the CAMs with the concurrent endorsement of transactions can be described with the following formulas:

$$t_{query} = 11.15\text{s} + 5.01 \cdot n_{process}$$

$$CPU_{query} = 9.79\% + 16.08 \cdot n_{process}$$

$$RAM_{query} = -69.60\text{MB} + 72.60 \cdot n_{process}$$

$$HDD_{query} = -6.06\text{MB} + 12.36 \cdot n_{process}$$

$$NBlocks_{query} = 0$$

$$SizeBlocks_{query} = 0\text{MB}$$

Results

The experiments concerning the validation (Fig. 7.19) and the execution of queries (Fig.7.19) have been analysed together because their graphs show a similar trend. First of all, the time required to concurrently endorse the transactions is linearly proportional to the number of concurrent processes endorsing the transactions. This behaviour is totally unexpected and different from the result obtained for the storage of CAMs. The deployment of the blockchain on a single hosting machine is probably the reason of this linear dependence between number of threads and execution time. The validation of CAMs and the execution of queries have proved to be computationally heavier (Fig. 7.19b and Fig. 7.20b) with respect to the operations necessary to store the CAMs in the blockchain (Fig. 7.18b). Thus, the limited number of cores of the hosting machine makes necessary for the operating system to execute numerous context switches to assign CPU cycles to every peer process. This determines an increase in the amount of time required by the operations. However, this behaviour should negatively influence also the experiments relative to the storage of CAMs. Another possible reason of this performance degradation has been identified in the sequential accesses to disk. The execution of queries and the validation of CAMs,

require to access the state database to read the CAM, and, in the case of validation, to update the CAMs marked as invalid. For this reason, the storage unit becomes a bottleneck in the case many concurrent processes tries to read data from the state database. The access to disk slightly affects the storage of CAMs because, in this case, it is not necessary to access the state database during the endorsement of transactions. Moreover, for storing CAMs, also the committing phase does not require to read data from the state database because but it is sufficient to store the new key-value pairs in the ledger (no read-write conflicts are possible). Definitely, the time of endorsement of the transactions of validation and querying, are heavily influence by the performance degradation consisting in the access to disk.
The experiment of validation has been conducted with both CouchDB and LevelDB state databases, to verify if the adoption of different implementations of the state database could improve the performance. Unfortunately, the trend of linear dependence between processes and the time of execution still remains.
Fig. 7.19c, Fig. 7.19d, Fig. 7.20c and Fig. 7.20d show that the RAM and storage memory utilisation are linearly dependent on the number of processes concurrently endorsing the transactions.

7.11 Final considerations on the scalability

The scalability test conducted in this chapter showed the limitations of the deployment environment adopted in this thesis. These performance evaluations, cannot be considered representative of the real scalability properties of Fabric. All the tests, in fact, have been conducted on a single hosting machine and the results of the experiments might have been compromised by the limited computational and storage capabilities of the deployment environment. For example, all the experiments demonstrated the linear dependence between the number of peers of the network and the performance of the blockchain, in terms of CPU utilisation and time of execution of the operations. The performance of Fabric should degrade only with a large number of peers, and not by varying the number of peers from 2 to 4 peers. Moreover, the communication between peers, hosted in the same hosting machine, is not affected by the latency and time of propagation of information that affects the blockchains composed by geographically separated nodes. Thus, the linear dependence between the number of peers and the time of execution of transactions was totally unexpected.
On the contrary, the number of orderers of the network, slightly affects the performance and the computational requirements of the blockchains; the CPU intensive operations, in fact, are executed by peers and not by orderers. However, the number of orderers that has been used in the experiments was restricted by their high main memory utilisation.
Another important limitation of the deployment environment has been identified as the difficulty of concurrently endorse the transactions. The simultaneous endorsement of transactions across different peers is one of the characteristics that allows to Fabric to reach good throughput levels. In the experiments that have been executed in this chapter, the decrease of the time necessary for the execution of transactions, obtained by concurrently endorsing transactions, was achieved only for the write-only transactions. On the contrary, the transactions that performed read operations and modified the content of previously stored data, showed a linear dependence be-

tween the number of processes and the time of execution of transactions. The bottleneck introduced by the limited computational capabilities and the concurrent access to disk, demonstrated the difficulty in testing the performance of Fabric in a scenario characterized by many peers concurrently endorsing the transactions.

Chapter 8

Conclusions

This chapter summarizes the results of the thesis, highlighting the strengths and the weakness of the blockchain platforms when they are used for general-purpose applications (Sec. 8.1). Sec. 8.2 summarizes the most important characteristics of Hyperledger Fabric, shows the limitations of the system architecture designed for the vehicular application and, finally, suggests other possible architectures. Sec. 8.3 concludes the thesis and identifies the future research directions concerning Hyperledger Fabric and the permissionless blockchain platforms.

8.1 Permissionless and permissioned blockchains

The analysis of different permissionless blockchains, such as Bitcoin and Ethereum, highlighted the limitations of these platforms: low throughput, high latency, full data replication and high energy consumption. In addition, the security of these platforms can be compromised if the number of blockchain users is limited; the proof-of-work and proof-of-stake consensus algorithms, in fact, allow to obtain good security levels only if the computational power, or the stake of participants, are well-distributed among the different users of the network. Indeed, in the event that the majority of the computational power is concentrated in the hands of an attacker, he can strike a 51% attack and monopolize the creation of the blocks. The probability of successfully mount an attack increases if many small-sized blockchains are created to store the data of a well-defined geographical area, since the number of users of every blockchain network is limited. Based on these considerations, a permissioned blockchain has been chosen for the development of the project of the thesis because it allows to overcome the limitations of the permissionless platforms, obtaining higher throughput levels, lower latency, lower energy consumption and permits to create many small-sized blockchains. Unlike the permissionless blockchains, in the permissioned environment only a set of trusted users can join the blockchain, maintain a replica of data and participate to the distributed consensus protocol. In this case, the role of a central authority is fundamental to control the accesses to the network and to release the certificates necessary for the identification and authorization of the peers. The control on the accesses and the absence of the tamper-proof property make the permissioned blockchains more similar to the traditional distributed databases rather than a real blockchain [74]. On the other hand, the limitations imposed by the permissionless blockchains, mainly attributable to their intrinsic consensus algorithm, demonstrate that a huge research effort is still necessary be-

fore the blockchain can replace the already existing distributed databases, especially when used for memory-intensive applications.

8.2 Hyperledger Fabric and the thesis project

Hyperledger Fabric has been identified as the best permissioned blockchain platform for the development of the thesis project because it has interesting functionalities, such as the channels and the execute-order-validate transactions flow. The latter allows to obtain a throughput level considerably higher than the other blockchain platforms, but still lower than the other distributed databases. Another disadvantage of Fabric, if compared to the other distributed databases, is the full replication of data, that does not permit to improve the storage capability by increasing the number of peers.
Additionally, in Fabric, there are no restrictions on the number of blocks that can be created in the unit of time, in contrast to the Bitcoin protocol that is characterized by the creation of 1 block of at most 1MB every 10 minutes. In Hyperledger Fabric, in fact, the blocks are created as soon as new transactions are endorsed. This feature increases the flexibility of the platform but worsens the problems related to the storage memory requirements.

8.2.1 Comments on the results of the experiments

The experiments of scalability, conducted in Chapter 7, estimated the performance of Fabric, in terms of time required to store the reports and to validate the CAMs. The most significant results demonstrated that the performance decreased by increasing the number of peers of the network or the number of peers concurrently endorsing different transactions. These results are not representative of the real capabilities of Fabric, because the tests have been executed by using a single hosting machine characterized by limited computational and storage memory capabilities. The execution of the experiments demonstrated that the proposed chaincode application is affected by some limitations which impact also on the system architecture. The chaincode function used to store the reports in the blockchain, in fact, reads a file saved in the peer container to store many reports with a single transaction. This strategy does not permit to multiple peers to endorse the transactions, because only one base station (i.e. peer) knows the content of every file and can correctly execute the code of the chaincode function responsible for storing the reports. All the other peers must need to trust a single endorser for every transaction, with subsequent loss of confidence in the reliability of the blockchain application. To increase the minimum number of peers that must endorse the transactions it is necessary to modify the architecture of the blockchain application and the chaincode endorsement policy. The latter, in fact, works at the granularity of the chaincode and specifies which is the minimum number of peers that must endorse the transactions.

8.2.2 Alternative architectures

In this section, two alternative systems architectures are proposed to solve the problem above-mentioned and allow many peers to endorse the same transactions.

Different chaincodes for different operations

A possible alternative is based on the Hyperledger Fabric functionality that allows to create different chaincodes able to interact with each other and to read stored data. For this purpose, two chaincodes, characterized by different endorsement policies, must be deployed. The first must be used to store the reports and is characterized by an endorsement policy that is satisfied if only one peer executes the chaincode function. The second is used to apply the position verification algorithms and requires that every transaction is endorsed by multiple peers. The interactions between the two chaincodes increase the complexity of the overall system but, at the same time, allow to specify different endorsement policies for the operations of storage and validation of the CAMs, increasing the reliability of the blockchain application.

Alternative storage strategies

Another possible architecture is characterized by a single chaincode and by an endorsement policy requiring that every transaction is endorsed by many peers. This alternative requires to pass the reports as parameters of the transaction proposals used to store the reports in the blockchain, increasing the amount of data broadcasted in the network and the transactions execution time. This means that the architecture is more influenced by the network bandwidth and makes it necessary to find a trade-off between the efficiency and security of the network.

Use of an SDK

The above-mentioned alternative architectures require to use one of the available SDKs (Java, Go or Node SDKs) to overcome the limitations of the Command Line Interface, allowing every transaction to be endorsed by many peers.

8.3 Future research directions

This thesis is concluded outlining some relevant topics that should be deeply analysed by the future research activities, in the context of Hyperledger Fabric and permissionless blockchains.

8.3.1 Hyperledger Fabric research topics

Hyperledger Caliper

As a starting point, the performance evaluation of the Hyperledger Fabric applications could be executed with **Hyperledger Caliper**, an open-source tool aiming to standardize the benchmarks of the blockchain applications [75]. In March 2018, this tool has been accepted as official Hyperledger project.

Pruning

Another interesting research topic is the **blockchain pruning**, i.e. the possibility to remove data stored in the blockchain to limit the storage memory utilisation. The thesis of Palm is a good starting point for understanding the pruning and its

benefits applied to the Hyperledger Fabric platform [76]. Among other things, the author suggests to remove a channel when the amount of data stored in it grows too much. This data can be persisted on external storage units for auditing purposes, and new channels can be created to store the new data [76]. The problem of full data replication is this way partially solved.

Side DB

Side DB is a new feature of Hyperledger Fabric version 1.1 that tries to improve the privacy of the data stored in the channels [77]. In the original Hyperledger Fabric consensus protocol, all the transactions are broadcasted to the same orderers irrespective of the channels they belong to. The orderers create the blocks and send them only to the peers taking part to the channel [77] but, if one the orderers is malicious, it could disclose the sensitive information to an unauthorized third party. Side DB solves this problem with an innovative strategy, by transmitting only the hash of transactions to the ordering service nodes, and introducing a peer-to-peer data exchange between the authorized peers [77]. In this way, the orderers are notified of the existence of new transactions, but only the authorized peers can access the transactions payload and store a copy of data on their private state database [77]. This feature is very important for all the applications focused on data confidentiality requirements.

Multi-host deployment environment

To improve the reliability of the experiments conducted to evaluate the scalability and performance of Hyperledger Fabric, different blockchains must be deployed in a multi-host environment, by using one hosting machine for every peer and orderer. In this way, it is possible to check if the performance of the blockchain degrades by increasing the number of peers and concurrent processes concurrently endorsing the transactions. It is necessary to deploy a network composed of many hosting machines to overcome the bottlenecks identified in this thesis and introduced by the concurrent access to disk and low computational capabilities of the deployment environment. Furthermore, by distributing the blockchain nodes across a wide geographical area is possible to analyse the effect of the network latency and bandwidth on the performance of the blockchain platform.

8.3.2 Permissionless blockchains scalability

The Lightning Network project aims to solve the scalability issues of the permissionless blockchains and, in particular, of the Bitcoin protocol. It is based on the off-chain transactions and on the creation of payment channels shared among the Bitcoin users. Every channel permits to execute money transfers between users, without recording all the transactions in the blockchain.

The funds of the users are protected by a fraud-prevention mechanism and by the use of multi-signature Bitcoin addresses, a particular type of address that requires every transaction is authorized by all the involved parties before being accepted by the Bitcoin network [78]. Every off-chain transaction updates the balance of every user belonging to the channel so that, if the channel is closed, all the participants can claim the money owing to them by broadcasting the last off chain transaction

to the Bitcoin network nodes [78]. Furthermore, Lightning Network introduced a penalty system to discourage users from stealing money from the other participants of the channel [78]. Despite these features, Lightning Network is characterized by some limitations that, for the time being, do not allow it to definitely overcome the scalability problems of the Bitcoin protocol.

A huge research effort is still necessary to solve the scalability problems of the permissionless blockchain and enable the adoption of the permissionless blockchains for general-purpose and data-intensive applications.

Bibliography

[1] Florian Tschorsch and Björn Scheuermann. Bitcoin and beyond: A technical survey on decentralized digital currencies. *IEEE Communications Surveys & Tutorials*, 18(3):2084–2123, 2016.

[2] Karl Wüst and Arthur Gervais. Do you need a blockchain? *IACR Cryptology ePrint Archive*, 2017:375, 2017.

[3] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system,2009. `https://bitcoin.org/bitcoin.pdf`, 2012. [Online; accessed October 2017].

[4] Vitalik Buterin et al. Ethereum white paper. `https://github.com/ethereum/wiki/wiki/White-Paper`, 2013. [Online; accessed October 2017].

[5] Hyperledger fabric. `https://www.hyperledger.org/`, . [Online: Accessed October 2017].

[6] Greenspan Gideon. Multichain private blockchain — white paper. `https://www.multichain.com/download/MultiChain-White-Paper.pdf`, 2015. [Online; accessed October 2017].

[7] Apiletti Daniele. Beyond relational databases. `http://dbdmg.polito.it/wordpress/wp-content/uploads/2010/12/Beyond-relational-databases-v16-12-02-SGBD-2x.pdf`, 2016. [Online; accessed 5-January-2018].

[8] What are zk-SNARKs? `https://z.cash/technology/zksnarks.html`. [Online. Accessed February 2018].

[9] How zerocash works. `http://zerocash-project.org/how_zerocash_works`. [Online. Accessed February 2018].

[10] Shehar Bano, Alberto Sonnino, Mustafa Al-Bassam, Sarah Azouvi, Patrick McCorry, Sarah Meiklejohn, and George Danezis. Sok: Consensus in the age of blockchains. *arXiv preprint arXiv:1711.03936*, 2017.

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

[12] Marko Vukolić. The quest for scalable blockchain fabric: Proof-of-work vs. bft replication. In *International Workshop on Open Problems in Network Security*, pages 112–125. Springer, 2015.

[13] Trent McConaghy, Rodolphe Marques, Andreas Müller, Dimitri De Jonghe, Troy McConaghy, Greg McMullen, Ryan Henderson, Sylvain Bellemare, and Alberto Granzotto. Bigchaindb: a scalable blockchain database. *white paper, BigChainDB*, 2016.

[14] Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 4(3):382–401, 1982.

[15] Bitcoin Wiki contributors. Wallet. `https://en.bitcoin.it/wiki/Wallet`, . [Online: accessed January 2018].

[16] Andreas M Antonopoulos. *Mastering Bitcoin: Programming the Open Blockchain*. " O'Reilly Media, Inc.", 2017.

[17] Bitcoin Wiki contributors. Transaction. `https://en.bitcoin.it/wiki/Transaction`, . [Online: accessed January 2018].

[18] Wikipedia contributors. Hash function — wikipedia, the free encyclopedia. `https://en.wikipedia.org/w/index.php?title=Hash_function&oldid=819072562`, 2018. [Online; accessed 8-January-2018].

[19] Wikipedia contributors. Sha-2 — wikipedia, the free encyclopedia. `https://en.wikipedia.org/w/index.php?title=SHA-2&oldid=819282899`, 2018. [Online; accessed 8-January-2018].

[20] Bitcoin Wiki contributors. Block. `https://en.bitcoin.it/wiki/Block`, . [Online: accessed January 2018].

[21] Bitcoin Wiki contributors. Block hash algorithm. `https://en.bitcoin.it/wiki/Block_hashing_algorithm`, . [Online: accessed January 2018].

[22] Merkle tree. `https://chrispacia.files.wordpress.com/2013/09/merkle-tree.jpg`. [image][Online: accessed January 2018].

[23] Cryptocurrency statistics. `https://bitinfocharts.com/`. [Online: accessed 10 January 2018].

[24] Ethereum wiki contributors. Decentralized apps (dapps). `https://github.com/ethereum/wiki/wiki/Decentralized-apps-(dapps).md`, . [Online; accessed January 2018].

[25] Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum Project Yellow Paper*, 151:1–32, 2014.

[26] Ethereum wiki contributors. Mining. `https://github.com/ethereum/wiki/wiki/Mining`, . [Online; accessed January 2018].

[27] Ethereum wiki contributors. Ethereum introduction. `https://github.com/ethereum/wiki/wiki/Ethereum-introduction.md`, . [Online; accessed January 2018].

[28] Sunny King and Scott Nadal. Ppcoin: Peer-to-peer crypto-currency with proof-of-stake. *self-published paper, August*, 19, 2012.

[29] Vitalik Buterin and Virgil Griffith. Casper the friendly finality gadget. *arXiv preprint arXiv:1710.09437*, 2017.

[30] Arthur Gervais, Ghassan O Karame, Karl Wüst, Vasileios Glykantzis, Hubert Ritzdorf, and Srdjan Capkun. On the security and performance of proof of work blockchains. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 3–16. ACM, 2016.

[31] Marco Conoscenti, Antonio Vetrò, and Juan Carlos De Martin. Blockchain for the internet of things: A systematic literature review. 2016.

[32] Coin Sciences Ltd. Multichain - private blockchain platform. `https://www.slideshare.net/coinspark/multichain-private-multicurrency-blockchain-platform`, 2015. [Presentation of Coin Sciences Ltd][Online; Accessed October 2017].

[33] Greenspan Gideon. Introducing multichain streams. `https://www.multichain.com/blog/2016/09/introducing-multichain-streams/`, 2016. [Blog post] [Online; accessed October 2017].

[34] Intel corporation. Hyperledger sawtooth documentation - introduction. `https://sawtooth.hyperledger.org/docs/core/releases/latest/introduction.html`, 2015-2017. [online: accessed October 2017].

[35] The Linux Foundation. Hyperleder sawtooth. `https://hyperledger.org/projects/sawtooth`, 2017. [online: accessed October 2017].

[36] Intel Corporation. Product change notification. `http://qdms.intel.com/dm/i.aspx/5A160770-FC47-47A0-BF8A-062540456F0A/PCN114074-00.pdf`, 2015. [online: accessed January 2018].

[37] Elli Androulaki, Artem Barger, Vita Bortnikov, Christian Cachin, Konstantinos Christidis, Angelo De Caro, David Enyeart, Christopher Ferris, Gennady Laventman, Yacov Manevich, et al. Hyperledger fabric: A distributed operating system for permissioned blockchains. *arXiv preprint arXiv:1801.10228*, 2018.

[38] Setting up the development environment. `http://hyperledger-fabric.readthedocs.io/en/latest/dev-setup/devenv.html`, . [online: accessed February 2018].

[39] Hyperledger fabric functionalities. `http://hyperledger-fabric.readthedocs.io/en/latest/functionalities.html`, . [online: accessed February 2018].

[40] Joao Sousa, Alysson Bessani, and Marko Vukolić. A byzantine fault-tolerant ordering service for the hyperledger fabric blockchain platform. *arXiv preprint arXiv:1709.06921*, 2017.

[41] Membership service providers msp. `http://hyperledger-fabric.readthedocs.io/en/latest/msp.html`, . [online: accessed February 2018].

[42] Hyperledger fabric functionalities. http://hyperledger-fabric.readthedocs.io/en/latest/channels.html, . [online: accessed February 2018].

[43] Ledger. http://hyperledger-fabric.readthedocs.io/en/latest/ledger.html, . [online: accessed February 2018].

[44] Building your first network. http://hyperledger-fabric.readthedocs.io/en/latest/build_network.html, . [online: accessed February 2018].

[45] Lance Feagan. Hyperledger fabric peer design. [online: accessed February 2018].

[46] Hyperledger fabric model. http://hyperledger-fabric.readthedocs.io/en/latest/fabric_model.html, . [online: accessed February 2018].

[47] Transaction flow. http://hyperledger-fabric.readthedocs.io/en/latest/txflow.html, . [online: accessed February 2018].

[48] Chaincode tutorials. http://hyperledger-fabric.readthedocs.io/en/latest/chaincode.html, . [online: accessed February 2018].

[49] Endorsement policies. http://hyperledger-fabric.readthedocs.io/en/latest/endorsement-policies.html, . [online: accessed February 2018].

[50] Endorsement policies. http://hyperledger-fabric.readthedocs.io/en/latest/arch-deep-dive.html, . [online: accessed February 2018].

[51] Chaincode for operators. http://hyperledger-fabric.readthedocs.io/en/latest/chaincode4noah.html, . [online: accessed February 2018].

[52] Zibin Zheng, Shaoan Xie, Hongning Dai, Xiangping Chen, and Huaimin Wang. An overview of blockchain technology: Architecture, consensus, and future trends. In *Big Data (BigData Congress), 2017 IEEE International Congress on*, pages 557–564. IEEE, 2017.

[53] Serguei Popov. The tangle. *IOTA*, 2016. [Online; accessed September 2017].

[54] Iotasupport community. Creating a new seed / wallet. https://iotasupport.com/gui-newseed.shtml. [Online; accessed October 2017].

[55] Dominik Schiener. A primer on iota (with presentation). https://blog.iota.org/a-primer-on-iota-with-presentation-e0a6eb2cc621, 2017. [Online; accessed October 2017].

[56] Steemit community. What is "snapshotting" in iota? https://steemit.com/technology/@steemhoops99/iota-snapshot-what-is-it, 2017. [Online; accessed January 2018].

[57] Narula Neha. Cryptographic vulnerabilities in iota. https://medium.com/@neha/cryptographic-vulnerabilities-in-iota-9a6a9ddc4367, 2017. [Online; accessed January 2018].

[58] David Sønstebø. Curl disclosure, beyond the headline. https://blog.iota.org/curl-disclosure-beyond-the-headline-1814048d08ef, 2017. [Online; accessed January 2018].

[59] LEEMON BAIRD. The swirlds hashgraph consensus algorithm: Fair, fast, byzantine fault tolerance. 2016. [Online; accessed December 2017].

[60] Leemon Baird. Overview of swirlds hashgraph. Technical report, Swirlds, 2016. [Online; accessed December 2017].

[61] LEEMON BAIRD. Hashgraph consensus: Detailed examples. Technical report, Swirlds, 2016. [Online; accessed December 2017].

[62] EN ETSI. 302 637-2 v1. 3.1-intelligent transport systems (its); vehicular communications; basic set of applications; part 2: Specification of cooperative awareness basic service. *ETSI, Sept*, 2014.

[63] ETSI TS. 102 894-2 (v1.2.1):intelligent transport systems (its); users and applications requirements; part 2: Applications and facilities layer common data dictionary. http://www.etsi.org/deliver/etsi_ts/102800_102899/10289402/01.02.01_60/ts_10289402v010201p.pdf, 2014. [Online; accessed November 2017].

[64] ETSI TS. 103 097 v.1.1.1 : Intelligent transport systems (its); security; security header and certificate formats. http://www.etsi.org/deliver/etsi_ts/103000_103099/103097/01.01.01_60/ts_103097v010101p.pdf, 2013. [Online; accessed November 2017].

[65] Wikipedia contributors. Enodeb — wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=EnodeB&oldid=799692926, 2017. [Online; accessed 5-January-2018].

[66] Sartori Davide. Lte: Caratteristiche della rete di accesso. Master's thesis, Università degli Studi di Padova, 2012. [Online; accessed 5-January-2018].

[67] M. Fiore, C. Ettore Casetti, C. F. Chiasserini, and P. Papadimitratos. Discovery and verification of neighbor positions in mobile ad hoc networks. *IEEE Transactions on Mobile Computing*, 12(2):289–303, Feb 2013. ISSN 1536-1233. doi: 10.1109/TMC.2011.258.

[68] F. Malandrino, C. Borgiattino, C. Casetti, C. F. Chiasserini, M. Fiore, and R. Sadao. Verification and inference of positions in vehicular networks through anonymous beaconing. *IEEE Transactions on Mobile Computing*, 13(10):2415–2428, Oct 2014. ISSN 1536-1233. doi: 10.1109/TMC.2013.2297925.

[69] Benjamin Leiding, Parisa Memarmoshrefi, and Dieter Hogrefe. Self-managed and blockchain-based vehicular ad-hoc networks. In *Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct*, pages 137–140. ACM, 2016.

[70] Pradip Kumar Sharma, Seo Yeon Moon, and Jong Hyuk Park. Block-vn: A distributed blockchain based vehicular network architecture in smart city. *Journal of Information Processing Systems*, 13(1):84, 2017.

[71] Tracy Camp, Jeff Boleng, and Vanessa Davies. A survey of mobility models for ad hoc network research. *Wireless communications and mobile computing*, 2(5):483–502, 2002.

[72] Ghemawat Sanjay and Dean Jeff. LevelDB. `https://github.com/google/leveldb`. [Online: Accessed February 2018].

[73] Hyperledger fabric peer chaincode. `http://hyperledger-fabric.readthedocs.io/en/latest/commands/peerchaincode.html`, . [online: accessed March 2018].

[74] Arvind Narayanan. "Private blockchain" is just a confusing name for a shared database. [Online. Accessed February 2018].

[75] Hyperledger Caliper. `https://www.hyperledger.org/projects/caliper`, . [Online: accessed March 2018].

[76] Emanuel Palm. Implications and impact of blockchain transaction pruning, 2017.

[77] Enyeart David, Sethi Manish, Natarajan Senthilnathan, and Ronda Troy. Privacy enabled ledger. `https://jira.hyperledger.org/secure/attachment/12720/PrivacyEnabledLedger20171022.pptx`, 2017. [Online: accessed February 2018].

[78] De Luigi Alberto. Lightning Network. `http://www.albertodeluigi.com/index/bitcoin/lightning-network-parte-1/`. [Online: accessed February 2018].