POLITECNICO DI TORINO

Master Degree course in Communications and Computer Networks Engineering

Master Degree Thesis

Proof of Presence based on Hyperledger Fabric Blockchain

Supervisors
Prof. Paolo Giaccone
Prof. Carla Fabiana Chiasserini

Candidate
Carla Delbani

Academic Year 2020-2021

Acknowledgements

First, I would like to thank Politecnico di Torino for giving me the opportunity of achieving my goal. I wish to thank all people whose assistance was a milestone in the completion of my thesis.
I wish to express my sincere appreciation to my supervisor, Professor Paolo Gaccione who has the substance of a genius: he convincingly guided and encouraged me to be professional and do the right thing even when the road got tough. Without his persistent help, the goal of this project would not have been realized. Finally, my special regards to my family.

Sincerely,
Carla Delbani

Abstract

In different areas such as the financial field, supply chain management, and healthcare are moving to a new technology to manage their own business in a secured and decentralized way by using the blockchain. Basically, it is a particular type of distributed ledger technology which is a protocol that allows the transactions to be distributed and validated among peers in a synchronized way and to be recorded. Once these transactions are recorded, they cannot be changed or spoofed. Due to these features, blockchain is being used in the validation and proofing of presence of an object. Proof of presence means to know whether a person or an object was in a certain time at a certain position. For example, we hear a lot of people are trying to make a bad review for an hotel or a restaurant and basically; they have not been there, for the manager to be sure of these reviewers whether they were really in the hotel or not, they basically need to proof their presence. Relying on GPS for tracking online shopping is not a good idea since the GPS can be jammed, spoofed, and hacked due to several factors. So, the blockchain helps in solving these problems by validating the presence of a person or an object. In this thesis we will deal with the Hyperledger fabric blockchain in proofing the presence of vehicles in a pre-defined area.

Contents

Chapter 1

Introduction

1.1 Introduction

The thesis deals with a running of several number of vehicles in a well-defined geographical area where each car is provided with a USIM 4G (Universal Subscriber Identity Module), it gives information about its position, time, speed, and ID. This information cannot be stored in a centralized database because they cannot be changed so we will be using the blockchain for the storage. Based on the data stored it can find out whether the information provided by the car are giving the proof of presence of it or it is trying to cheat. The first part of the thesis will introduce the concept of the blockchain, types of blockchains. Then we will introduce the architecture of the blockchain used for the implementation the protocol of the proof of presence. Finally, the results of the simulation will be discussed.

1.2 Outline of Thesis report

The structure of the thesis looks as follow:

- Chapter 2: It highlights on the idea of proof of presence; how blockchain replaces the GPS; then it introduces different types of blockchains.
- Chapter 3: It describes the system architecture used in the development of the thesis.
- Chapter 4: It describes the results of the simulation.
- Chapter 5: It concludes the thesis.

Chapter 2

Permissionless and Permissioned Blockchain

2.1 Overview

Proof of presence is an algorithm used to verify the position of an object at a certain position and at a certain time. For example, the customer uses it in tracking of objects, or it can be used for finding a stolen car or phone in a certain area. This chapter will discuss how GPS works and how is it being replaced by blockchain; then it introduces types of permissionless and permissioned blockchain.

2.2 GPS

2.2.1 How does GPS work?

GPS [1] is a space-based satellite navigation system that was installed by US department defence. GPS determines three-dimensional position: longitude, latitude, and altitude. Also, it supports navigation and tracking. To determine the position on the earth, GPS needs to know the orbital position of the satellite and the range to these satellites, to accomplish this, 31 satellites are distributed where each one of them emits signals to receivers. A person needs to know his location, the GPS satellite from his mobile phone starts to emit signals, the location is determined by the time difference between the time sent by GPS satellites and the time for which the GPS receiver receives the signal. By calculating the time, it is possible to calculate the distance using the formula: d = speed of light * time.

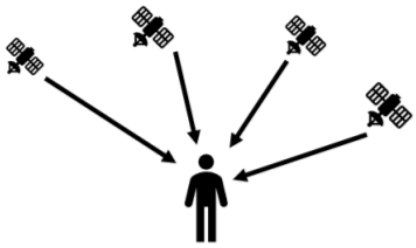

Figure 2.1. GPS Design

2.2.2 Weakness of GPS:

GPS[1] works well if the person who is requesting the location is an outdoor area with no buildings around and has a good connection service. Google Maps can have a fault of giving the position of 3-10 meters. There are different effects that cause GPS not to be certain.

1. Atmospheric Effect: the GPS signal needs to travel a long distance through the atmosphere to reach the GPS receiver; this will weaken the signal.

2. Multipath: GPS will work perfectly in an outdoor area, but it is not always the case that we have. There are buildings, trees, or any object that the GPS signal can collide with.

3. Cold starts:At turning on the GPS needs around 5 minutes to reinstall all information nescessary to work when it is being turned off for a long period.

4. Not enough satellites: To give an ideal position 7 or 8 satellites are needed.

5. Old GPS Device: Communication between GPS satellite and GPS receiver may have difficulty if the GPS receiver is old.

6. Low Battery on GPS devices: it may affect the functionality of GPS.

7. Poor service signal: Sometimes the GPS device is in an area where the signal connection is lost.

8. Jamming: sometimes the map misses certain addresses of building and streets; this will may create gaps in the coverage in some areas.

9. Spoofing: GPS can be hacked and spoofed; one the example of this is game PokemonGO where some people were are able to say that they were in this position, but they were not.

Due to these common problems, we need a new technology that we can rely on to save data in a secured way and not being changed or modified. These features are given by the blockchain technology. In the following sections, blockchain technology with its properties and types are introduced.

2.3 Blockchain Technology

2.3.1 Definition:

Blockchain was introduced in 1991 by Stuart Haber and W. Scott Stornetta[2]. Blockchain is a distributed ledger technology (DLT) where it stores records in an immutable way as series of chains. It has no central authority, and it is an open source where any one can access and read the data. It is known for its security where the records cannot be changed or tampered, if a hacker tries to modify data in any block; the previous data must be changed also so it is difficult for him to change inside the records. Figure 2.2 shows the structure of the blockchain.

Figure 2.2. Blockchain Structure

The basic idea of blockchain is to make a transaction with a free transaction fee between two parties without the control of a third party. For the transaction to be valid, it must be validated by members running in the network. Once they prove the transaction it is stored in a block with unique id and unique history. For example, PayPal is relying on the concept of blockchain where two customers can transfer money between each other with 0 transaction fee unlike the bank.

2.3.2 How does blockchain work?

Imagine that there is a workshop for construction that must be done, and a timeline table must be distributed to all the workers in the workshop. It must be updated whenever a worker finishes his own job after the validation of others that the work is done, basically, the blockchain works like this. Blockchain has important properties that let it be a unique technology. Three main properties of blockchain are:

- Decentralization:

Before the invention of the blockchain, a centralized authority must exist between two parties as a trust authority. For example, the bank is considered as centralized; let us say that person A wants to send money to person B, the bank must verify and accept the transaction with a transaction fee. Although the centralization system gives the trust property to people, but it has also vulnerabilities:

1. All data are stored in one place which makes it easier for a hacker to take information.

2. Maybe the system will go to a software update so the entire system will freeze.

3. The owner of the centralized system stops from working, so no one can access the information anymore.

4. What if the system got hacked, so all the data are changed and are lost.

Decentralized system solves the problem of referring to a third party and rely the trust on it. In blockchain the data are being distributed all among the peers of the network. Figure 2.3 shows the difference between centralized and decentralized systems.

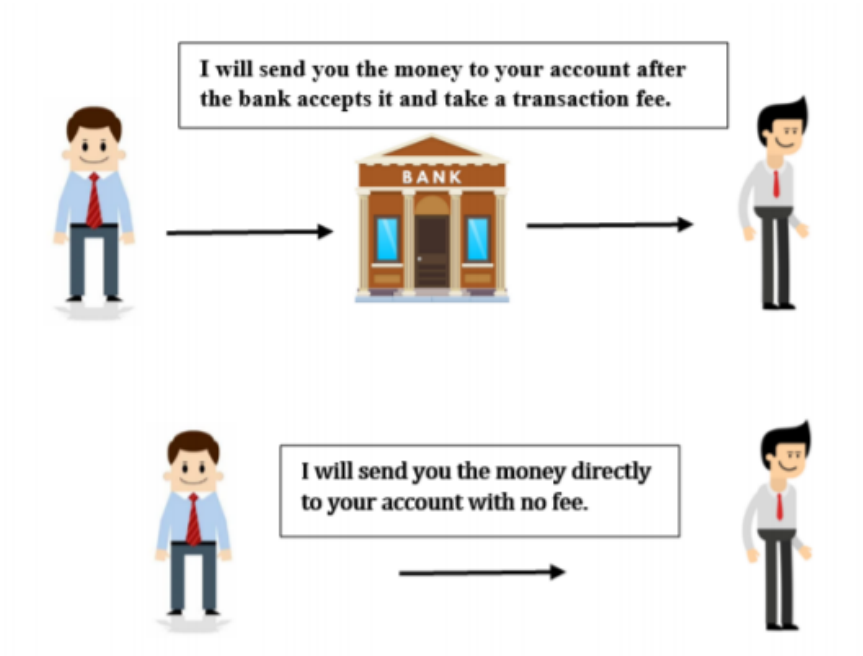

Figure 2.3. Difference between centralized and decentralized system

- Transparency: Transaparency means that the transactions done are public and can be seen. Figure 2.4 is a transaction done by Ethereum where the identity of the person is hidden via complex cryptography and represented only by their public address. The details of transaction such as the amount of money sent, gas limit, gas price etc are shown.

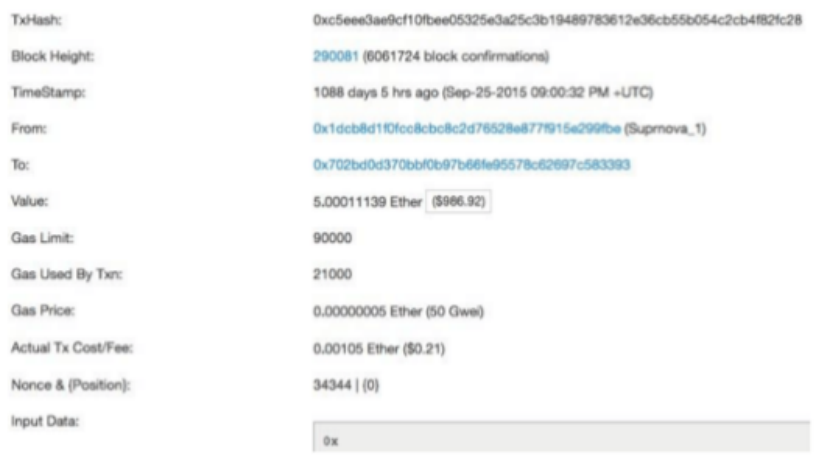

Figure 2.4. Ethereum Transaction [3]

- Immutability: Immutability is one of the most characteristics that led the blockchain to be spread and used nowadays, because once the data are recorded inside the blockchain they cannot be changed or tampered. Blockchain uses the idea of cryptographic hash function. Hashing means taking an input string of any length and giving an output of fixed length. One of the most used hashing functions is the SHA-256 which takes any input, and it gives an output of 256 bits length. In figure 2.5 the input is "hello" with a small letter in the beginning, it gives an output of 256 bits length.

Figure 2.5. Example of SHA-256 Algorithm

A little modification in the input results with a different hash. Any change in the input will. For example, in figure 2.6 the input is "Hello" where the first letter is capital, the output is different from above.

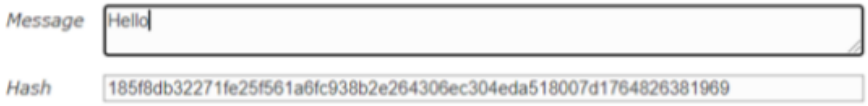

Figure 2.6. Example of SHA-256 Algorithm

As we said that blockchain consists of a series of blocks. Each block contains data, its own hash, and the hash of previous block. Data for example in Bitcoin is the information about the identity of sender and receiver of the transaction and the amount of money exchanged. The hash is calculated once a block is created. It is like a fingerprint where it makes the block unique. The hash of previous block makes sure for the nodes that nothing is being changed or hacked in the network. Let us take an example of three blocks as shown in figure 2.7, where each of the block has its own hash and the hash of the previous, where block 3 points to block number 2 and block 2 points to block number 1 except for the first block which does not point to any block because it is the smallest unit of a blockchain, and it is called the genesis block. If a hacker tries to modify the data inside block 3 this leads to the change of the hash stored in block 2 which will result also in changes in block 1 so this will change the chain completely.

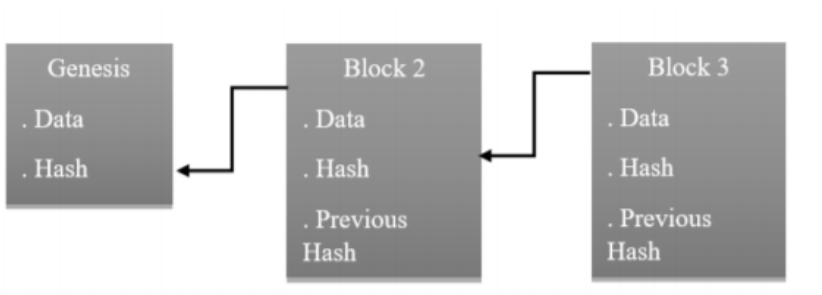

Figure 2.7. Presentation of blockchain linked by the hash function

2.3.3 Transactions of a blockchain

Transactions are piece of information that are being exchanged between users in the network, like cryptocurrencies. Each user is associated by asymmetric keys: private and public key. The public key is distributed to all the users in the network to be used as the public address whereas the private key is used to sign the issued transaction. Below is an example of how coin transaction between Alice and Bob is done. Figure 2.8 gives a summary of how transactions are done.

1. Alice generates her public address by computing the hash of her public key.
2. Bob generates his public address to share it with Alice by computing the hash of his public key.

3. Alice wants to issue a transaction to Bob, she signs the transaction with her private key.

4. Alice distributes her public key to all users in the network to verify her signature.

5. Bob and all the members in the network can verify the correctness of the signature and by using her public key they can verify that the hash of the public key of Alice corresponds to the source of transaction.

6. If the signature and the hash are both verified, then the transaction done by Alice is considered as valid.

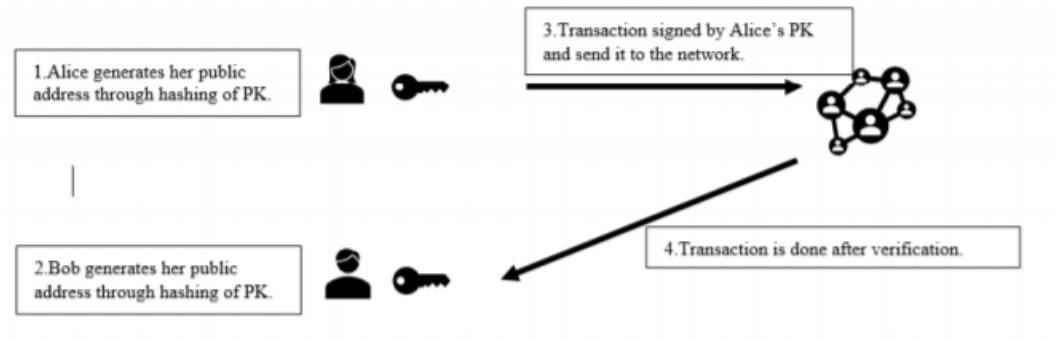

Figure 2.8. Transaction Process

2.3.4 Consensus Algorithm

Before any record is being stored inside the blockchain, the peers inside the blockchain must reach to an agreement about the state of the record this is known as consensus. Different consensus algorithms are used:

1. Proof of work: It is an algorithm introduced by Bitcoin in 2009 which is basically that the users, who are called miners, compete by solving a mathematical puzzle. When one of the miners find the solution, it should distribute it to all other miners to be validated and proved by them. If the solution is correct, then the miner is rewarded.

2. Proof of stake: It is used by Ethereum blockchain which relies on the selection of a node to do the validation process. Any user wants to participate must put a certain amount of coins as a stake.

3. Byzantine-fault tolerant: It is used whenever a crash or a malicious attack happens.

2.3.5 Smart Contract

Smart contract is a digital program that is stored inside the blockchain, it contains a set of rules on how the communication between nodes must be done. Smart contracts are immutable and distributed among all peers this means that when the smart contract is released it cannot be changed and the output of the smart contract must be validated by all the peers. Smart contract is being supported by Ethereum, Hyperledger fabric, FOAM blockchains.

2.4 Permissionless Blockchain

Permissionless blockchain is an open and public blockchain. It has the property of transparency where users can access the transactions except for the private keys. It is decentralized and it has the concept of consensus. Whenever a user wants to enter the network no identity validation is needed. Some of the permissionless blockchain introduces the concept of tokens which is digital assets are intended to incentivize users to become a part of the network. Bitcoin and Ethereum are examples of the permissionless blockchain.

2.4.1 Bitcoin

Bitcoin was created by Satoshi Nakamoto in 2009[4] and it is the first permissionless blockchain platform. It is a peer-to-peer network allowing users to exchange transactions between each other. It is a decentralized system where any user can join the network and anticipate in it. Bitcoin introduces the concept of consensus which means that an agreement must be reached to validate the issued transaction. Double spend attack is solved by bitcoin where a malicious user cannot spend twice the same coins. When the user wants to join the network a pair of asymmetric keys are issued: private and public key. Private key is used to sign the transaction and it should not be shared among the users. Public key is used to generate the public address of the user by using the hash algorithm and whenever a transaction is used the public key is used by other users to verify origin of the transaction. Address it is made from digits and strings where it is shared to anyone who wants to send the money. From the public key the address is generated through the one-way cryptographic hashing. A hash algorithm is a one-way function that produces a hash of an arbitrary-sized input. The algorithms that are used to make the bitcoin address are: Secure Hash Algorithm(SHA) followed by RACE Integrity Primitives Evaluation Message Digest(RIPEMD). [5]
"Starting with the public key K, we compute theSHA256 hash and then compute the RIPEMD160 hash of the result, producing a 160-bit (20-byte) number. "
A= RIPMD160(SHA256(K))
where K is the public key andA is the resulting bitcoin address.

2.4.1.1 Transaction Structure

Transaction means the transfer of ownership. Bitcoin introduces a concept of UTXO (Unspent transaction output. Suppose that Alice owns UTXO and she wants to make a transaction to Bob. For the transaction to be successful she must transfer the ownership using Bob's public key which can be used only with Bob's private key. This process is handed by two scripts which are: unlocking script which unlocks the ownership from the sender and at the receiver side locking script is used. The bitcoin transaction consists of version, input, output, and locktime. [4]

1. Version: it indicates the bitcoin software used.
2. Lock time: It indicates the time for locking the blockchain for a certain period before adding a transaction.
3. Input: The inputs of a transaction are the output of a previous transaction.
4. Output: The output of a transaction indicates the receipts of the transaction and the amount of money transferred. The output's structure consists of:
5. Value: Amount of UTXO newly owned by the receipt.
6. Locking size script.
7. ScriptPubKey (locking script): P2PKH,Pay-to-PubKeyHash is the most common used script, which contains the address to whom the money must be transferred.

The input's structure consists of:

1. Tx id: Each transaction has a unique id to know to whom the output of the transaction belongs to.
2. Vout (Output index): It is the index of UTXO.
3. Input: The inputs of a transaction are the output of a previous transaction.
4. Unlocking script: it is a script to unlock the UTXO received.
5. Unlocking size script.

2.4.1.2 Blockchain Header

The block header of bitcoin consists of:

1. Bitcoin version which is useful in keep tracking the updates.
2. Timestamp: it indicates when the event occurs.
3. Difficulty: The complexity in solving the mathematical puzzle in the consensus algorithm.
4. Merkle tree: it represents the history of transactions.

5. Previous block hash which connects the current block to the previous block.

6. Nonce: a random number that specifies the difficulty.

2.4.1.3 Proof of Work

Consensus algorithm is an agreement between the peers in the network about a state of distributed ledger. In 2009[6], bitcoin introduces a consensus algorithm called: Proof of Work. It is an algorithm for the peers to validate transactions and add new blocks into the blockchain. Peers who will do validation of the transactions are called miners, they will compete between each other to solve a mathematical puzzle; the difficulty of this puzzle depends on the size of the network, size of blockchain, and other factors; the solution is broadcasted by the miner who is able to find the correct solution to the other miners to check the correctness of the solution. Miners need to try and guess a pseudo number called "nonce", this number when is combined with the data and then passed through a hash function it must produce a result matching a certain condition. If the solution is correct the minor is rewarded. Each validated block contains a block hash that represents the work it has done. Proof of work helps in preventing attacks because the cost of a successful attack will be bigger than the cost of reward. Miners can use a variety of computer hardware's to mine Bitcoin:

1. CPU (Central Processing Unit): CPU has a limited processing power, so it is useless to mine a Bitcoin with it.

2. GPU (Graphics Processing Unit): GPUs are more powerful graphics processing cards.

3. FPGA (Field-Programmable Gate Array): It is more powerful than CPU and GPU to perform a specific task.

4. ASIICs (Application-Specific Integrated Circuit): An ASIC is a type of chip that is used for bitcoin mining, and it is known for its fastness and energy-sufficiency.

Disadvantages of proof of work:

1. High Energy Consumption: it uses a lot of energy which can be harmful for the environment.

2. 51% attack: When the network is large enough proof of work gives a high level of security but if the network is small, there is a big possibility for a hacker to gain majority of the network's computational power which is known as the 51

2.4.2 Ethereum

Ethereum is a public blockchain platform designed to make transactions without the need of a central authority. It is intended to execute program codes of different decentralized applications (DApps) which are called smart contracts.

2.4.2.1 Smart contract and decentralized applications

Blockchain smart contracts are software programs stored inside the blockchain, solidity programming language is used. It specifies rules and how any digital asset or transaction should be done. Ethereum is equipped for running smart contracts made by any application (DApps) that is kept on the blockchain. Ethereum has its own cryptocurrency which is "Ether". Any change in the state in the blockchain requires the usage of Ether also it is used as a reward for the user for adding a new block in the chain. Ether has a price that varies daily so when the price of Ether becomes low the users will start execution their transactions which is not a good idea. For this reason, "gas" is introduced. Gas is an internal currency in Ethereum. For the transaction to be done both Ether and gas are needed.

Transaction fees = amount of gas required * gas price

Amount of gas = limit gas

If the user provides less than the amount of gas to run a particular operation, then the process will fail, and the user will be given the message "out of gas".

2.4.2.2 Accounts

Ethereum introduces a new term which is the world state; it is a collection of account present in the network. It is considered as a database to save all the transactions done. Ethereum account has a 20-byte cryptographic address, account balance, and state transitions. There are two types of Ethereum account:

1. Externally Owned Account: It is controlled by the user's private key. It is like a bank account where the user deposits its money and make transaction through this account. The account is linked by the user's public key to the world state.

2. Contracts Account: It is like the externally owned account where it holds ether, and it is identified by its public key. CA does not have private code, but it contains a code for smart contracts.

2.4.2.3 Transactions

Transaction is an exchange of piece of information between two parties. In Ethereum there are three types of transactions:

1. An externally owned account deploys a smart contract.

2. Execution a function in the smart contract that needs the update of the ledger.

3. Transfer of Ether from one account to another. Four different scenarios occur as shown in figure 2.9:

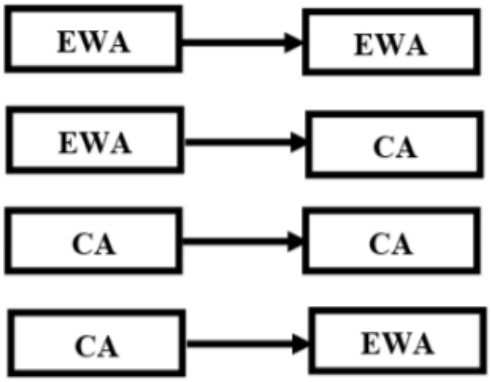

Figure 2.9. Different scenarios of transaction

2.4.2.4 Structure of transaction

Figure 2.10 shows the structure of ethereum transaction

1. From: account of the sender.
2. To: receiver account.
3. Value: amount of Ether.
4. Input: it is referred to contract bytecode where it stores data related to smart contract.
5. Block Hash of the transaction.
6. Gas: amount of gas used by the sender.
7. Block number
8. Hash of the transaction.
9. Transaction index which indicates the transaction number of executions of the current transaction.
10. Nonce: number of transactions made by sender.
11. V, R, S refers to digital signature and signing of transactions.

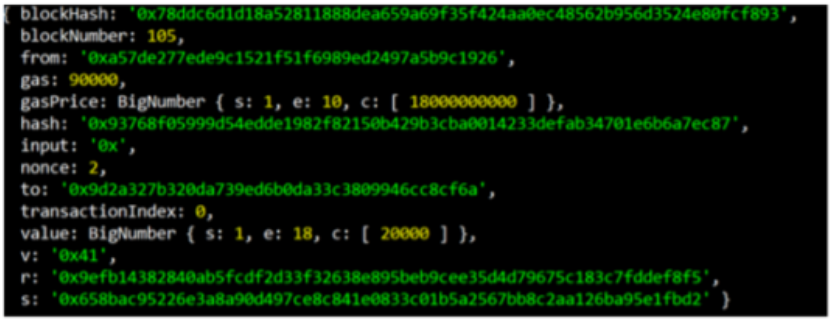

Figure 2.10. Transaction structure of Ethereum[7]

2.4.2.4 Ethereum Block Structure

A block contains multiple transactions. It contains different fields, figure 2.11 shows the structure:

1. Difficulty: determines the complexity of the mathematical puzzle.
2. Hash: hash of the block.
3. Number: block number in the chain.
4. Parent Hash: parents' block hash.
5. Miner: account identifier of a miner.
6. GasUsed: gas used in the transaction.
7. Gas limit: it determines the maximum number of transactions the block can contain.
8. Nonce: it is a random number that is used in solving the puzzle.
9. Total Difficulty.
10. Transactions
11. Receipt, state, and transaction root

Figure 2.11. Ethereum Block Structure[7]

There are two types of nodes in Ethereum:

1. Ethereum Virtual Machines (EVM): The execution of the smart contract and the transactions to be recorded are the responsibility of a system which is called the Ethereum Virtual machine (EVM). The programming language of the DApps or the smart contracts are converted to an EVM justifiable language and transactions are executed.

2. Mining Nodes: A miner is responsible for writing transactions in the Ethereum ledger. Miners try to compete by trying to solve a puzzle to add the transaction, when the first minor finds the solution a 5 Ether will be rewarded to it and the other miners will get rewarded from the gas fees of the transaction. Each mining node has its own ledger. When a new block is created each of the mining nodes must update their ledger.

2.4.2.5 Proof of Stake

Proof of stake is another consensus algorithm that is based on a random election of nodes to do the validation process; proof of the stack uses the term of forging rather than mining; any peer that would like to participate must lock a certain number of coins as a stake; the more the value of the stake the more chance of probability to be selected. Another approach of selection is based on "Randomized Block Selection" and "Coin Age Selection". Randomized block selection is based on the lowest the hash value and the highest stack is the node is being selected. Coin age selection chooses nodes based on how long the tokens are being staked. Coin age = number of days that the coin is stocked * number of staked coins. When the node finishes its forging the coin age is set to zero for a certain period this allow for other nodes to participate. Any node wants to participate in forging it will check the transaction if are valid, sign the block and then add it to the blockchain. As a reward, the node receives the transaction fees. The node who validates wrongly are punished by removing some coins.

How is transaction done?

Suppose that Alice wants to send Ether to Bob, first Alice will generate a transaction message and be placed in a pool. The miners in the network create a new block and take all the transactions from the pool. Then the miners start to compete to solve the mathematical puzzle, a time of 10 seconds is given to them to find the solution. When one of the miners find the solution; it broadcasts it to the other miners to check the correctness of it. If the solution is correct the miner who solved the problem will be rewarded. After this the verifier and other miners will append Alice's transaction into their ledger.

2.5 Permissioned Blockchain

Not everyone in the network can join the permissioned blockchain. So, the transaction done between two users are only seen by them and not by others. It requires a permission from the owner for joining the network. They are more secured, and they require less energy because the ledger is not saved in all the members of the network. Permissioned blockchain is known for its anonymity and transparency where there is a trust between the organization and each identity is protected using cryptography. The nice feature that it holds that it allows members in the network to create their own organization with their own requirements and this gives them to have a decentralization on their own network and not a full decentralization on the whole network. Moreover, the organization can decide to have validator nodes to validate or not. Corda, Hyperledger fabric, ripple, sawtooth blockchains are examples of permissioned blockchain.

2.5.1 Corda

Corda[8] is a permissioned network that is mainly designed for financial transactions. It allows the creation of organizations in the network; each organization can have its own smart contract and consensus mechanism according to their requirements. For a consensus to be reached there are two requirements needed:

- Validation of the transaction: Validation of the transaction is done by the parties issuing the transaction where the first party proposes the transaction and the another must accept it after validating it. Validation means that the transaction meets the requirements needed in the smart contract and signature is correct
- Uniqueness of the transaction: Uniqueness means that double spend of a transaction is not allowed. This means that the input of the transaction should be used only once, this control is done by a group of nodes in the network called notary clusters to check the input of the transaction.

2.5.1.1 Properties

- Assured Identity: Members use identities to represent the node. Member will get a private key along with the corda blockchain identity and any transaction done is

signed by the private key. Other members of the network will get the public key and the name of the node that makes the transaction.

- Privacy: Users within the same network can know each other and do transactions.
- Shared Logic: The system manages the characteristics of the agreement.
- Immutability: Any information on the ledger cannot be changed.
- Open System: System itself is an open source for participation and governance, this gives the property of the diverse usage of cases of platform.
- Inclusion: Members are free to know each other.
- Legal footing: deals recorded by the ledger are, by contract, accepted as admissible evidence.

2.5.1.2 Architectural Standards

- Scalable: it supports huge number of transactions.
- Secure: It provides high security feature for the transactions.
- Interoperable: multiple applications can be used on the same network.
- Longevity: On the same network there are multiple versions of corda running.

2.5.1.2 CorsDapps

Corda support the implementation of CorDapps, which are distributed applications that are built on Corda. This means that network members can perform different transactions on the platform according to their requirements.

2.5.1.3 Network Parameters

Message size, the usage of privacy options, or upgrading are network parameters that are agreed between the members of the network to have a successful communication.

2.5.1.4 Business Network in Corda

Nodes in the network can create different network on the same corda platform and infrastructure with their own standards according to their needs which include the privacy levels, asset type, network parameters. The blockchain members refer to these types of networks as business network. It is a network of companies where they can collaborate on a specific business purpose on the network, but they are also free to participate in other business networks and transact with nodes simultaneously.

2.5.2 Quorum

Quorum [9] is a permissioned blockchain that forks the public Ethereum blockchain. It has the concept of smart contract where they allow to create DApps on its platform. Smart contracts are written in Solidity or Serpent where each smart contract can have multiple instances running on the same quorum network. As we mentioned that quorum is the fork of Ethereum, so it uses the ether. In Ethereum, ether is being paid to the miners to make transaction, but in quorum ether is used to track the transaction where its value is specified in the beginning at the genesis block, and it will not be changed. Quorum uses constellation which means that when sending the transaction to a public address the data is encrypted.[7]

Consensus

Quorum supports two consensus protocols: IBFT (Istanbul Byzantine Fault tolerant) and RAFT.

- IBFT: IBFT is used in a network where there is a need of BFT, the network has validator nodes N where they are responsible for creating a block in the chain. The system can tolerate at most F crashed nodes where F= (N-1)/3. At every round of creating a new block the validators select one of them a "proposer". Proposer is responsible for creating a block and distributes to other validators for validation. To commit a block in the blockchain it must be signed by 2F+1 validators; and at each round a new proposer is selected. There are two algorithms for selection the proposer: round robin and sticky purposer. Round robin gives a certain time for each validator to be selected. But in the sticky proposer, once the proposer is selected it will stay for all rounds until it fails. From 1 to 10 seconds are given for the validator to create a new block, if it fails then a new round is started, and new validator is selected.

States:

- Awaiting Proposal: Validators are waiting for a new proposed block.
- Preparing: Validators must notify the acceptance of the received block.
- Ready: At this stage, validators accept the block, and the block is being in "locked-in" and cannot be replaced by another until an attempt of insertion has been conducted.
- Round Change: It is done when the insertion is failed or round timed out of consensus.

Transitions: There are 6 transitions that may occur:

1. From awaiting to preparing when the new block is proposed.
2. From awaiting to round change: The proposed block is in invalid or round time out.

3. From preparing to ready: Validators indicate the validity of the proposed block.
4. From ready to awaiting: Validators are ready to append the new block to the chain.
5. From ready to round change: Block insertion has failed.
6. From round change to awaiting proposal: 2F + 1 of validators agree on the next round number to be used.

IBFT prevents for forks to be happened by introducing a concept of block locking. It means that when an agreement is reached on a block it becomes in a "Locked in", where no other blocks will be considered for insertion until the attempt of insertion has ended.

- RAFT: RAFT is designed in a way that the network requires more than 50% of the nodes to be available when a new transaction is being ready for committing. In RAFT there are three kinds of nodes: leader, followers, and candidates. Leader is responsible for creation of a block and broadcasting to the followers. The block is being validated if more than 50% of the followers consider it as valid, so the leader will commit the block and sends a message to the followers to commit the block into their blockchains. The leader every 150 to 300 ms sends a message to the followers to inform them about its existence; each follower sets a time for receiving this message from the follower if the time expires the follower changes its status from follower to a candidate as a leader. The nodes become a leader when 50% of the nodes agree on it.

Chapter 3

Hyperledger Fabric Blockchain

This chapter introduces the concept of Hyperledger fabric blockchain, comparison between it and other blockchains. It introduces a related work like the thesis.

3.1 Definition of Hyperledger Fabric

Hyperledger fabric is a permissioned blockchain and an open source founded by Linux foundation. It does not depend on anonymous miners nor on a currency, each participant must be authenticated to validate transactions. It has the same concept of Ethereum in the usage of smart contracts which are called chain codes. Smart contracts can be written in different languages: java, JavaScript, or Golang. It gives the option of creating different channels where each channel can have its own consensus and rules depending on its needs.

3.2 Main Aspects of Hyperledger Fabric

3.2.1 Identity

Any user must have an identity to access the network; it is like a permission to involve in the network. In Hyperledger fabric digital identities encapsulated in a X.509 digital certificate issued by certificate authorities (CA). CA issues identities by generating a public and private key which forms a key-pair that can be used to verify to prove identity.

3.2.2 Member Service Provider

As we mentioned that CA issues identities by generating a public and private key, but the private key cannot be shared; so, member service provider (MSP) is responsible for proving and management of the identities. MSP indicates which identities can have access to the network. There are two MSP domains:

- Local MSPs: It is used for clients and nodes where it defines the permissions for them.
- Channel MSPs: It defines the permissions at the level of the channel.

3.2.3 Policies

Policies are set of rules that are agreed by the members in the network by specifying who has the permission to access an asset and what permissions they have. For example, they can decide how many organizations or members the network can have or change the format of a block. Policies give more security and privacy since not all the members are involved in the transaction's validation like in Ethereum or Bitcoin. Policies are implemented at different levels:

- System Channel Configuration: Any network must have an ordering service that has system channel configuration. The policies in the system channel configuration defines the criteria of creation the blocks and who are allowed to create them.
- Application Channel Configuration: It specifies the way of communication between the organizations.
- Access Control Lists (ACLs): It lets the users to have control over the network.
- Smart contract endorsement policies: It defines how many members are needed to execute the transaction and validate it.

3.2.4 State Database

In Hyperledger we can find the history of all transactions up to the current state by using what is so called blockchain or transaction log. Also, it is possible to access a database known as state database to know the state of the ledger at the current time which is called world state. The state can be updated, deleted, and created. They are defined as key-value. The world state holds a version number of the transaction so every time a new transaction happens a new version number is changed. When an application submits a transaction, it first commits to the world state and then it is updated in the ledger. State database has two ways of implementation:

- LevelDB: it is the default database which supports composite key queries and key range queries.
- CouchDB: it is the same as LevelDB, but it supports more the data-rich queries and indexes.

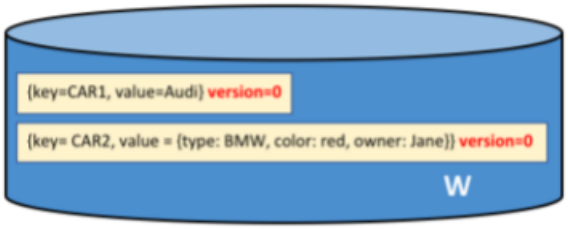

Figure 3.1. World State Structure [8]

For example, the world state contains two states. The first one with key= CAR1 and value AUDI while the other contains the key = CAR2 with type of BMW, colour is red, and the owner is Jane. Both the states hold the version 0, as shown in figure 3.1.

Transaction log is a group of blocks linked together through the hash of the block header.

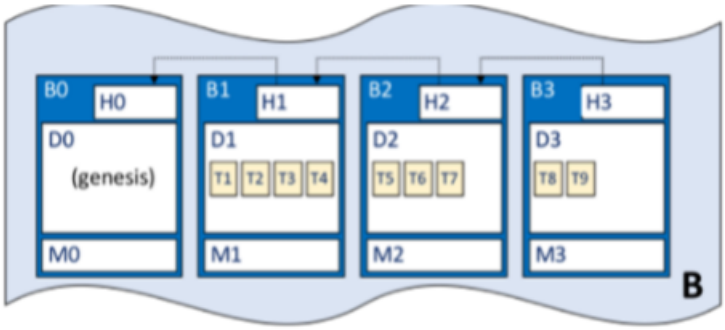

Figure 3.2. World State Structure [8]

Blockchain B contains B0, B1, B2, B3. Each block contains its transactions and the hash of the previous block, as shown in figure 3.2.

3.2.5 Structure of Blocks

The block consists of three sections, as shown in figure 3.3:

- Block Header: It contains three elements:

- Block number
- Current Block Hash
- Previous Block Header Hash

- Block Data: it contains ordered transactions.

- Block metadata: It contains the time of the creation of the block, certificate, key and signature of block user, as shown in Figure 3.4

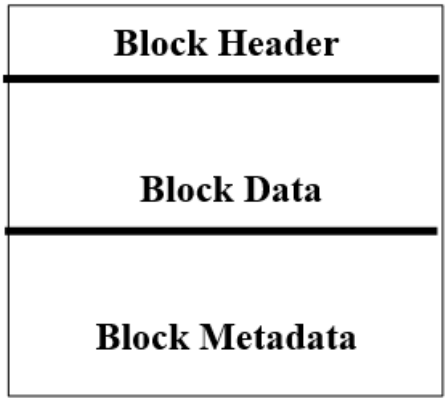

Figure 3.3. Transaction Log Structure [8]

3.2.6 Client SDK

A client software developer kit is a set of tools that allow users to create applications that invoke transactions. Hyperledger support both Node.js and Java SDK. SDK is a software that contains libraries for creation the smart contracts.

3.2.7 Transactions

A transaction is an exchange of piece of information between members in the network based on the rules implemented in the smart contract.

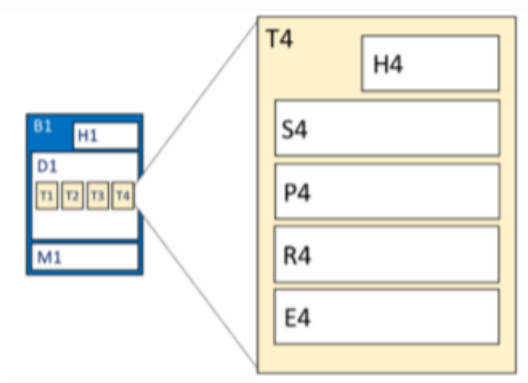

Figure 3.4. Transaction Structure [10]

Transaction T4 of block B1 contains the transaction header H4 which hold the chain code's version, transaction signature S4 which is a cryptographic signature of the client,

proposal transaction P4, and transaction endorsement E4, as shown in figure 3.5. Steps of a transaction:

1. Transaction Proposal:
    - SDK proposes a transaction and submits it to all the endorsing peers.
2. Transaction Endorsement:
    - The transaction is received by the endorsing peers.
    - Endorsing peers sign the result of the transaction and send it back to the SDK.
3. Transaction submission to the ordering service:
    - SDK collects all the transaction and send them to the ordering service.
    - Ordering service orders the transactions into blocks and then distribute them to all peers in the channel.
4. Validation:
    - Peers are now ready to validate the transactions before committing them into the ledger.
    - After validation the transactions, the block is committed into the ledger and the valid transactions are saved in the state DB.

3.2.8 Nodes

In Hyperledger fabric we can define four types of nodes:

- Committing peers: They may carry the smart contract; they are responsible for the commitment of transactions.
- Endorsing peers: They carry the smart contract; they can whether accept or deny the transaction proposal.
- Ordering nodes: They do not hold the smart contract; they communicate with committing and endorsing peers, and they ensure the commitment of blocks into the ledger.
- Anchor peers: Hyperledger can have different organizations; anchor peers are responsible for the establishment of communication between the organizations.

Validation has two kinds of roles:

- Endorsing: it means that endorsing peers check the compatibility of a transaction with the smart contract.
- Ordering: verifies transactions for committing them into the block.

3.2.9 Ordering service implementations

Different methods are used for the implementation of ordering service.

- Raft: It is a crash fault tolerant based on Raft protocol. It follows the model of leader and follower, where the leader is being elected, and its decision is distributed to the followers.
- Kafka: It is used for management purpose, and it uses the model of leader and follower.
- Solo: It is used for testing only and it contains only one ordering node.

3.2.10 Smart Contacts and Chaincode

A smart contract sets rules for the information exchange between different organizations. It is packaged into a chaincode which is then deployed into the blockchain. Smart contracts contain function as put, get, and delete states in the world state and it can query transactions as well. Every smart contract has an endorsement policy which specifies which organizations are allowed to participate in the transaction approvement. Smart contracts take the transaction proposal as an input and it executes a transaction proposal response which contains a read-write set with both the state that have been read, and the new states that are to be written if transaction is valid. Below we can see an example of a smart contract, as shown in figure 3.5

Figure 3.5. Smart Contract [11]

3.2.11 Fabric chaincode lifecycle

- Install and define a chaincode: Organizations must agree with each other on parameters that define the chaincode such as name, version, the endorsement policy.

1. Packaging the smart contract: Before the installation of chaincode on peers, it must be packaged in a tar file. The tar file contains two files: "metadata.json" that specifies the chaincode language, package label, and code path. The second file is "code.tar.gz" which contains the chaincode files.

2. Install chaincode on the peers: Peer administrator is responsible for install the chaincode on peers; if the installation was successful, it will return a command including the package identifier which is the package label combined with the hash of the package.

3. Approve the chaincode: Chaincode must be approved by each member of the channel; the agreement is done on the version, name, endorsement policy of the chaincode. Once they agreed they must send the agreement to the ordering service; then it will be distributed to all peers.

4. Commit the chaincode to the channel: After the approvement of the organizations, one of them must commit the chaincode to the channel. In the beginning the commitment request is sent to the peer of the channel and then it is submitted to the ordering service which commits the chaincode. The commit definition transaction needs to be submitted as the Organization Administrator.

- Upgrade a chaincode: When a new chaincode needs to be upgraded the same steps of chaincode lifecycle are done repackaging the chaincode, install new chaincode, approvement of the new chaincode definition, and finally commit the new chaincode to the channel.

3.3 Related Work

3.3.1 XYO Network

The XY oracle network [12] is a trust less, decentralized system that is based to provide proof of presence of object at a certain position and at a certain time. Whenever a user needs to a claim of location, he can issue a transaction which is called "queries" to get an information from the blockchain platform. In the network there are members known as aggregators where their role is to collect these queries and try to find an answer with the highest accuracy to this claim. These aggregators try to find the best answer through consensus and feed these answers back into the smart contract. XYO network has 4 primary components:

- Sentinels: known as data gatherers.
- Bridges: data relayers
- Archivists: data storers
- Diviners: answer aggregators

The network is provided by sensors, radios to collect the information around by the sentinels. Then this information will be passed to archivists for storing them through the mean of bridges. Diviners are responsible for analysing the location heuristics to generate answers and assign accuracy scores. Then the diviners relay these answers back into a

smart contract. Diviners use so called origin chain score which is determined through a set of zero knowledge proofs known as a proof of origin chain. XYO network relies on a public blockchain which is known as XYOMainChain which is responsible to store the query transactions with the data analysed by diviners with their score.

Sentinels: Sentinels are known as the location witnesses where they are responsible to collect information and then produce a temporal ledger. By producing these ledgers, the other components by the network can have a trust from where the origin of these information came from. For the sentinels to provide an honest information they are rewarded for XYO tokens whenever their information is used as answer for a query.

Bridges: Bridges are used as mean of transportation of information from sentinels to archivists. To give trust for the network the bridges add an additional proof of origin, and they are rewarded by token when this information is not hacked or altered.

Archivists: They play a role as storage of ledgers where they stored them in indexes. Archivists can ask each other of information of one of them could not find the information in their storage. They got rewarded for retrieval of data.

Diviners: Diviners are responsible to find the most accurate answer for the query. First the diviners poll the applicable blockchain platform for queries issued to XYO smart contract. Then they find the answer by interacting with the archivist to fetch the answer with highest accuracy. The diviners that fetched the answer with the best score in the shortest amount of time will have the ability to create a block on the main XYO blockchain through pow. Other diviners reach consensus on the validity of a block and digitally sign the block. The diver that was responsible for fetching the answer will send the transaction to the smart contract containing the answer with the score. It also sends a list of other diviners' signatures to prevent an attacker from issuing fake information into the blockchain by pretending to be a diviner. The smart contract can then verify the integrity of this information by checking the payload's signature list.

3.3.2 FOAM:

It is a decentralized protocol that is used for proof of location. Foam[13] is saying where I am on earth, and I also need to prove that I was at that place and other people need to be able to verify my presence. FOAM solves three problems with the existing location protocols:

- Location encoding: many applications and services need the location data, and there are several systems that provide it such as what3words, geohash but unfortunately 60% of the places in earth has no addresses. Google tries to add addresses, but it charges prohibitive licensing fees or because they are open source but lack financial incentives to gain traction.

- User Experience: it introduces a map to visualize location through an app.

- Location Verification: Lack of reliable mechanism of proof of location.

Elements of FOAM

1. Crypto-Spatial Coordinates: CSC is a new shared location standard. Each smart contract has a physical address. CSC consists of Ethereum address and geohash. The resolution of a CSC is very fine about one square meter (or about 11 square feet). Thanks to this, CSCs can define not only a building but also objects within that building. CSC is a hierarchical standard, meaning the shorter the CSC address, the larger the area it represents. Longer addresses represent a more specific location, like the hierarchical concept of area codes in phone numbers.

2. Spatial Index and Visualizer: It allows the representation of smart contract addresses on a map.

3. Proof of Location: There are two kinds of POL: static and dynamic. Static is based on points of interest by using token curated registry while the dynamic is based on beacons and time synchronization.

- Static: CSCs and TCRs together form a POI (Point of Interest) which can be a café, restaurant, or hotel. TCR has stake that let any user contribute in making a list of POI. TCR has three participants:

    1. Consumers: users that want to use the list.
    2. Candidates: want to be in the list where they submit a foam token deposit with a corresponding CSC.
    3. Cartographers: foam token holders, where they decide whether to include the candidates in the list or not by voting through tokens.

- Dynamic: It is a decentralized GPS alternative. Foam is working by adding a time synchronization protocol and hardware radio beacons. It is done by installation of 4 hardware devices "LORA" where they form an area called as zone anchor. Any zone anchor wants to contribute must pay foam tokens. Then these zone anchors start to synchronize the time between each other by exchanging beacon frames. Once synchronization is reached, they form a zone. A user is near to the zone wants to get a presence claim, he will pay a fee for the zone anchors. Zone anchors will start measuring the distance by measuring the time taken to receive the message. Then the zone authorities store the information in their own local ledger to reach consensus. Once the consensus is reached the presence claim is sent to the verifiers. The verifiers are responsible to check the data received by other zones to see if it is validated or not. Then this information is stored in Ethereum blockchain and becomes public. The user receives the result of its presence claim through a decentralized application.

Chapter 4

Proof-of-Presence blockchain-based Application

This chapter will introduce the architecture overview of the system implemented, the software setup to use Hyperledger fabric, and finally the results.

4.1 Architecture overview

Our system contains a huge number of vehicles that are moving in a pre-defined area, each car carries a message containing its position coordinates, speed, timestamp, and an identity. These cars followed a method called "Random Waypoint Mobility Model". Random waypoint means that the cars select any starting point from a pre-defined area, a random destination, and a random speed. Then it moves along this path to reach the destination; when it reaches it selects again a new destination and a new speed. Each car is supported with an USIM (Universal Subscriber Identity Module) 4G that can access the internet. The information collected while running the simulation are saved in a file that will be saved in the Hyperledger blockchain. Blockchain implements a smart contract that can read the data in the file and stored into their ledgers. After that, the smart contract's function is to discover whether the car while moving is trying to change its direction or it is moving in the correct way. Figure 4.1 shows a small presentation of the system.

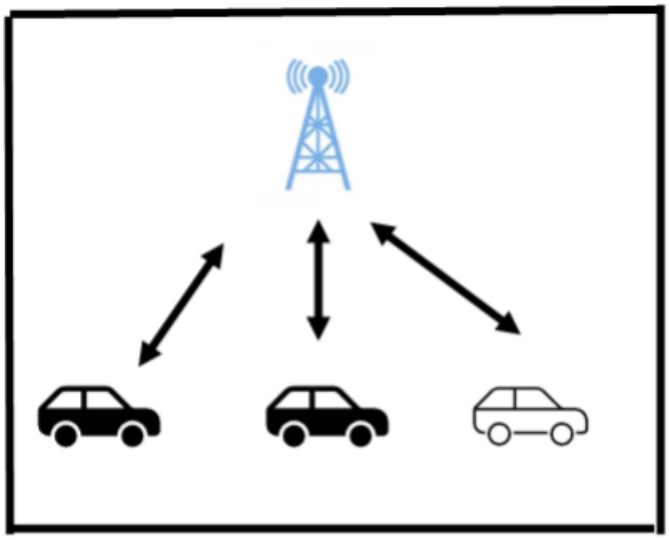

Figure 4.1. Vehicular Structure System

4.1.1 Vehicles

Vehicles are equipped by internet connection through an USIM 4G. They are moving in a pre-defined area following the random way point mobility model. Each car owns an id where it reports its position and its speed at each timestep of the simulation. This information is carried out and stored into the base station.

4.1.2 eNodeB station

The eNodeB (Evolved NodeB) is the evolution of the base station for LTE networks. eNodeB gives the internet connection to the vehicles and the authenticated cars give their information to the eNodeB. Then it is responsible to stores the information and validate it. The eNodeB plays two roles in the architecture: clients and peers of the Hyperledger Fabric Blockchain.

- Clients: The eNodeB executes transactions to store the information of each car in the blockchain.
- Endorsing peers: The eNodeB is responsible to execute the smart contract to execute transactions.
- Committing peers: The eNodeB validates the execution transaction by the endorsing peers and updates its local ledger.

4.2 Reasons of choosing Hyperledger Fabric blockchain

Hyperledger fabric blockchain is used for the implementation of the thesis which fits the best for the requirements. These are the reasons of using Hyperledger fabric:

- Privacy: It does not allow the unauthorized user to access the data recorded.
- Scalable: Hyperledger fabric can store thousands of transactions comparing to other blockchains.
- Permissioned: Any user wants to participate in the validation of the network must be authenticated through the certificate authority.
- Creation Organizations: it is a good feature that it allows to create different organizations within the same network and each organization has its own policies, members, and rules.
- Chaincodes: it is possible to create a smart contract that it can implement the algorithm of location validation.
- Different Consensus: it gives the possibility to choose between the consensus according to the requirements needed.
- Channels: it gives the possibility of creating different channels by having independent ledgers.
- Low latency: it takes a very small time to commit the block to the ledger.
- Transparency: Verification of the transaction are done by the peers of the network by being committing and endorsing peers.
- Decentralized: there is not a third authority that controls the flow of transactions.
- Hyperledger fabric can be used in different fields: healthcare, IoT, finance, supply chain management.

Also by comparing hyperledger fabric with all the blockchains that are discussed in the thesis, it is the best selection between them for several reasons:

- Ethereum and Bitcoin are both of them public and permissionless so anyone can access the network and anticipate in it.
- Corda is used only for financial sector so its usage is limited.
- XYO network depends on XYO token which its price may differ everyday. Let's say that the sentinal in the network finds the answer, so the sentinal and the other members who contribute in the process of proving the origin of transaction must be rewarded. So the reward will be distributed for the 4.
- Quorum has less performance comparing to Hyperledger fabric and it is used for finance and banking.

4.3 Hyperledger Fabric Project Setup

This section highlights on the procedure done to setup the software components to implement the simulation[14].

4.3.1 Software installation

- Operating System: Ubuntu 16.04
- Hyperledger Fabric version: 2.0

Software Prerequisites:

- Git (latest version)
- cURL (latest version)
- Docker (version 17.06.2-ce or greater is required)
- Docker compose (version 1.14 or greater is required)
- Node (version 8.9 or higher, version 9 is not supported)
- npm (version 5.x)
- GO (version 1.13.x)
- Python (version 2.7.x)
- Node.js
- Hyperledger Fabric Examples
- Hyperledger Fabric Binaries: cryptogen, configtxgen, configtxlator, peer.

4.3.2 Setup of the network

Several operations must be done to start the Hyperledger fabric network.

1. Generation of Crypto.
2. Generation of channels configurations.
3. Configuration of network participants.
4. Creation of channels.
5. Setup and execution of chaincode.

- Generation of Crypto: A tool called cryptogen is saved in a file called "crypto-config.yaml"; this tool allows to generate a unique root certificate (ca-cert) and keys for the organization. Each member belonging to this organization will be bind to it throught the certificate. These certificates and keys are stored inside a folder called "cryptoconfig".
  Below is the command to run the crytogen tool.
  cryptogen generetae config=./cryptoconfig.yaml –output=./crypto-config/ Configuration of the transaction: Configuration artifacts are generated by executing the "configtxgen" tool which creates the following:
  - Orderer which is known the genesis block.
  - Configuration of the different channels.
  - Anchor peer transactions. Below it is the command to execute the configtxgen tool. *../bin/cryptogtxgen profile .....*

- **Channel configuration creation:** To create channel configuration, we must set in the command the channel name "CHANNEL_NAME" as an environment variable.

- **Setup and execution of the chaincode:** The network contains different running containers like a container for peer, one for orderer, one for certificate authority... the configuration of these containers is defined in a file called dockercompose.yaml file. It consists of: Header, Certificate Authority container, peer container, orderer container, and CouchDB container.

Header: it indicates the version and the name of the network. Certificate Authority Container: it is the CA that releases the certificates to enable the communication between peers, clients, orderers. Peer container: For every peer it defines the name of the container, peer ID, address, ports, and the services.

- Orderer Container: For every orderer it defines the name of the container, ID, ports.

- CouchDB container: 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.

- **Participating in the channel:**

A channel must be created for the authorized peers to join it and start interacting with each other. To do this, the following steps are done:

1. Creation of channel: In this step, the orderers are requested to create the channel genesis block. Inside the command, it is necessary to specify the name of the channel and the configuration transaction file. After executing the command, a channel genesis block is created, and it is sent to the peers. ***peer channel create*** ——

2. Peers joining the channel: After the channel is created the members can join it to interact with each other. First they fetch the genesis block from the ordering service using command peer channel fetch. ***peer channel join*** ——

3. Set anchor peers: After the peers joined the channel, they must select one of the peers to be an anchor. This is done by updating the channel. ***peer channel update*** —–

Chaincode installation and instantiation: After the peers successfully joined the channel, they must interact with each other through the chaincode. From the definition of blockchain, the chaincode must be installed on every peer in the channel by using the following command: peer chaincode install —— After the installation of the chaincode, it must be instantiated on the channel using the following command: peer chaincode instantiate —– Example: Instantiation process was an initialization of key "a" with a value of "100". In the instantiation process, it is necessary to specify the endorsement policy for the chaincode. There are two options for writing the endorsement policy:

1. "AND ('Org1MSP.peer', 'Org2MSP.peer')" which means that endorsement must be done by both organizations.

2. "OR ('Org1MSP.peer', 'Org2MSP.peer')" which means that endorsement must be done by one of them.

Query of chaincode: To check whether the instantiation process was successfully done, the following command can be done by specifying the name of the channel and its key.
peer chaincode query ——
Invoke: This allows the execution of the transactions. The state DB will be updated after the validation of the transaction. For example, we want to remove 10 from the value of key "a" and give to key "b".
peer chaincode invoke —–
Query: To check whether the invocation process is done successfully, we can run the command query against the key "a" to get the value. If it is done successfully, it must show the result = 90.
peer chaincode query —–
Upgrade: It is possible to upgrade the chaincode when a modification to it is needed. First it should be installed to all peers and then be upgraded.
peer chaincode install —– peer chaincode upgrade —

4.3.3 Parameters used for the simulation

A python code is implemented to perform the random waypoint mobility model. This code take the following parameters as input from the user:

- Number of cars.

- Width of the area [m]

- Length of the area. [m]
- Simulation time [seconds]
- Speed [m/s]

Output of the code: The output of the simulator is in a text file containing the id of the car, its position X, its position Y, its speed at every 10 seconds. Figure shows that a car is running for 360000 seconds with a squared area 10*10 with a speed of 10 m/s. The output of the script gives 8 ways of the car movement, where it selects first the position X = 6.1429 [m] and position Y = 4.0522 [m] to reach a destination of (5.6618, 7.5705) with a speed of 10 m/s. Then it selects another destination which is (1.2897, 8.5186) with the same speed until it reaches the last the destination which is (7.0122, 2.1276). Figure 4.2 shows the movement of the car.

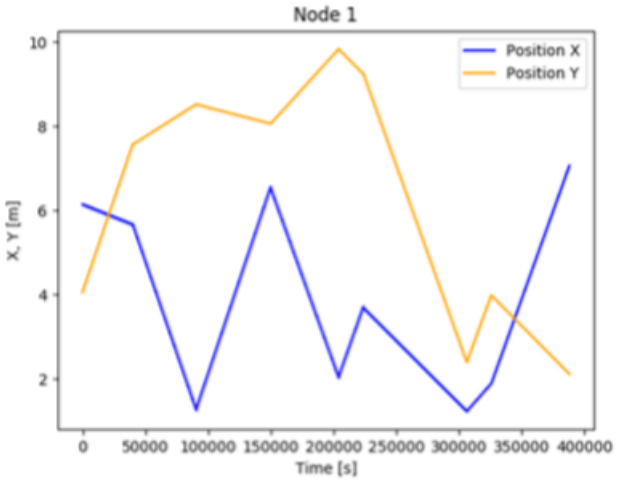

Figure 4.2. Graph showing the random waypoint mobility model

4.3.4 Smart Contract Structure

After the implementation of the random waypoint model, all the information is saved into a text file. A smart contract is implemented using Golang language to get all the data saved in the file and analyse them. The smart contract as follows. The smart contract takes the information provided by the file. It forms a unique asset called "Asset" by creation a composite key using the id of the car and a parameter called sequence which is increased by one at every creation of an asset from the same car using the following command function.
CreateCompositeKey("Asset" []stringlocation[j].carId,location[j].Data
where location is an array that contains all the data from the file The structure of the asset is in a JSON format, it is shown in figure 22.

l := Asset Data, carId, PositionX, PositionY, DestX, DestY, Time, Est-Speed, Type

During the simulation of the model, the position of the car was sampled every 10 seconds. As we can see that the time in the asset represents the time sampling. Estimated speed is the speed calculated in the smart contract using Haversine formula [15], as shown in figure 4.3. Haversine formula determines the distance between two points on a sphere given their coordinates X and Y.

$$a = \sin^2(\Delta\varphi/2) + \cos\varphi_1 \cdot \cos\varphi_2 \cdot \sin^2(\Delta\lambda/2)$$

$$c = 2 \cdot \text{atan2}(\sqrt{a}, \sqrt{(1-a)})$$

$$d = R \cdot c$$

φ *is the position X*, λ is the position Y, and R is the radius of Earth (6,371 km)

$\Delta\varphi = X2 - X1$

$\Delta\lambda = Y2 - Y1$

Figure 4.3. Haversine Formula

So, after the calculation of the distance, the estimated speed can be calculated using the following formula: speed = distance/time. After getting all the information needed for the car, it is time to see how proof of presence is implemented. Different scenarios are considered:

Scenario 1 As we said that the car is moving in a squared pre-defined area following the random waypoint mobility; our system needs to know whether this car is giving a position at a certain sampling time which is far from the exact position by 500 [m], as shown in figure 4.4.

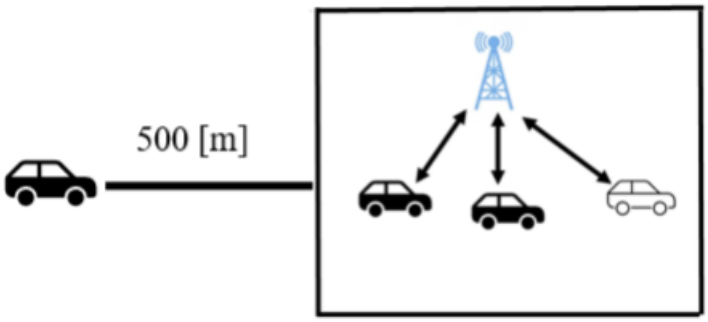

Figure 4.4. Representation of scenario 1

Scenario 2
The car is trying to move out of the area by a random radius between 0 [m] and 500 [m], as shown in figure 4.5.

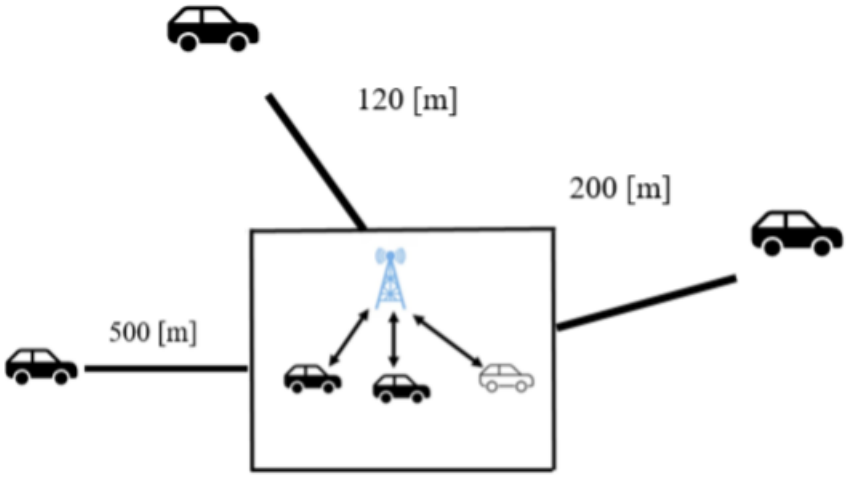

Figure 4.5. Representation of scenario 2

4.4 Results

4.4.1 Execution results of the code

This section introduces the results of the simulation of both scenarios. **Scenario 1:** We run the simulation for 100 hours with one car in a squared area 10*10. First, we install the chaincode on every peer and it was successful as shown in figure 4.6.

Figure 4.6. Chaincode Configuration

The car selects the starting point (6.3, 4.2 [m]) to reach a destination with coordinates (7.6, 2.2 [m]) with a speed of 10 m/s. When the simulation is done, we plot two graphs: Figure 4.7 is a graph showing the mobility of the car while figure 4.6 shows the speed calculated over time. At every 10 minutes, the car was getting out of the area by 500 [m]; so if we want to take an example of a car that is moving with a distance covered of 100 then it moves within 10 seconds to a distance far from the relative position by 500 [m] so the speed will be 50 [m/s] (Speed = distance/time : 600-100/10 = 50 m/s), as shown in figure 4.8.

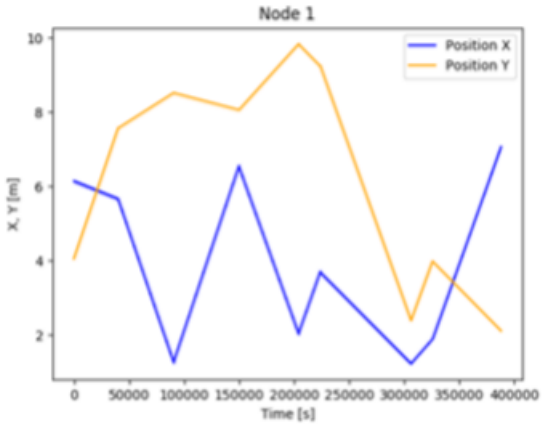

Figure 4.7. Random Waypoint Model

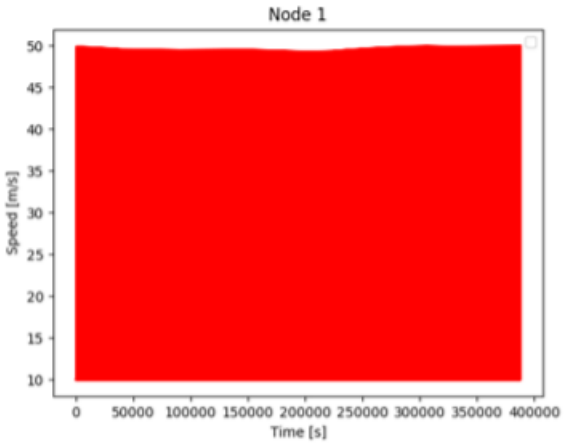

Figure 4.8. Variation of speed along time

Scenario 2 In scenario 2, instead of changing the variation of moving by 500 [m]; a random number between 0 [m] and 500 [m] was selected. Figure 4.9 shows the movement of the car for 10 hours from a starting point of (2.3, 8.6) to a reach a destination (6.2, 2.3) with a speed of 10 [m/s].

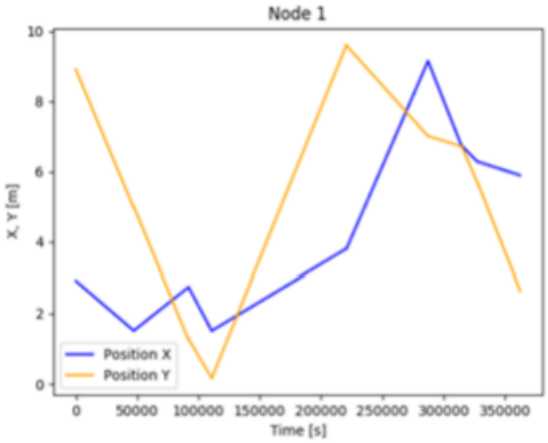

Figure 4.9. Random Waypoint Model

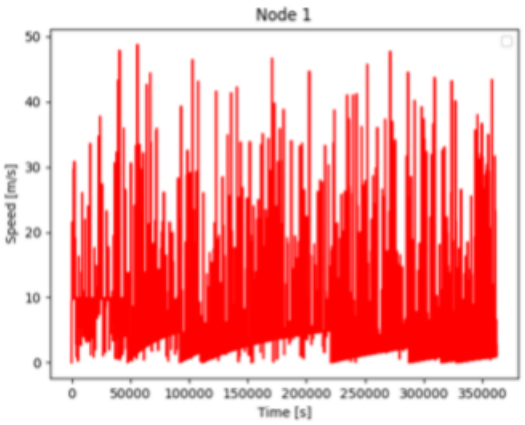

Figure 4.10. Variation of speed along time

The estimated speed will not be constant when the car is trying to cheat since the distance is randomly selected between 0 and 500m. At time 50000 seconds, the estimated speed reaches to 3.2 [m/s] whereas at time 200000 seconds the estimated speed reaches 45.7 [m/s], as shown in figure 4.10.

4.4.2 Results executed by the smart contract

In the smart contract, there is a function that reads from the text file and creates an asset as a JSON format. Then these assets are inserted into the blockchain after being validated by the peers. Figure 4.11 shows the structure of the asset in the couchDB.


```
{
  "_id": "\u0000CarId~Data\u00000\u0000110\u0000",
  "_rev": "1-3b7fc82d4cd3a5b57396731148e1300f",
  "Data": "110",
  "DestX": 2.29299168411,
  "DestY": 0.867713247217,
  "EstSpeed": 10,
  "MobileId": "0",
  "PositionX": 1.59845006405,
  "PositionY": 5.20259951028,
  "Structure": "",
  "Time": 110,
  "~version": "CgMBBgA="
}
```


Figure 4.11. Asset structure in CouchDB

As we said that the smart contract contains function that must be able to distinguish whether the car is cheating or not by calculating the estimated speed. The command peer chaincode invoke is used to query an asset from the couchDB, where it selects two assets, and it calculates the distance using haversine formula from the coordinates of each asset. Then it calculates the estimated speed by dividing the calculated distance by the difference of the sample time. After that, it checks the estimated speed whether it is greater than or less than 10 [m/s]. In the figure 4.12, it shows that the estimated speed is 9.999977532 [m/s]. According to the algorithm, the car must be considered as a good car; the smart contract was able to detect that the car is good.

Figure 4.12. Execution of function

Whereas in figure 4.13, function in the smart contract indicates that the car is a cheating car since the estimated speed is greater than 10 [m/s].

Figure 4.13. Execution of function

4.5 Classification Metric Results

As a result, to evaluate the simulation of the system of scenario, we use a classification metric which is by calculating: true positive probability, true negative probability, false negative probability, and false positive probability. The test classifies a car is good when the estimated speed is less than 10 [m/s]. Four different scenarios:

1. True Negative: It is a good car and the test reports that it is a good car.

2. True Positive: It is a cheating car and the test reports that it is a cheating car.

3. False Positive: It is a good car but the test reports that it is a cheating car.

4. False Negative: It is a cheating car but the test reports that it is a good car.

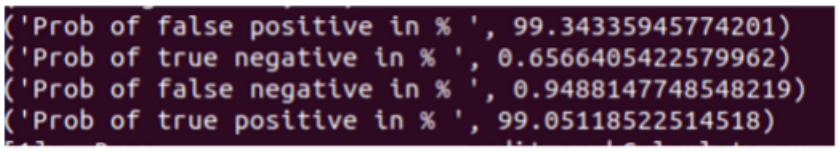

Figure 4.14. Results of the classifier

The classifier shows that 99.3% of the cars are good but the test indicates that they are cheating. 0.6566405% of the cars are good and the test says that they are good. 99.05118% of the cars are cheating and they are cheaters. For the others who are cheating but the test classifies them as a good car their probability is 0.9488147%, as shown in figure 4.14.

4.6 Smart Contract Code

```
package main

import (

        //format packaging used for printing messages on console
        "fmt"

        //shim package
        "github.com/hyperledger/fabric-chaincode-go/shim"

        //peer package
        peer "github.com/hyperledger/fabric-protos-go/peer"

        //conversion functions
        "strconv"

        "ioutil"

        //JSON Encoding
        "encoding/json"

        "math/rand"

        // KV Interface
        "github.com/hyperledger/fabric-protos-go/ledger/queryresult"

        "math"
)

// Earth radius in METERS
const radius = float64(6371000)

//AssetLocChaincode represents our chaincode object
type AssetLocChaincode struct {
}

//AssetLoc structure manages  the state
type AssetLoc struct {
        Data            string   `json:"Data"`
        CarId   string    `json:"MobileId"`
        PositionX       float64  `json:"PositionX"`
        PositionY       float64  `json:"PositionY"`
        DestX           float64  `json:"DestX"`
        DestY           float64  `json:"DestY"`
```

Figure 4.15. Code

```
	Time		int64	`json:"Time"`
	EstSpeed	float64	`json:"EstSpeed"`
	Type		string	`json:"Structure"`
}

//Invoke Method
func (asset *AssetLocChaincode) Invoke (stub shim.ChaincodeStubInterface) peer.Response {
	//var results string
	//var err error
	//Get the function name and parameters
	funcName,args := stub.GetFunctionAndParameters()

	if funcName == "Init" {
		return asset.Init(stub)

	} else if funcName =="createAsset" {
		return asset.createAsset(stub,args)

	//else if funcName == "readInsert" {
		//return asset.readInsert(stub)

	} else if funcName == "query" {
		return asset.queryAssetById(stub,args)
	}
	// This is not good
	return shim.Error(("Bad function Name !!!"))
}

//Add Sample Data
func (asset *AssetLocChaincode) Init (stub shim.ChaincodeStubInterface) peer.Response {
	indexName := "CarId~Data"
	assetLoc := [] AssetLoc {

		//Asset Data for testing
		//Data, ID, PosX, PosY, DestX, DestY, Time, Speed
		AssetLoc{"110", "0", 1.59845006405, 5.20259951028, 2.29299168411, 0.867713247217, 110, 10.0, ""},
		AssetLoc{"120", "0", 1.60169242381 , 5.20266321993, 2.29232219424, 0.867112707648, 120, 10.0, ""},
	}

	i := 0
	for i < len(assetLoc) {
		data := assetLoc[i].Data
		id := assetLoc[i].CarId
		dataIndexKey, err := stub.CreateCompositeKey(indexName, [] string{id, data})
```

Figure 4.16. Code

```
		assetBytes, _ := json.Marshal(assetLoc[i])
		if err != nil {
			return shim.Error(err.Error())
		}
		stub.PutState(dataIndexKey, assetBytes)
		i= i+1
	}

	return shim.Success(nil)
}

func (asset *AssetLocChaincode) readInsert(stub shim.ChaincodeStubInterface) peer.Response {

	content, err := ioutil.ReadFile("test1.json")
	if err != nil {
		fmt.Println(err.Error())
	}

	//Unmarshal the data into assetloc
	var assetloc []AssetLoc
	err = json.Unmarshal(content, &assetloc)
	if err != nil {
		fmt.Println(err.Error())
	}
	indexName := "MobileId~Data"

	for _, tmp := range content {
			var Data = tmp[0]
			var MobileId = tmp[1]
			var PositionX , _ = strconv.ParseFloat(tmp[2],64)
			var PositionY , _ = strconv.ParseFloat(tmp[3],64)
			var DestX , _ = strconv.ParseFloat(tmp[4],64)
			var DestY , _ = strconv.ParseFloat(tmp[5],64)
			var Time , _ = strconv.ParseInt(tmp[7],0,64)
			var EstSpeed , _ = strconv.ParseFloat(tmp[6],64)
			var Structure = tmp[8]

			l := AssetLoc{Data,MobileId,float64(PositionX),float64(PositionY),float64(DestX),float64(DestY),int64(Time),float64(EstSpeed), Structure}
			location = append(assetloc,l)

		}

	for j:= 0 ; j<len(assetloc); j++ {
		dataIndexKey, err := stub.CreateCompositeKey(indexName, []string{location[j].MobileId,location[j].Data})
```

Figure 4.17. Code

```
            dataIndexKey, err := stub.CreateCompositeKey(indexName, []string{location[j].MobileId,location[j].Data})
            if err != nil {
                    return shim.Error(err.Error())
            }

            Convert to JSON
            assetBytes, _ := json.Marshal(assetloc[j])
            stub.PutState(dataIndexKey,assetBytes)
    }

    return shim.Success(nil)
}


// Haversin function
func Haversine(PosX float64, PosY float64, DestX float64, DestY float64)(distance float64) {
    var deltaLat = (DestX - PosX) * (math.Pi/180)
    var deltaLon = (DestY - PosY) * (math.Pi/180)

    var a = math.Sin(deltaLat / 2) * math.Sin(deltaLat / 2) +
            math.Cos(PosY * (math.Pi / 180)) * math.Cos(DestY * (math.Pi / 180)) *
            math.Sin(deltaLon / 2) * math.Sin(deltaLon / 2)
    var c = 2 * math.Atan2(math.Sqrt(a),math.Sqrt(1-a))

    distance = radius * c

    return distance
}

func (asset *AssetLocChaincode) queryAssetById (stub shim.ChaincodeStubInterface, args []string) peer.Response {

    var resultJSON = "["
    if len(args) < 1 {
            return shim.Error("Incorrect number of arguments. Expecting 1")
    }

    counter := 0
    owner := args[0]
    indexName := "CarId~Data"

    resultsIterator, err := stub.GetStateByPartialCompositeKey(indexName, []string{owner})
    if err != nil {
            return shim.Error(err.Error())
    }
```

Figure 4.18. Code

```
for resultsIterator.HasNext() {
        var resultKV *queryresult.KV
        var err error

        // Get the next element
        resultKV, err = resultsIterator.Next()
        if err != nil {
                fmt.Println("Err: " + err.Error())
                return shim.Error(err.Error())
        }

        key, _, _ := stub.SplitCompositeKey(resultKV.GetKey())
        fmt.Println(key)
        data := ""
        data += (string(resultKV.Value))
        if counter > 0 {
                data ="," + data
        }
        resultJSON += data
        counter ++
}

resultsIterator.Close()
resultJSON += "]"

bytes := []byte(resultJSON)
var result []AssetLoc
json.Unmarshal(bytes, &result)

var t = rand.Int(len(result))

//First Record
var PosX1 = result[t].PositionX
var PosY1 = result[t].PositionY
var Time = result[t].Time

//Last Record
var PosX2 = result[t+1].PositionX
var PosY2 = result[t1].PositionY
var Time1 = result[t+1].Time


var finalDist = Haversine(PosX1, PosY1, PosX2, PosY2)
```

Figure 4.19. Code

```
        EstSpeed := finalDist / 10
        var structure = ""

        if EstSpeed <= 10 {
                structure = "Good Car"
        }
        if EstSpeed > 10 {
                structure = "Cheating Car"
        }

        a := AssetLoc{"1", owner, PosX1, PosY1, PosX2, PosY2, int64(Time1-Time), EstSpeed, structure}
        assetBytesValues , _ := json.Marshal(a)

        return shim.Success([]byte(assetBytesValues))


}

//Create asset
func (asset *AssetLocChaincode) createAsset (stub shim.ChaincodeStubInterface, args[]string) peer.Response {
        Data := args[0]
        CarId:= args[1]
        PositionX,_ := strconv.ParseFloat(string(args[2]),64)
        PositionY,_ := strconv.ParseFloat(string(args[3]),64)
        DestX,_ := strconv.ParseFloat(string(args[4]),64)
        DestY,_ := strconv.ParseFloat(string(args[5]),64)
        Time,_ := strconv.ParseInt(string(args[6]),0,64)
        EstSpeed,_  := strconv.ParseFloat(string(args[7]),64)
        Type := args[8]

        var a = AssetLoc {Data,CarId, PositionX, PositionY, DestX, DestY, Time, EstSpeed, Type}
        assetBytes, _ := json.Marshal(a)

        indexName := "MobileId~key"
        dataIndexKey, err := stub.CreateCompositeKey(indexName, []string{Data,CarId})
        if err != nil {
                return shim.Error(err.Error())
        }

        stub.PutState(dataIndexKey,assetBytes)

        return shim.Success(assetBytes)

}
```

Figure 4.20. Code

```
// Chaincode registers with the Shim on startup
func main() {
        fmt.Printf("Started Chaincode \n")
        err := shim.Start(new(AssetLocChaincode))
        if err != nil {
                fmt.Printf("Error starting chaincode: %s", err)
        }

}
```

Figure 4.21. Code

Chapter 5

Conclusion

Hyperledger fabric blockchain has clear benefits for being decentralized, enabling trust between nodes, secured, immutable where the data once are stored, they cannot be replaced or modified; makes a difference in the world of technology. It is being used in tracking shipments, supply chain management, financial sector, or in the health department. This thesis aimed to highlight on the concept of proof of presence. As we said, proof of presence means that to prove the existence of an object currently by at certain time without relying on GPS which can be hacked. Hyperledger fabric blockchain was used to validate the algorithm by implementing a smart contract that can specify whether the car is trying to cheat or not. Hyperledger fabric misses the concept of creation an application to see the verified locations; maybe in the future Linux company will invent an application like google Maps to spot the point of interest that we concern.

[1–14].

Bibliography

[1] Common problems with gps: How to improve your gps accuracy. https://hellotracks.com/en/blog/How-to-Improve-your-GPS-Accuracy/.

[2] History of blockchain. https://www.icaew.com/technical/technology/blockchain/blockchain-articles/what-is-blockchain/history.

[3] Hao G. Multi-authority attribute-based access control with smart contract. https://www.researchgate.net/publication/331858454_Multi-Authority_Attribute-Based_Access_Control_with_Smart_Contract, March 2019.

[4] Beyer. Blockchain before bitcoin: A history. https://blocktelegraph.io/blockchain-before-bitcoin-history/.

[5] Andreas M. Antonpoulos. *Matering Bitcoin*. O'Reilly Media, Inc., 2014.

[6] Proof of work (pow). https://www.investopedia.com/terms/p/proof-work.asp.

[7] Ritesh Modi. *Solidity Programming Essentials*. Packet, 2018.

[8] Hasib Anwar. Corda blockchain: Ruler of the financial enterprises. https://101blockchains.com/corda-blockchain/.

[9] A blockchain platform for the enterprise. https://hyperledger-fabric.readthedocs.io/en/release-2.2/ledger/ledger.html.

[10] Hyperledger fabric docs. https://hyperledger-fabric.readthedocs.io/it/latest/smartcontract/smartcontract.html,.

[11] A technical overview of the xyo network, blockchain's cryptographic location oracle network. https://medium.com/xyonetwork/whitepaper-b4793efe7609.

[12] Foam whitepaper. https://foam.space/publicAssets/FOAM_Whitepaper.pdf,.

[13] Hyperledger fabric docs. https://hyperledger-fabric.readthedocs.io/it/latest/getting_started.html.

[14] Movable type scripts. https://www.movable-type.co.uk/scripts/latlong.html.