Tenancy over Distributed Workflows using Blockchain Technology exemplified by Network Slicing

Master-Arbeit

Jordi Bisbal Ansaldo
KOM-M-634

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Fachbereich Elektrotechnik
und Informationstechnik
Fachbereich Informatik (Zweitmitglied)

Fachgebiet Multimedia Kommunikation
Prof. Dr.-Ing. Ralf Steinmetz

Tenancy over Distributed Workflows using Blockchain Technology exemplified by Network Slicing
Master-Arbeit
KOM-M-634

Eingereicht von Jordi Bisbal Ansaldo
Tag der Einreichung: 31.05.2018

1. Gutachter: Prof. Dr.-Ing. Ralf Steinmetz
2. Gutachter: Dr.-Ing. Amr Rizk
Betreuer: Prof. Paul Müller

Technische Universität Darmstadt
Fachbereich Elektrotechnik und Informationstechnik
Fachbereich Informatik (Zweitmitglied)

Fachgebiet Multimedia Kommunikation (KOM)
Prof. Dr.-Ing. Ralf Steinmetz

Erklärung zur Abschlussarbeit gemäß § 23 Abs. 7 APB der TU Darmstadt

Hiermit versichere ich, Jordi Bisbal Ansaldo, die vorliegende Master-Arbeit ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Mir ist bekannt, dass im Falle eines Plagiats (§38 Abs.2 APB) ein Täuschungsversuch vorliegt, der dazu führt, dass die Arbeit mit 5,0 bewertet und damit ein Prüfungsversuch verbraucht wird. Abschlussarbeiten dürfen nur einmal wiederholt werden.

Bei der abgegebenen Master-Arbeit stimmen die schriftliche und die zur Archivierung eingereichte elektronische Fassung überein.

Darmstadt, den 31.05.2018

_____

Jordi Bisbal Ansaldo

Contents

Abstract

Captivate readers attention.

1 Introduction

Distributed workflows enable the automation of complex business to business processes by dynamically adapting the running instances in real-time. In a supply chain environment, since different parties perform multiple activities or tasks that alter the product information during its lifecycle, this automation could enhance the system efficiency. Thus, the supply chain cycle could benefit from a system that stores, shares and synchronizes the data as the events occur: a distributed ledger.

Blockchain (BC) is a type of distributed ledger, which has been considered one of the most promising and disruptive technologies of the last years. Many market-leading companies, and experts have referred to it as the "Next Generation of the Internet" [Cla17], succeeding the World Wide Web era. After evaluating its potential benefits, different banks and enterprises, such as UBS, Microsoft or IBM, have already accomplished important investments in this innovative technology.

The revolution started in 2008, with the whitepaper publication by Satoshi Nakamoto [Nak08], who introduced a new digital payment protocol called Bitcoin. Satoshi Nakamoto was a name used by a person or group of people to first reference the performed work. Nowadays, its creator still remains unknown. One year later, a deployed software version based on the paper was launched, which enabled the use of Bitcoin to make trusted transactions between different peers on the Internet. This system was allocated in a blockchain, which is a public and distributed ledger that combined with cryptography models, removes the presence of middle-parties or intermediaries. In the case of Bitcoin, it eliminates the agents responsible for managing the money transactions, the banks.

Firstly, the blockchain architecture was restricted to only one application: online payments. However, after observing its main advantages in other use cases, an improvement of Bitcoin emerged: Ethereum [B$^+$14]. By contrast, Ethereum extends the power of decentralized transactions with a Turing-complete contract system, which uses Ethereum scripts, also called smart contracts, to perform any computation. A smart contract can be generated with non-restrictive and user-friendly programming languages, allowing developers to easily benefit from them. Therefore, it brings the user the opportunity to develop their own applications. Smart contracts are implemented in decentralized apps, which in contrast to normal web apps, its back-end is not allocated to a central server. In other words, by using them in the blockchain, their content is shared and stored in a more distributed and decentralized manner.

1.1 Motivation

Nowadays, blockchain has become a topic receiving a close review in the business world. Thousands of articles, research papers and books, such as: How the technology behind bitcoin is changing money, business and the world [TT16] or Blockchain: Blueprint for a new economy [Swa15], are catching the public eye. Nevertheless, as it is an emerging technology, and thanks to the smart contracts, a necessity to look towards new horizons exists.

Several scenarios are currently investigated from a blockchain perspective, e.g: candidate's voting or asset tracking [AM16]. In the former, voters send signed and encrypted ballots to the blockchain contract, who immediately verifies them. Simultaneously, it also preserves confidentiality, since the ballot can only be emitted from its owner. In the latter, each physical asset could be encoded in the blockchain enabling a fast and transparent tracking. For instance, Everledger [Lom15], a startup company from London, tracks diamonds by storing each diamond's digital identity on the BC, in order to prevent diamond theft.

After examining different blockchain applications in real-world scenarios, we observe that the technology provides many advantages when users are willing to share and alter real-time data in a distributed, secure and transparent manner. In this thesis, we will focus on an application example, where multiple enterprises are willing to solve the major challenges faced during a supply chain cycle.

1.2 Problem Statement and Contribution

Supply chain management is the process of linking organizations through information flows, in order to achieve a competitive strength or advantage that maximizes the customer value. Since the activities involved, go from the design or development of a product, up to its return on investment (ROI), a good coordination between those is extremely needed.

Furthermore, enterprises currently handle a large amount of information. This leads companies to suffer from considerable issues, such as scalability, data security or communication. For this reason, most of these organizations rely on third parties, which help them on the mentioned problems. However, what if this process could be efficiently accelerated in a secure and decentralized manner? Here, is where blockchain could play a crucial role.

In this research, the focus will be on IT companies facing this dilemma. The scenario includes, on one side, different customers, and in the other, organizations acting as providers. For example, eBay, one of the biggest multinational e-commerce corporations, acts as an intermediate for the product's purchase-sale. Therefore, eBay is responsible for managing the whole data. Nevertheless, can a user/company always safely trust third-parties? Why not distributing these privileges among multiple users, which cooperate for handling such complex tasks?

To prove this, in this thesis, we thoroughly investigate a service negotiation cycle, in which different service providers (SPs) are willing to embed virtual nodes across multiple InPs, in order to provide wide-area network services [DRP15]. This process is called network virtualization or network slicing, in which typically, the brokers, named VN Providers (VNPs) are responsible for performing the service negotiation that enables the VN embedding. Nevertheless, infrastructure providers are not willing to publicly disclose detailed information about their internal network topologies, along with the resources availability and costs, and hence to share this information with the VNPs.

For this reason, we further examine whether blockchain can enhance the process, by removing the presence of these third-parties (VNPs), while maintaining a coordinated process that ensures a secure storage of data. This negotiation will be based on a time-limited auction, where each virtual network request automatically creates a new smart contract on the blockchain that enables the bidding of the requested resources by the SP. In addition, a decentralized application, which guides users through the network virtualization process and directly interacts with the blockchain will be deployed.

Therefore, a good question for the thesis could be: How blockchain enhances the distribution of workflows between unknown parties while providing an efficient and secure environment? Exemplifying workflows, with the network slicing scenario.

1.3 Outline

This thesis is structured as follows. In chapter 2, relevant background information about the blockchain evolution, the network virtualization concept and the different auction mechanisms, is provided. Subsequently, in chapter 3, an overview of existing applications that integrate blockchain solutions for improving the supply chain management efficiency is given. Afterwards, a comparison of the previous related work on the multi-provider virtual network embedding is performed. Based on the observed problems and challenges, in chapter 4, a conceptual design of a multi-provider virtual network embedding approach using blockchain is introduced. In chapter 5, the implementation of the proposed brokerless system, together with the fundamental smarts contracts that ensure the system functionality, are presented. In chapter 6, the efficiency of the defined brokerless VNE system is evaluated. Lastly, in chapter 7, the conclusions, the main contributions and the possible future work of this thesis, are summarized.

2 Background

In this section, an overview of the blockchain technology evolution will be provided. It starts with the 1st blockchain generation related to cryptocurrencies, with Bitcoin as a leading representative. Then, in the 2nd generation, the so-called smart contracts, which are driven by the Ethereum platform, will be investigated. This extends the idea of money transfers, to any other application that can be writable as a piece of code.

Afterwards, the network virtualization concept will be introduced, in which resource negotiation between customers and providers is crucial. Due to this importance, a well-known public negotiation mechanism will also be presented, that is, auctions.

2.1 Blockchain: A decentralized and distributed ledger

A blockchain is a decentralized distributed ledger, which stores the entire history of transactions on the network. In other words, it is a simple database distributed among a network of computers, where each computer has an identical copy of this database. This contrasts with traditional centralized (e.g. SQL) databases that are controlled by a single entity. Thus, in a blockchain, there is no central server or agent in the middle of the communication. For example, imagine a scenario where a user wants to transfer money to another one, see Figure **??**. In a centralized system, the transaction will go first to the bank, who will update its internal database and subsequently perform the operation. In contrast, in a decentralized system, each user is able to directly transfer the money, since it possesses an updated copy of the database. Another example to replace a centralize design could be in the healthcare environment. There, patient records are stored in multiple databases, which always leads to a costly exchange of information between them. In this scenario, the blockchain could improve the process, preserving patients confidentiality in a secure and decentralized manner.

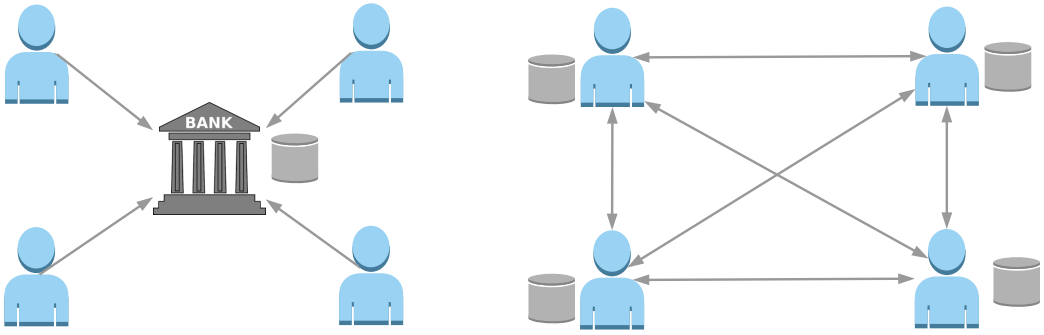


Figure 2.1: Centralized approach (traditional database) compared to a decentralized approach (blockchain).

At the beginning, the terms Bitcoin and blockchain were sometimes interchanged, as these words were used to refer: (i) the technology, (ii) the protocol for making transactions and (iii) the cryptocurrency. Therefore, before continuing, one statement needs to be clear: **Bitcoin is a cryptocurrency that uses the blockchain technology and cryptocurrencies are just one of the multiple blockchain applications.**

However, when a new technology appears, the first user's goal is normally to exploit its economic potential. For this reason, money transaction through cryptocurrencies was its first application. In the next subsection, we will understand what cryptocurrencies are, followed by an explanation of the Bitcoin's design architecture.

2.1.1 Blockchain 1.0: Cryptocurrencies

At of the end of January 2018, Coinmarketcap[1], a cryptocurrency market tracker, lists more than 1,400 cryptocurrencies with an aggregate value approaching USD 700bn. But what are cryptocurrencies? Cryptocurrencies are a variety of digital currencies pretending to work as a medium of exchange, such as Euro or USD does. As the name suggests, apart from being a virtual currency, they use cryptography to secure and verify its transactions. The main difference with traditional currencies is that they do not have any physical equivalent in the real world. Nevertheless, they can be used to pay goods and service, with the advantage of not being constrained with geographical or political borders. For example, GMO Internet[2], a Japanese company, will start paying parts of employees salaries in cryptocurrencies (Bitcoin).

In this thesis, we will set aside whether cryptocurrencies can become true currencies or not, and also the political, social or economic impact. The only focus will be on the technology behind it, as blockchain can be extended to much more than digital currencies. Thus, we will start from the genesis of the technology, with Bitcoin as its revolutionary innovator.

Bitcoin

Bitcoin took the world by surprise in 2008, after Satoshi Nakotomo's white paper publication [Nak08] and later its software release. Bitcoin (BTC) is a cryptocurrency used for making secure transactions across a peer-to-peer (P2P) network. In addition, Bitcoin uses its own protocol that operates in an overlay network, the blockchain. An overlay network is a computer network built in the top of another network, in this case above the Internet application layer, which is controlled by its users.

From another point of view, cryptocurrencies ledgers can be interpreted as state transition systems, where there is an initial state that after a transition function (money transfer), results in a new state. In Bitcoin, this initial state is a list of mined but still not spent transactions, each containing also the address supplying BTCs. This transaction output will be the receiving address along with the amount received. For this reason, Bitcoin is considered an unspent transaction output (UTXO) data model. Imagine an scenario where user A wants to send 10 BTC to user B. The first state is A and B current balance, e.g. $\{A = 10, B = 20\}$, and the transition function will take 10 BTC from A and insert it to B's account, generating a new state, now $\{A = 0, B = 30\}$. But what happens if A sends exactly the same payment to two different addresses (B and C) at the same time? This scenario is the so-called *double spending attack* and consequently, a transaction always needs to be verified by miners before being confirmed.

Mining and Transactions

Miners are specific blockchain users, responsible for monitoring and verifying all the transactions between users. And how all these miners cooperate efficiently? The answer to this question is one of the most remarkable Satoshi innovation key factors, which consists of the communication between nodes through a simple decentralized consensus protocol. This consensus protocol consists of multiple algorithms (e.g. Proof of Work), used by the miners in the Bitcoin network.

Therefore, Bitcoin needs to combine its chains with a consensus protocol, in order to synchronize the order of all transactions among the users. There, new transactions are stored in the last block of the blockchain, and a new block is mined on average every ten minutes. Over time, this creates an ever-growing chain of blocks, which are constantly updated. Thus the name: blockchain. Additionally, a complete history of the transactions is kept, so everyone can verify the last money movements. For instance, a blockchain can be compared to an endless domino game, where all the pieces are placed in vertical one after the other. Each of these pieces references the previous block, and if one block is removed (e.g. transaction error), all the subsequent ones will be affected. Hence, as miner's task is not

[1] https://coinmarketcap.com/

[2] https://www.gmo.jp

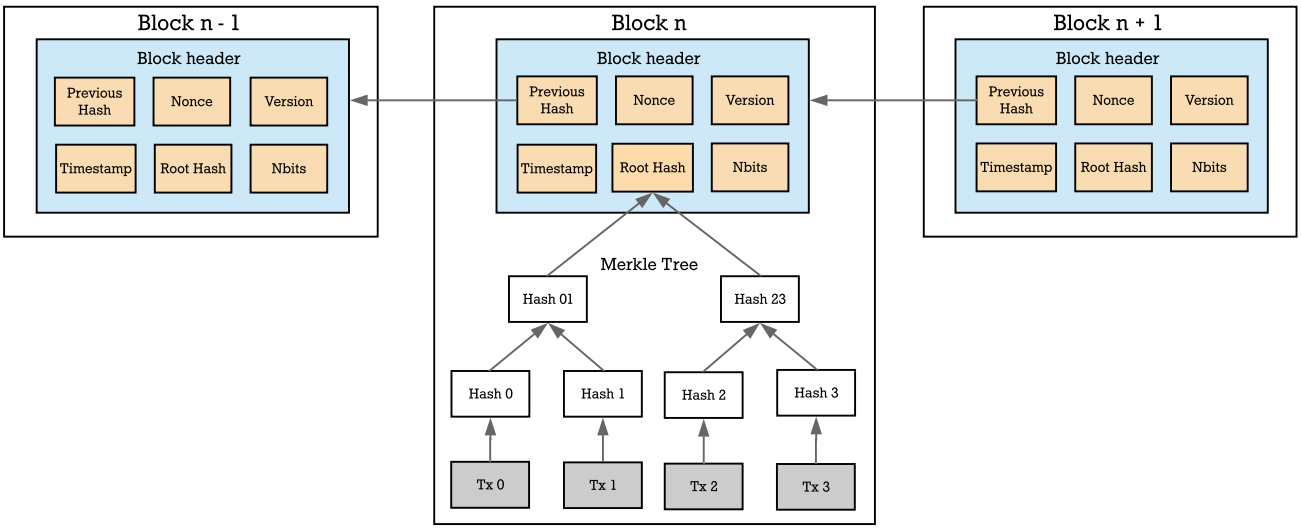


Figure 2.2: Bitcoin blockchain structure. Adapted from [HZRS17].

simple and requires computational power, if they are the first block solvers, they are also economically rewarded.

Apart from a list of transactions, each Bitcoin block contains a **block header**, whose hash is stored in the next block in order to maintain a consensus. Figure 2.2 shows how transactions are bundled into blocks, where each block header contains:

- A timestamp, to identify when the event occurs.
- A nonce, which is an arbitrary number used for the miner to create the block header.
- Previous block header hash, to keep track of already added blocks.
- A block version number that indicates the validation rules used.
- A Nbits field, whose value should be higher than the block's header hash (later explained).
- A list of all the transactions that have been created since the previous block. To save storage, this transaction list is typically stored in a Merkle tree, which is a binary tree hash [Mer87].

Furthermore, in Bitcoin, each block and transaction is restricted to a size of 1MB and 250-300bytes respectively, and for a block being valid it needs to satisfy the following requirements:

- Previous block referenced exists.
- Block's timestamp is greater than previous block one.
- Proof-of-work in this block is valid.
- If any transaction from the transaction lists returns an error, exit.
- If all previous steps confirmed, store state at the end of the block and return true.

In the third step of the block validation process, appears the term *Proof-of-work*. Bitcoin uses the *Hashcash* proof of work algorithm, see (2.1), to prove that a block miner spent some computational time creating a block. More precisely, Bitcoin protocol demands that miners find an input, which is a valid block header B_h formed by a random value (c) and a nonce (x), whose cryptographic hash (e.g $SHA256$) is less than a *target* value (non-encoded version of Nbits). This value is obtained from d, the blockchain difficulty. Then, the only way to create a valid block is simply trial and error until a valid Proof-of-Work is generated. Proof-of-work is also used in other contexts, such as for limiting email spam or denial-of-service attacks (DoS).

$$F_d(B_h) = F_d(c, x) = \text{SHA256}(\,\text{SHA256}(\,c|x\,)\,) < \frac{2^{224}}{d} \tag{2.1}$$

Key management

In Bitcoin, each user is identified by a single public address. However, how is this Bitcoin address generated? This process involves the following steps:

1. A random 256-bit private key is created. Since this key will be used to sign the transactions, it must be kept secret.
2. A 512-bit public key is generated from the private key, using the Elliptic Curve Digital Signature Algorithm (ECDSA[3]). This key is used for verifying private key signatures.
3. This public key is hashed to 160 bits using SHA-256/RIPEMD. Apart from size constraints, the reason for hashing the public key is that if there is a vulnerability in elliptic curves, user's money can still be safe, since only the hash is known.
4. Finally, the public key is encoded in ASCII using Base58Check. This output is the resulting Bitcoin address.

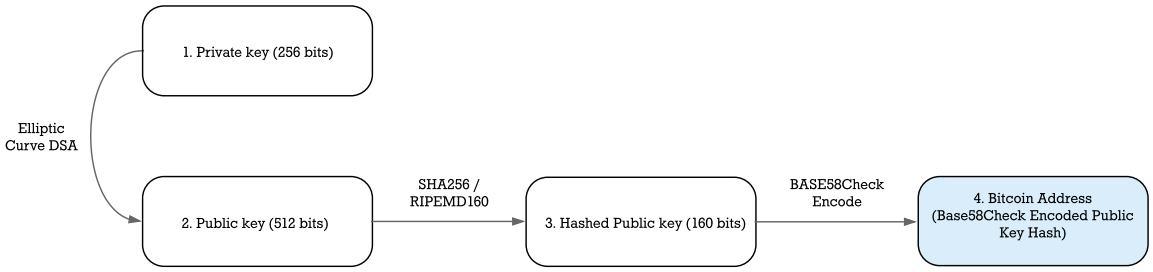


Figure 2.3: Bitcoin keys and addresses generation [Shi14].

After that, users are able to send transactions using Bitcoin. For example, imagine the last example where user A wants to send 10 BTC to user B. Firstly, the user creates a transaction willing to transfer 10 BTC to user address B. Secondly, he signs the transaction with his private key, attaching also his public key, and sends it to the blockchain. There, miners will verify the signature using A's public key and also that its hash matches user A address. If both, the signature and the hash, are correct, the transaction is accepted and added to the next block. Transaction examples can be found in Block Explorer[4].

2.1.2 Blockchain 2.0: Smart contracts

The Blockchain 1.0 had numerous limitations since essentially it only approaches the decentralization of money and payments. However, the architecture implemented by Bitcoin is extensible beyond financial uses cases.

Ethereum

In 2013 Vitalik Buterin, a Russian-Canadian programmer released the Ethereum white paper [B+14], where he describes an alternative platform running in a blockchain that allows building any kind of decentralized applications. In 2014, Gavin Wood published the Ethereum yellow paper introducing the Ethereum Virtual Machine (EVM), i.e. a blockchain with a built-in programming language. Ethereum first started as a crowdfunding project, which collects small amounts of money from a large number of users. Currently, Ethereum is the second most valuable cryptocurrency.

[3] https://www.maximintegrated.com/en/app-notes/index.mvp/id/5767
[4] https://blockexplorer.com/

Furthermore, it extends the power of decentralized transactions with a Turing-complete contract system. A Turing-complete system can be proven mathematically to be capable of performing any computation, which allows developers to create its own programs. The code of these functions are written into the so-called **smart contracts** and executed by the Ethereum nodes, each using its Ethereum Virtual Machine (EVM).

In essence, smart contracts are basically Ethereum scripts that whenever they are called, the function programmed inside of it is executed. A simple example of a contract is the logic inside a vending machine. For instance, a buyer wants a Coke bottle that costs 2 €. Once this buyer inserts 2 € and presses the button, a small program (contract) runs inside the vending machine and supplies the user with the Coke. In the smart contract context this function can be seen as:

```
1 contract Products {
2
3   function getItem(bytes32 buttonPressed, uint amount) returns (bytes32 item) {
4     if (buttonPressed == 'coke' && amount == 2) return coke;
5     return null;
6   }
7 }
```

Listing 2.1: Example of an Ethereum contract simulating a vending machine. The defined function returns a bottle of Coke if 2 € are inserted.

Therefore, smart contracts allow the writing of customized functions in a few lines of code. Despite the fact that Bitcoin and Ethereum have similar features, such as decentralization or being transaction-based, Ethereum makes use of smart contracts, which potentially opens up real-world use cases. Bitcoin and Ethereum can be comparable to a calculator (one application) and a smartphone (multiple applications) respectively. In addition to the already mentioned differences, there are other inequalities that need to be highlighted:

- Ethereum's block time is shorter, specifically it is around 14s. Remember that Bitcoin's mining time was around 10 minutes. Hence, as a consequence, Ethereum needs to handle large amounts of transactions per seconds, which means that the network can easily get separated in different subchains. To solve these scalability issues, *Casper*, a protocol from the new *Proof-of-Stake* (PoS) mechanism has been recently proposed [Ros17].
- Thus, Ethereum pretends to move from the *Proof-of-Work* to the *Proof-of-Stake* consensus algorithm. In contrast to PoW, where the miners that first solve a mathematical problem are rewarded, in PoS, validators propose and vote the next block. The voting power is directly proportional to the validator's stake (deposit). To become validators, users need to lock up their ether into a deposit. Therefore, in PoS there are validators rather than miners.
- Bitcoin's blocks and transaction sizes are indicated in bytes, however, in Ethereum, this depends on the contract complexity. This complexity is expressed in terms of *gas*, which is the amount of *ETH*[5], used for executing contracts in the Ethereum blockchain.
- Ethereum transactions compared to the mentioned Bitcoin fields consists of (i) the transaction value in Ether, (ii) recipient address, (iii) data arguments and (iv) execution cost.
- Ethereum is account-based and not transaction-based.

Also, a miner does not receive a block reward, but instead he earns all gas used for executing the smart contract. In the last point, we observe that Ethereum introduces a new concept called accounts. These accounts are unique 20-byte addresses in the blockchain, each of them having a balance controlled by ethers (ETHs). In Ethereum, there are two types of accounts (Figure 2.4):

[5] *ETH* or *Ether* is the digital currency used by Ethereum

- **Externally Owned Accounts (EOAs):** These accounts identify users while being controlled by their own public and private keys. Only EOA can send transactions to other accounts. A transaction in Ethereum can either send Ethers or call/trigger a contract account
- **Contract accounts:** These are the accounts, where the code (smart contract) is stored. Thus, once triggered (function call from another contract), its defined code is executed.

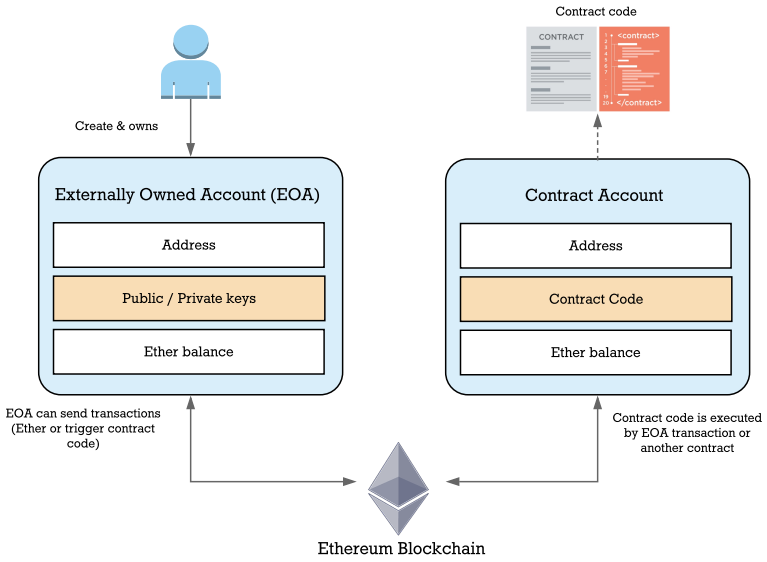


Figure 2.4: Ethereum accounts: Externally owned accounts and contract accounts.

However, what happens if this contract code results in an infinite loop?Ethereum solves this using gas. For example, suppose an EOA calling a contract account for money transfer. Apart from the ether transferred, it will also spend an additional amount of ether (gas). Thus, the ether used for a transaction, or transaction fee F is given by:

$$F = P \times L + V - R \tag{2.2}$$

Firstly, the gas limit L is the maximum amount of gas to spend on a transaction, which is by default 21000. Secondly, on Ethereum the gas price P is measured in units of Gwei, which is equivalent to $1e^{-9}$ ETH. Thirdly, the value V is the amount of ether transferred to the other EOA. Fortunately, all unused gas R during a transaction is refunded to the sender. Nevertheless, if a transaction returns an error, e.g: gas provided not enough to execute a contract, this provided ether will never be refunded.

In conclusion, a smart contact is just an account containing code, which lives on the blockchain and allows developers to create their own decentralized applications.

Decentralized applications

In the Ethereum white paper [B+14] decentralized applications DAPPs are split into three types. The first group includes financial applications, where users only want to manage money transactions, e.g. a customer paying the provider through a blockchain application. The second group is formed by semi-financial applications, in which money is involved but it is mixed with other requirements. For instance, an application where users receive incentives when solutions to computational problems are provided. And finally, in the third category, there are non-financial applications. For instance, an online voting platform providing a better transparency into elections, without compromising voter confidentiality.

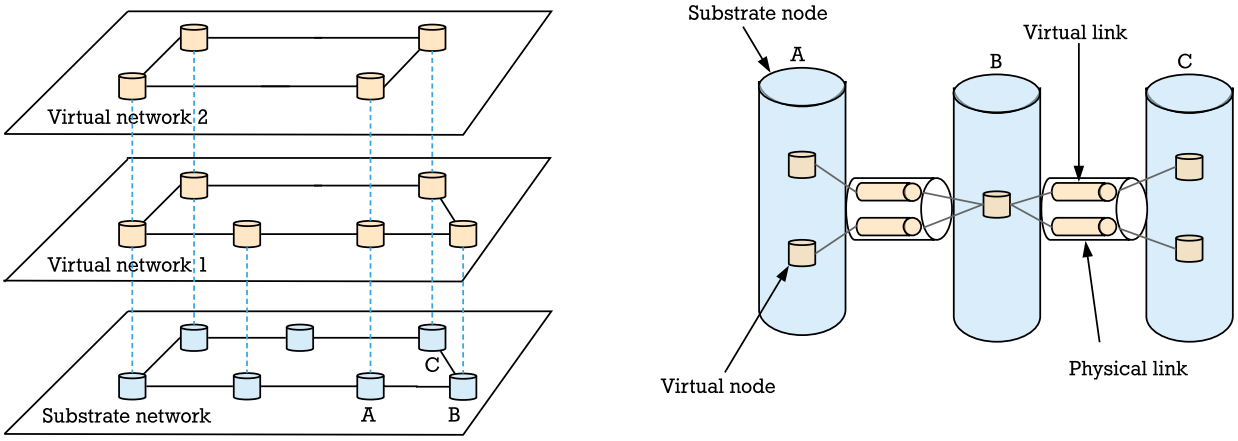
Despite the blockchain is a secure technology, the created applications are constantly compromised to multiple attacks. This is because the majority of smart contracts are insecure, due to its code being prone to bugs. For example, in June 2016, the DAO[6], a distributed autonomous organization instantiated on the Ethereum blockchain, was hacked. The attack, which was a smart contract bug, had resulted in over 50 million dollars of cryptocurrency theft. Thus, smart contracts need to be protected against malicious attacks before being uploaded on the blockchain.

In the following section, an example application or process currently performed by IT companies will be introduced.

2.2 Network Virtualization

In the last few years, network virtualization has started to grow in popularity since it enables simulating baremetal networks. These virtual networks, in contrast to physical networks, provide flexibility, manageability and a low cost.

Network virtualization handles two main concepts: node virtualization and link virtualization. The former implies sharing the node's physical resources located in the substrate network (e.g: CPU, storage, memory) to multiple virtual nodes in the virtual networks, such as physical node A in Figure 2.5a. The latter enables the transport of multiple virtual links in a single and shared physical link (Figure 2.5b).



(a) Network virtualization model.

(b) Network virtualization elements.

Figure 2.5: Network virtualization model and its basic elements. Adapted from [CJ09].

In a real scenario, there are two main actors (and one optional) participating in the network virtualization process.

- **Infrastructure providers (InPs):** Infrastructure providers (e.g: Deutsche Telekom or Telefonica) deploy and manage the physical nodes, sharing efficiently its resources to the virtual network. For InPs, this results in a significant decrease in operational and technology investment costs [DRP15].
- **Service Providers (SPs):** Service providers lease virtual resources from infrastructure providers to create its customized virtual network. This is done without the need to purchase physical network equipment.
- **Virtual network providers (VNPs):** Are brokers, which can optionally serve as intermediaries between the infrastructure providers and service providers.

[6] `https://ethereum.org/dao`

The network slicing process with the presence of a broker typically starts with the VNP obtaining the necessary network information from the SP. To optimize the software costs, the SP's virtual network request will be divided into groups of nodes (subgraphs). Afterwards, the modified network is transmitted to the multiple InPs, who are in charge of embedding the virtual nodes from the request in their physical resources.

The presented process is coordinated by an actor, the VNP, who stores all the network information to successfully allocate the incoming resources. However, there are other approaches, where the InPs do not want to rely on a central entity, since its confidential information is being disclosed and hence, known by the VNP. Conversely, the SPs and InPs make use of consensus mechanisms to perform the virtual resources trading. Thus, the network virtualization task is accomplished in a more decentralized manner, since no middle-party exists. In the following, we will briefly investigate an open negotiation technique: the auctions.

2.2.1 Auction Mechanisms

Service negotiation has already been examined in many researchs, such as [HS05], [OV02], which have worked with diverse auctions platforms. An auction is a public negotiation method involving buyers and sellers that enables resource trading between peer-to-peer users. While the buyer's goal is to find the desired service at the lowest price, the providers or seller's goal is to sell the service at the highest "possible" price. Thus, an auction should provide fairness in terms of technical and economic efficiency, for both the buyer and the seller.

Firstly, auctions can be divided into (i) *one-sided auctions* where only the buyers submit their bids (e.g: a painting auctioned in an art exhibition), or (ii) *two-sided auctions* in which buyers and sellers submit their bids. Normally, when any of the two sides (buyer or seller) cannot perform a good estimation of the service, a one-side auction is preferred. This is the case in network virtualization, where the SPs are not aware of the InPs physical nodes availability or complexity. Secondly, the auctions can be (i) *open-cry* auctions, where the bids are broadcasted to all users, or (ii) *sealed-bid* auctions in which bidders first commit bid values in secret that are later revealed (e.g: physical envelopes). In our scenario, the InPs should bid in a confidential manner, because they do not want to disclose internal network cost models. In addition, concerning auction types, our focus will be on the most common [CST80]:

- **English auctions**, also called *open-cry ascending-price* auction, is the classic bidding mechanism where the seller first sets a minimum starting value (reserve price), which buyers should overcome. Then, these buyers start to offer higher prices until nobody wants to offer a greater one. However, due to the competition, this auction model can result in a long process (e.g. low reserve price compared to real value) where users pay huge amounts.
- **Dutch auctions** are similar to the English auctions, but on the contrary, they are *open-cry descending-price* auctions. In this model, seller fixes a maximum value which is reduced slightly over time. Once a buyer accepts a price, the auction will finish. Nevertheless, to prevent ending with low prices for the sellers, they also establish the mentioned reserve price, which will be the minimum accepted amount. Dutch auctions are faster than the English ones, but buyers will also end up paying more to ensure that they win.
- **Vickrey auctions** has the particularity of corresponding to a *sealed-bid second-price* auction [Vic61]. Since it is sealed, during the bidding time buyers do not know other bids and eventually, how the auction is evolving. In addition, the Vickrey auction corresponds to a second-price tender. This means that like standard auctions, the bidders will offer a price for the service, and the highest bid will win. Nevertheless, this service will be rendered at the second highest value. For example, imagine a bidding for product A, where the winner has bidden $\{A = 15\}$ and the second highest bid is $\{A = 10\}$. Then, this winner ends-up paying only $\{A = 10\}$. In addition, it is shown that a Vickrey auction is strategy-proof, meaning that each bidder quotes the true cost of the service, and

if they do so, the resulting total values are maximised [Vic61]. From the bidder's point of view this means that he can safely bid the real value, knowing that if he wins, he will pay less than his bid. Thus, a Vickrey auction compared to English and Dutch auctions, is considered a fair-price system since it provides a reasonable price to the buyer by motivating bidders to bid truthfully.

- **Reverse auctions** as the name states, buyers and sellers roles are switched. In contrast to ordinary auctions, where buyers compete to obtain an asset, in reverse auctions, sellers compete to gain a business. In other words, the seller bids for his own provided services, which finally, will be given to the supplier offering the lowest price. For example, in the network virtualization process, InPs will bid for the lease of their physical nodes, and afterwards, the SP will rent the most economical ones.

Ultimately, after reviewing the preceding analysis regarding auction types, a ***one-sided reverse Vickrey*** auction seems to be the most suitable for brokerless virtual network embedding since it provides confidentiality and the biggest aggregate profit among sellers and buyers. This will be later investigated in chapter 4.

2.3 Summary

The blockchain is a transparent, secure and robust system where users are in charge of their own accounts and transactions. Beyond money transfer, which started with Bitcoin, the blockchain technology can be used as a software connector in multiple scenarios. Due to smart contracts, arbitrary applications could be implemented in the context of Ethereum blockchains, enabling developers using the blockchain concept in their own environment. In the next chapters, we will investigate a multi-provider virtual network embedding scenario where infrastructure providers are not willing to disclose internal network cost models.

3 Related Work

In the following, we explore some of the noted challenges that organizations encounter during their supply chain management (SCM) process, and how they currently approach these issues. Then, we will introduce a recent application in manufacturing, to demonstrate how blockchain could improve SCM efficiency. Henceforth, we focus on a particular supply chain management example: virtual network embedding across multiple InPs. A comprehensive overview of related work on already implemented centralized and decentralized frameworks will be presented. Finally, we will first enumerate the main blockchain benefits observed from the SCM manufacturing application point of view, and secondly, we will analyze the pros and cons of the existing VNE related work, in order to observe the problems and limitations that need to be addressed in our thesis.

3.1 Blockchain in Supply Chain Management

In the last years, managing the information during the lifecycle of a product represented a major challenge for many companies [KARF03], [Tut02]. This product information is constantly changing and in addition, it has to be accessible by different entities. As a consequence, this problem results in high costs for companies.

Typically, companies approach the problem in the following way: (i) storing and maintaining its data into their company-specific infrastructures, which is then communicated to the other supply chain partners, or (ii) sharing this data in a centralized database. In the former, each enterprise creates its own copy of the product data, using their own protocols and procedures. As a consequence, a lot of information asymmetries are found, which ends up destroying the benefits of knowledge or data sharing. In [LB92] and [Fia05] inadequate definition of customer services, poor coordination or organizational barriers are named to be some of the pitfalls that these users suffer from. On the other hand, the latter is a great solution as long as the companies trust the party who is maintaining the database. However, what if they are not willing to place this intermediary on their operations? Here, we foresee that blockchain could improve the process supplying data distribution and storage among these companies.

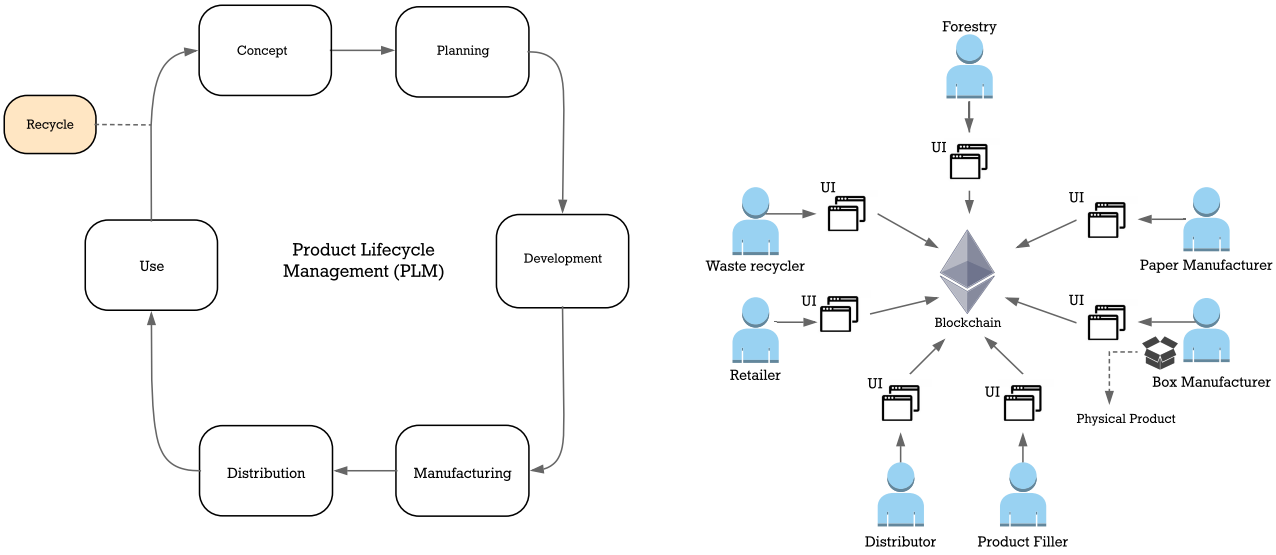
Apart from the supply chain, [Hei02] states another relationship between suppliers and customers, the Demand Chain Management (DCM), where we predict that blockchain could also play a significant role. In a DCM, the aim is to provide a customer service at the least cost. Hence, the suppliers need a real-time visibility of customer situations and needs. The main difference is that in SCM, the stakeholders wait to receive the order for proceeding with a product (push), whereas in DCM the suppliers observe and immediately operate in user's petition (pull) [WG17]. Currently, companies such as Everledger or Skuchain[1], offer blockchain services to manage and improve the supply chain performance.

In the following section, a blockchain application for a manufacturing supply chain system will be presented.

3.1.1 Blockchain Ready Manufacturing Supply Chain

In this part, a blockchain example [AM16], which stores and distributes specific product information during its lifecycle, will be discussed. This task also named product lifecycle management (PLM), involves different stages [Sta15], in which multiple parties modify the product's data, see Figure 3.1a. In this scenario, each actor interacts with a user interface, which is connected to the product's data from the blockchain. Each product is created in form of rules (smart contract) so that only specific users can access or modify the information. To access or alter this data, parties must authenticate themselves signing the requests with their private keys.

[1] http://www.skuchain.com/



(a) Product Lifefcycle Management (PLM) stages.

(b) Cardbox example.

Figure 3.1: Product Lifefcycle Management (PLM) stages and cardbox blockchain application example [Sta15].

During the lifecycle management, a product is owned by an entity, for instance the distributors. Moreover, when a product goes to another actor (e.g. from distributors to a consumer), both parties will sign a contract that updates product's ownership. Thus, the system ensures that in each phase, only the granted user modifies the contract.

At the end of [AM16], an example to better explain the mentioned approach is also presented. The application consists of the product lifecycle of a cardboard box, starting from the trees cut down until the box is recycled. In Figure 3.1b an overview of the implemented design is presented. In this business process, different actors are entailed. For instance, the box manufacturer first receives the physical object and then signs a contract to gain the product's access. Afterwards, he is allowed to alter it through the user interface (e.g. modify the box quality). Once finished, he will transfer the package (signed) to the next actor, in particular, the product filler, who will follow the same steps.

3.2 Network virtualization

In the last years, many investigations in network virtualization have been conducted by different authors [HLAZ11], [ZZSRR08], [CRB09]. Among these, it is important to highlight topics such as the trade-offs between single [CRB09], [HLZ08] and multi-provider VNE [DARP17], or the different techniques and algorithms used for achieving VN efficiency. Typically, they decompose the VNE into two tasks: VN partitioning and VN segment mapping [FBB+13].

Each of these studies proposes new interesting features compared to the prior research. Nevertheless, in this thesis, we will not contrast the different architectures (e.g. between a single or multiple InPs) or discuss the various VN partitioning and mapping algorithms. The main goal of this work will be to explore an existing network virtualization scenario, in particular, multi-provider VNE applications suffering from the limited information disclosure problem [DARP17], [ZXB10], [EDPM13], [CSB10].

3.2.1 Multi-Provider Virtual Network Embedding with Limited Information Disclosure

To deploy wide-area networks, service providers are willing to embed virtual resources across heterogeneous domains belonging to different infrastructure providers. As a result, the system avoids being

restricted to just a single provider topology, providing at the same time better efficiency. This process, also named inter-domain VNE[2], is descomposed in three main tasks [CSB10]:

- **VN request partitioning**: Divide the virtual network into groups of virtual nodes (subgraphs), such that virtual node and link requirements are satisfied, minimizing at the same time the software cost of the virtual network setup.
- **Inter-connection between subgraphs**: Establish paths between the previously created subgraphs.
- **VN segment mapping**: Embed the virtual resources (subgraphs) to the InPs physical nodes.

Furthemore, in such scenarios exists two types of communication [ZXB10]:

- **Horizontal communication** is the one established between infrastructure providers to guarantee the most efficient end-services. These relations can be: *public relations* established using a market mechanism (e.g: an auction), or *private relations* that are previously arranged, because the InPs already know each other. In both cases, the relation arises from the need to negotiate and cooperate to serve the SPs.
- **Vertical communication**, which emerges from the negotiation between SPs and InPs, where the former is willing to lease some virtual resources from the latter. This communication is typically facilitated by a third party (VNP).

One of the main challenges of inter-domain VNE, is that InPs are not willing to broadcast their resources information (e.g: nodes availability, cost) or their network topology to the outside world [DRP15]. Thus, the virtual resources need to be allocated in a constrained and limited scenario (LID). And if that was not enough, this InP's data is crucial to perform the partitioning of resources, in order to optimize the network software costs.

Centralized Slice Embedding

In [DRP15], a centralized VNE framework is implemented (Figure 3.2a). There, the mentioned LID partitioning problem is approached through a VPN, who accesses public information that is not considered confidential by the InPs:

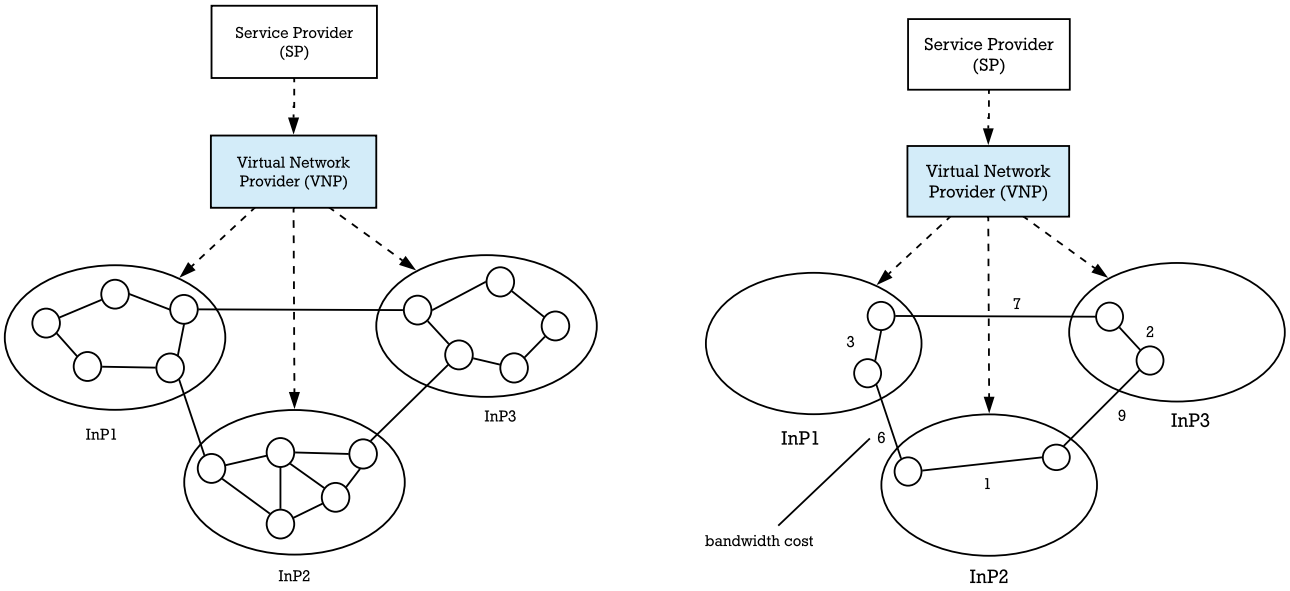
- **Virtual resources availability**: In this case, virtual resources examples are obtained from Amazon EC2 [ama18], which announces the attributes (CPU, memory, cost) of different instances types.
- **Substrate network topology**: Most of network's topology information is treated confidentially for the InPs. Nevertheless, there are certain aspects of the network which are not considered private, such as InPs peerings (including location) and its related link cost (Figure 3.2b).

Furthermore, in the [DRP15] implemented framework, the slice embedding is achieved by subsequently perfoming the following tasks: Firstly, the above-mentioned resource information is sent from the InPs to the VNP, who registers and stores it. Secondly, the VNP matches the service provider request with this collected data. Thirdly, the corresponding partitioning and mapping algorithms are applied (not discussed in this thesis).

In addition, the results prove that this centralized VNE approach provides embedding costs not much higher than in an ideal case scenario (InPs announcing all required information) and a high ratio between the number of slices requested and later allocated. Despite the positive findings, this scenario suffers mainly from scalability, since everything relies on a single centralized authority: the virtual network provider. In addition, it assumes that InPs will be constantly advertising their updated data to the VNP, which can lead to undesired costs.

Therefore, other approaches where the actors benefit from a more scalable and distributed slice embedding need to be further investigated.

[2] Inter-domain is between multiple InPs and intra-domain is inside a single InP.



(a) VNP's full view on substrate network.

(b) VNP's limited view on substrate network.

Figure 3.2: Virtual network Request across multiple InPs using a VN Provider. Adapted from [DRP15].

Distributed Slice Embedding

Many existing solutions [HLAZ11], [DRP15], [DARP17] rely on a centralized party (VNP) that stores the InPs disclosed information and then performs the partition of the virtual network request. However, what if the SPs and InPs are not willing to trust a centralized broker? In that case, this central actor could be removed enabling a direct communication between the SPs and InPs, which provides a better system scalability. Distributed network slice embedding has already been investigated in:

- In [EDPM13], a **consensus-based auction for distributed slice embedding (CAD)** is proposed. In particular, the physical nodes, which are owned by a single or different InPs, bid on virtual nodes. This value is subsequently stored in a vector b_i, where i is the physical node. Afterwards, once the bidding phase concludes, each physical node exchange the bids with its neighbors to reach an agreement for the auctioned virtual nodes. The mentioned bidding can be for a single slice (2 virtual nodes and 1 link) or on the entire slice (multiple virtual nodes and links). In the former, there is a limit on the number of biddable nodes, which although it has a great performance, it provokes multiple iterations (excessive time). Despite in a multi-provider VNE scenario, the latter is more suitable, the InPs willingness to disclose their internal information will hamper the performance. Hence, this approach provides better scalability (decentralization), but it is not proper for our scenario since it does not solve our LID problem.
- In [CSB10], a **policy-based inter-domain VN embedding framework (PolyViNE)** is presented. In PolyViNE, each InP embeds part of the VN request under its internal policies, while they cooperate with other InPs in a decentralized manner. In addition, forwarded decisions are location-based. For example, in Figure 3.3 each virtual node $\{A, B, C, D\}$, has a prefered geolocation. After that, the matched InPs will receive a notification for embedding the specific virtual node (e.g. InP3 with virtual node D). This whole process starts with the SP sending its VN request to multiple trusted InPs who reply back with the related embedding and its prices (bidding). Then, the SP will choose the most economical embedding. However, since there is no central broker, to ensure performance each SP needs to know $k^{SP} \geq 1$ InPs and each InP $k^{InP} \geq 1$ InPs. Also, most of the time a complete VN is not mappable by a single InP. In this case, the InP embeds its part and forwards the other to

a known InP. Despite, PolyViNE introduces a decentralized approach dealing with LID and uses the interesting location-assisted embedding, it has some drawbacks that need to be covered. Firstly, each InP has an extra overhead in communicating the rest of the request when the VN is not fully mapped. Secondly, the bidding to provide a competitive market is a good idea, but in this scenario, the bids can be publicly known by other InPs. Thus, a better bidding process preventing InPs from revealing its confidential data is needed. Finally, the requirement that SPs and InPs need to know at least 1 InPs can lead to an application poor performance.

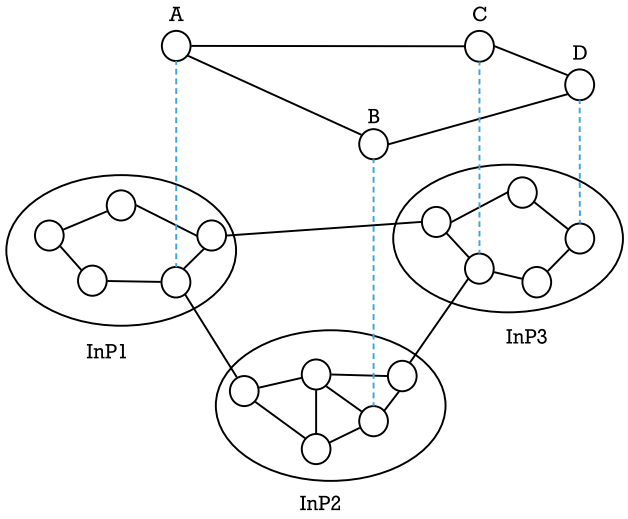


Figure 3.3: Multi-provider virtual network embedding [CSB10].

- Finally, in [ZXB10], an **automated service negotiation framework (V-Mart)** is implemented. The V-Mart is also decentralized and based on an auction mechanism, the Vickrey auction model. Though this approach ensures a fair market, thanks to the Vickrey auction model, it does not guarantee performance. The reason is that the VNE process deployed is a second-stage auction where explicitly all the InPs need to interact during the VN embedding. Thus, the VNE results in a high-demanding task. In addition, the sealed-bid is proposed to be done by a trusted 3rd party, which could cause again confidence issues. Nevertheless, the main V-Mart shortcoming is that it does not solve the LID problem, since it does not enforce inter-domain policies.

3.3 Analysis of Related Work

Through relevant examples in supply chain management, as the presented in section 3.1.1, we can observe the blockchain's potential mainly in areas, where entities are willing to exchange data without trusting a third party. Despite the research does not include detailed specifications, such as the blockchain's type or mining strategy, a useful system design has been provided. In addition, some of the main benefits compared to centralized technologies are contemplated:

- **Disintermediation:** Thanks to the consensus mechanisms, blockchain runs without a central administrator. Thus, intermediaries are not required for realizing transactions which suppresses all the significant costs of involving middle-parties [MS15a]. For instance, in the presented example, users were able to transfer ownership, just signing the product's contract.
- **Data management and redundancy:** Blockchain helps us in coordinating, validating and storing the distributed data in a decentralized manner. It also provides automatically redundancy since users own the same data. Reversely, in centralized systems, the users need to explicitly create a backup of the databases, for preventing the loss of data if the system crashes.

- **Unconstrained technology:** As actors can define and establish their own rules (smart contracts), new business models open up.
- **Privacy:** Thanks to cryptography, users can interact with untrusted partners. For example, in the alredy mentioned example, all actors were able to coordinate its services through the PLM process, which before would have been a complex task as they do not trust each other.
- **Transparency and integrity:** All the changes that are made to the contract data are directly visible to the other users. In addition, once a transaction is executed, its immutable since it cannot be altered or deleted.

On the other hand, in the second part of this chapter, we have investigated related work on multi-provider VNE suffering from LID. A comparison for each of the approaches presented before is shown in Table 3.1.

First of all, we can observe that decentralized systems provide obviously a better scalability because the application does not rely on a single point of failure. In centralized systems, this central entity has to be trusted since it stores and manages all the information. In contrast, all the presented related work on decentralized VNE, uses a bidding mechanism to reach an agreement between the SPs and InPs.

Despite being a decentralized system, [ZXB10] uses to some extent a trusted third party (TTP). This is because, it is the only approach offering a sealed-bidding between users, although with the use of a trusted third party. We also notice that in [EDPM13], [CSB10] the bids are exchanged between InPs to cooperate in the VN embedding, which results in undesired computational costs. In contrast, centralized systems [DRP15], perform the service negotiation without using auctions, since they trust a middle-party managing these operations.

Moreover, one of the main challenges is to solve the limited information disclosure issues. We notice that [DRP15], [CSB10] deal with the problem, accessing public information not considered confidential and performing the VNE under its internal policies respectively. In addition, they provide location-assisted VNE, which facilitate the VN request partitioning across different InPs. The last, it also introduces a notification system, where the InPs gets notified upon VN request, which avoids time spending in request synchronization.

In general, centralized systems provide a better throughout in small scenarios since all the information is managed by a central entity. Hence, data sharing or user's cooperation is not needed before performing any operation. Conversely, decentralized systems benefit from large scenarios. Firstly, because the more users participating in an auction, the more competitive the market is. Secondly, all the activities are diversified, and thus, the decision-making is more efficient.

Finally, we classified costs into virtual network embedding and economic. Overall, we observe that VNE costs are lower when using a centralized system [DRP15], although when the parties are efficiently coordinated in a decentralized scenario, the costs are also extremely reduced [CSB10]. Concerning economic costs, they are obviously higher when a centralized system is used, as the application performance depends only on a central entity, which needs to be financed for their computational and maintaining costs.

3.4 Summary

After comparing the different multi-provider VNE approaches exposed throughout this chapter, we find out that each of them introduces innovate features to handle network virtualization. However, there is no single solution that satisfies at the same time the following requirements:

- Scalability.
- LID problem approached with data confidentiality.
- Large scenarios performance.
- Computational and economic low costs.

Approach	System	Scalability	Trusted third party (TTP)	Sealed bidding	Bid exchange between InPs	LID problem solved	Location-assisted VNE	User notified	S-performance	L-performance	Low VNE cost	Low cost (€)
Multi-provider VNE framework with VNP [DRP15]	C	✗	✓	✗	✗	✓	✓	✗	✓	✗	✓	✗
CAD - Consensus-based auction for distributed slice embedding [EDPM13]	D	✓	✗	✗	✓	✗	✗	✗	✗	✓	✗	✓
PolyViNE - Policy-based inter-domain VN embedding framework [CSB10]	D	✓	✗	✗	✓	✓	✓	✓	✗	✓	✓	✓
V-mart - Automated service negotiation framework [ZXB10]	D	✓	*	✓	✗	✗	✗	✓	✗	✗	✗	✗

Table 3.1: Comparison of virtual network embedding approaches - Overview for system features implemented as *yes* (✓), *no* (✗), *to some extent* (*). Abbreviations: *D* = decentralized, *C* = centralized, *L-performance* = large scenario performance and *S-performance* = small scenario performance.

Therefore, the main focus of this thesis will be to investigate and later implement an approach that gets rid of the VNP, and which fulfills the above-mentioned virtual network embedding demands. Here, we foresee that blockchain could play a significant role.

4 Design

Based on the challenges highlighted in the previous chapter, a conceptional design of a multi-provider virtual network embedding approach using blockchain will be now introduced. First, we define the requirements and assumptions used for the design process. Afterwards, we will present the fundamental goals expected from the system, followed by a general overview of the architecture along with its functionality. In addition, important features for our application will be discussed, in particular, the blockchain type, the mining strategy, the authentication system and the auction model used.

4.1 Requirements and Assumptions

Since blockchain is a disruptive technology, apparently it seems that it could improve many existing systems. However, is the blockchain the only way of approaching scalability, security or data handling between organizations? Or why not use a traditional database rather than a blockchain? The truth is that there is not a single solution or better technology than the others, as it depends on user's requirements. Thus, it is important to understand in which scenarios the blockchain could replace existing infrastructures. For this purpose, in Figure 4.1 we present a flowchart [WG17], which pretends to guide users in determining whether blockchain is the appropriate solution to resolve their problems, and if yes, the suitable BC type. We will also follow the diagram to decide if our defined VNE scenario demands match the blockchain use case.

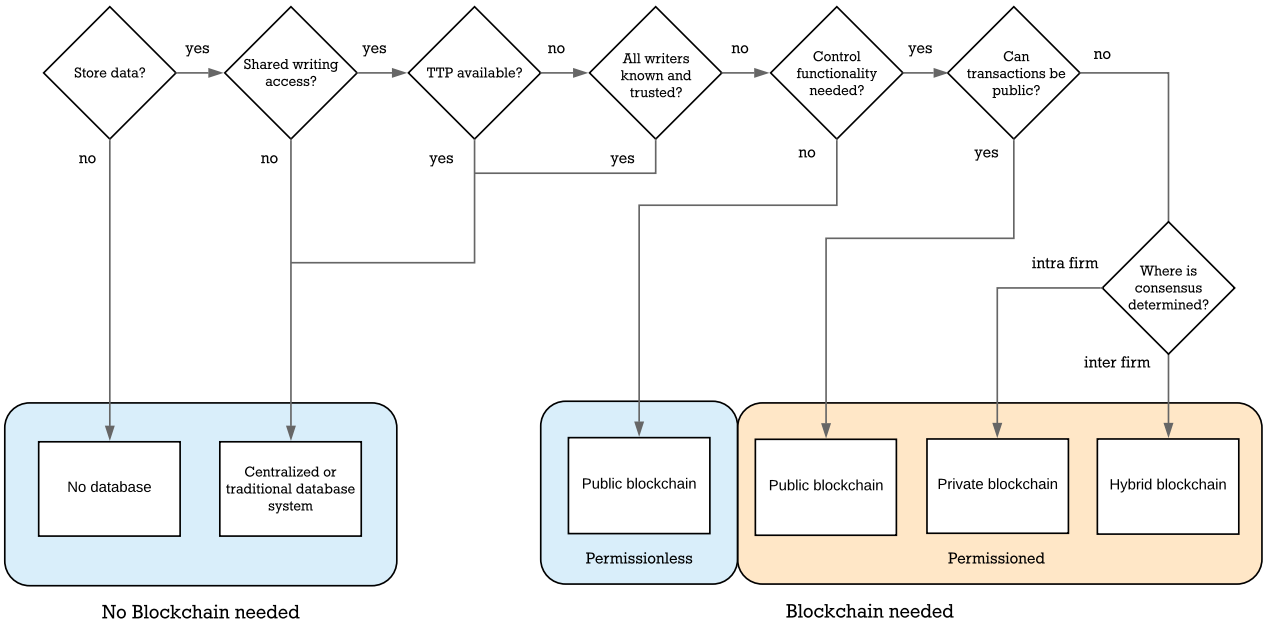


Figure 4.1: Flowchart to determine whether blockchain is the suitable solution. Adapted from [WG17].

Before starting with the flowchart, one statement needs to be clarified: if the user does not need to store data, no blockchain or database is required. The discussion arises when multiple writers want to exchange data without relying on a trusted third party (TTP). In this context, a writer is a user that performs the four basic functions of persistent storage: create, read, update and delete data (CRUD). If they trust a TTP, a traditional or centralized database will always provide a better throughput. However, if this is not the case, then the user should first consider if all the writers are known and trusted. If so,

again a typical database (e.g. SQL) will be better positioned. If all the last requirements are satisfied, or in other words, multiple untrusted writers are willing to update the state of a system without trusting a middle actor, it is then that using a blockchain is a reasonable solution.

In Figure 4.1, we also classify blockchains into two groups: *permissionless* and *permissioned* blockchains. To decide between these two approaches, the user should consider if he is willing to control his customized blockchain, creating his own specifications. For example, the main networks of Bitcoin and Ethereum are permissionless, where users are added to the network and thus, restricted to its policies. In Table 4.1 the main characteristics of each blockchain type are summarized. Scalability is an important topic in the blockchain domain, where we distinguish: node scalability and performance scalability. The former is the capacity to add new participants in the blockchain without losing performance, and the latter is the number of transactions executed per second. In addition, note that some features depend on the blockchain design decisions since permissioned blockchains capabilities are easily modifiable. For example, if consensus algorithms are used, the interaction between users should have a cost in order to ensure performance, i.e miners need to be rewarded with transaction fees.

Features	**Permissionless**	**Permissioned**
Type	Public	Public, private and hybrid blockchain
Control functionality/ Transaction processing	Fully decentralized	One or a small group of pre-selected entities
Access control	Open access	Authorized access
Number of users	High	Low
Number of untrusted users	High	Low
Writers	Any user participating	Permissioned group of people, e.g. bank customers
Centrally managed	No	(*)
Security	Consensus algorithms (e.g PoW)	(*)
Transaction fee	Yes	No (*)
Trusted	No	Yes
Node scalability	High	Low
Performance scalability	Low	High
Use cases	Cryptocurrencies (e.g. Bitcoin [bit09])	Transactions in business networks (Hyperledger Fabric [hyp17])

Table 4.1: Types of blockchain: permissionless vs permissioned. Adapted from [WG17]. (*) depending on blockchain design decisions.

Furthermore, as stated in Figure 4.1, permissioned blockchains are divided into:

- **Public permissioned blockchain:** Decentralized, all the participating users are allowed to read the system's state (transactions can be publicly verified), but just some of them have writing privileges.
- **Private blockchain:** Centralized, with access and writing permissions restricted to a set of users. Note that in the flowchart (Figure 4.1) this corresponds to an intra-firm consensus since it is only managed by a single organization. We understand by firm a traditional company or a pre-selected group of users.
- **Hybrid/Consortium blockchain:** Partly decentralized. The blockchain is controlled by a pre-established set of entities (inter-firm). In this model, each of these firms coordinates and ensures the system's functionality. Note that compared to the private blockchain, multiple firms maintain and serve the infrastructure rather than a single one. Thus, a consensus between these parties is also extremely needed. For example, currently, several banks deploy common blockchains to manage their shared data and reduce intermediaries costs. In this case, each bank will control a node, resulting in a better throughput in terms of scalability.

After presenting the blockchain use cases and types, we observe some similarities between a traditional shared database and a permissioned private chain, since they are both managed by a group of users, which simultaneously maintain the same confidential data. Despite private blockchains could be seen only as another term used to name shared databases, the truth is that each technology has its trade-offs, without mentioning scalability or cost issues. On one hand, centralized databases will always have better throughputs since, in addition to the basic transaction operations, blockchains need to perform cryptographic verifications, and ensure a consensus between the other participating nodes. On the other hand, private blockchains are less prone to malicious attacks, inasmuch as they spend a longer period in checking errors or validating transactions, than regular databases. The blockchain also organizes data into blocks, which is updated only appending new data. Therefore, as already noted, if users trust each other and a high-performance is desired, using a blockchain will not be the suitable solution.

4.1.1 Virtual Network Embedding with LID scenario

In section 3.3, we have concluded that although some research in network virtualization have been conducted, to the best of our knowledge, there is still no approach under LID that gets rid of the broker (VNP), providing scalability, confidentiality, and low-cost performance. Thus, after having spotted the main benefits of integrating blockchain in real-scenarios, among which disintermediation, privacy and transparency need to be highlighted, we assume that this technology could improve the VNE business process. Nevertheless, first, we should investigate whether our scenario could be integrated in a blockchain.

In network virtualization, we mainly encounter users with two different roles: Infrastructure Providers and Service Providers. Moreover, as the VNE is typically performed by the same parties across the countries, e.g. Deutsche Telekom, Vodafone or O2 in Germany, the scenario is restricted to a limited amount of users that barely changes. Therefore, we are facing an application where multiple known users but probably not trusted, want to reach a virtual network agreement, without relying on a third party. The decision of not including a TTP or VNP is mainly to avoid exchanging the InPs internal network cost models to an intermediate actor.

Observing the last mentioned requirements, we foresee that blockchain could be a suitable solution for our VNE scenario suffering from LID. However, an appropriate blockchain type still needs to be chosen. Obviously, the service negotiation between InPs and SPs is performed in a private environment, which automatically removes the permissionless and permissioned public blockchain options. Hence, the decision should be taken between a private or a hybrid blockchain.

One of the main requirements noticed from the previous related work is that our approach should address scalability issues. Accordingly, using a private blockchain, which is a centralized system managed by a single firm, will provoke that users rely again on a central point or trusted party, converting it to an unfeasible workaround. Thereby, the **partly decentralized hybrid blockchain** seems to be the most suitable solution. In this scenario, a firm corresponds to each interested InP or SP willing to possess and control a blockchain node, in order to ensure the network virtualization functionality.

4.1.2 Smart Contracts and Ethereum

After having spotted the most suitable blockchain type for our VNE environment, the non-trivial decision of selecting a blockchain platform needs to be taken. Prior to this, we need to consider whether our scenario could profit from the potential of smart contracts. Despite in subsection 2.1.2, it has been discussed what smart contracts are, it is also required to notice when they can be used. Thereupon, due to its extended dimension, a market study regarding smart contracts and its application fields will not be conducted. Nonetheless, we will enumerate the main usage examples of smart contracts in business to business processes [Tea17]:

- **Settlement agreement:** Since smart contracts can be personalized, it enables that participants can reach any kind of agreement without involving third-parties costs.
- **Frequently executed tasks:** Users regularly performing the same operations, can accelerate this process by depicting them on a smart contract code.
- **Self-operating databases:** Thanks to the blockchain, secured shared databases with low-costs are easily deployed. In addition, smart contracts can automate the interaction with this stored data, providing speed, real-time transparency and at the same time, opening new business models.

In the presented network virtualization example, as the users negotiate the lease of virtual nodes over and over to reach an agreement, this task could be perfectly reflected in a contract code. Apart from dealing with smart contracts, as later we will implement a prototype, the blockchain platform chosen needs to fulfill the following prerequisites [MLTJ17]:

- **Easy to build, use and learn:** A low effort in setting the platform is essential. Thus, the smart contracts should be written or derived from common languages in the world of programming, such as JavaScript or Python.
- **Support and documentation:** A consolidated platform, which is not in its early stages and with a high-maturity level achieved, is extremely needed. An active community of developers behind the project is also fundamental, e.g. with a considerable number of Github stars and forks.
- **Blockchain type:** Though it could be desirable that the main network is permissioned and hybrid or private, currently, in open source code platforms, the blockchain types can be modified. Thus, that the predefined main network type matches the desired one, is not a must-have feature.

Characteristics	Bitcoin	Ripple	Ethereum	Hyperledger Fabric	R3 Corda
Application	Payments	Payments	General purpose	General purpose	Financial services
Governance	Bitcoin developers	Ripple Labs	Ethereum developers	Linux Foundation	R3
Main network	Permissionless, public	Permissioned, public	Permissionless, public and private	Permissioned, private and hybrid	Permissioned, private and hybrid
Data model	UTXO-based	UTXO-based	Account-based	Account-based	UTXO-based
Configuration effort	Low	Low	Low	High	High
Support and Documentation	High	Medium	High	Medium	Low
Smart contract language	-	-	Solidity (JS), Serpent (Python) and LLL	Go, Java	Kotlin, Java
Currency	BTC	XRP	ETH	None	None

Table 4.2: Comparison of different blockchains platforms. UTXO stands for Unspent Transaction Outputs.

In table 4.2, a comparison of the most common blockchain platforms, in our opinion, is depicted. Since in our application a scripting language with smart contracts is preferrable, and Bitcoin and Ripple by default do not incorporate them, these platforms are automatically discarded. On the other hand, Hyperledger Fabric and R3 Corda are popular used platforms, whose main network perfectly fits our requirements. However, they provide customized blockchains with a high configuration effort, since its specifications need to be carefully designed. Thus, as a platform with a straightforward setup and usage is required, Ethereum seems to be the better positioned.

Ethereum is a mature and easy to build project based on smart contracts with a large number of developers behind, who are actively discussing and improving its functionality. Thanks to this, the

creation of new business models is extremelly facilitated for newcomers. Furthermore, Ethereum smart contracts are written in Solidity and Serpent, which are similar to JavaScript and Python respectively. Despite the fact that Ethereum's main network (public permissionless), is not the ideal for our scenario (permissioned and hybrid), the simple and easy set up of the platform's environment, will allow us to quickly deal with these issues. Finally, the Ethereum Virtual Machine (EVM) is a Turing-complete system, which helps us in obtaining the maximum benefit from the smart contract's potential.

Overall, thanks to the blockchain, more precisely Ethereum and smart contracts, the VNP could be replaced with a secure, flexible and coordinated system. In this scenario, the reliance will now be distributed among multiple blockchain nodes rather than just on a costly single entity. However, challenges, such as the scalability or the service negotiation performance, will have to be thoroughly investigated along the thesis.

4.2 System Design

Based on the requirements and assumptions defined in the previous section, the design of a multi-provider virtual network embedding system using the blockchain technology is presented in the following. At first, the fundamental goals that are expected from the system are defined, followed by a general overview of the architecture and later on, the essential design features are discussed.

4.2.1 Fundamental Goals

In general, the fundamental goals for network virtualization relies on simulating baremetal networks, providing improvements in terms of flexibility, manageability, and costs. In this scenario, the SPs are willing to embed virtual nodes across different InPs to create customized virtual networks. This embedding task consists first of the VN request partitioning and later, of the corresponding mapping to the InP's physical nodes, such that virtual node and link requirements are fulfilled. However, the problem appears when InPs are not willing to disclose detailed information about their resource availability or network topologies, which is indispensable for the VN partitioning efficiency [DRP15]. In this thesis, the VN partitioning is treated as a cost minimization problem, where the service provider pursues the minimum VN embedding cost[1]. In a real world, there are more parameters that could be taken into consideration, such as quality of the service or the proportioned capacity.

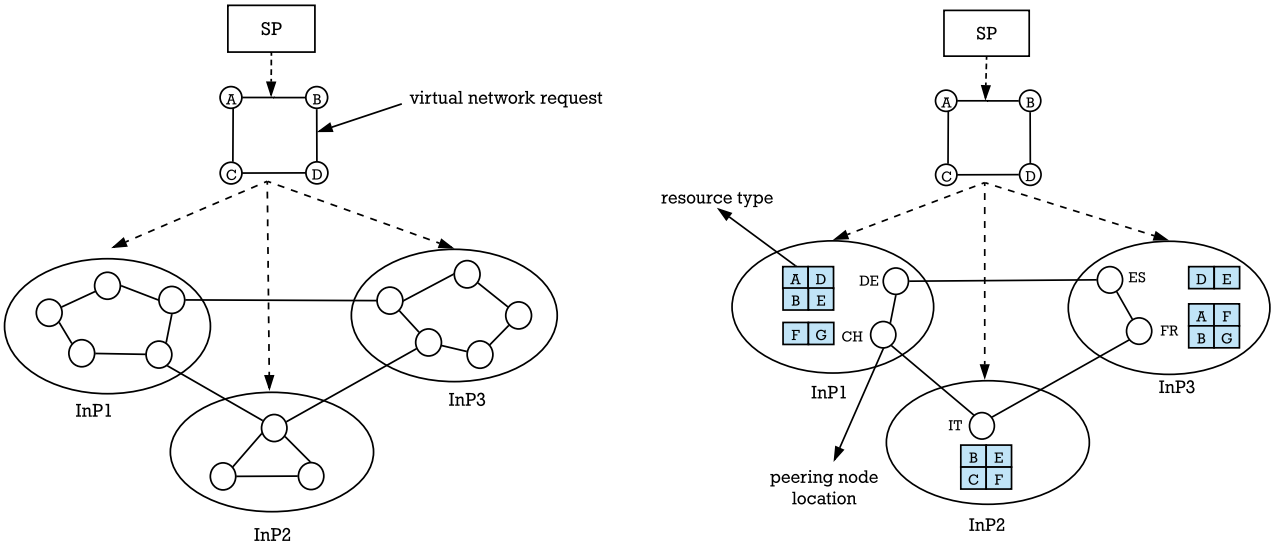
In a multi-provider scenario with full information disclosure (Figure 4.2a), since the infrastructure provider's internal information is publicly available, the VN request partitioning is efficiently computed. Conversely, the lack of data in a limited information disclosure scenario (Figure 4.2b), directly affects the VNE throughput. Nevertheless, there is some information not considered confidential by the InPs, such as the location or the virtual node types offered by the peering nodes. For instance, in Figure 4.2.b, the InP1 has two peering nodes with locations in Germany and Switzerland, and virtual node types $\{A, B, D, E\}$ and $\{F, G\}$ respectively. These virtual node types, as Amazon EC2 [ama18] does, are used to classify the resources in different groups, each having common attributes, such as CPU, memory, storage and network capacity. Thus, in this thesis, new embedding strategies and approaches to facilitate the service negotiation process between the InPs and SPs, will be developed.

Moreover, in section 3.3, it has been contemplated that decentralized approaches tend to use auctions to negotiate the network workflows. Regardless of the auction model applied for our VNE framework, there are several objective metrics that need be considered:

- **Efficient:** The virtual network embedding must be efficient in terms of synchronization, communication, and data sharing between the participating users.

[1] From now on, we use the term cost to refer the value that the SP must pay for embedding the virtual nodes across the InPs. This value is derived from the InP bids

- **Equality:** All participants, namely service providers and infrastructure providers, must have the same role's privileges. In other words, all SPs must be able to request the same amount of virtual networks over time and InPs bids must be treated equally in the application.
- **Market fairness:** The auction model requires that the SPs find the desired service at the lowest price and that the InPs obtain the highest possible revenue for the offered service.
- **Data confidentiality:** At the moment of bidding, the bids should be kept in secrecy in order to not alter the auction progress. Finally, the winners can be publicly known.
- **No trusted auctioneer:** Since confidentiality is required, a middle party or auctioneer should not be capable of managing the auction workflow and thus, accessing the private bidding values.
- **Automated:** The virtual network embedding process needs to be automated with a friendly and easy-to-use framework.



(a) Full information disclosure (FID).

(b) Limited Information Disclosure (LID) with peering node locations and resource types.

Figure 4.2: Substrate network visibility. Adapted from [DRP15].

4.2.2 System Architecture

After defining the main VNE problem that needs to be addressed, and describing the goals and metrics required for the design of our scenario, a system architecture is now presented, see Figure 4.3. Firstly, the proposed design shows the main components of the system, which are:

- **Private group of users:** The system is composed of a private group, in which participants can be infrastructure providers or service providers. Since in private groups only certain members are allowed to access and alter the data, in this design, new users must ask for permission before joining the network. Furthermore, once entering the group, the newcomer can optionally act as a miner or just use the application without the need of maintaining a blockchain node.
- **User interface (UI):** Through an API (for automation), each member will efficiently interact with the blockchain application. Depending on the user's role, i.e infrastructure providers or service providers, the API will provide different functionalities. For example, SPs have the ability to request virtual networks and InPs to bid for the proposed virtual nodes, but not the other way around. Thus, the available operations must be restricted to the user's privileges.

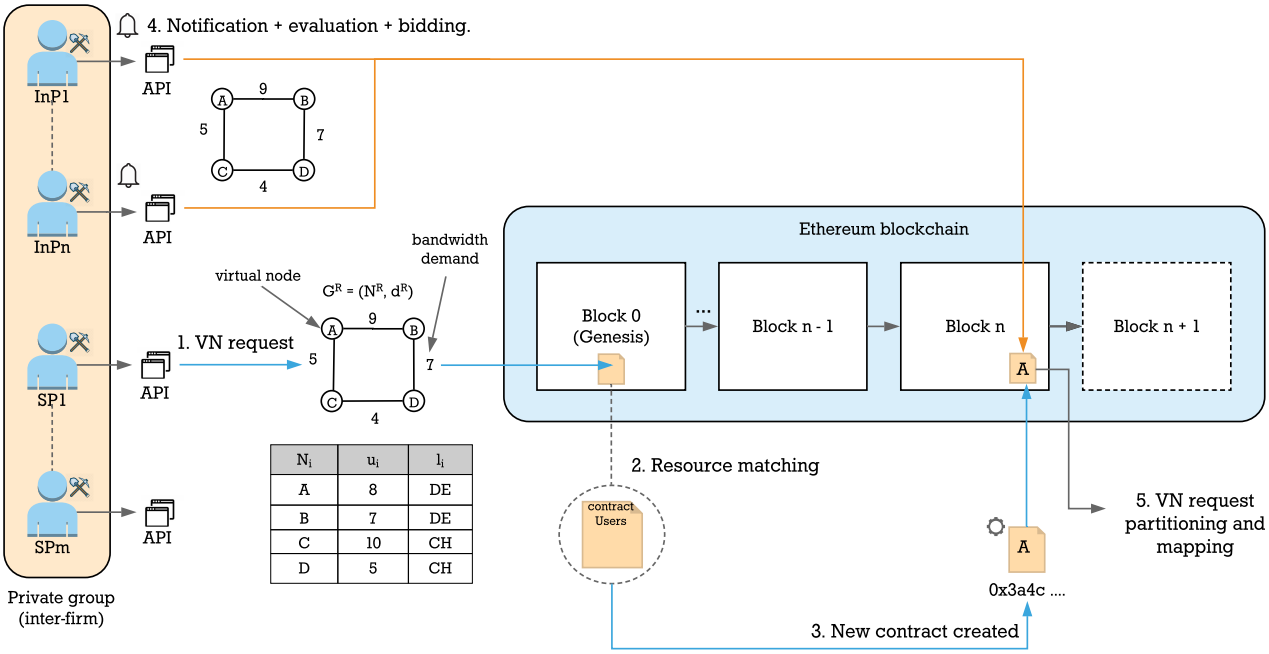


Figure 4.3: Our design for the system architecture. u_i states for upper bound cost and l_i for location.

- **Ethereum blockchain:** One of the main differences between the Bitcoin and the Ethereum blockchain is that the first is transaction-based while the second account-based. For this reason, apart from a list of transactions, each Ethereum's block also stores the most recent state, i.e. the current account list with the corresponding balances. Nevertheless, every blockchain starts with the block 0, also called *Genesis block*. Since it is the first block, it is the only one not pointing to a predecessor and it is typically hardcoded in a file called *genesis.json*. This file contains some of the blockchain specifications, such as the difficulty or the gas limit. By modifying the value of these parameters, users are allowed to create blockchains based on their requirements. More details about the genesis file content used in our system will be discussed in chapter 5.
- **Smart contracts:** Above, we have mentioned that each Ethereum block stores a list of accounts with their corresponding balances, among others. These accounts can be externally owned accounts (users) or contract accounts (smart contracts). In the case of smart contracts, they enable reflecting user's needs into code and can be created or called either from another contract or simply by an EOA. In our system, a static contract called *Users* is contained, which will be permanently requestable. This contract will store all the participant's information, and whenever a new user registers, its state will be modified. In addition, this contract includes the InPs entered information about their substrate networks, such as the peering nodes or the substrate node types, which will be used to match the virtual network requests.

To provide a clearer view of the last mentioned components, the VNE functionality needs to be introduced. This VNE process entails the following steps (Figure 4.3):

1. **VN request:** A service provider requests a virtual network, which can be represented in a graph as $G^R = (N^R, d^R)$, where N^R consists of a set of k virtual nodes, and d^R the set of all bandwidth demands between the virtual nodes $i, j \in N^R$. Here, the superscript R stands for request. In addition, each virtual node $N_i \in N^R$ is formed by a collection of attributes, such as the desired location l_i or the upper bound cost u_i, where the last term may be the maximum amount of money that the SP is willing to pay for virtual node N_i. Otherwise, if an upper bound cost for the whole network is specified u_G, the one for each virtual node will be treated as $u_i = \frac{u_G}{k}$.

2. **Resource matching:** Before the auction starts, first, the associated set of attributes (e.g. location) of the requested virtual nodes N^R are matched with the InPs entered data in the *Users* contract. For instance, virtual nodes $\{A, B\}$ and $\{C, D\}$ will be matched with InPs possessing physical nodes in Germany (DE) and Switzerland (CH) respectively. In case no location l_i is specified for the nodes N^R, the VN request will be broadcasted to all the registered infrastructure providers.
3. **Auction start:** Using the previous information, the corresponding smart contract, i.e. a *limited time auction*, is created. This contract contains: (i) the data related to the virtual network nodes and links, (ii) the address of the SP willing to lease virtual nodes, (iii) a list of all the InPs addresses matched, (iv) a lifetime and (v) a bidding period field. Since the bidding and the virtual network lifetime must terminate, in the first step, the SP must also add the bidding period and the lifetime fields to the request.
4. **Notification and bidding period:** After the new auction contract is created, the included InPs, which are the only users with the ability to bid for the resources, are notified. This notification can be transmitted either by a push-based technology, where the server or publisher updates information on the client (e.g. Publish/Subscribe system) or by a pull-based technology, in which the client initiates the request of information from the server. In this thesis, our system utilizes a pull-based technology (section 5.2), such that the application is more customer-centric without the need of using and maintaining external databases apart from the blockchain. Note that this notification system could also be done by creating complex contracts in the BC, although we considered this an extra work not required for this thesis since a pull-based system already fulfill the requirements. Regarding the bidding, despite having a full view of the VN request and knowing the other matched InPs, each InP is only allowed to bid for their paired resources. For instance, InPs from Germany can only bid for the resources $\{A, B\}$. Before bidding, these users typically evaluate the request requirements into their substrate network, depicted as $G^S = (N^S, L^S)$, such that N^S and L^S are the set of physical nodes and physical links respectively. Although InPs are interested in serving requests at maximum profit, a single InP can barely serve all the virtual nodes requirements. As a result, the virtual network just may be supplied by different infrastructure providers. More details about the bidding mechanism will be presented in section 4.2.5.
5. **VN request partitioning and mapping:** Once the auction has finished, each virtual node will be assigned to the winning InPs so that the VN embedding cost is minimized. These winners will be publicly known by all the other participating users and thereupon, the service provider will contact them to ultimate the VN setting up, i.e. to perform the VN segment mapping.

To summarize, the presented system design uses the blockchain technology, in particular, smart contracts, to enhance the VN partitioning efficiency in a multi-provider scenario with limited information disclosure. Nevertheless, up to now, this is only a conceptual design with a lot of features that must be further discussed.

4.2.3 Blockchain Type and Mining Strategy

The Ethereum's main network, as Bitcoin, is public and permissionless, where all the users have access to the transferred values between accounts, since value transfers cannot be blinded. However, since our described scenario has a small number of users, the fully public decentralized approach will directly affect the application performance. Thus, as concluded in section 4.1, our focus is on **permissioned hybrid blockchains**, where a set of inter-firm entities control and maintain the blockchain functionality.

On the other hand, the default Ethereum's consensus algorithm will be used, which is at the time of writing, the **Proof-of-Work** protocol. Nevertheless, since the blockchain can be customized, in chapter 6, other mining strategies will be further investigated on their suitability.

In summary, although the Ethereum main network is public and permissionless, it can run as a private or hybrid blockchain, with the technology mechanisms and protocols remaining unaltered.

4.2.4 Authentication System

In the following, we will discuss how a user is registered and authenticated to the system, see Figure 4.4.

First of all, in order to deploy and use the presented design, a single or a group of InPs need to set up the environment. Afterwards, this consortium of InPs or a public authority, such as a regulatory agency, is responsible for inviting others users (InPs or SPs) to join his network, i.e. to create an Ethereum account (EOA) for the application.

However, once these users are registered, the communication between them must be secured. For this reason, the blockchain technology applies the **public-key cryptography**, also called asymmetrical cryptography, which generates a public and private key pair. The first, as the name suggests, it is publicly disclosed, and the second must remain confidential to its respective owner. Since these keys are mathematically related, the information encrypted with a private key can only be decrypted with a public one, and vice-versa. In addition, the Ethereum address is derived from this key pair, more specifically, from the last 160 bits of the *SHA3-256*[2] hash of the public key. Thus, once a user creates a new Ethereum account, the blockchain generates a public and private key pair for this user.

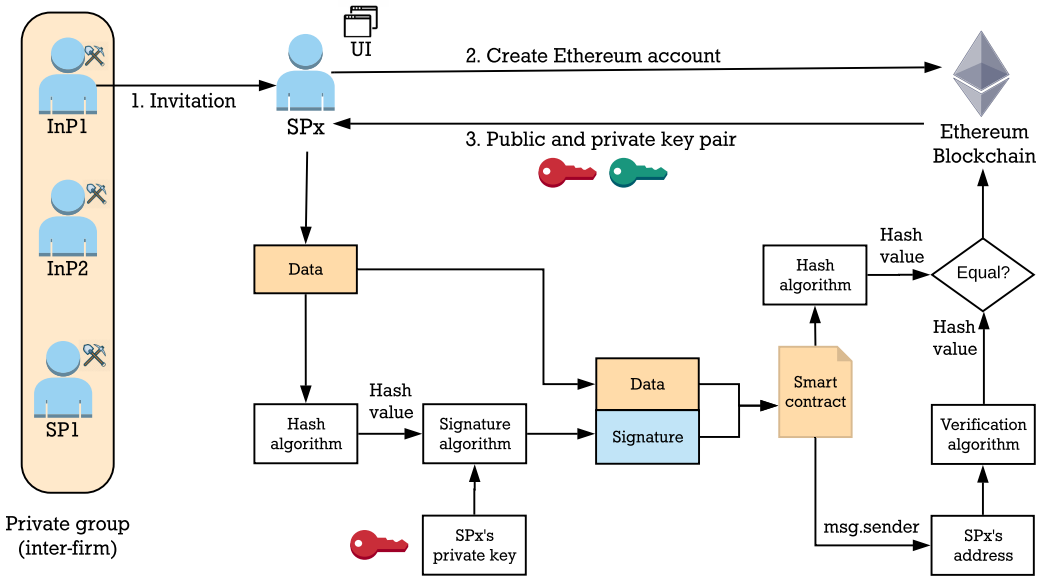


Figure 4.4: Our authentication design and used public-key cryptography.

Furthermore, to validate the origin and integrity of the messages, users must always digitally sign their operations, by providing a **digital signature**. In particular, Ethereum uses the *Elliptic Curve Digital Signature Algorithm (ECDSA)*. In this process, see Figure 4.4, a sender uses his private key to sign a hashed version of the data, which produces the signature. This signature, together with the original data, is sent to the receiver, who uses the sender's address (related to public key) and the data to verify that the information has been signed by who claims to be the owner and that the content has not been altered (hash value comparison).

It is important to note that smart contracts, if well-defined, also provide security in terms of access control and confidentiality. Despite the fact that Ethereum users can create and own smart contracts, the content is not always accessible or alterable. For instance, when users are not willing to broadcast the contract's information or simply when they want to avoid that every party is able to alter the state of the contract. Thus, these operations can be restricted only to a set of users, whose addresses need to be known by the contract account. In our VNE scenario, the matched InPs are the only users allowed to bid on specific resources.

[2] `https://en.wikipedia.org/wiki/SHA-3`

Although the blockchain is a decentralized and distributed technology, where the actors do not need to know or trust each other, thanks to cryptography, a secure and unalterable communication between all the participants is achieved. In addition, since Ethereum uses smart contracts, the data access can also be restricted.

4.2.5 The Vickrey Auction Model

In section 2.2.1, after reviewing different auction types, we have concluded that a **one-sided reverse Vickrey auction** seems to be the most suitable for brokerless virtual network embedding. In the following, a more thorough examination in adapting this auction model to the multi-provider VNE scenario with LID will be presented.

Firstly, we seek an efficient virtual network partitioning where the requested virtual nodes are assigned to the participating InPs, such that the VNE costs are minimized. Despite an auction could use a multi-dimensional process, i.e. more parameters, in this report, only the price quotes of the notified InPs will be used to determine the minimum VNE cost. Hence, the InPs act as sellers, who submit bids for the services requested by the SP, and once the auction finishes, the VN will be splitted between the winning InPs. In this section, our focus will be on the techniques and auction models applied for the VN partitioning (Table 4.3 summarizes all notations).

Symbol	Description
G^R	virtual network request formed by a set of virtual nodes N^R and a set of bandwidth demands d^R
N^R	set of requested virtual nodes
d^R	set of bandwidth demands between the virtual nodes $i, j \in N^R$
N_i	virtual node $N_i \in N^R$
N_i^t	virtual node N_i type
d_i	virtual node demand for node N_i
d_{ij}	bandwidth demand $d_{ij} \in d^R$ between the requested virtual nodes $i, j \in N^R$
P_l	l virtual nodes groups
u_i	upper bound cost for the virtual node N_i
u_G	upper bound cost for the virtual network G
l_i	desired location for the virtual node N_i
$b_n(N_i)$	$n's$ bid value for the virtual node N_i
$b_n(G)$	bid value for the virtual network G
$v_n(N_i)$	$n's$ real cost for the virtual node N_i
$v_n(G)$	$n's$ real cost for the virtual network G
$C_n(N_i)$	cost of virtual node N_i when assigned to InP n
$C_n(G)$	cost of virtual network G when assigned to InP n
$C^*(G)$	minimum cost of virtual network G
$\gamma_n(N_i)$	InP n profit margin at the time of bidding on virtual node N_i
$\gamma_n(G)$	InP n profit margin at the time of bidding on virtual network G
G^S	substrate network formed by a set of substrate nodes N^S and a set of links L^S
N^S	set of substrate nodes
L^S	set of substrate links between the substrate nodes $i, j \in N^S$
$\alpha_{s*}(N_i)$	dynamic cost for embedding N_i in the paired substrate node N_s^*
$\beta_{s*}(L_{ij})$	dynamic (fixed in evaluation) cost for sharing the paired substrate link L_s^*

Table 4.3: Notations.

Since in our scenario, the SPS request the embedding of different virtual nodes $N_i \in N^R$, we are facing a multi-unit auction. Thus, before defining the minimum VN cost $C^*(G)$ using the *one-sided reverse Vickrey* auction, it is important to note that $n = 1, ..., N$ InPs can submit bids per virtual node $b_n(N_i)$ or for a group of virtual nodes $b_n(P_l)$, where P_l are the $l = 1, ..., 2^k - 1$ possible virtual nodes combinations.

- **Per virtual node:** In a Vickrey auction with multiple items, the simplest form to bid is submitting a quote for each unit. In our scenario these units are virtual nodes, and in contrast to standard auctions, the **lowest bid** wins the service, inasmuch as the SPs are the ones who pay for the services offered by the InPs. Thus, to ensure not ending up with inflated prices, the SPs denote either an upper bound cost per virtual resource u_i, or a global one for the virtual network u_G, where $u_G = \sum_{i=1}^{k} u_i$. For instance, let $v_n(N_i)$ and $b_n(N_i)$ be, respectively, bidder n's real cost and bid value for the virtual resource N_i. Since it is a *low-bidding second-price auction*, the cost $C_n(N_i)$ that the service provider pays to InP n, is given by (4.1). This value must also include the link costs, which after being evaluated by the InPs, are subsequently added to the quoted price $b_n(N_i)$. In this thesis, we assume that the intra-link costs within an InP are negligible and that the inter-links, due to the lack of information of the other InP's substrate topologies, must be roughly estimated by the InPs. On the other hand, it is important to note that in our system, when $b_n(N_i) = b_j(N_i)$, the first bidder will be the provisional winner. For instance, in an English auction the same decision is taken, since when a bidder overcomes a price, the next participant must always bid a higher value than the current one. Thus, the InP's goal is to smartly quote $b_n(N_i) = v_n(N_i) \times [1 + \gamma_n(N_i)] \leq u_{N_i}$, where the symbol $\gamma_n(N_i) \in [0, 1]$ represents the profit margin that InP n takes at the time of bidding for N_i, knowing that the lower $\gamma_n(N_i)$ is, the higher the probability of winning, although at a smaller revenue. The cost $C_n(N_i)$ that a SP pays to InP n is given by:

$$C_n(N_i) = \begin{cases} \min b_j(N_i), & b_n(N_i) < b_j(N_i) \leq u_i : \ \forall j \neq n, \ \exists b_j(N_i) \neq \infty \\ b_n(N_i), & b_n(N_i) < b_j(N_i) \leq u_i : \ \forall j \neq n, \ \nexists b_j(N_i) \neq \infty \\ \infty, & \text{otherwise} \end{cases} \tag{4.1}$$

- **Per group of virtual nodes:** The Vickrey auction model behaves differently when a seller offers a package of items at a lower price, also called package pricing. In this context, rather than ending up paying the second price of the auction, the winner pays the **opportunity cost** for the units won, or in other words, the "damage" that his bids impose on the rest of the bidders. Formally, it is described as follows. Let x be a vector of outcomes again with $n = 1, ..., N$ players bidding $b_n(X_n)$. The mechanism will seek for the outcome $x^* \in argmax_{x_1,...,x_N} \sum_n b_n(X_n)$ that maximises the efficiency. Then, the cost for bidder n will be $p_n = \sum_{j \neq n} b_j(x^*_{-n}) - \sum_{j \neq n} b_j(x^*_j)$, where the term x^*_{-n} refers to the most efficient outcome without the presence of bidder n. Generally speaking, this expression can be seen as the result of the auction when bidder n is not participating in the auction, minus the total cost for the other bidders when n participates. However, in [AM+06], it has been proven that the Vickrey auction for multiple items suffers from serious weaknesses, e.g. zero revenues or unfair prices, which results in a poor performance. Due to the high degree of complexity, in this thesis, we investigate the package pricing when an InP bids for the whole VN network $C_n(G)$, rather than for partial biddings $C_n(P_l)$. For instance, as shown in Figure 4.5 example, we will just consider bids for the entire network $b_n(ABCD)$ and not the other possible combinations, such as $b_n(AB)$ or $b_n(AB) + b_n(ABC)$. Hence, in this case the cost $C_n(G)$ will be treated exactly as in the per virtual node auction, represented as:

$$C_n(G) = \begin{cases} \min b_j(G), & b_n(G) < b_j(G) \leq u_G : \ \forall j \neq n, \ \exists b_j(G) \neq \infty \\ b_n(G), & b_n(G) < b_j(G) \leq u_G : \ \forall j \neq n, \ \nexists b_j(G) \neq \infty \\ \infty, & \text{otherwise} \end{cases} \tag{4.2}$$

Finally, an algorithm that combines individual bidding and package pricing, and calculates the minimum VN cost $Cm(G)$, will be designed. Firstly, in case no bid per package is provided, the cost for the VN network can be generally expressed as $C^*(G) = \sum_{i=1}^{k} C_n(N_i)$, which is the sum of all minimum virtual nodes costs. However, when the two bidding types are used, the minimum cost will be calculated as follows 4.3.

$$C^*(G) = \min\{C_n(G), \sum_{i=1}^{k} C_n(N_i)\} \tag{4.3}$$

The last expression shows that the system first computes independently an individual and a package auction. Afterwards, the resulting costs for these two tenders are compared, whose minimum value gives us the final VNE cost $C^*(G)$. In case $C_n(G) = \sum_{i=1}^{k} C_n(N_i)$, the virtual network will be embedded to the InP offering package pricing, since the performance is always better within a single InP. Moreover, the common InPs bidding strategies and the encountered weaknesses of the Vickrey model for package pricing, need to be investigated in this design. Algorithm 1 shows the pseudocode of the InP n resource matching and subsequently bidding strategy, where N_i^t corresponds to the virtual node type of N_i.

Algorithm 1: InP n resource matching and bidding strategy

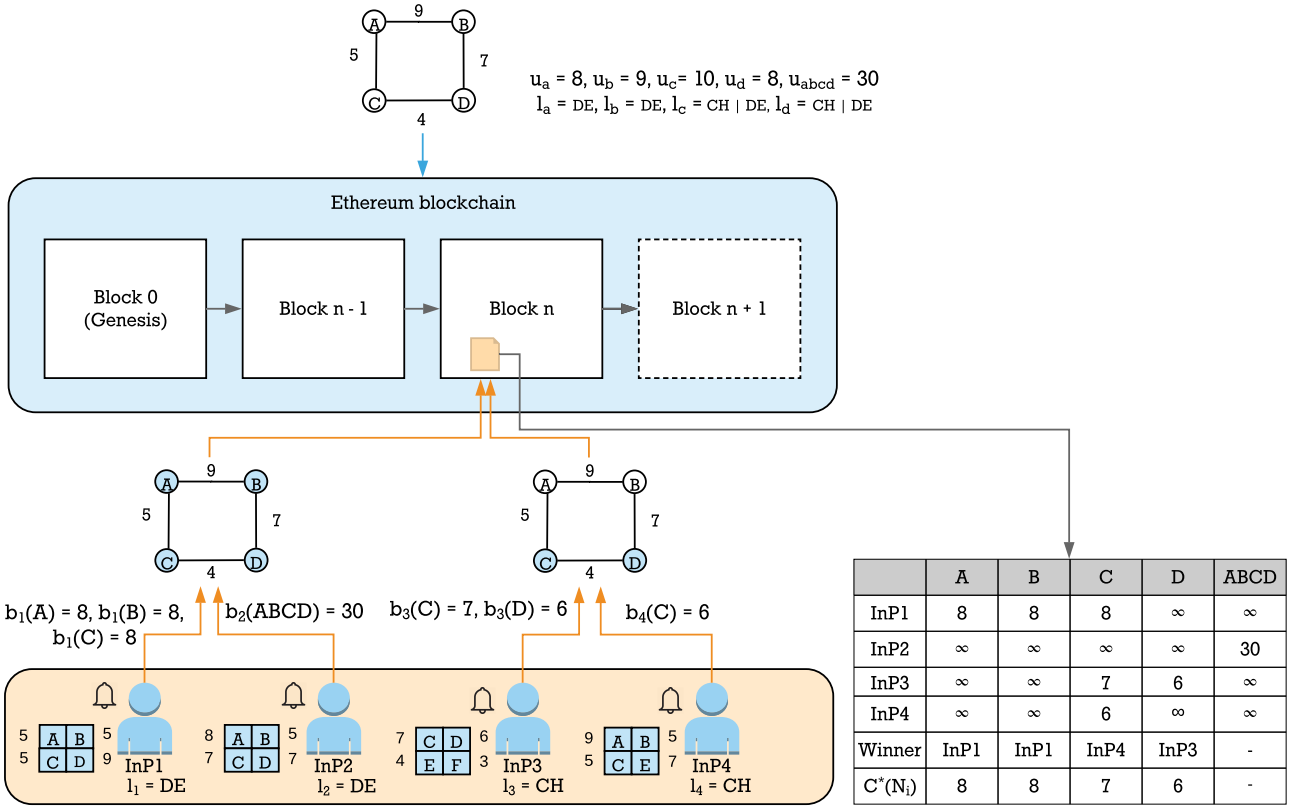
```
1: if N_s ← ∀l_i ∧ ∀N_i^t then
2:     return b_n(G) = υ_n(G) × [1 + γ_n(G)] ≤ u_G :  υ_n(G) = Σ_{i=1}^{k} υ_n(N_i), γ_n(G) ∈ [0,1]
3: else
4:     for N_i ∈ N^R do
5:         if N_s ← l_i ∧ N_i^t then
6:             b_n(N_i) = υ_n(N_i) × [1 + γ_n(N_i)] ≤ u_{N_i} :  γ_n(N_i) ∈ [0,1]
7:     return b_n(N^R)
```

Firstly, according to Algorithm 1, an InP bids either for the entire virtual network $b_n(G)$ or for individual virtual nodes $b_n(N_i)$ depending on the amount of paired resources. In other words, if all the virtual nodes locations l_i and types N_i^t are matched with certain InP substrate nodes N_s, then the system enables that the infrastructure provider bids for the entire virtual network. If not, an InP is only allowed to quote prices for the paired virtual nodes.

To better explain the bidding strategy of Algorithm 1, it is worth taking a look at the example from Figure 4.5. There, four InPs are competing to win virtual nodes $\{A, B, C, D\}$, each using profit margin values of $\{\gamma_1 = 0.6, \gamma_2 = 0.\widehat{1}, \gamma_3 = 0, \gamma_4 = 0.2\}$. In this example, the profit margins are identical for each N_i and G. At the end of this auction, since the individual quotes are lower than the package pricing offer $C^*(G) = min\{30, 29\}$, the virtual network is partitioned across different InPs, i.e. $A \rightarrow InP_1, B \rightarrow InP_1, C \rightarrow InP_4, D \rightarrow InP_3$. In addition, the prices that the SP must pay to each InP are the ones resulting from the *low-bidding second-price auction* $C_n(N_i)$.

In the above-mentioned example, is important to note that although the InP_1 applies the highest profit margin, it is the one with the best outcome. Thus, since the virtual node and virtual network bids are determined first by $\upsilon_n(N_i)$, which is also used in $\upsilon_n(G) = \sum_{i=1}^{k} \upsilon_n(N_i)$, and second by $\gamma_n(N_i)$ and $\gamma_n(G)$, these values need to be examined:

- **Real cost $\mathbf{v_n(N_i)}$**: Since network virtualization is a real-time process where the demand and availability of the resources are constantly changing, our approach encourages that users apply **dynamic pricing models**. A dynamic pricing model, in contrast to fixed pricing, is a strategy where the prices are flexibly adapted to the current market demands, rather than always charging the same costs. In our approach, InPs could use algorithms that take into account the SP demands and the current



	A	B	C	D	ABCD
InP1	8	8	8	∞	∞
InP2	∞	∞	∞	∞	30
InP3	∞	∞	7	6	∞
InP4	∞	∞	6	∞	∞
Winner	InP1	InP1	InP4	InP3	-
$C^*(N_i)$	8	8	7	6	-

Figure 4.5: Our designed auction algorithm that combines the Vickrey model with package pricing.

availability of the resources, to better define their product prices, and hence, to have a better market positioning or to open up new markets. For instance, they could use the demand and supply principle in microeconomics, which states that if the demand for a product short in supply is high, the price increases. In our scenario, this demand is related either with the computing or the bandwidth requirements, and the supply with the resource utilization. Thus, the real cost $\upsilon_n(N_i)$ is given by (4.4), where $\alpha_{s*}(N_i)$ and $\beta_{s*}(L_{ij})$ are the dynamic prices for the paired substrate node N_s^* and link L_s^*, and whose values will be updated according to the resource availability (see chapter 6). Apart from demand and supply, InPs could also use automated processes based on other factors, such as competitive pricing or customer behaviors. In the former, since the reserved prices for the virtual nodes are publicly known when an auction finishes, it allows InPs to learn from competitors in order to be more proficient in incoming auctions. In the latter, the virtual node upper bound costs entered by the SPs on their VN requests, can also be used to adjust the InP prices on customers needs. Overall, due to the high elasticity of the prices, InPs should recalculate periodically their costs, in order to compete in the modern market. This cost estimation can be performed based on many other factors, which can be thoroughly examined in future research.

$$\upsilon_n(N_i) = \alpha_{s*}(N_i) + \sum_{j=1}^{k} \frac{\beta_{s*}(L_{ij})}{2} \qquad \forall j \in N^R,\ \exists d_{ij} \neq \infty :\ i \neq j \tag{4.4}$$

- **Profit margins $\gamma_n(N_i)$ and $\gamma_n(G)$:** The profit margin is a measure to indicate the profitability of a product, which is extracted from the net profit divided by the cost. Equations (4.5) and (4.6), give the values of the profit margins for a virtual node and for the entire network. It is important to note that these values are provisional (at the time of bidding), since if a second lowest quote exists, as it is a *second-price auction*, the values $b_n(N_i)$ and $b_n(G)$ will be replaced by $C_n(N_i)$ and $C_n(G)$ respectively.

Typically, a high-profit margin produces a higher revenue, although at a smaller probability of winning. Nevertheless, from the example in Figure 4.5, we can observe that if an InP performs a good market research (most probably InP_1), it can also obtain virtual nodes through high-profit margins. Thus, in chapter 6, we will examine how the risk taken by the InPs at the time of determining the profit margins, affects the VN embedding outcome.

$$\gamma_n(N_i) = \frac{b_n(N_i) - \upsilon_n(N_i)}{\upsilon_n(N_i)} \qquad \gamma_n(N_i) \in [0,1] \tag{4.5}$$

$$\gamma_n(G) = \frac{b_n(G) - \upsilon_n(G)}{\upsilon_n(G)} \qquad \gamma_n(G) \in [0,1] \tag{4.6}$$

Lastly, we will analyse the weaknesses of the Vickrey auction model for package pricing in our design [AM+06], and observe how the potential attacks in Internet auctions could hinder our system performance [BM00]:

- **Bidder non-monotonicity:** Adding other bidders in a *Vickrey auction* with package pricing can result in low or zero revenues, such that bidders pay the opportunity cost. Hence, the system has potential vulnerabilities that the bidders can exploit. In our design, as the virtual network and the corresponding virtual nodes are treated as single independent items, adding more bidders will only increase the resource competition.
- **Multiple bidding identities:** In many Internet auctions, a single bidder can use multiple pseudonyms or accounts to bid and increase their profits. In our system, as users can only be registered once (through an invitation), it is not possible for an InP to have multiple Ethereum accounts to compromise the final outcome. Thus, this is a controlled system where InPs, such as Deutsche Telekom or Swisscom, are identified by a single account.
- **Bid shielding:** It consists in the withdraw of bids in the last minute. Since our application interacts with a blockchain, where submitted data is immutable, this has no influence on the system.
- **Bid sniping:** The auctions started by the SPs have a fixed duration, after which no further bids are accepted. In this case, an InP could enforce a denial of service attack, by sending multiple last-minute bids, in order to prevent that the system computes other bids and hence, winning the auction. For this reason, in our system, each InP will have to first pay his bid to the smart contract, and a percentage of this total quantity will be refunded after the sealed bid is opened.

4.3 Summary

Virtual network embeding (VNE) enables SPs to embed virtual nodes across different InPs to create customized virtual networks. Typically, these InPs are not willing to disclose detailed information about their resources availability or network topologies, which directly affects the VNE performance. In addition, the service providers seek to minimize VNE embedding costs, such that the VN partitioning is treated as a cost minimization problem.

In this thesis, an approach to improve the VNE throughput under the limited information disclosure scenario has been introduced. More precisely, a complete system using the Ethereum blockchain has been presented, which thanks to smart contracts, enables the creation of new embedding strategies to facilitate the service negotiation between the InPs and SPs. In our case, an algorithm adapted from the Vickrey auction model has been designed, which improves the VN partitioning efficiency. In the following chapters, the implementation of this design will be evaluated.

We do not claim that our approach is the only way of performing VN partitioning. However, it enables brokerless VN embedding in a distributed environment. Furthermore, to the best of our knowledge, this is the first approach using blockchain in the VNE context, and thanks to smart contract's flexibility, it can serve as a starting point for more upcoming investigations.

5 Implementation

In the following, the implementation of a brokerless VNE system under limited information disclosure is introduced. In chapter 4, after investigating the different types of blockchain and platforms, we have presented a complete design for the system architecture, which uses the Ethereum blockchain, and hence, smart contracts.

In this part, first, we define the design decisions and environment where our approach is implemented. Afterwards, we present the system architecture, describing how a hybrid blockchain with multiple nodes is configured in the Ethereum platform, and how the authentication system works.

In addition, we thoroughly explain the two developed contracts for our design: the Users and Vickrey contract. The first acts as a permanent database contract, with the only logic of updating its state with new user data. The second is an application contract used as a template for every auction that is started (multiple Vickrey contracts). In particular, the code is equal for all the created contracts, although with different states restricting the user interaction to the allowed ones. Thus, through these contracts, we represent the tailored Vickrey auction model, in order to improve the VN partitioning efficiency and minimize the costs between the InPs and SPs. Finally, we discuss the implementation workflow, that is, the interactions between the system components, along with the limitations observed in our implementation.

5.1 Design Decisions

In section 4.1.2, we have discussed the reasons why is logic to choose Ethereum as the main platform of our hybrid blockchain, among which, we highlight the maturity and the building simplicity of the project. Thanks to this, the blockchain concept is fastly understood and tested, resulting in a straightforward and intuitive scenario setup. In addition, since Ethereum is utilized by many users, on the Internet, a large number of starter tutorials can be found. Thus, the basic steps for becoming familiar with Ethereum are extremely facilitated, which help us on start writing our first smart contracts.

In the design system architecture (Figure 4.3), we have stated that for automation, each user efficiently interacts with the blockchain through an API. Therefore, it is important to first study the environment workflow, i.e. which technologies enable clients to create, read and update data from the blockchain.

The first aspect to consider is how to run the blockchain nodes that maintain the BC platform. These nodes are responsible for downloading the blockchain into the system on a regular basis, verifying and processing the incoming transactions and synching with the other nodes. However, there are multiple clients that run the Ethereum clients/nodes across different operating systems, such as Go-Ethereum [Fou14], Parity [Eth16] or Pyethapp [Fou15], written in Go, Rust and Python respectively. Despite all those packages are experimental, in our thesis, we use **Go-Ethereum (Geth)** since it is the official client software provided by the Ethereum Foundation (no third parties). Moreover, Geth provides a simple set up for multiple clients in private networks, see section 5.2.1 for more information.

After having configured the blockchain, the next step is to understand how to write, compile and store the smart contracts. Firstly, as outlined in the design chapter, there are mainly two programming languages for writing smart contracts: Solidity (JavaScript) and Serpent (Python). At the time of writing, Serpent has been deprecated in favor of Viper [Fou17], a new experimental Python programming language developed by the Ethereum Foundation. Despite Viper is approached to be simpler with more auditable and secure smart contracts, due to its low maturity, it currently does not have a large number of features, documentation and community support. Thus, as we seek for quick usability to start testing our scenario, the most widely supported, documented and stable language, is **Solidity**.

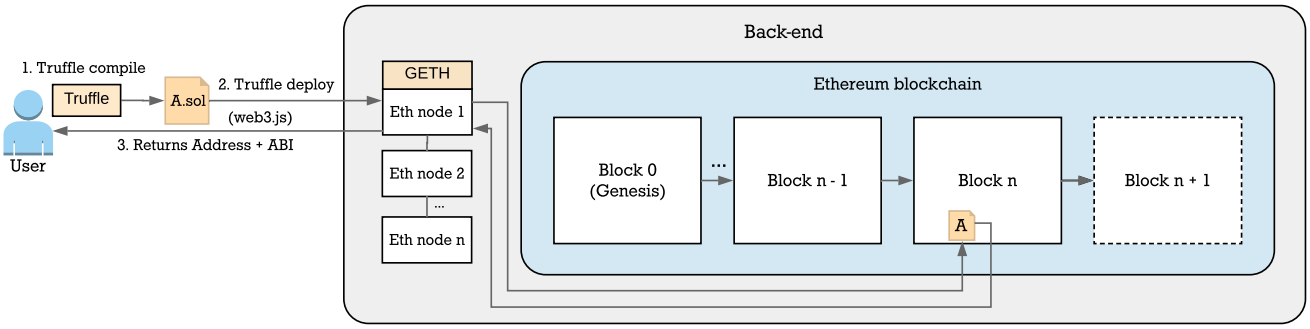


Figure 5.1: Smart contract compilation and deployment using the Truffle framework.

Besides that, Solidity can be integrated with great frameworks, such as Remix[1], which is a browser-based compiler and Integrated Development Environment (IDE) that enables the creation, debugging and testing of Solidity smart contracts. Nevertheless, since users need a friendly GUI to perform different actions, apart from interacting with smart contracts, we cannot use a browser IDE for our final system application. In our case, we will only utilize Remix for the testing and debugging of smart contracts.

Fortunately, there are several frameworks that simplify the complexities of compiling, deploying and testing smart contracts into the blockchain, as well as reaching and accessing them subsequently from the application. Among those, we want to highlight Truffle [MS15b] and Embark [IM16]. They are both comparable in terms of features and usage simplicity. However, Embark allows a quite easy development in front-end applications, although, in other systems (e.g. back-end server), the features integration is more complex. On the contrary, Truffle can also be placed in these systems, providing a straightforward project creation, contract deployment and interaction on private networks, with plenty of detailed documentation on the Internet. Therefore, as our application may grow over time with new functionalities, **Truffle** seems to be the most appropriate solution.

To better explain the workflow on deploying smart contracts, Figure 5.1 reflects an example of a user uploading one smart contract through the Ethereum node. The process comprehends the following steps: Firstly, after writing the smart contract, the user compiles it. Consecutively, if no error appears, the user deploys the contract into the blockchain. Finally, once placed into the last block, the Ethereum node returns the address of the created contract together with the application binary interface (ABI) in JSON format. From now on, the user can request and modify the state of the contract using the corresponding ABI and address through web3.js. **Web3.js** is a JavaScript library that enables the interaction with local or remote Ethereum node, using an HTTP or IPC connection. In addition, due to the changes in deployment as the project evolves, truffle also enables defining migrations, which are JavaScript files that help users to deploy the desired contracts into the network.

At this point, we have already compared and selected the technologies used for the deployment of our Solidity smart contracts into the Geth blockchain. Nevertheless, the definition of another fundamental part of our system is still required: the front-end. Since we do not observe any advantage on users having the presented application on mobile phones, we opted for developing a traditional web application rather than a mobile application. In this way, users can easily interact with the blockchain by accessing the application on their browsers.

Thus, the first and main requirement for the front-end is that the programming language is accepted and supported by most of the browsers without the need of compilers or plugins. In web development, JavaScript is the most used, and hence, better positioned in this respect. Nowadays, there are many JavaScript frameworks extending the HTML and CSS capabilities, such as AngularJS, ReactJS or VueJS. Due to the consistency, efficient productivity, simplicity, and maintainability, among others, we have chosen **AngularJS** as the default language for our web application. AngularJS is a popular JavaScript-based

[1] `https://remix.ethereum.org/`

open-source framework released by Google, with a large number of features and a built-in functionality for creating responsive and reactive web applications.

Lastly, we need a computer software that connects all the defined components, handling and transmitting the requests to the blockchain nodes: a **middleware (MW)**. Thanks to the middleware, the Geth nodes will be configured to only accept requests initiated from the domain of this component, which will prevent the system suffering from many security issues. Thus, Truffle will be placed on the middleware and the web application will use it as a gateway for the API management. This middleware could also check the user's identity (authentication system), in order to ensure that only the allowed peers interact with the Ethereum nodes. Regarding the programming language, as our environment has been mostly developed in JavaScript, we employ **Node.js**, which is an open-source JavaScript environment that runs on the server side.

Apart from the reasons mentioned above, a MW enables the integration and correlation of different data sources (blockchain and non-blockchain events) in real time, such as blockchain clients, databases, partner APIs or a publish/pubscribe system. Later on, this data could be used for the InPs maintaining the application, to find insights and patterns in historical data, i.e to apply machine learning. Therefore, a middleware is a key success factor for managing a blockchain project, and later to integrate it with existing services and platforms.

5.2 Architecture

After having discussed the reasons for choosing certain technologies and software components, in Figure 5.2, we present an overview of our system architecture. As most of the web applications (webapp), the structure consists of two main components: the front-end and the back-end. The first can be run in a traditional centralized web server (e.g. AWS EC2 instance) or in a more distributed environment, such as the InterPlanetary File System (IPFS) [Ben14], which enables the hosting of decentralized applications (dapps) on the Internet. In particular, the front-end is formed by a webapp based on Angular 2, a more sophisticated version of Angular.JS, responsible for displaying, the functionalities used for the clients (private group) to interact with the application, on the browser. In addition, the front-end contains another fundamental element for the system functionality, a middleware, which can also be placed on the server, although in a different port. As explained in the section above, in the middleware deployed in Node.js, we also set up the Truffle framework, in order to manage the interaction with the Ethereum nodes, improve the system's security and integrate the dapp with other services further on.

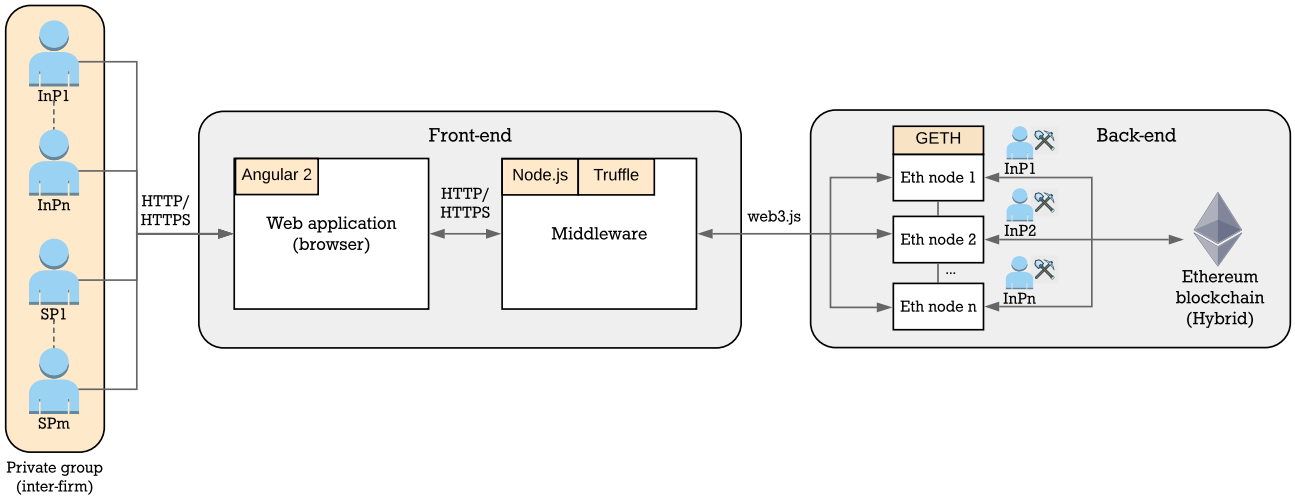


Figure 5.2: Architecture overview of the technologies and software components used for our system.

On the other hand, in Figure 5.2, we have placed together in the Back-end server, the different Geth nodes and the Ethereum blockchain. However, it is important to note that in our scenario, the different nodes are maintained and served by different firms, and most probably, domains. For instance, company A and B are responsible for controlling nodes 1 and 2, which are placed on http://www.companyA.com:3005 and http://www.companyB.com:3005 respectively. Another point to bear in mind is that every new Ethereum node must be subsequently added to the web3 configuration of the MW. If not, the system will become more centralized with the transactions always sent to the same node, e.g. Ethereum node 1, which can be prone to DoS attacks. Nevertheless, the middleware has no need to be immediately modified, because although a node does not receive directly transactions, these are added to a shared pool by all the Ethereum nodes, from where they are subsequently processed.

Thus, in the following, we describe how to set up a hybrid blockchain where all the miners are synchronized and work coordinated.

5.2.1 Hybrid Blockchain with Multiple Mining Nodes

In the first place, to initialise a private blockchain separated from the public Ethereum chain, there are three components that need need to be customized:

1. **Data Directory:** It is where the chain data of our blockchain is stored. In our case, each firm has a specific directory in the machine running the mining node, where all this information is conserved.
2. **Network ID:** It is a scalar value that identifies the customized network, which must be random but different than other predefined networks, such as 1 or 4 used for the Ethereum public main network or the public Geth Ethereum testnet respectively. In our environment, this value needs to be known for all the mining nodes.
3. **Genesis Block:** The Genesis block, also called block 0, is the start of the Blockchain. It is hardcoded into the client in a file called genesis.json, whose content must be identical for all the mining nodes to ensure that they work on the same blockchain. Since it is written in JSON format, the specifications for the genesis block are written in key/value pairs. Despite most of these parameters are later on arbitrary, they need to be first defined for the genesis block [Gen16]: (i) *nonce*: Recap from 2.1.1 that the nonce is a 64-bit hash, which proves that a sufficient amount of work has been performed on creating the next block header. (ii) *mixhash*: It is an intermediary hash calculation (256-bit) for finding the nonce and used to determine if an attacker mines blocks with incorrect nonces, which has a lower computational cost. (iii) *timestamp*: A scalar value equal to the current Unix time to identify when a block is mined. The values introduced until now are meaningless in the genesis block, since they will be later on arbitrary or specific to the current time (timestamp). (iv) *difficulty*: An integer that defines the increase or decrease of the mining block complexity depending on the timestamp differences between the current and the parent block. (v) *parentHash*: The keccak 256-bit hash of the parent block header. This is meaningless in the genesis block, since block 0 has no parent. (vi) *extraData*: An optional value with a maximum size of 32-bytes. (vii) *alloc*: It is a list of accounts with pre-allocated Ether. (viii) *coinbase*: A 160-bit address to which all the mining rewards are sent. In our case, as the mining rewards are divided between the different nodes this value is not set. (ix) *gasLimit*: A value equal to the maximum gas for transaction.

In Listing 5.1, we present the genesis file used for our implementation. There, we define the *difficulty* to be equal to 0x800, which is the minimum value for the Ethereum PoW (explained later in section 6.3.2). Note that in PoW, a miner needs to find a valid block header, whose cryptographic hash is less than $\frac{2^{224}}{d}$, where d is the block difficulty (Equation 2.1). Thus, a small difficulty gives us a higher value for this term, which results in a higher probability of finding the block header. In chapter 6, we will further examine how this parameter influences the block generation time, and which is the most suitable value for a private network.

```
{
    "nonce": "0x0000000000000042",
    "timestamp": "0x0",
    "parentHash": "0x0000000000000000000000000000000000000000000000000000000000000000",
    "extraData": "0x00",
    "gasLimit": "0x8000000",
    "difficulty": "0x800",
    "mixhash": "0x0000000000000000000000000000000000000000000000000000000000000000",
    "alloc": {
    },
    "config": {
        "chainId": 987654321,
        "homesteadBlock": 0,
        "eip155Block": 0,
        "eip158Block": 0
    }

}
```

Listing 5.1: Genesis file used for our implementation.

Apart from the already introduced fields, in our genesis file, we have introduced a *config* parameter, which specifies the configuration variables for the Ethereum network, and has no relation with block 0. In particular, *chainId* is the chain to work, typically a random number, and it is totally different from the network ID. For instance, a peer can use the network ID 58342 but later mine in chain 1. When setting the *homeSteadBlock* field to 0, means that the chain is using the Homestead release of Ethereum, which is the same of the main Ethereum network. Finally, the EIP parameters stand for Ethereum Improvement Proposals, from which we use the *eip155Block* and *eip155Block* that are used to prevent replay attacks and modify the way to deal with empty accounts (less space in BC) respectively.

After having defined the genesis file and in order to start running the blockchain, one Ethereum node must build it. An example of the two commands needed are illustrated below:

```
$ geth --datadir <path-to-directory> init genesis.json
$ geth --port 3000 --networkid 58342 --datadir="<path-to-directory>" --maxpeers=3 --rpc --
    rpcport 8545 --rpcaddr 127.0.0.1 --rpccorsdomain "http://localhost:8000" --rpcapi "eth
    ,net,web3,personal" --gasprice "0" --nodiscover console
```

Listing 5.2: Example of the two Geth commands used to start running the blockchain.

The first command specifies the data directory where the blockchain data is initialized and stored. The second starts running the blockchain with the specified parameters. In this example, the blockchain is configured to run locally in port 3000 and with network ID 58342. In addition, a gas price equal to 0, means that the transactions in the blockchain, if not explicitly specified, have not cost. Since we are building a private blockchain with different users, it is not logical to charge those when they are performing operations. For instance, when a user registers to the application or when a SP begins a new auction. Also, adding the field –nodiscover disables the peer discovery mechanism, i.e. nodes are only connected by manual addition.

After executing these commands for each of the Ethereum clients, which should have the same genesis file and network ID but different domains, ports and data directories, these nodes need to be synchronized. To achieve this, Geth supports a feature called static nodes, which can be configured into <path-to-directory>/static-nodes.json. These static nodes are added through the so-called **Enodes**, which are Ethereum nodes written in a URL scheme.

An example of the static-nodes file in a blockchain client, which adds two other Enodes, has the following format:

```
[
"enode://38
    dd47f0ed04ec88ede2f75970da08f23ca87de8f96bf576dc769b0f9df807782d3729bc83b99042a3eb193ff
     091cb4715a0906e5e902fb5adc04af91f3a7ba7@[::]:3010?discport=0",

"enode://
    dcd4a6714af990cc036dcb3b46c8ac6f929d7b9c1119ba3bdf8ffde9252fa17204bac84c3afae9c53b38c4a
     0dacccd2c52ac832ec723c5ff5e29546402c09737@[::]:3015?discport=0"
]
```

Listing 5.3: Example of a static-nodes.json file that adds two other blockchain nodes.

Afterwards, once all the nodes are connected and have synchronized the blockchain status, they can start to mine all the incoming requests. The distribution of the mining privileges between these nodes will be later introduced and evaluated in chapter 6, where a JavaScript file is preloaded on each of this clients.

5.2.2 Authentication system

In section 4.2.4, we have presented the conceptual design of how a user is registered and authenticated in our system. More specifically, the first is performed through a user invitation and the second providing a digital signature to validate the origin and integrity of the messages. In the following, we introduce the implementation of such a system in our architecture, in which the web application sends the information to the middleware, who uses web3 to interact with the Ethereum nodes.

Firstly, a consortium of InPs invites a user to join the network, e.g. a service provider, sharing a link that allows the creation of an Externally Owned Account (EOA) by only entering a password. After that, the user receives the created address, which can only be unlocked by entering this password. In Geth, an account needs to be unlocked in order to send transactions from it to the blockchain. Therefore, for not entering the password each time a user sends data to the BC, we create an authentication system in which when a user logs in, the corresponding account is directly unlocked for the desired time.

Nevertheless, before a user is logged in, he needs to register to the application entering the following fields: (i) an email, (ii) the previously created Ethereum address and (iii) the user's role, to distinguish between InPs and SPs, to later provide different application functionalities. Note that the Ethereum address is extremely important, since if a user forgets this address, all the information regarding the account, e.g. ETH earned, will be lost.

After being successfully registered to the application, now the user can sign in, and hence, utilize the application. This process employs the digital signature concept, although it slightly varies depending on the technology used (e.g. Parity or TestRPC). In Figure 5.3, we illustrate how this mechanism works, where SPx authenticates to the system by entering on the browser the Ethereum address and the password to unlock this account.

The cycle starts with hashing a randomly generated data that is later signed with the unlocked address, to generate a signature (web3.eth.sign(address, hash value)). This signature is sent via HTTP together with the hash value and the address to the middleware. Then, the authentication is performed, which consists on recovering the Ethereum address from the signature and a specific hash of the data. If the obtained address matches the one provided on the HTTP request, or in other words, if the user is proved to be the owner, since he is the only that knows the password to unlock its account to sign the message, a JSON web token is generated. In particular, a self-contained token is provided, which as the name suggests, includes user information (address). Therefore, from now on, the user will attach this token on the HTTP header of every request to the blockchain. This token will be validated by the middleware, who decrypts it through the secret key already used to encrypt (symmetric encryption). In addition, once a user performs a request, the token is updated with a new expiration time equal to one hour.

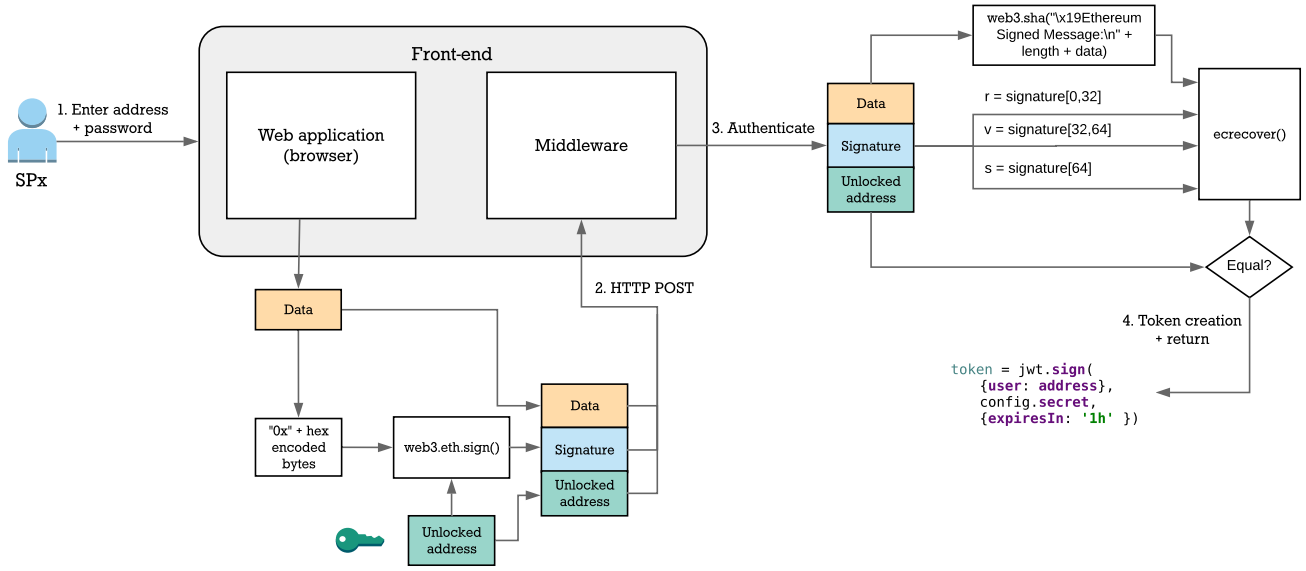


Figure 5.3: Overview of the implemented authentication system.

To summarize, thanks to the middleware and the web tokens, in the authentication process, the Ethereum clients are only responsible for creating the Ethereum accounts and subsequently unlock them. All the other actions are performed between the web app and the middleware, which apart from preventing that users spend time on entering the password for each transaction, it ensures that the Ethereum clients, and thus, the smart contracts are only efficiently used for specific tasks. For instance, performing the signature verification in a contract requires a higher computational effort, which can lead to a system overload if executed frequently.

5.2.3 Users Contract

In the design of the system architecture, see Figure 4.3, we have defined a permanently requestable smart contract named *Users*, whose responsibility is to store the participant's information, such as the entered InP substrate networks (peering nodes and links), used to match different virtual node types in specific locations.

Given the importance of this contract, on the next pages, we present the fundamental data structures and functions that describe its functionality. To understand the code, it is assumed that the reader has a basic knowledge of JavaScript or even better Solidity.

To begin with the contract, we have defined the structs *User* and *UserAuction*, see Listing 5.4. The first stores the email and role of the registered user and it is accessed by giving its Ethereum address (*users* mapping). The second provides all the auctions (open and close) of a determined user by entering again the address as the key of the *userAuctions* mapping. If the endTime is smaller than the current Unix time(), no more bids can be placed to the contract.

This last struct is accessed by the function *getUserAuctions()*, which gets all the auctions of a user. Since the auctions are multiple, in this function we define three dynamic arrays on the memory, which are filled with the corresponding auctions once entering the loop. Despite it is not illustrated, once a service provider starts a new auction, the application appends the address of this new auction to the *UserAuction* struct of the InPs with matched resources. Thus, once an infrastructure provider enters the application, the *GetUserAuctions()* is called, which exclusively returns the auctions where this InP is allowed to bid. This behavior easily accomplishes the pull-based notifications required, retrieving the new auctions address, and subsequently the specific information, in real-time. In addition, the *Vickrey contract* also stores these InPs, to prevent the bidding of other users (see section 5.2.4).

```
struct User {
  bytes32 email;
  bytes32 role;
}

struct UserAuction {
  address auctionAddr;
  bool isOpened;
  uint256 endTime;
}

mapping(address => User) private users;
mapping(address => UserAuction[]) private userAuctions;
address[] public userAddresses;

// Method that gets the auctions addresses specific to the user requesting it
function getUserAuctions() public constant returns (address[], bool[], uint256[]) {
  var auctions = userAuctions[msg.sender];
  address[] memory auctionAddrs = new address[](auctions.length);
  bool[] memory isOpened = new bool[](auctions.length);
  uint256[] memory endTime = new uint256[](auctions.length);
  for (uint i = 0; i < auctions.length; i++) {
    auctionAddrs[i] = auctions[i].auctionAddr;
    isOpened[i] = auctions[i].isOpened;
    endTime[i] = auctions[i].endTime;
  }
  return (auctionAddrs, isOpened, endTime);
}
```

Listing 5.4: Data structures and functions related to the user data in the Users contract.

Moreover, the *Users* contract includes the *PeeringNode* and *Link* structs, which combined form the InPs substrate networks, see Listing 5.5. The first includes the different resources types in a specific peering node location, such as $\{A, B, C, D\}$ in Germany. The second includes the *from* and *to* fields corresponding to the ids of the peering nodes connected by this link.

```
struct PeeringNode {
  bytes32 idPeeringNode;
  bytes32 location;
  bytes32[] resourceTypes;
}

struct Link {
  bytes32 idLink;
  bytes32 from;
  bytes32 to;
}

mapping(address => PeeringNode[]) private peeringNodes;
mapping(bytes32 => Link[]) private links;
```

Listing 5.5: Data structures related to the peering nodes and links in the Users contract.

Furthermore, in Listing 5.6, we find the *getMatchedVirtualNodes* function, which returns the id of the virtual nodes that were matched with an InP data. For instance, if InP1 has a peering node in Germany with resource types $\{A, B, C, D\}$, and a VNR contains $\{A, B, C\}$ in $\{DE, DE, CH\}$ respectively, only the virtual nodes A and B are return. The aim is to use this data, to restrict the InPs to only bid for the paired virtual nodes, on the subsequently created *Vickrey* contract.

Finally, it is also important to stress the modifier *isUserInP*, which is placed in determined functions, such as to add peering nodes (not shown in code), in order to prevent that SPs perform operations not related to their roles.

```
1  // Modifier that when placed in a function, restrict the call to only the InPs
2  modifier isUserInP(address sender) {
3    require(users[sender].role == "InP"); _;
4  }
5  // Method that returns the virtual nodes matched with the substrate networks
6  function getMatchedVirtualNodes(address InP, bytes32[] idVirtualNodes, bytes32[]
       locations, bytes32[] resourceTypes) public constant returns (bytes32[]) {
7    bytes32[] memory virtualNodes = new bytes32[](locations.length);
8    for (uint i = 0; i < locations.length; i++) {
9      for (uint j = 0; j < peeringNodes[InP].length; j++) {
10       if (locations[i] == peeringNodes[InP][j].location) {
11         var peeringNode = peeringNodes[InP][j];
12         for (uint k = 0; k < peeringNode.resourceTypes.length; k ++) {
13           if (resourceTypes[i] == peeringNode.resourceTypes[k]) {
14             virtualNodes[i] = idVirtualNodes[i];
15           }
16         }
17       }
18     }
19   }
20   return virtualNodes;
21 }
```

Listing 5.6: Function to get the paired virtual nodes in the Users contract.

5.2.4 Vickrey Contract

In the following, we present the most relevant data of the *Vickrey* contract. To create a contract that initiates the auction corresponding to the virtual network request, the ABI of the *Vickrey* contract is loaded each time a service provider requests a VN. Thus, in contrast to the single *Users* contract, in the blockchain, there are multiple *Vickrey* contracts, each having the same data structures and functions, although with different states. In addition, before its creation, the middleware always requests the *Users* contract, to add to this new contract the data that restricts the bidding to only the paired InPs.

Firstly, when the contract is created (Listing 5.7), the constructor function *Vickrey* is called, which stores the SP owning the contract and the auction end time, and adds the virtual nodes and links of the VNR in the first structs (functions *addVirtualNodes* and *addVirtualLinks* not shown in code).

```
1  address public owner; //SP address
2  uint256 public endTime;
3
4  // Constructor method called when contract is created
5  function Vickrey(bytes32[] idVirtualNodes, bytes32[] resourceTypes, bytes32[] locations,
       uint8[] upperBoundCosts, uint8[] cDemands, bytes32[] idLinks, bytes32[] from,
     bytes32[] to, uint8[] dBandwidth, uint256 time) public {
6    owner = msg.sender;
7    endTime = time;
8    addVirtualNodes(idVirtualNodes, resourceTypes, locations, cDemands, upperBoundCosts);
9    addVirtualLinks(idLinks, from, to, dBandwidth);
10 }
```

Listing 5.7: Data structures and functions called when the Vickrey contract is created

Subsequently, the middleware adds the allowed InPs per virtual node and the ones enabled to bid for the whole virtual network, functions *addInPsPerVNAllowed* and *addInPsPackageAllowed* respectively. Specifically, these methods change the state of the mappings *allowedInPsPerVirtualNode* and *isInPPackageAllowed*, which prevent the bidding of non-allowed InPs.

Afterwards, once an InP bids for a virtual node, the function *isInPAllowed* is called, which search the InP address inside the addresses given from the mapping *allowedInPsPerVirtualNode*.

```
1  mapping(bytes32 => address[]) public allowedInPsPerVirtualNode;
2  mapping(address => bool) public isInPPackageAllowed;
3
4  // Add InPs allowed to bid per virtual node
5  function addInPsPerVNAllowed(address[] allowedInPs, bytes32[] idVirtualNodes) private {
6    for(uint i = 0; i < idVirtualNodes.length; i++) {
7      allowedInPsPerVirtualNode[idVirtualNodes[i]].push(allowedInPs[i]);
8    }
9  }
10
11 // Add InPs allowed to bid for the whole virtual network
12 function addInPsPackageAllowed(address[] packageAllowedInPs) private {
13   for(uint i = 0; i < packageAllowedInPs.length; i++) {
14     isInPPackageAllowed[packageAllowedInPs[i]] = true;
15   }
16 }
17
18 // Method that checks if an InP is allowed to bid on the passed virtual node.
19 function isInPAllowed(address InP, bytes32 idVirtualNode) public constant returns (bool)
       {
20   for(uint i = 0; i < allowedInPsPerVirtualNode[idVirtualNode].length; i++) {
21     if (allowedInPsPerVirtualNode[idVirtualNode][i] == InP) {
22       return true;
23     }
24   }
25   return false;
26 }
```

Listing 5.8: Data structures and functions called to prevent the bidding of non-allowed InPs.

Therefore, from now on, the paired InPs are allowed to bid for the corresponding virtual nodes or for the whole virtual network (if enabled) through the web application. These InPs call either the function *commitIndividualBids* or *commitPackageBid*, see Listing 5.9, which can only be executed if the current time is lower than the *endTime*. In addition, note that to restrict that an InP bids multiple times for the same resources, the mapping *hasBidded* is defined, which returns true or false whether the passed InP has bidded or not.

Lastly, once the auction finishes, i.e. endTime < now, the SP needs to close the contract. In particular, he sends a request to the middleware, which gets all the bids from the contract and performs the Vickrey algorithm introduced in section 4.2.5. Afterwards, the resulting winners and reserved prices are added to the contract and the boolean field *isOpened* from the struct *UserAuction* is set to false (Listing 5.4).

5.3 Implementation Workflow

After having introduced the two fundamental contracts of our scheme, we can now fully comprehend the interaction between the system components during the VN partitioning cycle.

The process begins with the InPs entering their peering nodes (location and resource types handled) and the peering links to the *Users* contract. This information is later used to match the VNR to the entered substrate networks, allowing the bidding only to the InPs with paired resources.

```
  struct Bid {
    address InP;
    uint8 value;
  }

  mapping(bytes32 => Bid[]) private bids; //per virtual node
  mapping(address => uint8) private packageBid;
  mapping(address => bool) private hasBidded;
  modifier onlyBefore() { require(now < endTime); _; }

  // InP commits individual bids
  function commitIndividualBids(bytes32[] idVirtualNodes, uint8[] values) onlyBefore() {
   if (!hasBidded[msg.sender]) {
      for(uint i = 0; i < idVirtualNodes.length; i++) {
        if (isInPPackageAllowed[msg.sender]||isInPAllowed(msg.sender,idVirtualNodes[i])) {
          bids[idVirtualNodes[i]].push(Bid({InP: msg.sender, value: values[i]}));
          hasBidded[msg.sender] = true;
        }
      }
    }
  }

  // InP commits package Bid
  function commitPackageBid(uint8 value) public onlyBefore() {
    if (!hasBidded[msg.sender] && isInPPackageAllowed[msg.sender]) {
      packageBid[msg.sender] = value;
      hasBidded[msg.sender] = true;
      }
  }
}
```

Listing 5.9: Data structures and functions called for the individual and package bids of the Vickrey contract.

Let us consider the whole workflow since the SP request a virtual network, see Figure 5.4, based on the example presented in Figure 4.5. Despite it is not illustrated, it is understood that the requests are sent from the web application to the middleware.

Once the SP requests the virtual network, the middleware first calls the function *getMatchedVirtualNodes* for each registered InP. Based on this information, the corresponding *Vickrey* contract is created, which returns its address (V in the example). Subsequently, this address is added to the *UserAuction* struct from the *Users* contract. Since the constant function *getUsersAuctions* retrieves the auctions each time a user enters the application, the matched InPs will be further notified of the newly created contract V (pull-based notification).

Consequently, the InPs enter the individual or package bids to the application, which execute the functions *commitIndividualBids* or *commitPackageBid* respectively. Once the auction has expired, the SP is enabled to finalize it by applying the VN partitioning algorithm defined in section 4.2.5. In particular, the middleware retrieves all the entered bids per virtual node and for the whole virtual network, and compares these values to calculate the final cost given by Equation 4.3. If $C_n(G) \geq \sum_{i=1}^{k} C_n(N_i)$, the *addPackageWinners* function is called. On the contrary, if $C_n(G) < \sum_{i=1}^{k} C_n(N_i)$, the corresponding winners and reserved prices are entered through the function *addIndividualWinners*. In case no solution exists, i.e. $C_n(G) = \infty$ and $\exists C_n(N_i) = \infty; \forall N_i \in N^R$, the service provider is directly informed.

Finally, in all above cases, the auction in the *Users* contract is closed. Therefore, once entering the application, the InPs will observe that the auction has been finished, and if they access it, the corresponding winners and reserved prices will be shown. In addition, as stated during the thesis, this information could be further used for each InP to study the market fluctuations, and hence, to better define their prices for the incoming virtual network requests.

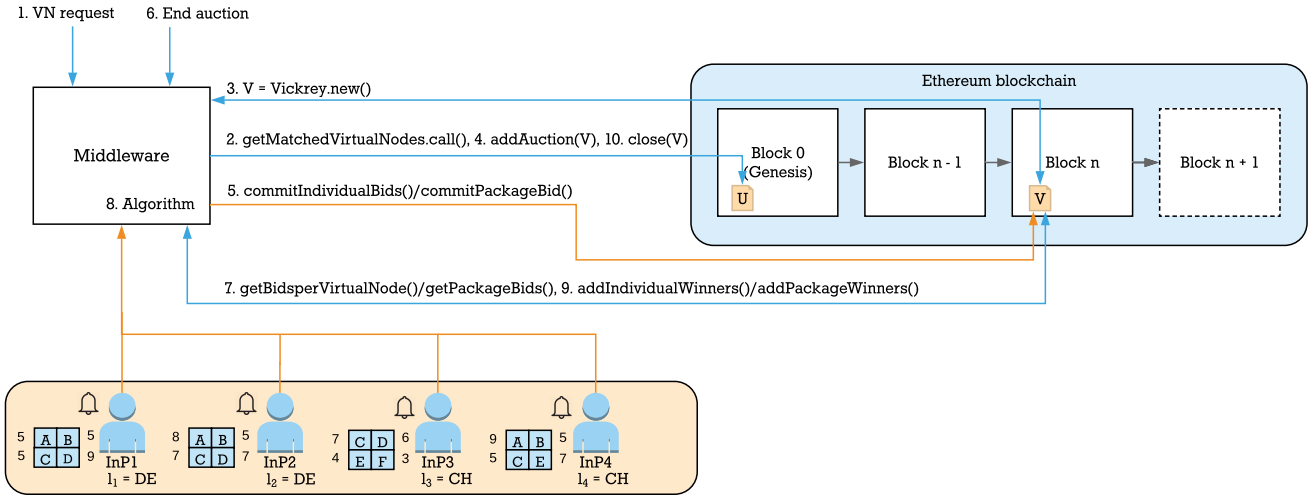


Figure 5.4: Virtual network request workflow and system components interaction

5.4 Summary

In this chapter, the implementation of a brokerless VNE system under limited information disclosure is presented. First, we have defined the design decisions and environment selected for our approach, which consists of a web application connected to an Ethereum blockchain through a middleware. Since this backend utilizes smart contracts, we have further detailed how to manage these on the blockchain, and which is the content of the two fundamental contracts (*Users* and *Vickrey*) that enable the defined VN partitioning algorithm integration. In addition, to improve the system's security by excluding smart contracts to directly interact with the external world (users), an authentication system has been developed. This provides compact and self-contained JSON tokens to ensure the data transmission between the users and the blockchain in a secure manner. Lastly, the interaction between our system components along the virtual network partitioning process has been specified.

Nevertheless, since the defined system is experimental and still in its early phases, several constraints and improvements are already contemplated. Regarding the smart contract security, as any other programming code, it needs to be constantly tested to highlight potential security vulnerabilities and optimized to efficiently achieve the desired aims at lower costs (CPU, memory...). Apart from that, the *Users* and *Vickrey* contracts need to be consistent with the current market demands, and as a consequence, they most probably need to be redefined with new algorithms and protocols. Therefore, it is vital that the authorities responsible for maintaining the system fully understand the system set up, although its complexity. In addition, since currently, the Ether values have not a significant impact on the application, such that there are not high benefits for users on mining, it is probable that only a small group users end up ensuring the system functionality. Thus, incorporating incentives can prevent that our system suffers from scalability. Finally, due to the pull-based system notification, the InPs are only notified once accessing the application rather than when certain events occur, which can result in a poor performance. However, thanks to the middleware, the integration of other approaches can easily solve the issue.

Therefore, to better reflect and examine the system functionalities and constraints, an evaluation of the different participating components is extremely required. In particular, our defined VN partitioning algorithm must be analyzed, together with the blockchain specifications that enhance the performance.

6 Evaluation

In this chapter, we evaluate the efficiency of the previously presented brokerless VNE system, which uses the blockchain technology to improve the multi-provider VN embedding throughput under the limited information disclosure scenario. First, we present the evaluation setup, which illustrates the architecture used and describes the substrate and virtual network topologies. In addition, in this section, we introduce a pricing model used for the InPs to dynamically quote their services, and a consensus algorithm, which enhances the standard PoW transmission time in private blockchains while maintaining fairness between the mining nodes. Afterwards, we define the goals and metrics used for characterizing the performance, costs and fairness of the system. Lastly, we will examine and interpret the results of our evaluation.

6.1 Evaluation Setup

Our evaluation is carried out on a single computer with four Intel Core i7-7500U CPU at 2.70 GHz and 16 GB of main memory, using the middleware and back-end components from the prototype described in the implementation chapter. Figure 6.1 depicts the architecture of the evaluation set up, which consists of a middleware that sends the corresponding transactions to the blockchain, through different mining nodes. In this setting, if one node cannot serve the transaction, i.e. it is not available, the request will be sent to the following node to ensure the system functionality.

Moreover, since our system is formed by a hybrid blockchain, multiple nodes (InPs) maintain and serve the infrastructure. In this scenario, we have adjusted the number of available mining nodes ranging from one to five depending on the aim of the test, since this amount of nodes is sufficient to flawlessly compare the different blockchain throughputs. Thus, these mining nodes are set up in different ports from a single machine, each pointing to the same genesis file. In a real world, these nodes will be controlled independently by different firms which cooperate to regulate and ensure the application performance.

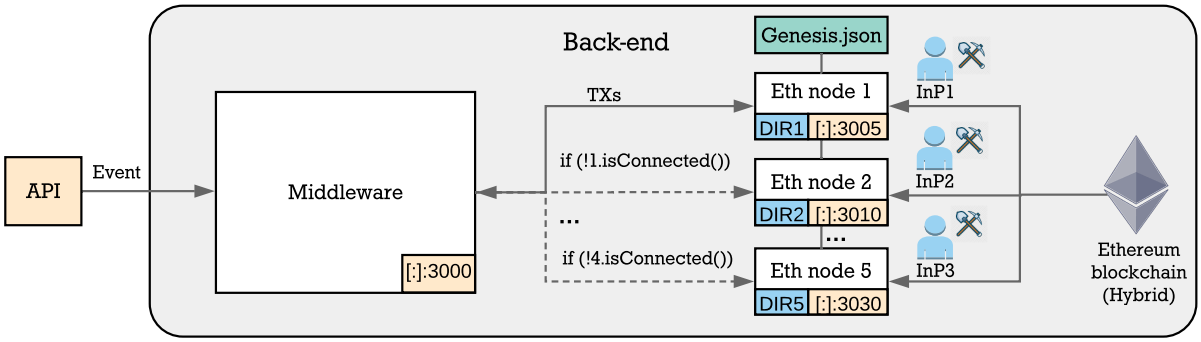


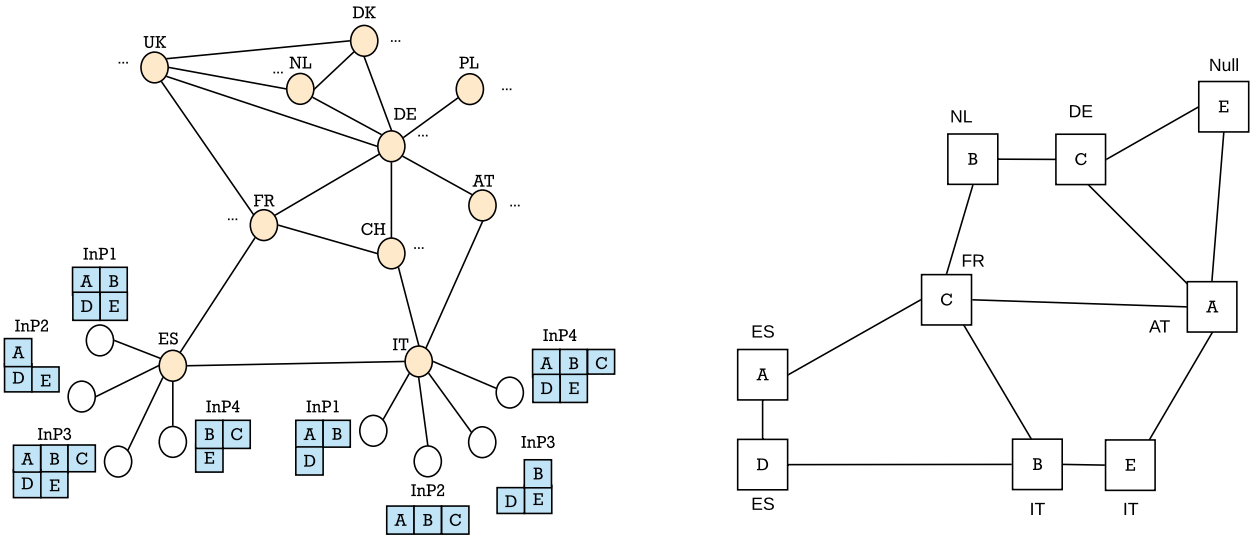
Figure 6.1: Architecture of the evaluation set up.

In the following, we discuss the substrate network and virtual network request topologies, along with the parameters specifications used in our evaluation. Note that, the values assigned for these parameters have been selected in order to illustrate the subsequent evaluation metrics (section 6.2).

- **Substrate network:** To simulate a real scenario, we have generated a substrate network based on the network topology from the GÉANT[1] backbone, which interconnects the National Research and Education Networks (NRENs) across Europe. We have used this model since it is formed by already known non-commercial research and education networks, which do not reveal confidential data.

[1] https://www.geant.org/

The European NREN dataset model was downloaded from the Internet Topology Zoo [zoo16], which is a project that collects data network topologies derived from publicly available records. Furthermore, since our evaluation scenario must contain different InPs competing for embedding virtual nodes in diverse locations, we have tailored the European NREN dataset model and their interconnections to fulfill our requirements. Figure 6.2a shows that our substrate network contains nodes located in ten different European countries (yellow nodes), in which InPs possess peering nodes (white nodes). These peering nodes are formed from 3 to 5 resource types (blue boxes), whose availability and costs are, as stated throughout the thesis, not broadcasted. In addition, each node contains a maximum capacity μ^s_{max}, i.e. a maximum resource utilization, to identify if the InP can serve incoming VN requests in the selected node.
Apart from those parameters, we need reference costs for every resource type, in order to perform the service negotiation between SPs and InPs. For this reason, we have researched the pricing models and costs, provided by different cloud providers. In particular, Amazon EC2 [ama18] offers a broad selection of instance types along with the corresponding on-demand prices. These instances contain hourly prices per computing capacity, which exactly meets our requirements. Thus, we have extracted the average of the minimum and maximum hourly unitary costs per resource type σ_s in the European region, to obtain real values for our presented evaluation scenario. Specifically, our $\{A, B, C, D, E\}$ node types correspond to the general purpose, compute optimized, memory optimized, accelerated computing and storage optimized Amazon EC2 instance types. The mentioned evaluation parameters for the substrate network are summarized in the left part of Table 6.1.



(a) European NREN topology [zoo16].

(b) Example of a virtual network request.

Figure 6.2: (a) The evaluation topologies are sampled as subgraphs from the European NREN topology [zoo16]. (b) Example of a virtual network request with node types $\{A, B, ...\}$ and locations $\{DE, CH, ...\}$

- **Virtual network request:** A VN request is composed of a set of virtual nodes and the corresponding virtual links. A virtual network request example is depicted in Figure 6.2b. Each virtual node contains exactly the same locations of the substrate network, plus a null one used for embedding the resource type in any location. In addition, the virtual node computing demand d_i and the bandwidth demand d_{ij} are also given from a uniform distribution. In a real world, the bandwidth demand between the virtual nodes is typically given by a traffic matrix. However, note that in section 4.2.5, we have stated that intra-link costs within an InP are negligible and that the inter-links, due to the lack of information of the other InP's substrate topologies, must be roughly estimated by the InPs. For simplicity, in this chapter, the inter-links have fixed unitary costs and these are added after partitioning rather than during the bidding process. In other words, if two virtual nodes are

embedded to the same InP, no link cost will be added. Nevertheless, if two nodes are contained into two different providers, then the link cost $\beta_{s*}(L_{ij})$ is given by Equation 6.1 (derived from Equation 4.4), where c is the constant corresponding to the fixed unitary cost.
Furthermore, we model the VN requests events through a periodic arrival rate λ, where each arrival occurs always at the same fixed positive time. Regarding the lifetime of the VN requests, they have also two fixed values $L_1 = 10s$ and $L_2 = 25s$. The reason for using periodic arrival rates and fixed lifetimes rather than, for example, Poisson processes and arbitrary lifetimes, is to better study and illustrate the effect of the resource consumption in a non-optimum scenario. The evaluation parameters for the VN requests are summarized in the right part of Table 6.1.

$$\beta_{s*}(L_{ij}) = c \times d_{ij} \qquad i, j \in N^R,\ \exists d_{ij} \neq \infty :\ i \neq j \tag{6.1}$$

Substrate Network		**Virtual Network Request**	
InPs	3 to 7	Virtual nodes	uniform distrib. [5,10]
Peering node locations	{ES, FR, IT, CH, DE, AT, PL, NL, DK, UK}	Virtual node locations	{ES, FR, IT, CH, DE, AT, PL, NL, DK, UK, NULL}
Peering nodes per InP	10 (1 per location)	Computing demand d_i	uniform distrib. [1,3]
Node types	$\{A, B, C, D, E\}$	Virtual node types	$\{A, B, C, D, E\}$
Node types per peering node	uniform distrib. [3,5]	Bandwidth demand d_{ij}	uniform distrib. [1,10]
Node capacity μ^s_{max}	uniform distrib. [5,100]	Arrival rate λ	1 to 5 requests per second
Unitary cost per node type σ_s	uniform distrib. [10,12]	Lifetimes L_1 and L_2	10s and 25s respectively

Table 6.1: Evaluation parameters: Substrate network and virtual network request.

6.1.1 Pricing Model

In section 4.2.5, we have discussed that our approach encourages users to apply dynamic pricing models for network virtualization. In dynamic pricing, there are many strategies to set flexible service costs based on current market demands. In this chapter, we will adapt the demand and supply labels to match our scenario. In particular, the first corresponds to the computing demand d_i for a resource type, and the second to the resource utilization μ_s in the substrate node maintaining the service.

In contrast to the demand and supply concept, in our evaluation, the capacity does not increase according to the demand, since the resources have fixed maximum capacities μ^s_{max}. Thus, the costs will be given according to the current resource utilization, which is based on the residual capacities. Figure 6.3 depicts an example of the dynamic cost per hour $\alpha^h_{s*}(N_i)$ for the resource type C. From this chart, we can observe how the price gradually increases as the resource is utilized. In this case, the probability of embedding the virtual node is lower, and hence, the resource will wait to embed further VN requests until the node is more released. Equations 6.2 and 6.3 (derived from Equation 4.4) reflect the mentioned behaviour, where σ_s, μ_s, μ^s_u and μ^s_{max} are the unitary cost, resource utilization, used capacity and maximum capacity respectively. In the equations below, note also that the dynamic price per hour $\alpha^h_{s*}(N_i)$ is multiplied for the lifetime L of the virtual network request. In case the lifetime L is less than one hour, then the unitary cost σ_s must be adapted accordingly.

Therefore, thanks to the presented algorithm, the InPs define their product costs regarding the current demands and resources utilization, which provides real-time and on-demand values. These costs are further used to quote prices for the requested virtual node while taking into account the desired profit margins. In future research, this cost estimation can be performed based on many other parameters, which will enable InPs to have a better market positioning.

$$\alpha_{s*}(N_i) = \alpha^h_{s*}(N_i) \times L = (\sigma_s + \frac{\mu_s}{1-\mu_s}) \times L \qquad \mu_s \in [0,1],\ N_i \in N^R \tag{6.2}$$

$$\mu_s = \frac{\mu^s_u + d_i}{\mu^s_{max}} \qquad \mu^s_{max} \in [5,100],\ \mu_u \in [0, \mu^s_{max}],\ d_i \in [1,3] \tag{6.3}$$

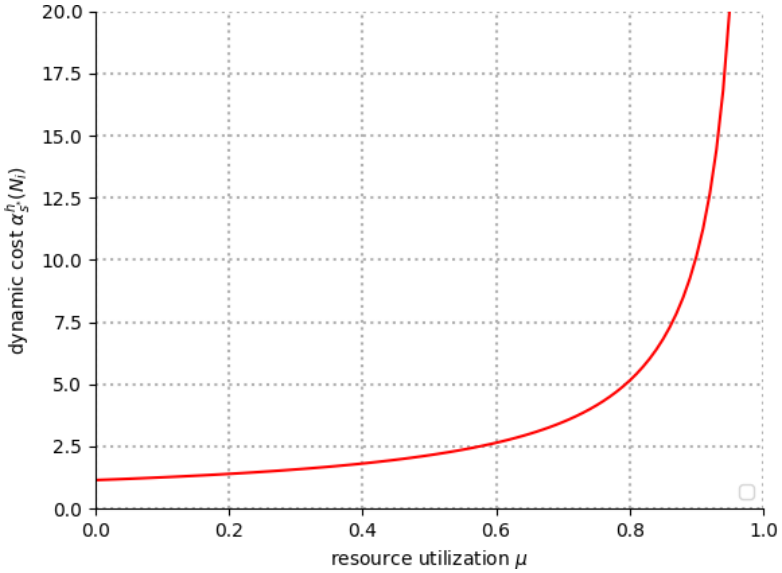


Figure 6.3: Our defined pricing model where dynamic costs increase as the resources are utilized. This is an example for resource type C

6.1.2 Proof of Elapsed Time

At the beginning of chapter 4, we have discussed the different blockchain types, in order to select the most suitable type for our application: **the hybrid/consortium blockchain**. In addition, it is obvious that depending on the blockchain type and the desired functionality, different consensus mechanisms need to be used to fulfill the application requirements. For instance, the Ethereum main network uses the *Proof-of-Work* consensus algorithm, which provides a great scalability between a large number of unknown users, granting them with equal mining probabilities. Nevertheless, since miners need to solve hard cryptographic puzzles to add a new block on the blockchain, this consensus model provides high energy consumptions and slow throughputs, which is obviously not an appropriate solution for private blockchains.

For this reason, after examining consensus mechanisms that provide medium-high transaction rates in a scalable and untrusted model, we have ended up with the **Proof of Elapsed Time (PoET)** protocol. Other consensus algorithms, such as Proof-of-Stake, Byzantine Fault Tolerance or Proof-of-Authority [Bal17], have also been investigated. However, the first two were excluded due to the integration complexity into our current Ethereum platform and the third since it sacrifices the trust of our system (not hybrid).

The Proof of Elapsed Time consensus mechanism is used in the Hyperledger Sawtooth project [saw18], designed by Intel. PoET is a lottery based consensus, where the next miner (also called leader) is randomly selected. The workflow is simple: First, each mining node waits a random time (sleep). Afterwards the node with the shortest wait time, wakes up and commits the next block. Finally, the other mining nodes are notified.

To work properly, the leader election needs to be totally random, where a miner cannot manipulate this waiting time. To achieve this, the Hyperledger Sawtooth project relies on specialized hardware, in particular, on the Intel's Software Guard Extensions (SGX) [sgx18]. Through this trusted environment, the participants are supplied with fair waiting times. Nevertheless, in our scenario, we will assume that there are no malicious nodes manipulating these waiting times. To summarize, we will tailor the idea of PoET to fit in our Ethereum platform.

6.2 Goals and Metrics

Due to the lack of previous work on multi-provider VNE using the blockchain technology, the main goals of this chapter, are to check the feasibility of the introduced solution and to demonstrate its practical potential. Thus, we set up the previous environment (Figure 6.1) based on the architecture presented in 5.2, to explore the benefits and implications of the proposed system.

Moreover, to study the behavior and performance of our approach, each InP makes use of the previously introduced pricing model (section 6.1.1), with homogenous prices. Apart from the real cost of the asset, note that each InP defines his own profit margins $\gamma_n(N_i), \gamma_n(G)$, to perform the bid. Therefore, we will also examine how the deviations in these values affect the final outcome of the VN partitioning.

On the other hand, another fundamental factor for the throughput of our system are the different blockchain specifications. From these properties, one of the key elements that directly influence the functionality is the consensus mechanism employed by the mining nodes. For this reason, we first investigate why *Proof-of-Work (PoW)* protocol hinder the system performance, and check if altering the block difficulty optimizes the results. In addition, we also inspect whether the aforementioned *Proof of Elapsed Time (PoET)* consensus enhances the PoW protocol while maintaining fairness between the mining nodes.

Following these definitions, we further summarize the metrics used in our study:

- **Acceptance rate**: Based on our approach and the proposed pricing model, we examine the VN embedding efficiency through the acceptance rate. This VN acceptance rate is the percentage of embedded virtual nodes from the total incoming VN requests. Nevertheless, these values are significantly influenced by many circumstances. Between these, we demonstrate how the following parameters alter the performance: (i) the virtual network request lifetime L, (ii) the number of participating InPs I_p and (iii) the VN requests arrival rate λ.
- **Bidding strategy**: The bidding strategies are evaluated based on how the profit margins $\gamma_n(N_i), \gamma_n(G)$ employed by the different InPs, produce significant revenues on certain scenarios.
- **Blockchain performance**: In this thesis, the blockchain performance will be verified basically in terms of the block generation time (mining) and the number of forks. The former, since each block contains many transactions, is the more accurate manner to calculate the speed of the transactions. The latter, once the block generation time decreases, the number of new blocks solutions increases, which can as a consequence, create forks on the blockchain. Another parameter that also steps up the number of forks is the network latency. Nevertheless, since our mining nodes are on a single machine and these values are not large in a real world, we will neglect the network latency between our mining nodes.
- **Mining fairness**: We consider our system fairness regarding the mining nodes contribution into the blockchain growth, i.e. the number of blocks that each node mines. Since, our approach corresponds to a hybrid blockchain, where each InP coordinates and ensures the system's functionality, the effort on mining blocks must be equally shared between the blockchain nodes. Thus, in a fair system, each node roughly mine the same number of blocks.

Furthermore, it is important to note that thanks to the blockchain and smart contracts flexibility, the results obtained thereupon are not definitive findings, since they can always be improved by upcoming investigations.

6.3 Evaluation Results

In this section, we present the evaluation results of the Vickrey auction model based on a demand-resource pricing model, and using the Ethereum platform in a private network. Through the aforementioned metrics and parameters, we provide different results for various scenarios during the evaluation process. These parameters are summarized in Table 6.2.

Symbol	Description
σ_s	unitary cost for a resource type in substrate node N_s^*
μ_s	variable $\in [0,1]$ expressing the resource utilization in substrate node N_s^*
μ_{max}^s	variable $\in [5,100]$ expressing the maximum capacity of a resource type in substrate node N_s^*
μ_u^s	variable $\in [0,\mu_{max}]$ expressing the used capacity of a resource type in substrate node N_s^*
$\alpha_{s*}^h(N^R)$	dynamic hourly unitary cost for virtual network request N^R
φ	acceptance rate $\varphi \in [0,1]$
I_p	participating InPs
λ	virtual network request arrival rate (number of requests in 5s)
L	virtual network lifetime
B_m	mined block
t_B	block generation time
D_B	block difficulty
D_{B_p}	parent block B_p difficulty
f_B	number of forks in B blocks
w_n	PoET window time of mining node n
x_i	miner IDs

Table 6.2: Evaluation notations

6.3.1 Acceptance Rate and Bidding Strategy

Initially, to measure the acceptance rate of our model under different conditions, we first study the influence of the VNR lifetime L. The reason to start proving this is that since our defined pricing model (section 6.1.1), is directly linked to the resource utilization, the VNR lifetime most probably has a significant impact on the performance.

Therefore, we have defined a scenario where three participating InPs (I_p) with zero profit margins ($\gamma_n(N_i), \gamma_n(G) = 0$) embed 100 VNR across 200 runs. In the first half of the runs, these requests have a small lifetime $L_1 = 5s$. Then, to compare the outcome with a larger lifetime, a second lifetime $L_2 = 20s$ has been set for the remaining runs. In addition, these requests are sent periodically every second, i.e. with arrival time $\lambda = 5$.

Figure 6.4 illustrates the mean acceptance rates φ with CI=0.95, when the lifetimes L_1 and L_2 are applied. As expected, this value is higher for L_1 inasmuch as smaller lifetimes ($L_1 < L_2$) utilize the resources during a shorter period, and consequently, the nodes are earlier released. More specifically, first, the acceptance rate decreases for both cases as the resources are gradually filled, which hampers the embedding of the following VNRs. Afterwards, we contemplate that both lifetimes reach steady states as the number of VNR grows. This is because the lifetime of the first incoming VNR expires, which reduce the nodes used capacities μ_u^s. Since this cycle is repeated over the time, it produces the named steady state. In particular, L_1 and L_2 start to experience this phenomenon around 10s and 25s respectively, which exactly corresponds to the lifetime of our requests.

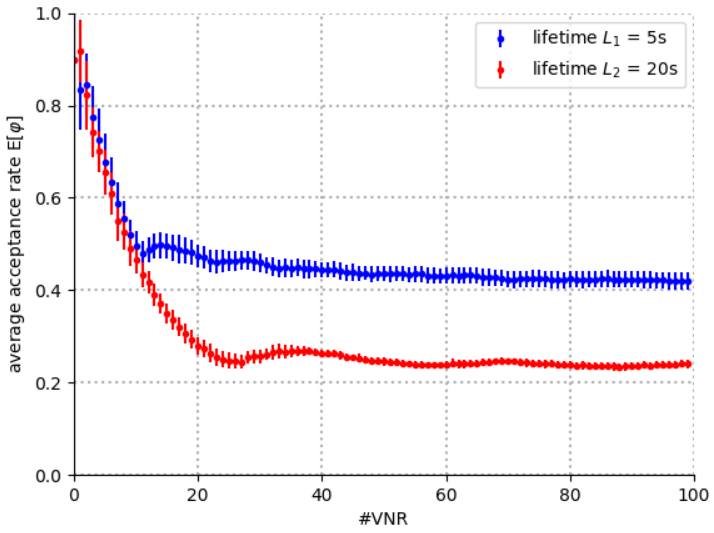


Figure 6.4: Average acceptance rate $\varphi \in [0,1]$ with CI $= 0.95$ describing the fraction of requests accepted. After a transient phase the request acceptance rate is dependent on the request lifetime L.

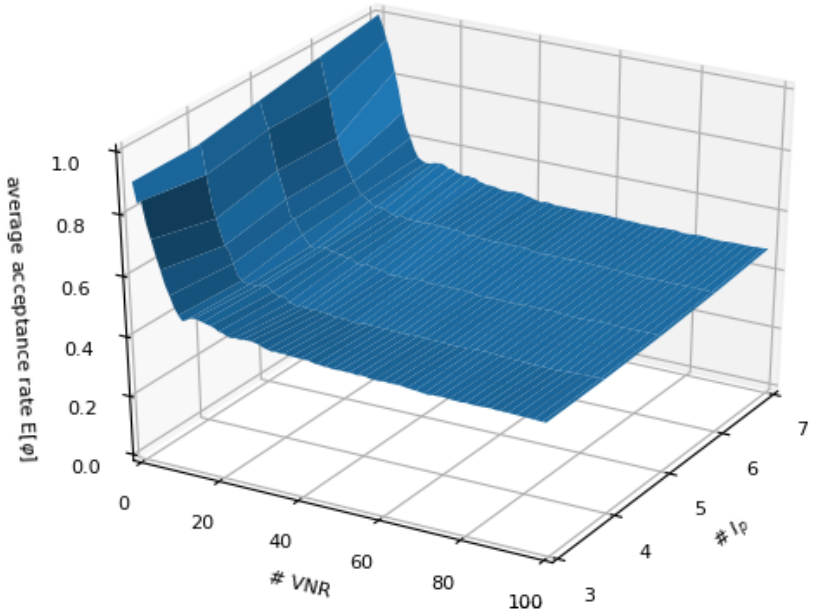


Figure 6.5: Average acceptance rate $\varphi \in [0,1]$ for $L = 5s$ and $\lambda = 5$, describing the fraction of requests accepted for infrastructure providers $I_p = [3,4,5,6,7]$. The acceptance rate is slightly higher as I_p increases.

In Figure 6.4, we have seen that at the beginning, a high volume of requests in a short period, cause that the acceptance rate scales down, since the resource utilization augment. For this reason, we need to investigate how the throughput is enhanced, when there are more resources, or in other words, as the number of I_p increases. Thus, we will repeat the experiment again, now in presence of 3 to 7 InPs, using $\lambda = 5$ and L_1. Despite the fact that with large lifetimes L_2 the differences would have been obviously higher, in this trial, we have applied a lifetime L_1, in order to observe whether the acceptance rate also varies for small lifetimes. Figure 6.5 illustrates the average acceptance rates φ with 3 to 7 I_p.

Firstly, we observe that with $I_p = 3$ the acceptance rate φ exactly matches our introduced Figure 6.4. Nevertheless, the interesting point is once I_p has a greater value. In this case, we notice that the initial acceptance rate is larger since the number of free resources is bigger. Besides, we also contemplate the

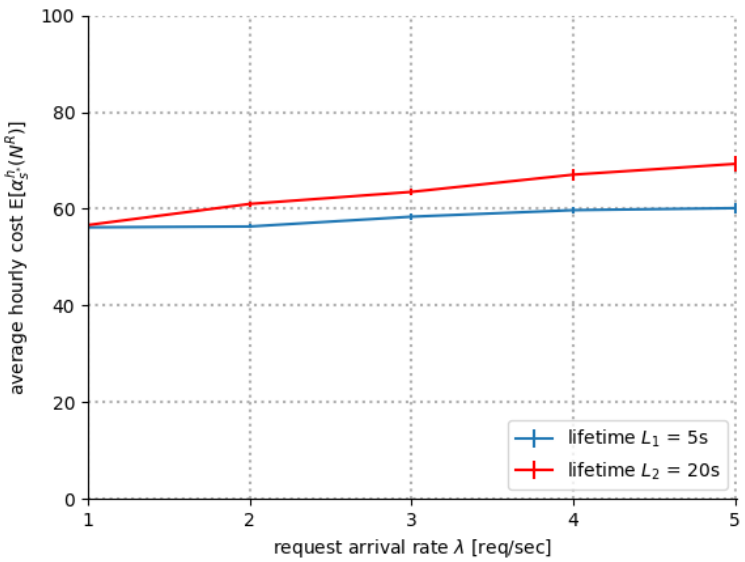


Figure 6.6: Impact of a periodic arrival rate λ on the average hourly costs. A higher request arrival rate causes a cost increase due to the resource utilization.

slight slope reflected during the VNR, although Figure 6.5 provides no statistical evidence of a significant slope for a determined number of I_p. Thus, we can assert that for L_1 the acceptance rate φ slightly improves initially, even though in the end, the steady-state difference is not steep. However, we can deduce that for larger lifetimes the slope variations will be more meaningful.

On the other hand, we have above-mentioned that our costs are directly linked to the resource utilization. For this reason, an analysis of our proposed pricing model is extremely required. In addition, in the following test, we also investigate the arrival rate λ footprint on the VNR costs, tailoring these values from one to five, i.e. from one to five requests every 5s. These values are chosen with regard to five seconds, because the minimum processing time for one request in our system is approximately 1s, meaning that $\lambda = 5$ corresponds to this minimum (1 req/s).

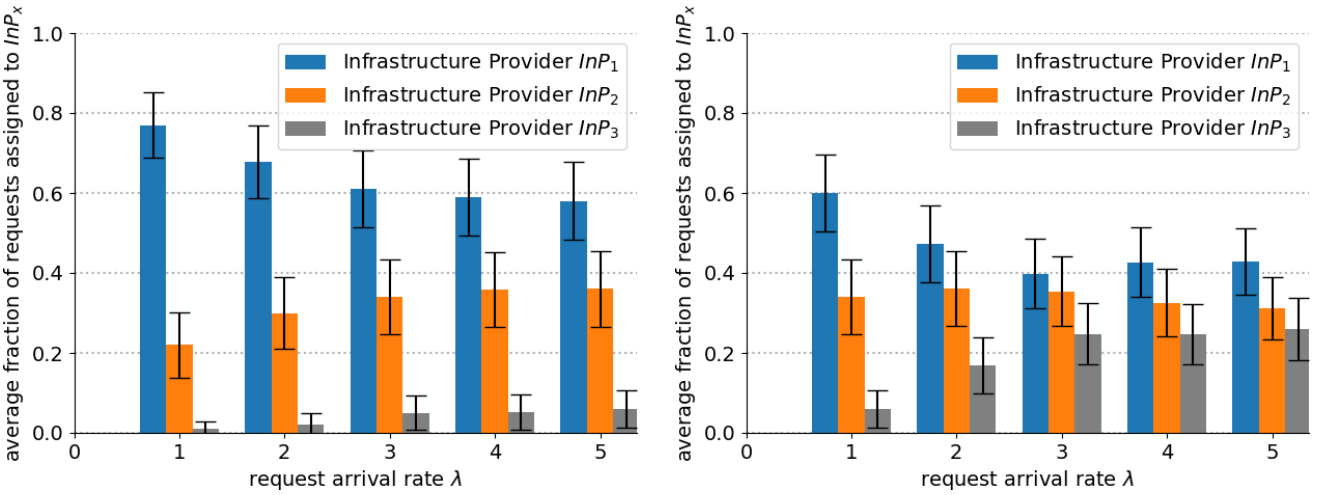
Figure 6.6, depicts the mean hourly costs for the requested virtual nodes N^R with lifetimes L_1 and L_2, towards the different arrival rates. Note that this value corresponds to the hourly cost $\alpha^h_{s*}(N^R)$ rather than the final $\alpha_{s*}(N^R) = \alpha^h_{s*}(N^R) \times L$. In this manner, the impact of the different arrival rates and lifetimes is best reflected, since the cost difference is directly proportional to the resource utilization. Besides that, to better precise the mean costs in a graph, we have reduced the randomness of N^R, fixing the number of VNRs to 5 rather than applying a uniform distrib. [5,10]. Thus, since in Table 6.1, we have defined that the nodes unitary costs σ_s values are around 10 and 12, and the number of $N^R = 5$, the minimum costs of $\alpha^h_{s*}(N^R)$ start roughly at 50.

In case of small arrival rates ($\lambda = 1$), as the resources are less utilized, then the Equation 6.2 results in $\alpha^h_{s*} \approx \sigma_s$, and hence, to 50 independently of the applied lifetimes. However, once the arrival rate value increases, the deviations from L_1 to L_2 are higher. The main reason is that the second term of the Equation 6.2, tends to infinity when the resources are occupied, which occurs frequently by large lifetimes. For instance, when $\lambda = 5$, the difference between L_1 and L_2 on the average cost is approximately 21%.

In the previous experiments, we have evaluated the acceptance rate φ regarding the lifetimes L_1 and L_2, and different I_p. In addition, we have studied how the costs are altered with different arrival rates λ.

Nevertheless, we have considered so far that the InPs apply zero profit margins ($\gamma_n(N_i), \gamma_n(G) = 0$), which indicates that the bidder strategies need to be further assessed. Thus, we have set up a similar scenario with the same arrival rates λ, lifetimes L_1 and L_2 and $I_p = 3$, where each InP has individual and package profits $\gamma_1 = 0.1, \gamma_2 = 0.2, \gamma_3 = 0.4$ respectively.

Moreover, to prevent that InPs use immense profit margins, we have established a VN upper bound cost $u_G = 80$, which is a slight increment of the maximum cost reached in Figure 6.6. Figure 6.7 depicts



(a) lifetime $L_1 = 10$s.

(b) lifetime $L_2 = 25$s.

Figure 6.7: Average fraction of VN requests assigned to each InP with different arrival rates λ and lifetimes $L_1 < L_2$. The InPs are heteregenous with different profit margins $\gamma_1 = 0.1, \gamma_2 = 0.2, \gamma_3 = 0.4$. With large lifetimes L and arrival rates λ, InPs with higher profit margins start to embed virtual network requests.

the percentage of embedded virtual network requests for each InP. To remark the differences, in this plot, we have divided the results for L_1 and L_2, which corresponds to Figures 6.7a and 6.7b respectively. In the first, we observe that for low arrival rates the values between the I_p are completely different because the resources are not yet fully occupied. Hence, the InP with lower profit margins, in this case, InP1 embeds most of the VNR. In contrast, for larger λ, the second InP, since the InP1 resources are utilized, start to gradually increase the number of embedded VNRs. Finally, InP3 embeds the remaining VNR, although this amount is not significant since the other InPs can still deal with the requests.

Meanwhile, when the VNR contain larger lifetimes L_2 (Figure 6.7b), the denouement is more weighted. From the beginning, since the InP's resources are largely populated, the incoming requests need to be served by other InPs. Naturally, first the InPs with lower profit margins handle the petitions. However, as the number of resources is not sufficient, InPs with bigger profits start to embed a meaningful fraction of them.

To summarize, in this section, we have studied the acceptance rate based on several parameters, and an example outcome for InPs with different bidding strategies has been introduced. After examining these results, we first conclude that a huge number of available resources, i.e. multiple InPs, are essential for achieving great acceptances rates. This can be contemplated, after disrupting the system performance through different arrival rates λ and lifetimes L_1 and L_2, which detects the changing behaviors in the worst case scenarios. In addition, thanks to the bidding strategy experiment, we have observed that InPs with high-profit margins also embeds VNR under certain conditions (large lifetimes L and arrival rates λ). For this reason, we reinforce the idea introduced during this thesis, which states that a well-executed market research can be translated with higher revenues, and thereby, to a better market positioning.

6.3.2 Blockchain Performance and Mining Fairness

In the following, we first study how the Proof of Work (PoW) specifies the block generation time. This value is related with the time that a miner needs to find a valid block header, whose cryptographic hash is less than $\frac{2^{224}}{d}$, where d is the block difficulty (Equation 2.1). Thus, the block difficulty D_B, which changes per each block following a difficulty adjustment algorithm, fundamentally provides the block generation time. In Ethereum, this is given by Equation 6.4, where D_{B_p}, T_p and T_b are the parent block difficulty, parent block timestamp and current block timestamp respectively.

$$D_B = D_{B_p} + \frac{D_{B_p}}{2048 \times \max(1 - (\frac{T_b - T_p}{10}), -99)} + 2^{(\frac{B}{100000} - 2)} \tag{6.4}$$

The above equation can be divided into two parts. The left part provides the difficulty adjustment, which increases or decreases depending on the time difference between the current block timestamp T_b and the parent block timestamp T_p. In particular, when $T_b - T_p < 10s$, the difficulty is adjusted upwards by $D_{B_p} - \frac{D_{B_p}}{2048 \times 1}$. Converserly, for $10s <= T_b - T_p < 20s$ the difficulty remains unaltered and for $T_b - T_p > 20s$, it is adjusted downwards from $D_{B_p} - \frac{D_{B_p}}{2048 \times -1}$ until $D_{B_p} - \frac{D_{B_p}}{2048 \times -99}$. Regarding the right part of Equation 6.4, also called the difficulty bomb, it exponentially increases the difficulty every 100000 blocks. The aim is to cause that mining a block is harder over the time, which will create less blocks, and hence, less mining rewards. In addition, this value is also intended for switching from PoW to Proof-of-Stake (PoS) at some point in the future.

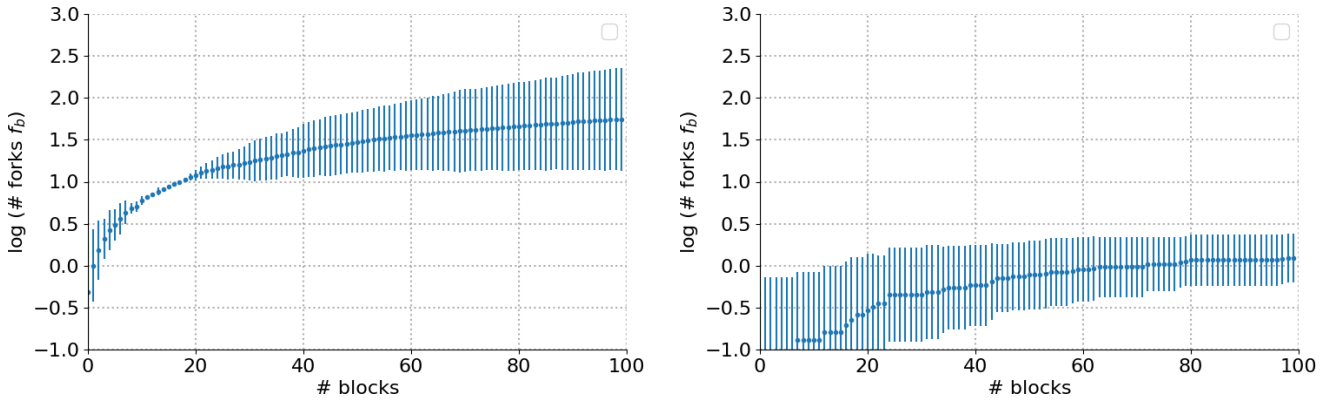
Despite in a private network, the number of transactions is less than in a public network, we must bear in mind that every time that the state of a smart contract is changed, it is considered a transaction. Since this action frequently occurs in a private environment, and even if the difficulty is configured with a small value in the genesis block, the difficulty changing algorithm provokes ending up with undesired mining times (around 12s in the Ethereum main network). For this reason, we claim that using the standard PoW in private networks is not logical, and we must consider other consensus algorithms.

The first approach is to establish an almost negligible difficulty with a fixed value from the beginning. To achieve this, we have modified a file from the open source code of Ethereum that defines the PoW consensus model (go-ethereum-master/consensus/ethash/consensus.go). Specifically, the CalcDifficulty function (line 2 in Listing 6.1) now returns 0x400, which is the hexadecimal value of 1024. Due to the bound divisor of Equation 6.4, in the PoW algorithm, this value cannot be smaller than 2048. For this reason, in this test, we have chosen a value that is a half of this minimum, which already give us huge differences. However, for further evaluations, this value could be set to even a smaller value (also in genesis block), such as big.NewInt(1), although the results will be almost equal.

Furthermore, since a continuous and disproportionate growth of the blockchain is in most cases illogical for private networks, we have preloaded a JavaScript function that restricts the mining only with the presence of transactions (line 6 in Listing 6.1). For instance, miner.start(miningThreads) begins the mining with mining threads being the number of used processor cores, and eth.getBlock("latest").number gets the block number of the latest mined block.

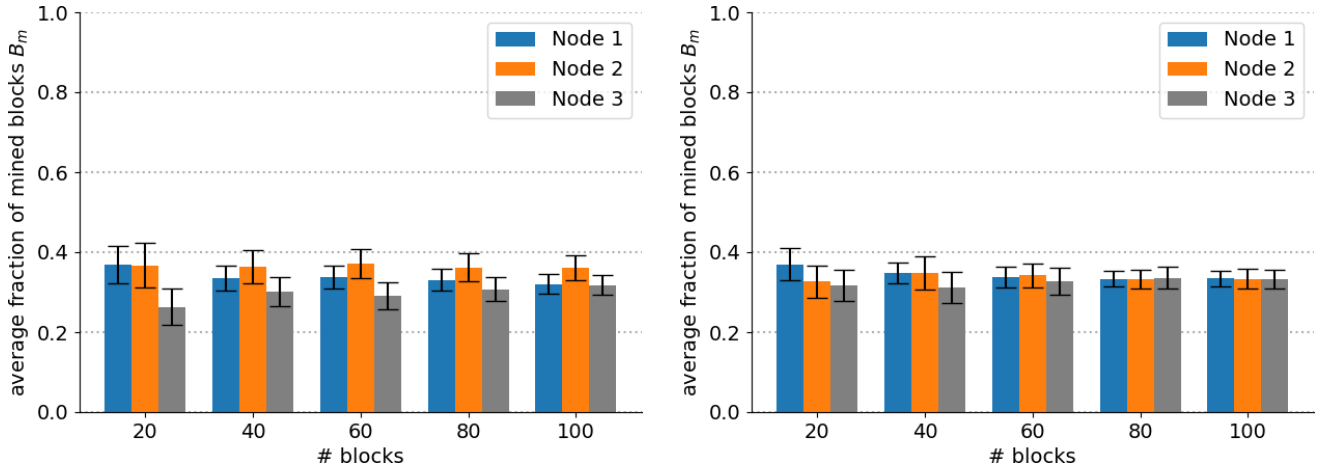
```
1  func (ethash *Ethash) CalcDifficulty(chain consensus.ChainReader, time uint64, parent *
       types.Header) *big.Int {
2      return big.NewInt(0x400)
3  }
4
5  var miningThreads = 1;
6  function tailoredPow() {
7      if (eth.getBlock("pending").transactions.length > 0) {
8          txBlock = eth.getBlock("pending").number
9          if (eth.mining) return;
10         miner.start(miningThreads)
11         while (eth.getBlock("latest").number < txBlock) {}
12         miner.stop()
13     }
14     else {
15         miner.stop()
16     }
17    }
```

Listing 6.1: Defined functions that specify the block difficulty and the tailored PoW.



(a) Proof of Work (PoW). **(b)** Proof of Elapsted time (PoET).

Figure 6.8: Number of forks f_b per block with three mining nodes using PoW with tiny difficulty and PoET. Note that the difference depends on the parameters of PoW, i.e. difficulty. and PoET, i.e. the window size w_n



(a) Proof of Work (PoW). **(b)** Proof of Elapsted time (PoET).

Figure 6.9: Average fraction of mined blocks B_m among three mining nodes using PoW with tiny difficulty and PoET. Using PoET the average fraction is more well-balanced due the random leader election.

In the following, we analyze the blockchain performance of the proposed solution, in terms of block generation time, number of forks and mining fairness. This evaluation is carried out with three mining InPs, defined locally in three different ports (Figure 6.1).

- **Number of forks**: The maximum number of forks for b blocks are $f_b = b \times (n-1)$, where n are the corresponding miners (3 in our test). Figure 6.8a, illustrates the logarithmic average number of forks f_b with CI=95% along 100 blocks. This value is $\log_{10}(f_b) \in [0, 1.75]$, more explicitly from 1 to 56 forks. If we compare the obtained values, with the worst case scenario $f_{100} = 100 \times (3-1) = 200$, this already corresponds to a 28% of the maximum possible forks with only 3 InPs. This produces a significant amount of misused work, which as in private networks the amount of miners is smaller, can result in important losses on throughput. In addition, it is also important to note that this value will increase as the number of mining peers grows.
- **Mining fairness**: In Figure 6.9a, we contemplate the mean percentage of mined blocks B_m along 100 blocks. Since the system is a modified version of the PoW with almost negligible difficulty, it still conservates the fairness given by the PoW algorithm. This fairness is more well-balanced, as the number of mined blocks grows. By contrast, in the beginning, the resulting distribution is more arbitrary, although this does not affect our system over time.

- **Block generation time**: Since the difficulty is fixed for all the mined blocks, the block generation time t_b is practically constant $t_b \in [1,2]$. Compared to the standard PoW, the t_b has decreased considerable, which entirely fulfills the requirements of a private blockchain.

Therefore, the tailored PoW consensus model offers great block generation times while maintaining a fair system. Nevertheless, since the difficulty is extremely low, multiple miners resolve blocks at the same time, which directly hinders the system performance with a huge expenditure of resources. Moreover, as having a higher number of mining nodes directly implies extra losses, this causes that the system suffers from scalability.

For this reason, we introduce a second approach, which implicitly, evaluates the PoET consensus model explained in section 6.1.2. To prove this, first, we define an identical window time w_n for each of the $n = 3$ mining nodes. In this workflow, only when a new transaction arrives, the mining nodes wait an arbitrary time between 1 or 10 seconds ($w_n \in [1,10]$), before appending a new block. Since in PoET there is no cryptographic work, the block generation time is similar to the waiting period. Hence, the window range must not be too large (slow throughput), and not too short (many forks occurring). This is why a $w_n \in [1,10]$ appears to be a proper starting point.

The following code, see Listing 6.2, shows the implemented JavaScript functions. From this, it is important to understand line 7, which is the responsible for calling the function startMining() after waiting an arbitrary time given by the window w_n. Once entering the function, if the old txBlock is still equal to the pending block number, then the mining proceeds.

```
1  var timeout_set;
2  var miningThreads = 1;
3
4  function simulatePoET() {
5      if (eth.getBlock("pending").transactions.length > 0) {
6          if (timeout_set || eth.mining) return;
7          var txBlock = eth.getBlock("pending").number;
8          timeout = setTimeout(startMining(txBlock), getRandomInt(1000,10000));
9      }
10     else {
11         miner.stop()
12     }
13 }
14
15 function startMining(txBlock) {
16     if (txBlock === eth.getBlock("pending").number) {
17         if (eth.mining) return;
18         miner.start(miningThreads);
19         while (eth.getBlock("latest").number < txBlock) {}
20         miner.stop();
21         timeout_set = false;
22     }
23     else {
24         timeout_set = false;
25     }
26 }
27
28 function getRandomInt(min, max) {
29     var time = Math.floor(Math.random() * (max - min)) + min;
30     timeout_set = true;
31     return time; //The maximum is exclusive and the minimum is inclusive
32 }
```

Listing 6.2: Defined functions to apply the Proof of Elapsted Time (PoET) in an Ethereum client.

Hereafter, we further analyze the PoET algorithm based on the last introduced metrics:

- **Number of forks**: Figure 6.8b depicts the logarithmic number of forks along 100 blocks using the PoET consensus model. At first glance, we can observe a pronounced decrease, using the same three mining nodes. In particular, after 100 blocks, the mean value is around 0.1, which corresponds to 1.25 forks. Compared to the number of f_b using the tailored PoW, now the percentage is only 0.625% with respect to the maximum, rather than 28%.
- **Mining fairness**: Thanks to the randomness given by the window time w_n and the non-presence of malicious nodes, in this scenario, the miners have equal mining privileges. In Figure 6.9b, we can contemplate how the mining tasks are successfully distributed since the beginning. Thus, the introduced approach also preserves fairness without the need of solving any cryptographic work.
- **Block generation time**: In this case, the block generation time is not given by any complex algorithm. Conversely, the value is directly proportional to the shortest waiting time randomly chosen by the miners. For this reason, we will study the changes on the block time, when the number of mining InPs increases or decreases. In particular, Figure 6.10 illustrates the box plot of the resulting average block times t_b with one to five miners, in comparison to the analytic block times (red dots). Since analytic block times do not include computational periods, these values are smaller than the ones extracted from our application. In case of a one-dimensional model (1 InP), the analytic block times are directly related with the mean values. On the other hand, in a two-dimensional model (2 InPs) this is given by Equation 6.5, where k is the maximum value of the window w_n. For instance, with 2 InPs and $w_n \in [1, 10]$, the block time is $t_b = 3.85s$. Back to the block time depicted in Figure 6.10, we observe that once the number of InPs increases, since the probability of obtaining shorter waiting times is higher, the block time is reduced accordingly. Nevertheless, due to the big size of our defined window time, when the number of InPs is small, the block time is too long. For instance, with only one miner the block time is approximately 5.5s, which is a poor throughput as with a single miner no forks can occur. Thus, regarding the forks, the presence of more miners creates more f_b on the blockchain (not shown in the graph), although in our example, thanks to the wide defined window time w_n, the effect is not severe.

$$E[t_b] = \sum_{i=1}^{k} \frac{(2k+1)-2i}{k^2} \tag{6.5}$$

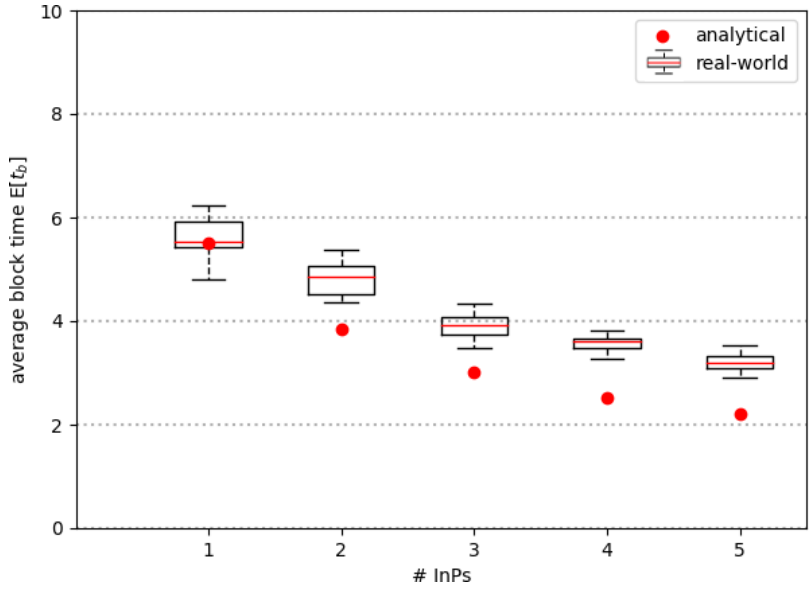


Figure 6.10: Average block time with different InPs mining and using the PoET consensus model. The difference between the analytical and real-world model is probably due to the overload in block generation (computation) and in informing the other mining nodes of the mined block.

Therefore, our proposed PoET consensus model maintains fairness on the system. However, since the blockchain time t_b is given by w_n, an approach that defines the window size maintaining fast throughputs and avoiding forks must be investigated.

In a best case scenario, from a set of windows $w = \{w_1, w_2, ..., w_n\}$, the system distributes one window for each miner, being the difference between each window Δw greater or equal to the averaged sum of computational times t_{com} in one node ($\Delta w = w_{i+1} - w_i \geq t_{com}$). For instance, imagine a scenario with 3 InPs, $t_{com} = 1$ and window set $w = \{1, 2, 3\}$, where the last two blocks have been mined by miner 1 and 2 consecutively. Then, for the next block each InP should be supplied with a window that ensures fairness on the system, i.e. $w_1 = 2, w_2 = 3, w_3 = 1$. Despite the optimum window values are simple to identify, the problem lies in the distribution of those, such that decentralization and scalability is preserved. Thus, we immediately discard placing a central node that distributes the window times for each mining node.

On the contrary, we introduce the pseudocode for a decentralized PoET window time distribution, where the last miner distributes the next window periods creating a pair p for each of the n miners. A possible solution to perform the allocation could be writing the pair set in the *extraData* field of each block, before mining it. Then, once each miner receives a notification from the latest mined block, they could pull the window time matching their address.

Algorithm 2: Pseudocode for a decentralized window time distribution across multiple mining nodes

1: $w = \{w_1, w_2, ..., w_n\} : \ \Delta w = w_{i+1} - w_i \geq t_{com}$
2: $x = \{x_1, x_2, x_j, ..., x_n\}$
3: $x^{'} = x - \{x_j\}$
4: $p = \{p_1, p_2, ..., p_n\}$
5: $p_n = \{x_j, w_n\}$
6: **for** $i = 1; \ i < n$ **do**
7: $\quad k = rand\{x^{'}\}$
8: $\quad x^{'} = x^{'} - \{k\}$
9: $\quad p_i = \{x_k, w_i\}$

In Algorithm 2, x corresponds to a set containing the miner addresses, where x_j is the last miner, i.e. the user executing the algorithm. First, the miner creates a pair p_n, which contains his address and the maximum value of the window. Afterwards, a loop process starts, where each miner is matched with a new window time. It is important to note that in this example, the cycle does not utilize any past data regarding the miners preceding the last block, although this could be further implemented, keeping track of them by storing their addresses in the memory.

In essence, in this section, we have demonstrated that the consensus model used in a private network directly affects the blockchain throughput. First, we have studied how the block time is derived from the standard PoW used by Ethereum. Since the difficulty has a big impact, we have tried to minimize this value by tailoring the Ethereum PoW. Due to the almost non-existent cryptographic work, this has directly caused that multiple miners create different solutions at the same time (forks).

For this reason, we have introduced PoET, which has considerably reduced the number of forks while maintaining fairness on the system. Since in this protocol each miner waits an arbitrary time given by the window size, this value has an extremely important role in determining the block time. To study the system behavior, we have first performed our test with $w_n \in [1, 10]$, where each miner picks and waits independently a random time inside of this range. Nevertheless, as the window time is fixed for all the scenarios, this produces a poor throughput when there is a small number of peers, and several forks when the number of participating nodes is higher. As a consequence, we have presented a possible solution for optimizing these values, by establishing a prior agreement on the window times distribution in a decentralized and distributed manner.

7 Conclusions

Hint:

This chapter should summarize the thesis and describe the main contributions of the thesis. Subsequently, it should describe possible future work in the context of the thesis. What are limitations of the developed solutions? Which things can be improved? The section should have a length of about three pages.

7.1 Summary

7.2 Contributions

7.3 Future Work

IOTA-TANGLE BigchainDB Hyperledger? The Front-end application could be hosted in a decentralized storage such as IPFS or SWARM. A DApp has its backend code running on a decentralized peer-to-peer network. Contrast this with an app where the backend code is running on centralized servers. A DApp can have frontend code and user interfaces written in any language (just like an app) that can make calls to its backend. Furthermore, its frontend can be hosted on decentralized storage such as Swarm or IPFS. BC storing chunks of data

7.4 Final Remarks

Bibliography

[AM+06] Lawrence M Ausubel, Paul Milgrom, et al. The lovely but lonely vickrey auction. *Combinatorial auctions*, 17:22–26, 2006.

[AM16] Saveen A Abeyratne and Radmehr P Monfared. Blockchain ready manufacturing supply chain using distributed ledger. 2016.

[ama18] Amazon ec2 instance types. `http://aws.amazon.com/ec2/instance-types`, 2018. Accessed: 2018-01-16.

[B+14] Vitalik Buterin et al. A next-generation smart contract and decentralized application platform. *white paper*, 2014.

[Bal17] Arati Baliga. Understanding blockchain consensus models. Technical report, Tech. rep., Persistent Systems Ltd, Tech. Rep, 2017.

[Ben14] Juan Benet. Ipfs-content addressed, versioned, p2p file system. *arXiv preprint arXiv:1407.3561*, 2014.

[bit09] Bitcoin is an innovative payment network and a new kind of money. `https://bitcoin.org/e/`, 2009. Accessed: 2018-03-26.

[BM00] Colin Boyd and Wenbo Mao. *Security issues for electronic auctions*. Hewlett-Packard Laboratories Bristol (UK), 2000.

[CJ09] Jorge Carapinha and Javier Jiménez. Network virtualization: a view from the bottom. In *Proceedings of the 1st ACM workshop on Virtualized infrastructure systems and architectures*, pages 73–80. ACM, 2009.

[Cla17] Jen Clarck. Blockchain Technology: the Next Generation of the Internet, March 2017. accessed:2017-11-27.

[CRB09] NM Mosharaf Kabir Chowdhury, Muntasir Raihan Rahman, and Raouf Boutaba. Virtual network embedding with coordinated node and link mapping. In *INFOCOM 2009, IEEE*, pages 783–791. IEEE, 2009.

[CSB10] Mosharaf Chowdhury, Fady Samuel, and Raouf Boutaba. Polyvine: policy-based virtual network embedding across multiple domains. In *Proceedings of the second ACM SIGCOMM workshop on Virtualized infrastructure systems and architectures*, pages 49–56. ACM, 2010.

[CST80] Vicki M Coppinger, Vernon L Smith, and Jon A Titus. Incentives and behavior in english, dutch and sealed-bid auctions. *Economic inquiry*, 18(1):1–22, 1980.

[DARP17] David Dietrich, Ahmed Abujoda, Amr Rizk, and Panagiotis Papadimitriou. Multi-provider service chain embedding with nestor. *IEEE Transactions on Network and Service Management*, 14(1):91–105, 2017.

[DRP15] David Dietrich, Amr Rizk, and Panagiotis Papadimitriou. Multi-provider virtual network embedding with limited information disclosure. *IEEE Transactions on Network and Service Management*, 12(2):188–201, 2015.

[EDPM13] Flavio Esposito, Donato Di Paola, and Ibrahim Matta. A general distributed approach to slice embedding with guarantees. In *IFIP Networking Conference, 2013*, pages 1–9. IEEE, 2013.

[Eth16] Ethcore. Parity. https://github.com/paritytech/parity, 2016.

[FBB+13] Andreas Fischer, Juan Felipe Botero, Michael Till Beck, Hermann De Meer, and Xavier Hesselbach. Virtual network embedding: A survey. *IEEE Communications Surveys & Tutorials*, 15(4):1888–1906, 2013.

[Fia05] Petr Fiala. Information sharing in supply chains. *Omega*, 33(5):419–423, 2005.

[Fou14] Ethereum Foundation. Go ethereum. https://github.com/ethereum/go-ethereum, 2014.

[Fou15] Ethereum Foundation. Pyethapp. https://github.com/ethereum/pyethapp, 2015.

[Fou17] Ethereum Foundation. Viper. https://github.com/ethereum/vyper, 2017.

[Gen16] What does each genesis.json parameter mean. https://lightrains.com/blogs/genesis-json-parameter-explained-ethereum, 2016. Accessed: 2018-05-17.

[Hei02] Jussi Heikkilä. From supply to demand chain management: efficiency and customer satisfaction. *Journal of operations management*, 20(6):747–767, 2002.

[HLAZ11] Ines Houidi, Wajdi Louati, Walid Ben Ameur, and Djamal Zeghlache. Virtual network provisioning across multiple substrate networks. *Computer Networks*, 55(4):1011–1023, 2011.

[HLZ08] Ines Houidi, Wajdi Louati, and Djamal Zeghlache. A distributed virtual network mapping algorithm. In *Communications, 2008. ICC'08. IEEE International Conference on*, pages 5634–5640. IEEE, 2008.

[HS05] David Hausheer and Burkhard Stiller. Peermart: The technology for a distributed auction-based market for peer-to-peer services. In *Communications, 2005. ICC 2005. 2005 IEEE International Conference on*, volume 3, pages 1583–1587. IEEE, 2005.

[hyp17] Business blockchain frameworks hosted with hyperledger. https://www.hyperledger.org/, 2017. Accessed: 2018-03-26.

[HZRS17] Ronny Hans, Hendrik Zuber, Amr Rizk, and Ralf Steinmetz. Blockchain and smart contracts: Disruptive technologies for the insurance market. 2017.

[IM16] Richard Ramos Iuri Matias, Jonathan Rainville. Embark. https://github.com/embark-framework/embark, 2016.

[KARF03] Mikko Kärkkäinen, Timo Ala-Risku, and Kary Främling. The product centric approach: a solution to supply network information management problems? *Computers in Industry*, 52(2):147–159, 2003.

[LB92] Hau L Lee and Corey Billington. Managing supply chain inventory: pitfalls and opportunities. *Sloan management review*, 33(3):65, 1992.

[Lom15] Natasha Lomas. Everledger is using blockchain to combat fraud, starting with diamonds. *Tech Crunch*, 2015.

[Mer87] Ralph C Merkle. A digital signature based on a conventional encryption function. In *Conference on the Theory and Application of Cryptographic Techniques*, pages 369–378. Springer, 1987.

[MLTJ17] M Macdonald, Lisa Liu-Thorrold, and R Julien. The blockchain: A comparison of platforms and their uses beyond bitcoin. *Working Paper*, (February), 2017.

[MS15a] Michael Mainelli and Mike Smith. Sharing ledgers for sharing economies: an exploration of mutual distributed ledgers (aka blockchain technology). 2015.

[MS15b] Bicholas d'Andrea Mike Seese. Truffle. https://github.com/trufflesuite/truffle, 2015.

[Nak08] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008.

[OV02] Elth Ogston and Stamatis Vassiliadis. A peer-to-peer agent auction. In *Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 1*, pages 151–159. ACM, 2002.

[Ros17] Ameer Rosic. What is ethereum casper protocol? crash course. https://blockgeeks.com/guides/ethereum-casper/, November 2017. Accessed: 2018-02-09.

[saw18] Hyperledger sawtooth. https://www.hyperledger.org/projects/sawtooth, 2018. Accessed: 2018-05-01.

[sgx18] Intel® software guard extensions. https://software.intel.com/en-us/sgx, 2018. Accessed: 2018-05-01.

[Shi14] Ken Shirriff. Bitcoins the hard way: Using the raw bitcoin protocol, February 2014. accessed:2018-02-01.

[Sta15] John Stark. Product lifecycle management. In *Product Lifecycle Management (Volume 1)*, pages 1–29. Springer, 2015.

[Swa15] Melanie Swan. *Blockchain: Blueprint for a new economy*. " O'Reilly Media, Inc.", 2015.

[Tea17] Roster Team. Blockchain at berkeley. https://blockchain.berkeley.edu, 2017. Accessed: 2018-02-18.

[TT16] Don Tapscott and Alex Tapscott. *Blockchain Revolution: How the technology behind Bitcoin is changing money, business, and the world*. Penguin, 2016.

[Tut02] A Tuttle. Who do you trust. *Industrial Distribution*, 91(3):17, 2002.

[Vic61] William Vickrey. Counterspeculation, auctions, and competitive sealed tenders. *The Journal of finance*, 16(1):8–37, 1961.

[WG17] Karl Wüst and Arthur Gervais. Do you need a blockchain? *IACR Cryptology ePrint Archive*, 2017:375, 2017.

[zoo16] The internet topology zoo. http://www.topology-zoo.org/index.html, 2016. Accessed: 2018-04-30.

[ZXB10] Fida-E Zaheer, Jin Xiao, and Raouf Boutaba. Multi-provider service negotiation and contracting in network virtualization. In *Network operations and management symposium (NOMS), 2010 IEEE*, pages 471–478. IEEE, 2010.

[ZZSRR08] Yaping Zhu, Rui Zhang-Shen, Sampath Rangarajan, and Jennifer Rexford. Cabernet: Connectivity architecture for better network services. In *Proceedings of the 2008 ACM CoNEXT Conference*, page 64. ACM, 2008.