
Università degli Studi di Catania
Dipartimento di Matematica e Informatica

Dottorato di Ricerca in Informatica
XXXV Ciclo

---

# The Ethereum Technology applied to Enterprise Decentralized Solutions.

## Healthcare and Renewables Use Cases.

Giovanni Marotta

Tesi di Dottorato

Relatore: Chiar.mo Prof. Emiliano Tramontana
Correlatore: Chiar.mo Prof. Giuseppe Pappalardo

**Acknowledgements**

*My immense gratitude goes to Prof. Emiliano Tramontana and Prof. Giuseppe Pappalardo whose emphatic, yet not condescending support, allowed me to complete my journey without losing direction. A special thanks also goes to Dr. Andrea Fornaia, who sustained me in both good and gloomy moments.*

### A remarkable oddity

*What a weird coincidence occurred during the writing of my thesis! The reader can notice that the first bibliographic reference is to one of the seminal works in the blockchain genesis, which was written by Stuart Haber and W. Scott Stornetta in 1990 at Bellcore Labs, Morristown, NJ, USA. Well, the fact is that I was also working there, exactly at the same time, in an office very similar to the one shown in the picture. A teammate of mine swears he can recognize Stuart Haber from the picture, honestly I can't. I just like to think we were next-door neighbors...*

Photo by Bob Ono

Stuart Haber, left, and partner Scott Stornetta discuss their 'time-stamp' for electronic documents, which they developed in the Bellcore Research Laboratory in Morristown

*...memories aside, I can still exhibit a proof of evidence of those times spent unknowingly close to these two visionary guys: the Bellcore mug that I proudly keep on using for my afternoon teas!*

# Abstract

Public blockchains have recently emerged as a disruptive technology in the distributed systems arena because of the adoption of a predominant decentralized approach. Initially dealing with the creation of a disintermediated cryptocurrency and financial market, the blockchain technology has rapidly caught the interest of a growing scientific and technological community willing to develop novel applications, in varied contexts, based on the decentralization paradigm, thus undermining the long established client-server model. The birth of Ethereum, the first public blockchain introducing executable programs into its core technology, has fostered the development of other similar platforms specifically addressed to private implementations, capable of mitigating well-known technological issues of the native public blockchains, especially in terms of scalability, energy demands and performance costs.

This thesis aims at demonstrating that the Ethereum blockchain technology is also suitable to host effective and performing decentralized applications in enterprise contexts, without resorting to custom-built blockchains for permissioned environments, provided that ad-hoc system enhancements are introduced in the devised solutions. The novel applications are therefore equipped with tailored features, such as properly designed smart contracts, optimal use of the blockchain storage, and smooth integration with external decentralized file systems, thereby ensuring that the developed Ethereum-based solutions exhibit a secure, decentralized and scalable behaviour.

Two use cases, namely “contact tracing” and “renewable energy”, have been investigated. Intensive test activities have thoroughly proven that a sensible use of the Ethereum platform delivers the expected outcomes in the explored enterprise contexts.

# Contents

**Abstract** **iii**

**1 Introduction** **1**
1.1 The decentralization paradigm . . . . . 1
1.1.1 Distributed and decentralized systems . . . . . 2
1.1.2 Decentralization in the blockchain . . . . . 4
1.2 Blockchain and cryptocurrency . . . . . 6
1.3 New frontiers in blockchain applications . . . . . 7
1.3.1 Smart contracts . . . . . 8
1.3.2 Drivers of enterprise blockchain development . . . . . 9
1.3.3 Use cases . . . . . 10
1.4 Personal contributions . . . . . 12

**2 Background** **15**
2.1 Blockchain fundamentals . . . . . 15
2.1.1 Consensus algorithms . . . . . 16
2.1.2 Blockchain types . . . . . 18
2.1.3 Transactions . . . . . 20
2.1.4 Tokens . . . . . 21
2.2 The Ethereum platform . . . . . 22
2.2.1 Ethereum clients . . . . . 23
2.2.2 Account-based model . . . . . 25
2.2.3 Transaction-driven state machine . . . . . 26
2.2.4 Token standards . . . . . 28
2.2.5 Digital wallets . . . . . 30
2.2.6 Ethereum's smart contracts . . . . . 31
2.2.7 Ethereum's blockchain . . . . . 33
2.2.8 Open issues . . . . . 35
2.3 Decentralized Applications . . . . . 38
2.3.1 Solidity contracts . . . . . 39
2.3.2 Web3 Providers . . . . . 43

2.3.3 Front-end operations . . . . . 44
2.4 Digital identity . . . . . 45
2.4.1 Self-Sovereign Identity . . . . . 46
2.4.2 Decentralized Identifiers and Credentials . . . . . 46

**3 Related Work** **48**
3.1 Contact tracing solutions . . . . . 48
3.1.1 Blockchain-unequipped . . . . . 49
3.1.2 Blockchain-equipped . . . . . 51
3.2 Renewable Energy Sources solutions . . . . . 53
3.3 Blockchain design topics . . . . . 55
3.3.1 Smart contract immutability . . . . . 56
3.3.2 Smart contract computational cost . . . . . 58
3.3.3 Smart contract storage . . . . . 59
3.3.4 Security and scalability design . . . . . 60

**4 Contact Tracing solution** **62**
4.1 Hybrid decentralized version . . . . . 64
4.1.1 System overview . . . . . 65
4.1.2 Device Localization . . . . . 67
4.1.3 Contact tracing . . . . . 70
4.1.4 User interface . . . . . 72
4.1.5 Absolute localization tests . . . . . 72
4.1.6 Comparative analysis . . . . . 74
4.1.6.1 Comparison with centralized solutions . . . . 75
4.1.6.2 Comparison with decentralized solutions . . . 75
4.1.7 Solution discussion . . . . . 77
4.2 Highly decentralized version . . . . . 78
4.2.1 DApp_v1 . . . . . 79
4.2.1.1 Back-end decentralization . . . . . 80
4.2.1.2 Blockchain components . . . . . 82
4.2.1.3 Operational workflow . . . . . 84
4.2.1.4 Blockchain-related software . . . . . 84
4.2.1.5 Gas consumption tests . . . . . 89
4.2.2 DApp_v2 . . . . . 94
4.2.2.1 Contact tracing in DApp_v2 . . . . . 96
4.2.2.2 Modified blockchain components . . . . . 97
4.2.2.3 Blockchain-related software . . . . . 99
4.2.2.4 Gas consumption tests . . . . . 102
4.2.2.5 Decentralizing the system DB . . . . . 103
4.2.3 Comparative analysis . . . . . 107

4.3 Complete solution wrap-up . . . . . 109

**5 DER management solutions 112**
5.1 Blockchain environment . . . . . 114
5.1.1 Energy Web Decentralized Operating System . . . . . 114
5.1.2 Energy Web applications . . . . . 116
5.2 Smart Aggregator DApps . . . . . 117
5.2.1 DER marketplace DApp . . . . . 120
5.2.1.1 Application description . . . . . 120
5.2.1.2 Demand/offer matching . . . . . 121
5.2.2 Smart metering DApp . . . . . 123
5.2.2.1 Application overview . . . . . 124
5.2.2.2 Data collection and storage workflow . . . . . 125
5.2.3 Grid flexibility DApp . . . . . 129
5.2.3.1 DApp overview . . . . . 135
5.2.3.2 Blockchain implementation . . . . . 139
5.2.3.3 Back-end architecture . . . . . 144
5.2.3.4 Experimental assumptions . . . . . 145
5.2.3.5 Experimental results . . . . . 149
5.3 Solution discussion . . . . . 156

**6 Conclusions 158**

**Bibliography 161**

# Chapter 1

# Introduction

Concepts such as blockchain, cryptocurrency and distributed ledger technology (DLT) have irreversibly impacted the academic and technological debate among scientific communities, and entered the daily lexicon of industry professionals and policymakers.

Although early proposals under the broader DLT umbrella have been present since the early 90s' [1], it was not until the publication of the *"Bitcoin whitepaper"* [2] in 2008 that the IT community had to deal with a rising tide of innovation in the way crypto-content could be managed on the net, with regard to ownership, security, transparency and reliability, thus allowing peer-to-peer (P2P) distributed systems to unveil promising perspectives in application development, until then quite underrated.

Whilst the initial hype was on new ways to create and exchange digital money with no intermediation from *"trusted"* financial third-parties, nowadays the interest has dramatically shifted towards a variety of new application fields in which the blockchain technology has an opportunity to unleash its transformative and disruptive capabilities and favour its mass adoption.

## 1.1 The decentralization paradigm

In the contemporary practice of distributed systems, a growing trend has emerged towards *administrative* decentralization, i.e., taking the full con-

trol of an organization away from a central trusted authority, in favour of a multiplicity of peer controlling agents. Accordingly, while conventional client-server applications assume the centralized administration model for the server, the recent and steady growth of the blockchain technology has made the above-stated *administrative* decentralization paradigm (in the following *"decentralization paradigm"* for short) emerge alongside its counterpart. Running a decentralized system entails several benefits, such as removal of single points of failure, increased efficiency, and distributed trust in decision making, to name but a few.

At the heart of most decentralization schemes, distributed consensus plays a primary role. Business-wise, consensus implies the final agreement between two or more involved parties on the value of parameters of interest to all (i.e., the balance of a bank account). Typically, a central organization, such as a bank, assumes the role of custodian of truth with respect to all the involved parties. This implies that customers and other stakeholders have to trust such a custodian faithfully.

The decentralization paradigm launches the challenge to reach a consensus, among the involved parties, on the values of agreed-upon parameters, without resorting to a central authority. This is a revolutionary idea since it entails a radical change in the way facts are assessed without the traditional trusted party.

### 1.1.1 Distributed and decentralized systems

To better understand the decentralization paradigm the notion of distributed systems needs to be introduced and explained, since the two concepts are related but often erroneously overlapped to a very large extent.

A distributed system is a computing architecture made up of two or more nodes interacting in a coordinated fashion in order to achieve a common goal. From the end user's point of view, a distributed system is seen as a single logical platform. Nodes are able to exchange messages with each other and they have individual processing power and storage capabilities. In this model, there is still a central authority that coordinates the overall processing

results. A node that exhibits unpredictably faulty or malicious behaviour is classified as Byzantine and it can damage unexpectedly or intentionally network operations. Designing a performing and fault tolerant distributed system can turn out to be quite compelling, hence many efforts have been put into the solutions of these issues since the early stages of the research activities on the the field. The CAP theorem, originally formulated as an intuition [3] and later proved [4], states that a distributed system cannot have the three fundamental properties, i.e., Consistency, Availability and Partition, simultaneously. These properties ensure data consistency among all the copies shared by the nodes, no downtime in system accessibility, and fault tolerance to byzantine nodes.

Unlike canonical distributed systems, a decentralized system adopts a paradigm that does not provide for a central administrative authority; conversely, control is distributed among many nodes.

In DLT systems varying levels of decentralization can be accomplished depending on the design of the entire application ecosystem, the topmost being total disintermediation, that is, absence of any central or intermediate authority when it comes to transaction validation, execution and storing. Disintermediation in the monetary and financial sectors, regardless of the growing dissemination, is exposed to practical obstacles, due to heavy regulatory and compliance requirements. On the other hand, this model can be successfully applied to various industries beyond the economic scope and accepted at regulatory level with fewer restrictions.

As far as consensus is concerned, a high degree of decentralization is achieved when a group of peer nodes compete with each other to be selected for the provision of protocol activities. This competing approach ensures that a single intermediary will never exclusively supply the required service. This mechanism is reflected by a leader-based consensus mechanisms, whereby competing nodes need to win a lottery for the election of a leader that is appointed to propose and store the final value of any data to be shared and stored in the distributed ledger.

### 1.1.2 Decentralization in the blockchain

Blockchains are inherently distributed systems, more specifically they are decentralized distributed systems [5]. In fact, blockchains adopt replication of computing, communication and storage resources as a method to achieve fault tolerance, while decentralized consensus algorithms are used to ensure consistency. The combination of these two mechanisms is known as state machine replication. The Blockchain technology in fact encompasses the idea of designing a platform that can implement a replication of the virtual state machine in all nodes.

As for the consensus process, reaching an agreement about the final state of new incoming data among competing nodes is a particularly difficult task in distributed systems and involves executing sophisticated agreed-upon protocol procedures.

Early precursors in the arena of consensus mechanisms are the works of: (i) Paul Baran, who introduced the idea of cryptographic signatures and decentralized networks in 1964 [6]; (ii) Lamport, who developed the conceptual problem - and some solutions to it - of the weak Byzantine generals in 1982 and later published it in 1983 [7]; (iii) Castro and Liskov who presented the Practical Byzantine Fault Tolerance (PBFT) algorithm in 1999 [8]. The latter describes a new replication algorithm which is able to tolerate Byzantine faults in asynchronous environments, such as the Internet, and incorporates several important optimizations that improve the response time of previous algorithms. After years of academic research, consensus mechanisms have eventually found their practical employment with the advent of Bitcoin in 2009 [2], in which the Proof of Work (PoW) algorithm was deployed as the selected mechanism to achieve the distributed consensus among the P2P nodes making up the Bitcoin network.

In the wide context of blockchain technologies, several consensus mechanisms have been proposed later on [9, 10]. For each of them, the designated protocol will elect the leader among the competing nodes at each occurrence based on specific criteria (e.g., computational work, reputation, randomness, quality of service). More detailed explanations of some of the most used con-

sensus algorithms in blockchain are given throughout the present document.

Although the blockchain, with its consensus algorithms equally carried out by peer P2P nodes, radically shifts the control paradigm towards the decentralization side, it must be noted that its underlying technologies are made up of conventional systems, mostly centralized. These elements include communication networks, computation and storage architectures. A full decentralization would instead require that the ecosystem around the blockchain be also decentralized in its underlying components, as shown in Figure 1.1.

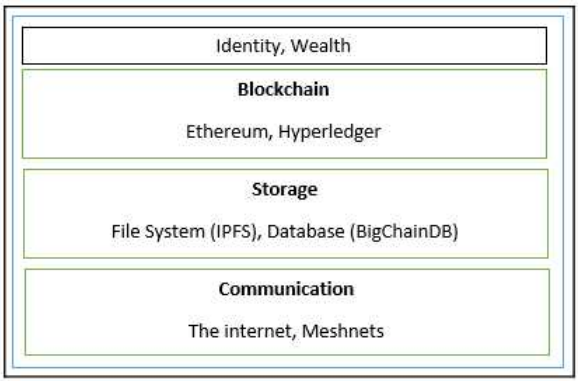


Figure 1.1: Decentralized ecosystem [5].

Unfortunately, the communication layer serving the blockchain nodes is inherently centralized in most cases, therefore an appropriate theoretical alternative could be represented by a more tightly meshed network topology. However, deploying these layouts at a planetary scale is very costly, so that this next step toward decentralizing communication networks in the blockchain ecosystem could be very impractical.

Needless to say that, as a distributed ledger technology, the blockchain can store data directly in its own repositories, and this may suffice to achieve full storage decentralization. Unfortunately, for cost and performance reasons, a blockchain is not designed to store large amounts of data, in contrast to traditional databases. Resorting to distributed hash tables (DHTs)[1], as originally used in P2P file sharing solutions[2], looks like a good complemen-

[1] https://en.wikipedia.org/wiki/Distributed_hash_table
[2] https://en.wikipedia.org/wiki/Peer-to-peer_file_sharing

tary choice for storage, provided that "high availability" requirements be fulfilled. Being "high availability" compliant in its design, Inter Planetary File System (IPFS)[3] is one of the most used storage technology along with the blockchain.

Finally, decentralization in computation is inherently achieved in the blockchain thanks to the replication of the protocol business logic at any P2P node, in which the virtual machines run the consensus algorithm as well as additional blockchain-related software of varying degrees of complexity, from simple scripts to sophisticated smart contracts.

## 1.2 Blockchain and cryptocurrency

The ultimate impulse to the practical development of blockchain platforms was undoubtedly provided by the deployment of Bitcoin in 2009 [2]. In terms of market capitalization, Bitcoin remains the most visible implementation of the blockchain technology for the time being. As for the decentralized consensus, the common parameter of interest is the balance of user accounts.

With the advent of electronic-cash (e-cash) in the Internet, the double spending issue, that is, the concrete chance to make a 'carbon-copy' money transaction to different parties at nearly the same time, has strongly required the intervention of central banking platforms to play the role of custodians of truth with regard to detecting invalid (i.e., replicated) transactions and inconsistency of account balances.

Conversely, Bitcoin establishes the decentralized consensus on account balances by settling the double spending problem without resorting to a central authority. The implemented mechanism does not store account balances in the Bitcoin blockchain, but computes them on-the-fly on each transaction execution, by reading the results of each and every previous transaction, which is immutably stored in Bitcon's DLT forever.

The digital money transacted by Bitcoin lies within the broader concept of cryptocurrency, which, in turn, has developed around the research work on

[3] `https://ipfs.tech/`

the e-cash topic over the last four decades. Already in 1983 David Chaum [11] addressed the two critical points in e-cash schemes, such as accountability and anonymity, by introducing two fundamental cryptographic operations, namely blind signatures and secret sharing. The former allows signing a document without actually seeing it, whereas the latter tackles the double-spending problem. More work on cryptographic operations was produced following Chaum's original findings, such as the "hashcash", introduced by Adam Back in 1997, and described more formally in 2002 [12], as a Proof-of-Work (PoW) system to control e-mail spam. PoW requires spending a significant quantity of computational power to find an hash for each mail to be sent. This idea inhibits spammers since it makes inconsiderate for them to spend such an expensive amount of processing resources. Though computing the hash takes much effort, verifying it is an easy and quick operation from the receiving side.

Before Bitcoin, other schemes to create cryptocurrency by making use of "hashcash" can be found in Wei Dai "b-money" proposal [13]. Further ideas were later introduced by Nick Szabo [14] and Hal Finney [15], all of them facing issues that could not make the implementation of such schemes applicable in practical ways or, at least, could not be classified as fully decentralized.

The concept of "hashcash" gained final visibility since it was selected to implement the decentralized mechanism regulating the leader election in the Bitcoin consensus protocol. At each round, the competing node (i.e., miner) that manages to solve the computation puzzle first is elected to create (i.e to mine) a new block of data to be added permanently to the chain. It is noticeable that Bitcoin was the perfect combination of previous ideas and concepts from e-cash schemes and distributed systems, which gave birth to the transformational notion of blockchain.

## 1.3 New frontiers in blockchain applications

Not only decentralized consensus has fostered the advent of cryptocurrency blockchains, such as Bitcoin, but it has also initiated a new era of distributed applications, which are extending the scope of use of the blockchain far be-

yond its original intent. The computing and communication arena has in fact experienced the overwhelming entry of the disruptive blockchain paradigm, which is changing the point of view of the scientific community on distributed applications, in cases where transparency, fault tolerance, and freedom from net censorship are of paramount importance. Due to these concerns, the new paradigm of *decentralized applications (DApps)* began to emerge for the building of novel and exciting Internet-based apps since the introduction of smart contracts in the blockchain ecosystem.

### 1.3.1 Smart contracts

The concept of smart contract, originally theorized in 1997 by Nick Szabo [14] to refer to *"a set of promises, specified in digital form, including protocols within which the parties perform on these promises"*, has played a central role in the development of DApps, within the new paradigm of Web3[4], which is intended as a more secure, fair and transparent web, based on the idea of multiplying profit centers by dividing value, in an open network. This more democratic vision has fostered research work since the early days of the millennium, but only with the growing availability of advanced technologies, such of AI, IoT, decentralized systems, and advanced cryptography, the popularity of Web3 has spread steadily over the last years. The term Web3 was in fact introduced in 2014 by Gavin Wood, the co-founder of Ethereum [16], a technology environment whereby smart contracts can be thoroughly specified, developed and deployed in a decentralized environment.

Notwithstanding the fact that there is no standardized definition of smart contracts, an articulate way of describing them should include both technical and enforcement aspects. The latter deal with the idea that smart contracts should execute the business logic of legal agreements deterministically if contractualized conditions are met, without any intermediation. Whether or not the agreement terms enforced by smart contracts can be legally enforceable will be a matter of heated debate between differing school of thoughts in the coming years. A considerable amount of focused research is in fact being

[4] `https://en.wikipedia.org/wiki/Web3`

carried out in the formal design of smart contracts in order to make their implementation legally viable and technically effective.

If we only stick to technical aspects related to blockchain applications, an essential yet satisfactory definition of smart contracts can be that of programs of arbitrary length and complexity, whose executable code is saved in the blockchain storage [17].

### 1.3.2 Drivers of enterprise blockchain development

Similarly to *e-mail* for the conventional Web, Bitcoin in 2009 was the "killer application" that established a new mindset in the way digital technologies can be modeled to provide a new application paradigm, though the first accountable blockchain application dates as back as 1990 with the aforementioned work of Haber and Stornetta [1] on a digital time-stamping service deployed on a distributed ledger.

Blockchain is nowadays capable of providing solutions to various problems in such a radical way to reshape the corporate agenda on distributed applications, as highlighted by Dr. W.Scott Stornetta: *"...what began thirty years ago as an effort to fix digital records has blossomed into an entire industry that aspires to disrupt not just all the world's recordkeeping, but all the industries and social structures that depend on recordkeeping..."* [18].

Corporate investments in blockchain are in fact changing the idea of delivering services to the community and, in commercial sectors, creating a promising outlook of revenues even in the near term. These encouraging breakthrough is a consequence of eventually leaving behind the notion that *enterprise* or cross-organization (*consortium*) blockchains do not unlock the entire potential of democratic disintermediation as public blockchains (see Subsection 2.1 for detailed definitions of blockchain types).

A fundamental driver of the blockchain revolution is the shared source of trust coming from decentralized consensus, which is at its highest expression in public blockchains, but still retains a heavy attractive weight in enterprise or consortium blockchains, which explains the remarkable amount of research and investments being spent on them. In addition, optimal system perfor-

mance and extended privacy are better achieved in enterprise blockchains, which, in some implementations, don't even use criptocurrency to incentivize the consensus process from the validating nodes [19].

Decentralized applications can find space as a replacement of traditional client-server ones whenever a free fall of mutual trust among parties should be alleviated by the adoption of a technological paradigm that places transparency, immutability, integrity, and shared consensus at its forefront.

As a matter of fact, in the last five years there has been increasing excitement around the enterprise blockchain potentials from many innovative companies, industry giants and visionary government agencies. The increasing number of proposed proof of concepts (POCs) are steadily raising the interest for blockchain technologies and proving capable of real transformation in many industrial, commercial and administrative scenarios.

### 1.3.3 Use cases

One of the first applications beyond the e-cash handling, which harnessed the unlimited potentialities unleashed by the introduction of Ethereum, has been the creation of Decentralized Autonomous Organizations (DAOs) that rely on organizational rules enforced by the automated execution of smart contracts. The initial attempt in 2016 was explicitly called the DAO[20], a crowdfunded initiative aimed at financing a project to provide a platform for investment, which unfortunately suffered from a hard-coded software bug that allowed skillful attackers to drain a considerable amount of money off the DAO and forced the Ethereum designers to change the system software [20]. It is acknowledged that the development of advanced DAO projects, in which, for instance, the application can fully replicate the behavior of a company board, should require the adoption of more sophisticated technologies, such as artificial intelligence, to make them look like the real thing [21].

Blockchain technology is also used in the medical sector to store patient data. The ability to maintain an incorruptible, decentralized and transparent registry of patient data, i.e., the Electronic Health Registers (EHRs), makes blockchain technology an ideal tool to fulfil the standard requirements in

this application area. Decentralization makes it easier for patients, doctors and health professionals to share information quickly and safely. Estonia, for instance, which is a forerunner in this field, began to harness the power of distributed ledger technology in healthcare and other fields as far as 2012 [22].

Informed consent about personal data collection is a recent concept, having a high impact on data management and treatment transparency. It is essential that every interested party has a dynamic control over consent, by being capable of expanding it, decreasing it or revoking it completely. A blockchain-based solution, as in [23], allows the history of informed consent given by patients in genomic research experiments to be traced in an immutable and non-repudiable fashion.

Another pervasive theme which occurs throughout many blockchain applications is "digital identity', such as in Know Your Customer (KYC) processes whereby the sharing of proof of customer identities among banks can be secured via a distributed ledger. For more complex applications some enterprise platforms, such as the ones promoted by the Hyperledger project[5] and the Energy Web project[6], make their own built-in, complementary services available to provide support to the implementation of digital identity mechanisms, such as the enrollment and registration of identities during network operations, and the management of changes like credential additions, drops, and revocations.

The green energy exchange is becoming one of the leading application fields where digital identities can play a distinctive role. Production through renewable energy sources (RES) is, in fact, becoming increasingly decentralized and lands quite naturally in the blockchain territory as far as energy trading, information storage, and increased transparency of energy flows are concerned. The change in the electricity market is characterized by the growth of so-called "smart grid" and by the emerging possibilities for decentralized energy generation. In the said scenario a large number of actors will be involved in the production and consumption of green energy. Such actors are generally small and without a well-established and trusted model

[5] `https://www.hyperledger.org/`
[6] `https://www.energyweb.org/`

to participate in the smart grid. Adoption of the blockchain can then play a fundamental role in many aspects of the green energy marketplace in the presence of parties that are unknown to each other. Furthermore, the distributed ledger technology can be used to certify the source of energy production from producers to consumers, for the sake of guaranteeing its green origin.

Other fertile sectors for enterprise blockchain applications are: (i) supply chain management (SCM) in the fields of food tracking [24] and counterfeit pharmaceuticals [25]; (ii) logistics, where the distributed ledger technology can be used to track assets by using their digital identities [26]; (iii) documentation management, in which the issuance of documents certifying the completion of procedural steps in complex cross-organization processes can be securely tracked and timestamped [27].

## 1.4 Personal contributions

This thesis focuses on the research and experimental activities carried out during my doctorate placement, which are related to the design of viable and effective blockchain-based solutions in a few selected, non-financial, application scenarios, such as (i) healthcare and (ii) renewable energy. In both scenarios, specialized simulations or prototypes have been designed, developed, and deployed on field and/or on suitable testbeds in order to validate the novel contributions behind their design.

The presented solutions aim to establish that even enterprise-level requirements can be met by decentralized applications deployed on top of standard public blockchain protocols that admittedly suffer from scalability and high computation costs, such as Ethereum[7]. To accomplish these goals, the decentralized applications have been designed to be endowed with a balanced combination of ad-hoc features, which contribute to enhance the scalability and security properties of the proposed solutions, as well as to provide a reassuring level of accuracy and efficiency, thus making their employment suitable and effective for their reference scenarios.

[7] `https://ethereum.org/en/`

The described experiments and their findings are the result of work which has been published in conference proceedings and journals. As for the epidemiological surveillance sector, three papers have been published as the corresponding author:

- Marotta G, Billeci F, Criscione G, Merola F, Pappalardo G, and Tramontana E. *"NausicaApp: a hybrid decentralized approach to managing Covid-19 pandemic at campus premises"*, in 2020 Asia Conference on Computers and Communications (ACCC), 2020 Sep 18 (pp. 124-129). IEEE.

- Marotta G, Fornaia A, Moschitta A, Pappalardo G, and Tramontana E. *"NausiChain: a Mobile Decentralized App Ensuring Service Continuity to University Life in Covid-19 Emergency Times"*, in 2021 the 4th International Conference on Software Engineering and Information Management (ICSIM), 2021 Jan 16 (pp. 74-81).

- Fornaia A, Marotta G, Pappalardo G, and Tramontana E. *"A Decentralized Solution for Epidemiological Surveillance in Campus Scenarios"*, in IEEE Access, vol. 10, pp. 103806-103818, 2022, doi: 10.1109/ACCESS.2022.3208167.

On the renewable energy sector the published paper is:

- Calvagna A, Casablanca E, Marotta G, Pappalardo G, and Tramontana E. *"Providing Trust in a Dynamic Distributed Energy Production Scenario by means of a Blockchain"*, 2022 Workshop on Blockchain for Renewables Integration (BLORIN), Palermo, Italy, 2022, pp. 13-18, doi: 0.1109/BLORIN54731.2022.10028019.

At the time of writing, the paper describing the latest experimental outcomes in the renewable energy sector, as presented in the thesis, is being submitted to a specialized journal (namely, IEEE Transactions on Smart Grids). The paper's title and authors are reported below:

- Calvagna A, Marotta G, Pappalardo G, and Tramontana E. *"A Blockchain-Based Solution for Modulating the Energy Flows in a Smart-Grid"*.

The remaining published paper is a survey on typical issues of blockchain-based applications in the Electronic Health Records (EHRs) sector. The

main contribution provided is a classification of typical issues, such as users' anonymity, data access and security, identity management, that similarly traverse many corporate applications:

– Mandarino V, Marotta G, Pappalardo G, and Tramontana E. *"Issues Related to EHR Blockchain Applications"*, in 2021 2nd Asia Conference on Computers and Communications (ACCC), 2021 Sep 24 (pp. 132-137). IEEE.

The remaining of this document is structured as follows. Chapter 2 provides a comprehensive insight into the blockchain technology, with special focus on the Ethereum platform and the design principles of decentralized applications. A final section gives an overview of the Self-Sovereign Identity (SSI) approach. Chapter 3 collects the prominent work related to the discussed topics. Chapter 4 describes and discusses the main findings of the solution designed over the blockchain for the epidemiological surveillance sector. Chapter 5 deals with the solutions designed for the digitalization of the green energy market, whose main objective is the creation of a management system for decentralized energy resources. Chapter 6 draws the conclusions of the dissertation focusing on experimental and innovative aspects spread throughout this document.

# Chapter 2

# Background

This chapter provides the technical references that are essential to handle the background technologies employed throughout the dissertation. Sections 2.1 and 2.2 pinpoint the fundamentals of blockchain and smart contracts technologies, as well as their known limitations. Section 2.3 provides an overview of the decentralized application model. Finally, Section 2.4 gives some hints on the self-sovereign identity topic.

## 2.1 Blockchain fundamentals

A blockchain is a peer-to-peer distributed ledger that registers cryptographically signed transactions in a sequence of linked blocks, namely the chain, in an append-only fashion.

The main components of a typical blockchain can be listed as follows: (i) a P2P network that connects the blockchain nodes; (ii) a chain of cryptographically secured blocks that acts as a decentralized ledger; (iii) a set of consensus rules that validate and create the blocks; (iv) messages, in the form of transactions, that trigger state transitions; (v) a state machine that processes transactions according to a built-in script language.

It's a fact that the diverse existing blockchains differ from each other by the components included in their implementation and for other aspects that make them exhibit different behaviour in terms of openness, accessibility,

performance, programmability and security. These components are provided by the P2P nodes' software, also known as *client*. In Bitcoin the client implementation project is developed by the *Bitcoin Core* open source initiative[1], whereas in Ethereum, rather than a reference implementation, there exists a mathematical description of the client in the Yellow Paper [16], which allows a number of different developers to implement Ethereum clients according to the reference specification.

Typically, each of the chained blocks stores data, such as a timestamp, a nonce, a set of transactions and a cryptographic reference (*hash*) to the previous block. Since every hash is unique, every block is linked to a unique parent all the way up to the "genesis block", which is the first one in the chain. Once a valid block is created, each node connects it to the previous blocks and updates its own private copy of the blockchain, thus ensuring consistency with the other peer nodes' copies. The properties provided by the hash function render the blockchain immutable because if an adversary modifies even a single bit in a block, the block's new hash will be different and the hash pointer stored in the subsequent block will be incorrect. As a result, this technology makes the origin of data, their integrity, and their authenticity certain. Blockchain immutability ensures resilience against modification or removal of the transaction data stored in the blocks, thus making the blockchain tamper-proof and tamper-evident.

### 2.1.1 Consensus algorithms

Blockchain technology combines different well-known concepts such as digital signatures, cryptographic hashing functions, and decentralized consensus algorithms, which validate all the registered transactions without requiring a central authority. The addition of new data to the chain is in fact a decentralized activity carried out by all peer nodes running the agreed-upon consensus algorithm, whose global convergence ensures the reliability of the blockchain and the correctness of the data saved in the chained blocks.

The two most popular (up to know) blockchain platforms, namely Bit-

[1] `https://bitcoin.org/en/bitcoin-core/`

coin [2, 28] and Ethereum [16, 17], have been designed on the basis of the Proof of Work (PoW) consensus algorithm [29], which goes back to the "hash-cash" mechanism introduced by Adam Black [12] (see Section 1.2). PoW is a computationally intensive mechanism, based upon finding a nonce value, such that, when hashed together with the candidate block's data, the resulting hash value is smaller than a current target value set by the protocol. Generating this proof of work is known as *mining*. Conversely, every other node can easily verify the solution of the puzzle. If the verification process succeeds, then the node which found the nonce is rewarded and every other node adds the newly created block to their copy of the chain.

Due the decentralized nature of the PoW consensus, there is usually a short-term disagreement in time between nodes as to what the current state of the blockchain is. For this reason it is recommended to wait for a few more blocks before a confirmed transaction in a block can be regarded as permanently included in the blockchain. A new block is in fact added to the longest chain of blocks, which makes less likely that the newly added block will be "detached" from the longest chain. Mining is then a necessary cost to extend the chain and prevent malicious double-spend attempts from adding fake blocks to the chain.

The high energy consumption of PoW, while discouraging dishonest nodes from transmitting malicious blocks to the network, represents a major drawback that has led to the adoption of alternative consensus algorithms, such as Proof of Stake (PoS) and Delegated Proof of Stake (DPoS) [30], Proof of Authority (PoA) [31], and so on [9, 10]. Proof of Stake (PoS) is probably the consensus algorithm that is receiving the utmost attention from the blockchain community, as far as the right blend of transaction costs, peer reliability and protocol fairness is concerned. With PoS, in order to be elected as a candidate creator of the next block, the requirement for the nodes is to stake a certain amount of coins as a guarantee of upright participation to the mechanism. The higher the stake, the higher chances a user stands of being chosen. If elected nodes follow the protocol, they are rewarded, while if they behave in a dishonest way they would be penalized by the protocol by "burning" the coins at stake. The adoption of PoS favours scalability and

security features of the consensus mechanism implemented in the blockchain, but to the detriment of decentralization, in accordance with the *"blockchain trilemma"*.

This assertion, initially formulated by Vitalik Buterin, the co-founder of the Ethereum blockchain, is a condition that concerns security, scalability and decentralization. It states that any improvement in one of these three aspects will negatively impact on at least one of the other two [32]. This means that a fine tuning among these factors has to be performed when designing a blockchain system solution, for it is quite difficult to have a scalable, secure and decentralized blockchain at the same time. A trade-off must be reached and accepted in favour of at least two of these aspects [33]. Oddly enough, the blockchain trilemma has never received a formal definition in scientific literature, but is often cited in research works and surveys on blockchain features, especially on scalability issues [32, 34].

Unlike PoW, which performs well in decentralization and security, the PoA consensus algorithm is located closer to the scalability and security vertices of the "trilemma", since it requires much less computation than PoW by removing the "crypto-puzzle" contest among competing nodes. The task of block creation is executed by predetermined authorized nodes, namely *sealers* or *validators* in PoA acceptation [19].

Proof of Elapsed Time (PoET) [35] is instead a consensus algorithm that sacrifices security in favour of scalability and decentralization. At each cycle, validating nodes wait for a given random time. The first node for which the waiting time has elapsed is selected to validate a block. The election mechanism requires dedicated hardware and high trust among the validating nodes, since they can cheat to each other on the random generation of time intervals.

### 2.1.2 Blockchain types

A varied number of blockchain designs have emerged over the past years that can be classified according to two similar dimensions. The first one is about governance, that is to say the set of nodes allowed to participate to

the consensus mechanism, and it distinguishes between *permissionless* and *permissioned* blockchains. The former type allows every node joining the P2P network to become a block maker, whereas the latter restrict this chance only to a set of authenticated users. The second dimension, *public* versus *private* blockchains, determines whether or not to enable any node or client to access the information stored in the blockchain [36].

In a less granular classification "permissionless" often encompasses the "public" meaning, since it is quite unlikely that a permissionless blockchain might restrict public access to nodes. In this acceptation, in a public (permissionless) blockchain anyone can write the transactions, read data, or run a validator node. We will use the term "public" to mean also "permissionless", unless differently specified. For instance, an administrative office may want to use a *permissioned public blockchain*, which can restrict the registration of transactions to a few selected nodes, while making the stored data accessible to all clients, so as to provide transparency and auditability.

Private blockchains typically require permissions to access, and are usually managed by few nodes, often belonging to the company that created the blockchain network, which, in this case, can be also classifiable as *enterprise* blockchains. These may designed to be more scalable and secure, at the expense of decentralization. A private blockchain that, at the consensus level, is managed by equally-powerful validators, which could be owned by multiple organizations operating in the same industry, is called a *consortium* blockchain. In this permissioned typology, only parties that undergo a well defined Know Your Customer (KYC) process, as in normal business operations, can be authorized to operate a consortium node. While defective in transparency and disintermediation, enterprise and consortium blockchains better fulfill some crucial requirements of many corporate applications, such as data privacy, access control, transaction scalability, system performance and protocol updating[2].

Paradoxically, the launch of a public blockchain such as Ethereum has unveiled the potentials of DLTs in enterprise businesses and fuelled the devel-

[2]Since consortium and enterprise blockchains share similar administrative and application requirements, we will refer to one of the two types also meaning the other.

opment of a multitude of private consortium blockchain projects, i.e., Corda[3] and Quorum[4], which are suited to large financial sectors. For the time being, the Hyperledger project[5], launched in 2016 and hosted by the Linux Foundation, represents the most relevant effort to provide a comprehensive framework for the development of enterprise blockchains.

### 2.1.3 Transactions

A transaction is a fundamental component of a blockchain, since it represents the communication mechanism through which a value or an information is transferred between two blockchain parties, such as users and/or smart contracts. Each transaction that propagates through the blockchain nodes is verified on the basis of a predetermined set of rules, and only valid transactions can be selected for inclusion in a new block.

Blockchain users need to be identified so as to distinguish senders from recipients when a transaction is sent over the decentralized network. Such unique identifiers are known as *addresses*, which are usually public keys or derived from public keys. This mechanism should ensure users' pseudo-anonymity, even if they reuse the same address for multiple transactions. Unfortunately, some research in de-anonymizing blockchain users has shown that pseudo-anonymous users can be successfully identified by means of sophisticated linkability techniques applied to the addresses of dispatched transactions [37]. So, a single user would be better off generating a new address for each transaction, although this practice is mostly unpractical.

In first generation blockchains, such as Bitcoin, transmitted transactions are processed by interpreting special opcodes contained in the *transaction scripts*. The opcodes are a predefined sets of commands that nodes execute to transfer values from one address to another, as well as to perform other simple hard-coded operations. More general-purpose blockchains, such as Ethereum, provide for Turing-complete languages that overcome the limits of transaction scripts. However, the security of such languages in the immutable

[3] `https://www.r3.com/products/corda/`
[4] `https://consensys.net/quorum/`
[5] `https://www.hyperledger.org/`

context of the blockchain may raise severe security issues, which are the object of outstanding ongoing research. Replicated virtual machines are used to run the Turing-complete code generated by the "smart languages".

Regardless of the complexity of the native language, each blockchain can be regarded as a state machine whose overall state is modified from its initial configuration to a final one, as a result of the execution of a transaction performed by all the decentralized nodes.

### 2.1.4 Tokens

In common parlance the term "token" usually refers to coin-like items of poor intrinsic value, which are often restricted to usage within specific organizations and represent a single "physical" item. For these reasons they are not practically exchangeable. Conversely, "tokens" distributed on blockchains have given new meaning to the term, namely a blockchain-based abstraction that can be owned [17].

The most obvious use of "blockchain tokens" is as digital private currencies. In fact, blockchains enable the transfer of value between its users via tokens, as in the first blockchain currency, bitcoin (BTC), which is a token itself. Tokens are also used to reward the miners contributing to implement the PoW decentralized consensus, in which they are generated according to a standard cryptographic algorithm, which gives evidence to contributors (i.e., miners) of their dedication to the decentralized effort.

However, generating and transferring cryptocurrency is not the only possible use. Tokens can be programmed to serve many different functions, often overlapping. For example, a token can equivalently convey a voting right, an access right, an identity, an equity, a collectible, a resource ownership, and so on.

Tokens whose provenance cannot be tracked can be categorized as *fungible* if any single unit of the token can be exchanged for another without any difference in its value or function. Non-fungible tokens (NFTs) are instead tokens that represent a uniquely identified item, either tangible or intangible, and therefore they are not interchangeable (e.g., NFT collectibles). Tokens

representing digital items intrinsic to the blockchain, such as digital currency, are governed by consensus rules, thus not carrying additional risks coming from third-party intermediation. Alternatively, tokens can be also used to represent items external to the blockchain ecosystem, such as real estate, corporate voting shares, trademarks, and gold bars. The ownership of these items is governed by law, custom, and policy, so that token issuers and owners may ultimately depend on "real-world contracts" outside the "smart contract" jurisdiction.

One of the most important features of blockchain-based tokens is the ability to convert external assets into intrinsic assets and thereby remove the third-party intermediation risk. As an example, consider how a corporation equity could be transformed into a voting token in a DAO.

## 2.2 The Ethereum platform

Ethereum is a blockchain that shares many structural components with its forerunners, i.e., a logical P2P infrastructure to connect the network nodes, a consensus algorithm for the synchronization of state changes among all nodes[6], a set of cryptographic primitives to enforce security, and a cryptocurrency. However, this is rather intended as a *utility currency* for the usage of computing and storage resources of the distributed platform.

In fact, Ethereum has not been designed to be just another e-cash blockchain for digital currency transactions. Its co-founder Vitalik Buterin conceived it in late 2013, at a time when the criptocurrency fanatics were trying to build novel applications upon Bitcoin having to cope with the intentional constraints of the network and trying to find workarounds. For this reason, Ethereum has become the first blockchain that introduced a Turing-complete language and the concept of a virtual machine, which gives limitless possibilities for the development of smart contracts and decentralized applications. These new possibilities are provided within the fundamental properties of a public blockchain, namely immutability, transparency, auditability, and

[6] As of Q2 2022 Ethereum has transitioned from PoW to PoS - see `https://ethereum.org/en/developers/docs/consensus-mechanisms/pos/`.

neutrality.

Although devised for public contexts, Ethereum has turned out to be also well suited to the development of corporate solutions for the reasons discussed hereafter and throughout the dissertation. First of all, it is (like Bitcoin) open-source, so that any company can run its own private Ethereum-based implementation, in which the consensus mechanism can be adapted to the development requirements, suitable to both public or private environments (e.g., PoW can be, and indeed, as of today, has been, replaced by PoA). Moreover, Ethereum is coming up with native solutions to make the platform more enterprise-friendly and interoperable with the main consortium blockchains, such as Hyperledger. Finally, the Ethereum organization makes a few testnets publicly available and compliant to a large variety of experimentation needs.

For the above reasons and, as well, its mature programming and tooling technology, Ethereum has been chosen as the testbed technology over which the experiments, carried out in the solutions described in this document, are implemented, deployed and tested.

### 2.2.1 Ethereum clients

A node is a device or a program that communicates with the Ethereum network. In the acceptation of a program containing all the blockchain software components, nodes are also known as clients. Software that can act as an Ethereum node includes Go Ethereum[7] (Geth) and OpenEthereum[8] (previously Parity).

In general, we can divide node software into two types: *full nodes* (see fig. 2.1) and *light(weight) nodes*. Full nodes execute the decentralized consensus protocol, mine and verify blocks that are broadcast onto the network. That is, they ensure that the transactions contained in the blocks (and the blocks themselves) follow the rules defined in the Ethereum specifications by running all the instructions to make sure that they arrive at the correct, agreed-upon

[7] `https://geth.ethereum.org/`
[8] `https://openethereum.github.io/`

next state of the blockchain. Full nodes that preserve the entire history of transactions are known as full archiving nodes. These must exist on the network for it to be healthy.

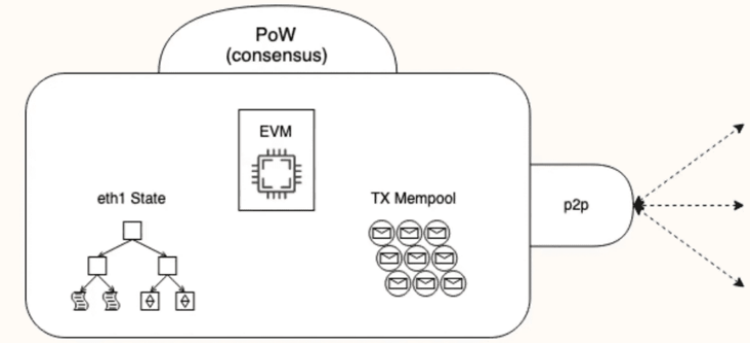


Figure 2.1: Simplified full-node client.

Full node clients can also provide wallet functionalities via specialized software allowing users to perform transactions on the blockchain (see Subsection 2.2.5). Furthermore, a node's specific software component, namely *Web3 Provider*, allows other programs, outside the blockchain ecosystem, to interact with the blockchain, as in a proxy. A full node also encompasses its own copy of the Ethereum Virtual Machine (EVM), which is so seen as a global, single-instance computer, with all its replicated instances reaching the same state.

Light nodes, in contrast, have a narrower scope than full nodes. In fact, they can validate only block headers and can prove the inclusion of transactions in the blockchain, thus determining their effects in the global state. On the other hand, they may not have a copy of the current blockchain state. They rely on full nodes to provide them with missing details. The advantage of light nodes is that they can get themselves up and running much more quickly in computational/memory constrained devices.

An even lighter alternative is to resort to a *remote client*, which does not store a local copy of the blockchain or validate blocks and transactions. These clients can handle users' accounts (EOAs), as well as create and broadcast transactions. They can be flexibly used to connect an external entity to full nodes of public or private blockchains.

For development and testing purposes only, a full node can be replaced by a *testnet node*, which provides access to a smaller public (or private) test blockchain.

### 2.2.2 Account-based model

As introduced in Section 1.2, Bitcoin never stores account balances in its blockchain, since it evaluates them according to the Unspent Transaction Output (UTXO) model, which is an on-the-fly computation of all the available digital money received from transactions sent to the account's owner and not yet spent. Conversely, Ethereum enforces its decentralized consensus around the concept of account state, which is referenced by its unique cryptographic address and maintains its balance and other related data in specific storage structures, as more extensively detailed in Subsection 2.2.7.

Ethereum opted for this more intuitive model for the benefit of developers of complex smart contracts, especially those that require state information or involve multiple parties. UTXO's stateless model would force transactions to include state information, thus unnecessarily complicating the design of the smart contracts. In addition to simplicity, the account-based model is more efficient, as each transaction only needs to validate that the sending account has enough balance to pay for the transaction. The potential drawback for this approach is the exposure to double spending attacks which can be counteracted by means of the "nonce" mechanism. In Ethereum, in fact, every account has a public viewable nonce and every time a transaction is made, this is increased by one. This mechanism can then prevent the same transaction being submitted more than once.

There are two types of accounts in Ethereum: (i) externally owned accounts (EOA), each of which is associated with a private key; and (ii) contract accounts, to which no private key is associated.

The former usually belongs to a user or a software agent, is handled by a digital wallet (see Subsection 2.2.5) and can generate transactions. The latter is controlled by the logic of the smart contract that is registered on the blockchain at the time the account is created. Since it is not associated to

any private key, it cannot generate any primary transaction. Both account types have associated addresses and balances and can send and receive ethers (ETH), Ethereum's native digital coins.

Global state transitions are always initiated by transactions generated by EOA accounts, which may trigger other chained transactions, and involve direct transfers of monetary values and/or information between accounts.

### 2.2.3 Transaction-driven state machine

As mentioned in Subsection 2.2.3, transactions play a pivotal role in the overall blockchain functioning. Due to the more complex transitions of the Ethereum state machine, transactions assume more extended functionalities than the ones executable on less smart platforms, such as Bitcoin. From the state machine perspective, a transaction is the only element that can trigger a change of state, from sending money to another account to causing a contract to execute in the Ethereum Virtual Machine (see 2.2.6). Architecture-wise, the EVM is a single-threaded, completely virtual machine with no system interface and scheduling features. It is also structured as a *stack-machine*, as opposed to a register-machine.

Upon the reception of a valid transaction, each node's replicated EVM is instantiated with all the information required to start the processing. At successful completion a new state of the global machine is reached. This behaviour justifies the definition of Ethereum as a *transaction-driven state machine*. If the execution does not succeed, the global state is reverted to the situation prior to the execution start. This is possible since the EVM runs in a sandboxed instance, which can be discarded in case or failure.

A key notion in EVM is the processing cost paid by the sender for the execution of transactions to the block creator, thus preventing infinite loops at run-time and hindering malicious attackers in their double-spending attempts. To quantify this cost, Ethereum has introduced a unit, called *gas*, through which the amount of computational effort required to execute each instruction on its virtual machine (*gas cost*) is measured. The overall cost of a transaction, called *gas fee*, amounts to the costs of the EVM instructions

making up the transaction. Gas costs of instructions are predetermined by the network and only alterable through a protocol upgrade. When users send a transaction, they must also set the *gas price*, i.e., the price they are willing to pay in *gwei*[9] for each *gas unit* of their transaction.

In addition to gas price, users set a maximum amount of gas units planned for execution, called *gas limit*. If an execution requires more gas than the gas limit specified by the user, a special out-of-gas (OOG) exception is thrown and the virtual machine's state is *reverted* to the one prior to the execution. In this case, the user will still have to pay the gas to the validating node as a countermeasure against potential Denial of Service attacks [38]. Note that a *block gas limit* is also enforced by the Ethereum's blockchain protocol, which provides an upper bound to the possible amount of data and processing that any transaction can incur during a smart contract execution.

To summarize, a transaction can be defined as a digitally signed message originated by an EOA, disseminated over the Ethereum network, validated and executed by the client nodes and eventually recorded on the Ethereum blockchain.

From a technical standpoint, a transaction is a serialized binary message, packed according to the proprietary Recursive Length Prefix (RLP) encoding scheme, which exhibits the following structure [17]:

- Nonce: a sequence number, issued by the EOA originating the transaction, which is taken from the EOA's account state and used to prevent double-spending;
- Gas price: the price of gas (in gwei) the originator is willing to pay for each gas unit;
- Gas limit: the maximum amount of gas units the originator is willing to spend for this transaction;
- Recipient: the 20-byte destination Ethereum address;
- Value: the amount of ethers to send to the recipient (it could be null);
- Data: the variable-length binary data payload that carries information related to a smart contract call (it could be empty);

[9] A gwei (gigawei) is worth $10^{-9}$ ETH

– Signatures: `v, r, s`, namely, the three components of an ECDSA digital signature [39] of the originating EOA, which, properly combined, provide the sender's public key and its associated address.

Transactions can carry value and data in the following three sensible combinations. A transaction with only value behaves as a traditional cryptocurrency transfer, that is a *payment* between two accounts. A transaction with only data is addressed to a contract account and is called a smart contract *invocation*. A transaction with both value and data is both a payment and an invocation and is addressed to a contract account. The meaning of an invocation's data payload will be explained in Subsection 2.2.6.

As for the propagation of transactions over the communication network, Ethereum uses a *flood routing* protocol within the logical P2P meshed infrastructure. Assume that an Ethereum node creates or receives a signed transaction. The transaction is then validated and broadcast to all the other Ethereum nodes that are directly connected (logically) to the originating node, called neighbors. Each neighbor node validates the transaction as soon as they receive it, store a copy and broadcast it to all its neighbors. As a result, within just a few seconds, an Ethereum transaction propagates to all the Ethereum nodes around the globe. Valid transactions will eventually be included in a block by miners and therefore recorded in the Ethereum blockchain along with changes to the global state ensuing the transaction execution.

### 2.2.4 Token standards

Although blockchain tokens existed before Ethereum, a multitude of new tokens has followed the introduction of the first standard for their creation from the Ethereum organization. Tokens are different from ethers because the former are outside the Ethereum protocol, though they can be associated to ethers. Sending or owning tokens, as well as handling token balances, are specifically dealt with at smart contract level.

The introduction of well-written new tokens in smart contracts are currently based on the *"ERC20 standard"*, which was the first standard, intro-

duced in November 2015 by Fabian Vogelsteller, as an Ethereum Request for Comments (ERC)[10]. ERC20 is a standard for *fungible tokens*[11], meaning that different units of an ERC20 token are interchangeable and have no unique properties[12]. This standard defines a common interface for contracts implementing a fungible token, thus allowing any compatible token to be accessed and used in the same way. The interface consists of a number of functions that must be present in every implementation of the standard, as well as some optional functions and attributes that may be added by developers. The ERC20 token standard does not (explicitly) track the provenance of any token.

Conversely, the ERC721 proposal[13] is a standard for *non-fungible tokens (NFTs)*, also known as *deeds*, which reflect the concept of "ownership of property". Currently, NFTs are not recognized as "legal documents", although things are moving towards a radical change whereby legal ownership, based on blockchain digital signatures, will be recognized. The ERC721 standard places no limitation on the nature of the "thing" whose ownership is tracked by a deed, provided that it can be uniquely identified by means of a 256-bit label.

Token standards represent the lowest common denominator for a token implementation. Thus, all wallets, user interfaces, and other software components can interface in a predictable manner with any contract that follows the specification. Another major goal is to favor interoperability between contracts. From a development point of view, these standards are meant to be descriptive, rather than prescriptive, thus making the implementation of the contract creating the token not relevant.

[10] `https://eips.ethereum.org/erc`

[11] `http://bit.ly/2CUf7WG`

[12] The ERC20 eventually became Ethereum Improvement Proposal 20 (EIP-20), but the two denominations are interchangeable.

[13] `http://bit.ly/2Ogs7Im`

### 2.2.5 Digital wallets

A digital wallet is a client software used to store private or public keys and network addresses. It also performs various operations, such as receiving and sending cryptocurrency. The notion of digital wallet dates back to Bitcoin and it has been taken up by most ensuing blockchains with similar features.

There are different types of wallets that can be used to manage private keys and carry out transactions on the network. Deterministic wallets derive keys out of a seed value via hash functions. The seed is generated randomly and is commonly represented by human-readable mnemonic code words, as defined in BIP39[14] [15]. Hierarchical deterministic wallets, defined in BIP32[16] and BIP44[17], generates the master key from a seed, which in turn generates relative keys. The complete hierarchy of private keys in such a wallet type is easily recoverable from the master private key.

Wallets also differ for the device or the software chain in which they are used. Hardware wallets, for instance, are tamper-resistant devices used to store keys. The device can be custom-built or can also be a secure element in NFC phones[18]. Online stored wallets, on the other hand, provide the users with a web interface to manage their accounts and perform the typical wallets operations. Finally, mobile device wallets can provide various methods to make payments, most notably the ability to use smart phone cameras to scan QR codes.

In Ethereum, a "wallet" can be conveniently defined as a generic program that stores private keys and manages their associated accounts. Similarly to Bitcoin wallets, which do not hold cryptocurrency, Ethereum ones do not contain ethers or tokens either. According to the account-based model (described in Subsection 2.2.2), the ethers or other tokens are in fact recorded on the Ethereum blockchain in the State Trie (see Subsection 2.2.7, so that

[14]Bitcoin Inprovment Proposals (BIPS) suggest guidelines for how Bitcoin can and should evolve.

[15]`https://github.com/bitcoin/bips/tree/master/bip-0039`

[16]`https://github.com/bitcoin/bips/tree/master/bip-0032`

[17]`https://github.com/bitcoin/bips/tree/master/bip-0044`

[18]Near-field communication (NFC) protocols enable communication between two electronic devices over a distance of 4cm or less.

the wallet software can only retrieve the information about any account's balance by querying the blockchain.

In a nutshell, an Ethereum wallet is a specialized software application to manage EOAs. It serves as the primary user interface to manage keys and addresses, track the balance, create and sign transactions to be transmitted over the P2P network. In addition, some Ethereum wallets can also interact with contracts, such as ERC20 tokens.

### 2.2.6 Ethereum's smart contracts

As repeatedly mentioned, Ethereum is the first distributed ledger technology that has introduced the possibility to run smart contacts on its peer nodes. Each smart contract is identified by a unique address that references an account to which digital coins and storage space are linked. At low level, any smart contract is a bytecode that is sequentially executed by the replicated virtual machine once it has been invoked by a transaction sent to its address.

The language complexity of smart contracts and the virtual machine architecture make up a Turing-complete system that could potentially run in infinite loops and never terminate, potentially freezing the entire network. To prevent this, Ethereum has introduced the notion of *gas*, which has previously been discussed in Subsection 2.2.6.

Smart contracts are typically written in a high-level language, such as Solidity[19]. Once compiled, they are deployed on the Ethereum platform by using a special transaction, deliberately coded for contract creation, which has some peculiar features. First of all, contract creation transactions are sent to a special destination address called the zero address, that is, the recipient field, in the transaction structure, carries the address 0x0. The contract creation transaction hauls, in its data field, the smart contract's bytecode that has to be registered in the designated blockchain's structure referable to the storage account (see Subsection 2.2.7). It is also possible to include an amount in ethers, in the value field, in order to feed the new contract with an initial balance.

[19] `https://docs.soliditylang.org/en/latest/`

If standard transactions deliver data to a contract address, the EVM will interpret this occurrence as a smart contract invocation, more specifically as a single invocation of a public function contained in it. In this case, the data payload is a serialized hexadecimal encoding of: (i) a 4-byte function selector obtained by hashing the function's prototype[20]; (ii) the function arguments which are passed to the invoked functions according to its prototype.

Chained calls of smart contracts can programmatically occur. However, the first contract of the chain execution has to be triggered by a transaction generated by an EOA. In fact, smart contract can never run on their own.

It is essential to pinpoint that a contract's code is immutable and can never be patched. However, it is possible to delete a contract's code from its address along with the related account state. Deleting a contract will cause a gas refund to the caller, thus fostering the release of node resources. It must also be noted that this operation does not delete the contract's transaction history, which is immutably stored in the non-ephemeral section of the blockchain storage, as better explained in Subsection 2.2.7.

Upon transaction completion, a transaction receipt is produced, which also contains *log entries* providing information about the actions performed by the contract's execution. In particular, a series of up to four topics correspond to special high-level objects, named *events*, generated by the Solidity code. These log entries can be regarded as a cheaper form of storage than the contracts' one. Moreover, these event-created logs, which are permanently recorded in the blockchain (regardless of the possible deletion of the originating contract), can be be watched and retrieved by external applications that interface with them via specific programmatic mechanisms. The design of this series of four topics provides for research indexes, which facilitate the browsing and the retrieval of information generated by the associated contracts. Further details of the event watching mechanism will be provided in Section 2.3.

Both functions' invocation and events' retrieval leverage the specifications contained in the contract's Application Binary Interface (ABI), which

[20]The function's prototype id defined as the string containing the name of the function, followed by the data types of each of its arguments.

provides a JSON-like interface that makes the communication among heterogeneous software components possible. A typical example is when the front-end of a decentralized application wants to interact with a smart contract permanently stored in the Web3 ecosystem. Further details on the ABI structure are given in Subsection 2.3.1.

### 2.2.7 Ethereum's blockchain

The layout of the Ethereum's blockchain is much more complex than Bitcoin's since, alongside the expected blocks, it contains particular data structures, called *tries*[21] that collectively maintain the Ethereum global state [40]. Ethereum implements a particular trie version, known as the Patricia Merkle trie[22], because of its compactness and simplicity.

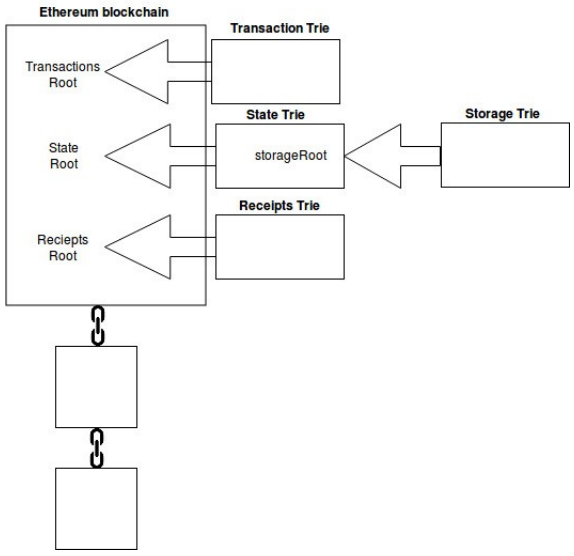


Figure 2.2: Ethereum's blockchain

Each trie manages different portions of the Etherum's storage, namely the Transaction Trie, the State Trie, and the Receipt Trie, which also differ based on the volatility of the data they handle. In fact, Ethereum's blockchain

[21]The trie is a data structure which is used for storing and retrieving sequences of characters.

[22]`https://ethereum.org/en/developers/docs/data-structures-and-encoding/patricia-merkle-trie/`

deals with two different types of data: *permanent* and *ephemeral*. An example of permanent data, which is what typically expected in an immutable blockchain, would be a transaction that is recorded in the Transaction Trie. This trie is never altered. Conversely, an example of ephemeral data would be the balance of an account, which is stored in the State Trie and is altered whenever a transaction deposits to or withdraws money from that particular account. The State Trie is the only portion of the Ethereum's blockchain that can be altered.

Figure 2.2 shows that a huge part of the information is not directly stored in the blocks of the Ethereum's blockchain, since the tries are outside it. Only the root nodes' hashes of the different tries, which act as a reference to them, are stored inside the blocks. In turn, the root node's hash of the Storage Trie (where all of the smart contract data is kept) points to the State Trie.

Each Ethereum block has its own separate Transaction Trie (see Fig. 2.3), which contains all the transactions included in the block. The order of the transactions in a block is decided by the miner that assembles the block and it determines how the EVM schedules the execution of the transactions once a block is mined.

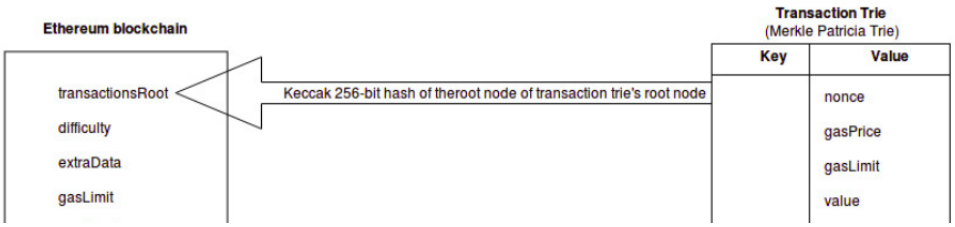


Figure 2.3: The Transaction Trie: one per each block

The transaction receipts emitted upon execution of each transaction of the block, as explained in 2.2.6, are stored in the permanent Receipt Trie associated to the block.

The root node's hash of the ephemeral State Trie, which is the hash of the entire trie at a given point in time, is cryptographically dependent on all internal State Trie data, and it is so used as unique reference to the trie at the time the block was created and all the transactions were executed.

The State Trie (see Fig. 2.4) contains a *key-value* pair for every account

which exists in the Ethereum network. The *key* is a single 20-byte identifier, namely the address of an Ethereum account, whereas the *value* is created by RLP-encoding the following account details of an Ethereum account: nonce; balance; storageRoot; codeHash.

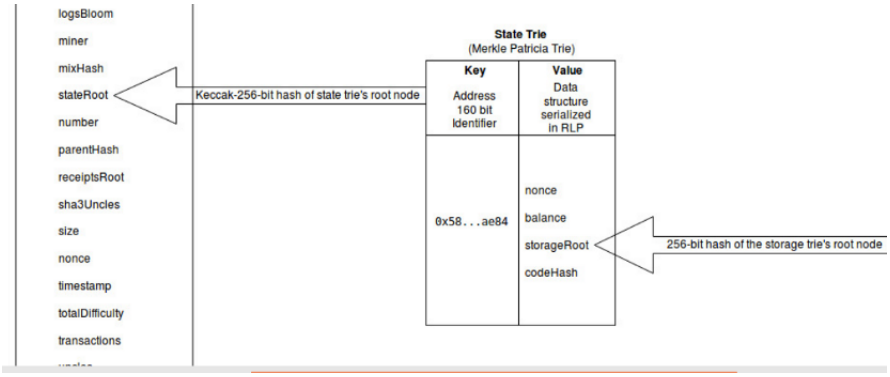


Figure 2.4: The State Trie: one per blockchain.

As previously stated, the *nonce* keeps the next value of the nonce field that a transaction, created by the associated account, can use to prevent the double-spend issue. The *balance* contains the current amount of digital currency (if any) held by the account. The next two fields are meaningful only in case of a smart contract account. In fact, *storageRoot* contains the root node's hash of the account's Storage Trie (see Fig. 2.5) where the data handled by the contract are stored in a key-value RLP encoding. Finally, *codeHash* stores the code's hash of the associated smart contract code.

### 2.2.8 Open issues

Ethereum's blockchain technology, in its constantly evolving stages, tries to address several issues.

A major one, especially in public blockchains such as the PoW Ethereum, is represented by transaction costs. These can drastically rise during network congestion, when the average price of gas tends to increase, because users will incentivize the mining nodes by offering above-average gas prices. Miners, in turn, in order to maximize rewards, will prioritize transactions that offer higher gas prices, leading to an increase in fees. Transaction fees can

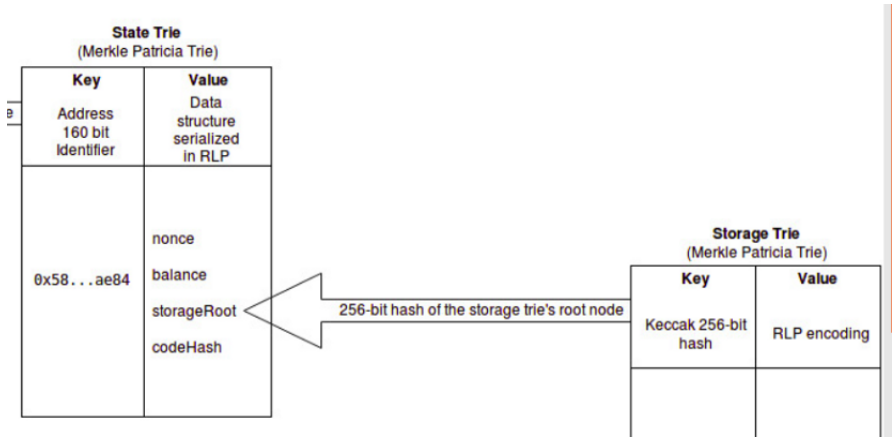


Figure 2.5: The Storage Trie: one for each contract's account.

range from less than one dollar to hundreds of dollars and have exhibited an uptrend until transition to PoS[23]. The transaction costs depend on a few factors, some controlled by the blockchain protocol, namely the gas units related to transaction execution, while others are affected by fluctuations of internal and/or external markets. For instance, in Ethereum, the gas price per unit is a matter of supply/demand among miners and users. It could be dependant on the kind of consensus algorithm implemented in the blockchain, since less-energy intensive protocols, such as PoS or PoA, could lead to a lower reward request from the validating nodes. Other factors, such as the fluctuation of the cryptocurrency exchange rate, on the monetary market, involve economic aspects that go far beyond the sole issue of internal supply/demand of computing power.

In the domain of "smart" blockchains, program bugs trigger some peculiar software development challenges. Due to the immutability feature of blockchains, once a smart contract has been deployed on the blockchain, it cannot be modified (although it can be deleted). Consequently, patching a deployed contract is impossible, and for this reason, a significant number of smart contracts are considered to be vulnerable. In 2016, a symbolic execution analysis tool, specifically developed to identify potential vulnerabilities in the entirety of the smart contracts deployed on the Ethereum blockchain

[23] `https://blockchair.com/ethereum/charts/average-transaction-fee-usd/`

up to that point in time, showed that 45% of 19,366 smart contracts were vulnerable with at least one security issue [41]. A few years ago, Ethereum suffered a supposedly involuntary hack, which caused an inexperienced developer to freeze multiple accounts with an estimated amount of ethers around 513,774 (equivalent to $162M at that time) managed by the Parity Wallet application. In July 2017, a hacker exploited the "Parity Wallet hack" and managed to hack a multi-signature[24] wallet taking full control of it. The Parity Wallet hack is a paradigmatic example of the issues, associated with the lack of properly standardized best practices, that may occur in smart contract development.

With the adoption of the ERC20 token standard, thousands of tokens have been launched especially to raise funds in various Initial Coin Offers (ICOs)[25]. Tokens are meant to function just like ethers, but they come with certain differences that compromise their secure handling. While ethers are transferred via a transaction that has a recipient address as its destination, token transfers occur within the specific token contract state and have the token contract as their destination, not the recipient's address. Furthermore, the token contract tracks balances and emits events to the transaction logs. Even ERC20-enabled wallets do not become aware of a token balance unless a specific token contract to watch is added by wallets' users. Another issue with ERC20 is the need to pay in ethers to send tokens, since it is not possible to pay for a transaction's gas with a token. This may change at some point in the future, but in the meantime this can cause conceptual and practical anomalies in the way users deal with tokens. Some of the previous issues are specific to ERC20 tokens, but others are more general and relate intrinsically to the Ethereum protocol. Fixing them requires changes to fundamental structures within Ethereum, such as the distinction between EOAs and contracts, and between transactions and messages.

Further issues, in terms of scalability and security, can be ameliorated if a more appropriate consensus algorithm is chosen, such as PoA in private Ethereum implementations [42]. However, according to the "blockchain

[24] "Multi-signature" refers to requiring multiple keys to authorize a transaction.

[25] `https://www.investopedia.com/terms/i/initial-coin-offering-ico.asp`

trilemma", such a choice will lower the value of the decentralization dimension. Many decentralized applications, however, in the vast arena of industry and administration, can cope with a downsized level of decentralization more easily, without prejudice to the other features that make Ethereum a suitable platform for enterprise solutions.

## 2.3 Decentralized Applications

A decentralized application (DApp) is typically an Internet application in which (part of) the back-end runs on a blockchain. A decentralized consensus algorithm is in fact vital to detect and prevent peers from making invalid changes to the application data and sharing wrong information with others. The *Web3* paradigm, introduced by Ethereum's co-founder Gavin Wood [16], is purposely meant to address the need for a new ecosystem for DApps development. Smart contracts are just a way to decentralize the application logic, whereas *Web3 DApps* are about decentralizing all other aspects, such as storage, messaging, naming, and so on.

While the vision of totally decentralized applications may take longer to establish, a minimal DApp can be as complex as a stand-alone JavaScript application that relies on the blockchain for its data and logic functionalities. However, the ultimate purpose of a DApp is to provide a rich client application that can save data locally and use multiple back-end services to achieve decentralization, included smart contracts and other blockchain functionalities.

A typical architecture of a complete DApp is depicted in Fig. 2.6.

The client's front-end usually runs in the user device as a JavaScript, or other similar languages, application. It interacts with the blockchain (both smart contracts and other data structures) for core data and back-end's business logic [43]. It could also rely on local or public storage services to manage large amounts of off-chain data, which can optionally be replicated in a decentralized manner. The DApp's client can be stored in any web server, thus eschewing single points of failure for the front-end as well.

To effectively deploy a DApp on the Web, further architectural compo-

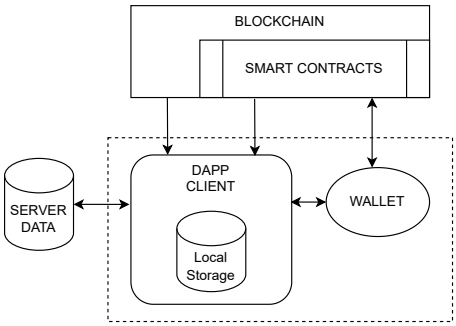


Figure 2.6: DApp architecture

nents are needed, such as a *Web3 Provider* and a *digital wallet*. The detailed analysis on the back-end's blockchain technologies, such as smart contracts, blockchain data structures, node access mechanisms, will be carried out with reference to the Ethereum technology. Fig. 2.7 shows a viable configuration of a DApp ecosystem in Ethereum, whose elements will be discussed in more details in the remainder of the chapter.

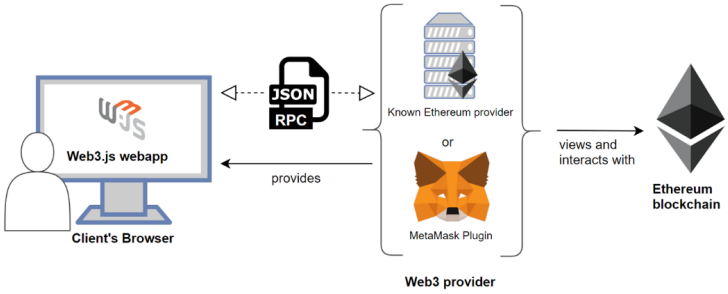


Figure 2.7: DApps' Ethereum ecosystem

### 2.3.1 Solidity contracts

As already discussed in Subsection 2.2.6, a smart contract is typically written in a high-level language and then compiled into the native EVM bytecode. The contract is then deployed on the blockchain by means of an EOA-generated contract creation transaction. The EOA that creates a contract never gets ownership on it, although this right can be explicitly coded in the contract.

Solidity is an object-oriented, high-level language that was initially proposed by Ethereum's co-founder Gavin Wood to write smart contracts with features to directly support execution in the decentralized environment of the Ethereum world computer. It is also used for coding smart contracts on several other blockchain platforms, such as Hyperledger Fabric. Solidity is a curly-bracket language, mainly influenced by JavaScript, C++, and Python, It is statically typed, supports inheritance, libraries and complex user-defined types, including arbitrarily hierarchical *mappings* and *structs*.

The Solidity compiler, namely *solc*, converts programs written in the Solidity language to EVM bytecode and also provides the Application Binary Interface (ABI) of Ethereum smart contracts. The ABI is essential to provide front-end languages with a JSON-coded interface that exposes access to data structures and machine code functions. Solidity follows a versioning model called *semantic versioning*[26], which specifies version numbers structured as three numbers separated by dots: major.minor.patch. The latest version, at the time of writing, is v0.8.17. Solidity programs contain a pragma directive that specifies the minimum and maximum versions of Solidity that it is compatible with the contract to be compiled.

Let's have a look at the Solidity language by means of a simple example that sets and changes ownership to a deployed smart contract[27]. This toy contract does not take into account programming flaws that can make it vulnerable to security threats.

```
// Version of Solidity compiler this program was written for
pragma solidity >=0.4.22 <0.7.0;
 // set and change contract's owner
contract Owner {
    // allocate contract's state variable
    address private owner;
    // event for EVM logging
    event OwnerSet(address indexed oldOwner, address indexed newOwner);
    // modifier to check if caller is owner
    modifier isOwner() {
            require(msg.sender == owner, "Caller is not owner");
        _;
    }
    // set contract deployer as owner
```

[26] https://semver.org/

[27] For the complete Solidity syntax refers to the official documentation site at: https://docs.soliditylang.org/en/latest/index.html

```
    constructor() public {
        // 'msg.sender' is sender of current call, contract deployer for a
            constructor
        owner = msg.sender;
        emit OwnerSet(address(0), owner);
    }
    // change owner
    function changeOwner(address newOwner) public isOwner {
        emit OwnerSet(owner, newOwner);
        owner = newOwner;
    }
    // return owner's address
    function getOwner() external view returns (address) {
        return owner;
    }
}
```

Listing 2.1: A very simple smart contract to change contract's ownership.

A quick code analysis reveals that many elements of Solidity are similar to existing programming languages, such as JavaScript, Java, or C++. After the pragma declaration, the first keyword declares a *contract object*, quite similar to a class declaration in other object-oriented languages. In the example, in fact, there exists a `constructor()` that assigns, at creation time, the contract's ownership to the sender `(msg.sender)` of the contract creation transaction, which is actually a *message* in Ethereum terminology. As anticipated in Subsection 2.2.6, no ownership right is in fact automatically assigned to the contract's creator by the Ethereum protocol. The contract's owner is kept in the private variable `owner`, which represents the contract's state at each execution. This variable is written in the Storage Trie associated with the contract's account (see Subsection 2.2.7) and it is *ephemeral* by nature.

The other two functions serve to allow an external program to dynamically change the contract's owner (`changeOwner()`), and to return the current contract's owner to any function caller (`getOwner()`). In functions' declaration the attributes `public` and `external` create automatic handler for external programs to directly interact with them.

Also notice the reserved word `event` and its associated data structure. The information in the event structure, created by the contract via the command `emit`, will be permanently stored in the transaction's logs of the associated Receipt Trie (see Subsection 2.2.7). Besides being permanent, even upon later deletion of the generating contract, these logs are a cheaper form

of storage and can be effectively browsed, by means of search filters, by front-end programs, external to the blockchain, at any time in the future. These is why smart contracts should ever publish (emit) their outcomes as events.

As already explained, the Solidity compiler creates, as an artifact, the ABI associated to the contract, which is specified as a JSON array of function descriptions and events. Therefore, the purpose of an ABI is to define the way each function will accept arguments upon invocation and how it will return its result.

For the contract in Listing 2.1, the corresponding ABI JSON-array is the following.

```
[
    {   "inputs": [],
        "stateMutability": "nonpayable",
        "type": "constructor" },
    {   "anonymous": false,
        "inputs": [
            {   "indexed": true,
                "internalType": "address",
                "name": "oldOwner",
                "type": "address" },
            {   "indexed": true,
                "internalType": "address",
                "name": "newOwner",
                "type": "address" } ],
        "name": "OwnerSet",
        "type": "event" },
    {   "inputs": [
            {   "internalType": "address",
                "name": "newOwner",
                "type": "address" } ],
        "name": "changeOwner",
        "outputs": [],
        "stateMutability": "nonpayable",
        "type": "function" },
    {   "inputs": [],
        "name": "getOwner",
        "outputs": [
            {   "internalType": "address",
                "name": "",
                "type": "address" } ],
        "stateMutability": "view",
        "type": "function" }
]
```

In the decentralized applications paradigm, the ABI is a data structure that any front-end program must include to exchange data with the smart contracts' function and the blockchain's logs.

In addition, Solidity exposes the set of global objects that a contract may

need to access during its execution in the EVM. These include the `block`, `msg`, and `tx` objects. In the shown contract code, `msg.sender` represents the EOA's address that initiated this contract call by signing the contract creation transaction. Finally, Solidity includes a number of EVM opcodes as predefined functions, such as `selfdestruct(recipient_address)`, which deletes the current contract from the blockchain.

### 2.3.2 Web3 Providers

Among the Ethereum node's client functionalities, the provision of a specific software component allows other programs, outside the blockchain ecosystem, to interface with smart contracts and other blockchain structures. An Ethereum client also acts as a *Web3 Provider*, that is, a gateway between the Web's front-end and the Web3's back-end. A Web3 Provider offers in fact an Application Programmable Interface (API), provided via a set of Remote Procedure Call (RPC) commands, which allows the DApp's front-end to interact with the Ethereum blockchain (see Fig. 2.8).

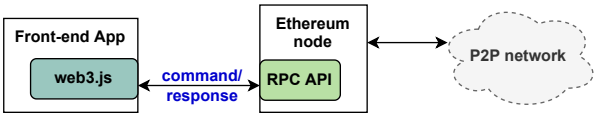


Figure 2.8: Programmable Web3 interface.

The RPC commands are encoded in JavaScript Object Notation (JSON); the API is then referred to as the JSON-RPC API. Usually, the RPC interface is offered as an HTTP service on localhost port 8545, but it can be done though other communication protocols (IPC, WebSocket). Native access to the JSON-RPC API can be made by means of a generic command-line HTTP client (e.g., Curl) by sending the request message to the Ethereum client and receiving the response message from it.

In the following example a request is addressed to the available Web3 Provider to know the client version the external application is talking to on port 8545.

```
// Request
curl -X POST -H "Content-Type":"application/json" --data '{"jsonrpc":"2.0","
    method":"web3_clientVersion","params":[],"id":1}' http://localhost:8545
// Response
{"id":1, "jsonrpc":"2.0", "result":"Mist/v0.9.3/darwin/go1.4.1"}
```

Interfacing with the blockchain at such a native level can be a chore for application developers. Therefore, high-level libraries for JavaScript and other front-languages are available to hide the cumbersome mechanism of JSON-RPC API, as more extensively detailed in the next subsection.

The Web3 Provider service can be provided not only by full Ethereum clients, such as Geth or OpenEthereum, but also by remote clients (see Subsection 2.2.1), such as Metamask[28], or cloud-based clients, such as Infura[29].

MetaMask is in fact a plug-in browser extension which, in addition to wallet functionalities (see Subsection 2.2.5), provides web-based tools and apps with Web3 access to different Ethereum networks (e.g., live, testnet, custom).

### 2.3.3 Front-end operations

Fig. 2.9 shows a Dapp's front-end that connects to any Ethereum client (full, remote, cloud-based) by means of Web3 high-level libraries. It needs to interact or include a wallet software to get transactions signed and EOA accounts managed.

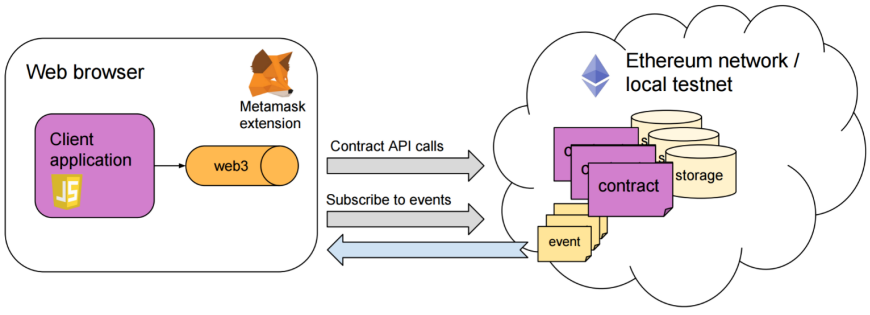


Figure 2.9: Front-end / back-end interaction in DApps.

[28] https://metamask.io/
[29] https://infura.io/

A front-end application is typically either a traditional web app written in HTML/CSS/JS or a Node.js[30] application. Even if less common, standalone front-end applications can be developed in other programming languages, such as Java or Kotlin for mobile applications. For each selected language a specific Web3 high-level library must be imported in the source code (e.g., web3.js, ethers.js, web3j).

*web3.js* is the most common convenience JavaScript library providing a Web3 object that can be used to interact with an Ethereum client. It works by exposing high-level methods and objects that have been enabled over RPC, thus allowing the developer to avoid using the underlying JSON-RPC API encoding.

## 2.4 Digital identity

The treatment of digital identities is at the foundations of the Know Your Customer (KYC) principles in digital financial services, which entail strict requirements when it comes to user authentication in any multi-party relationship. The ensuing procedures are employed for the purpose of ensuring that the actors involved in a digital interaction are actually who they claim to be. The topic of proving digital identity on the global network is being addressed, with varying levels of decentralization and security requisites, by several internet-based systems (e.g., OpenID Connect[31], KYC-chain[32]).

In a fully decentralized approach, Identity Access Management (IAM) policies should be resistant to censorship, meaning that it doesn't exist any privileged party in the system that has the power to selectively limit the information that others have access to. A blockchain, by its very nature, meets these requirements [44]. In fact, some platforms have been specifically designed or complemented with features to support the implementation of decentralized identity policies, e.g., Namecoin[33], Hyperledger Indy[34], Energy

[30] `https://nodejs.org/`
[31] `https://openid.net/connect/`
[32] `https://kyc-chain.com/`
[33] `https://www.namecoin.org/`
[34] `https://www.hyperledger.org/use/hyperledger-indy`

Web Decentralized Operating System[35], the latter being used in the original DApps presented in this work at Chapter 5.

### 2.4.1 Self-Sovereign Identity

Self-sovereign identity (SSI)[36] identifies the digital movement that promotes the idea about individuals owning and controlling their identities, as opposed to a typical administrative paradigm where most of our official identifiers (driver's license, birth certificate, usernames, etc.) are issued and maintained by a central authority, and where individuals' data can be shared without their knowledge or consent. The SSI paradigm envisages identity systems where individuals can maintain their own digital identities, and where data verification and sharing can occur via a secured digital consent [45].

Therefore, SSI platforms tend to be designed under a decentralized paradigm, since digital credentials are better handled using typical distributed ledger technologies, such as digital wallets, trustless consensus mechanisms, and immutable storage and business logic [46]. In this respect, the Sovrin Foundation[37] is an independent organization established to ensure that the self-sovereign identity is transparently public and pervasively accessible. It governs the Sovrin Networks, a public service utility providing the realization of the SSI principles on the Internet.

### 2.4.2 Decentralized Identifiers and Credentials

Decentralized Identifiers (DIDs)[38] and Verifiable Credentials (VCs)[39] are two of the most important components of SSI, so much that their core architectures, data models, and representations are being standardized as technical recommendations by the W3C organization, which establishes guidelines and best practices for an open, inclusive and trustworthy web.

[35] `https://www.energyweb.org/`
[36] `https://en.wikipedia.org/wiki/Self-sovereign_identity`
[37] `https://sovrin.org/`
[38] `https://www.w3.org/TR/did-core/`
[39] `https://www.w3.org/TR/vc-data-model/#core-data-model`

A DID is a digital, verifiable identity that is user-generated and not coupled to any centralized institution, which is entirely controlled by individuals or organizations without an external authority. It can be used to identify any subject, such as a non-tangible asset, a customer, or an organization.

Unfortunately, very little is known about the subject through a digital identity alone. This is why VCs need to come into play. They are secure and machine-verifiable digital credentials, that is, something we can prove about a subject or individual (e.g., a certificate, a property, an expiration date, etc.), which are standardized as statements or claims to be be associated to DIDs. VCs are secured through cryptographic protocols (e.g., a digital signature) and validated through a digital trust mechanism (e.g., a blockchain's consensus algorithm). Once a VC is successfully verified, it can then be used as an official record linkable to the subject's digital identity, thus enriching its set of associated claims.

# Chapter 3

# Related Work

In this chapter, multiple contributions that have been consulted and compared with the topics discussed in the thesis are presented and commented. Comparisons with the original solutions being devised during the doctorate's placement will be detailed in the relevant chapters.

The first two sections of the chapter collect the related work lying within the contract tracing and renewable energy application fields, respectively. Moreover, an additional section classifies the related work according to the technical issues the use of blockchain intends to mitigate or, when applicable, to the specific design patterns developed or used in the described solutions.

## 3.1 Contact tracing solutions

Since the outbreak of the Covid-19 pandemic, one of the main issues to be faced in digital solutions has been that of maximising the tracing of contacts among potentially Covid-19 positive individuals, while, at the same time, preserving their privacy. Many countries and public institutions have developed and deployed contact tracing solutions that differ especially due to the balance between an effective response to the pandemic emergency and the upholding of individual rights and liberties. Design choices, in the discussed solutions, mainly concern the *degree of decentralization* and the *user communication channels*. While the overall *system performance*, and the

*effectiveness* of the solutions are undoubtedly relevant parameters to measure, the goal of paramount importance is nevertheless the degree of *privacy preserving* that can be guaranteed to users' data and identities.

Contact tracing solutions basically consist of two components: (a) mobile apps to be installed on user personal devices, and (b) an administration server. The computational and communication functionalities, as well as the storage capabilities of both components, depend on the adopted solution, as discussed below, and may vary according to cultural and political backgrounds. Centralized solutions, for instance, which are often based on the *absolute location* of their users, rely on the trustworthiness of central servers and authorities, but commonly elicit privacy concerns [47]. They are mainly targeted at maximising efficiency in the contact tracing follow-up of Covid-19 positive people (e.g., mobility patterns, contagion risks, location monitoring). In contrast, decentralized solutions, which most of the times depend on the users' *relative location*, cannot provide a global view of contact tracing but guarantee a deeper safeguard of individual privacy [48].

The degree of decentralization of such solutions also depends on the integration of specific blockchain components into the overall system design, so that we can divide the analyzed solutions into two subcategories: *blockchain-unequipped* and *blockchain-equipped*.

### 3.1.1 Blockchain-unequipped

**EPIC** is a solution for *indoor environments* designed and prototyped by Altuwaiyan et al. [49]. It uses hybrid short-range wireless technologies, namely WiFi and Bluetooth, with the aim to provide fine-grained response to direct-only contact tracing goals. EPIC is mostly centralized in that, even though localization data are collected and stored at client side, they are processed at server side to assess if contacts with infected users have occurred, and if these have to be regarded as critical. Because of this architectural choice, much emphasis is placed on cryptographics techniques.

**WifiTrace** is a *network-centric solution* for contact tracing that relies on passive WiFi sensing with no client-side involvement [50]. It has been de-

signed and prototyped for *campus environments*, and is mainly used for post-processing device trajectories and reconstructing on-campus people flows. It strongly relies on passively collected WiFi enterprise network logs. This approach assumes central and third-party authorities to be fully trusted. Similarly, **vContact** [51], which uses WiFi to recognize smartphones, provides for a mobile app to store location data and check whether a location later marked as infected had previously been visited by app's users.

Singapore's **TraceTogether**[1] counts on *centralized server capabilities* for the purposes of contact tracing and contagion risk advertisements to individuals who have come close to reported infected people. Even though relative localization data are collected at user side and cryptographically exchanged with the server, once users are reported positive, both they and their contacts will be de-anonymized by the server, owned by the Ministry of Health. This highly beneficial choice for the contact tracers does not guarantee the right to anonymity at all, and yet is considered absolutely legal in Singapore [52], whereas the adoption of a similar solution in Australia (i.e., **COVIDsafe**[2]) has had to undertake some steps to address a few legitimate privacy concerns.

The South Korean Corona-100m, or **Co100** for short, is by far the most centralized and the least privacy preserving one among the solutions analysed so far. Absolute localization data are collected from GPS or cellular networks by user devices and sent to the central authority with the aim to provide a *publicly available website*[3], which can help people to find out where infected (anonymized) people have been in the previous days. Also, de-anonymized data belonging to confirmed Covid-19 patients can be immediately provided to health investigators [53].

Decentralized Privacy-Preserving Proximity Tracing (**DP3T**) is a seminal proposal [54] for the majority of *decentralized solutions* which have followed, since it states the principle that storing and processing of direct contacts are under mobile apps' responsibility, thus leaving the ownership of the collected data with the end-user apps. In this proposal the central server only plays

[1] `https://www.tracetogether.gov.sg`
[2] `https://www.health.gov.au/resources/apps-and-tools/covidsafe-app`
[3] `https://coronamap.site`

the role of a relay system whenever a user self-declares as Covid-19 positive. DP3T employs a dedicated *Bluetooth app-to-app communication channel* for direct exchange of proximity data, which are retained by each app in its local storage for later processing, upon notification from the central server, of contact matching. Many country solutions, such as Italy's **Immuni**[4], share a similar design, with the apps exchanging and storing anonymized IDs and other location data, reporting infected cases and tracing suspicious contacts.

The decentralized solutions under the DP3T umbrella exhibit some very interesting features, but also a few drawbacks mainly arising from resorting to Bluetooth, according to security issues raised by Vaudenay [55], which will be extensively detailed and debated against the security features of the decentralized prototyped solution presented in Section 4.1.

### 3.1.2 Blockchain-equipped

**BeepTrace** [56] is a very articulated distributed solution, which aims to position itself as a reference blockchain framework for any initiative in the contact tracing arena, since it claims to be able *"to become a piece of open interface information tracing hub for all privacy-preserving contact tracing providers globally"*, regardless of the consensus mechanism, the positioning system and the methods to share data among them. Such an intent looks quite attractive but still very demanding in its making. From an architectural point of view, BeepTrace makes use of two different chains. The tracing blockchain is where all users store their positioning data which have previously been encrypted and anonymized by means of sophisticated algorithms involving PKI Central Authorities. When a user is tested positive, its[5] positioning data must be endorsed by a diagnostician, who decouple them from the user pseudonym and put them in the second blockchain (i.e., notification blockchain) to the benefit of so-called geodata solvers. The latter determine the contagion risk level that the notification blockchain makes available to the involved users.

[4] `https://www.immuni.italia.it`

[5] The neutral singular gender is purposely used in similar cases hereinafter.

A proposal by **Song et al.** [57] pursues contact tracing objectives that include both location-based and individual-based information, which is stored in a blockchain database as a trustless repository for both direct and indirect contact tracing. The solution also features risk contagion algorithm which operates on the collected data.

On the same track, a proposal by **Arifeen et al.** [58], based on a Bluetooth app-to-app, traces user encounters and exports all the contact information, collected by each single user in the latest 14 days, to the public blockchain, thus mitigating the typical privacy preserving issues of DP3T-based solutions [55].

**PRONTO-C2** [59] is an interesting proximity-based contact tracing proposal, which combines a mechanism that allows users to autonomously and confidentially call each other to alert the presence of a detected infection by means of the Diffie-Hellman protocol[6], along with a bulletin-board implemented through a blockchain. The basic idea of the contact tracing proposal is to replace the generation of users' pseudonyms with that of unique encounter identifiers generated by the DH protocol. Other cryptographic measures make the solution quite robust, whereas a few problems are encountered in the way the DH protocol should fit in the Bluetooth payload.

A significant contribution to the use of the blockchain in epidemiological surveillance is provided by a remarkable technical report by **Micali** [60]. The author observes that DP3T-based solutions do not enable a global view of the pandemic evolution for epidemiological follow-ups, since all the relevant information about encounters is stored on the users' mobile phones, making it unfeasible to obtain aggregate information. Hence, the report's suggestion is to introduce a public blockchain in the system design, whereby it is possible to upload synthetic information about suspicious encounters. An up and running web-app, **iReport-Covid**[7], has been deployed on the Algorand blockchain [61] under the report's recommendations.

The blockchain can be also employed to counteract the location forging issue in absolute localization algorithms by providing time-based *Proof-of-*

[6] `https://www.hypr.com/security-encyclopedia/diffie-hellman-algorithm`
[7] `https://ireport.algorand.org/en`

*Location (PoL)* protocols, as originally proposed by Amoretti et al. [62], for generic location-based services. More recently, PoL has been proposed for contact tracing in **Bychain**, a permissionless blokchain for location-based tracing. The protocol witnesses, that is, devices sending location information (e.g., WiFi Access Points, BLE-equipped nodes, LTE base stations), are all equipped with GPS and identified by a {*public key-private key*} pair.

Finally, a proposal by **Marbouh et al.** [63] implements a blockchain-based system that makes use of Ethereum smart contracts and *oracles*[8] to assess the reliability and trustworthiness of the information received by the public and government agencies. The main goal of the solution is therefore to enable dashboards and mobile decentralized apps to retrieve aggregate data coming from registered external sources that have incrementally obtained a high degree of reputation, according to a specific logic developed on an appointed smart contract.

## 3.2 Renewable Energy Sources solutions

The potential applications of blockchain in the energy sector have always been regarded as crucial in terms of new opportunities to manage the "smart grid" complexity and foster new business models and marketplaces, such as engaging new small actors in the renewable energy sources (RES) market [64]. The blockchain can therefore act as an enabler for the creation of novel energy communities marketplaces, in which the transparency features, introduced by the decentralized technology, can guarantee accountability while preserving privacy requirements. In fact, the World Energy Council[9] claims that the use of the blockchain in the energy sector can become the ultimate facilitator for its decarbonization by allowing the integration of decentralized energy sources into the power grid. In particular, two very promising application fields deserving a deeper investigation are: (i) the handling of grid flexibility demand/response mechanism, and (ii) the provision of platforms for more efficient billing processes. the blockchain technology may also be used for

[8] `https://en.wikipedia.org/wiki/Blockchain_oracle`
[9] `https://www.worldenergy.org/`

issuing certificates of origin, particularly for green energy production and renewable energy sources.

In addition to the many surveys on the application of the blockchain to the energy sector [64, 65, 66, 67], conceptual work has been delivered on RES specific topics, which are related to those investigated in the the original DApps presented in Chapter 5. Within this large body of literature, **Decusatis and Lotay** [68], have tackled security issues for an Ethereum blockchain that hosts a decentralized energy management application, presenting an approach to digital identity management that would require smart meters to authenticate with the blockchain ledger and mitigate identity spoofing attacks.

**Yang et al.** [69] have explored the potentiality of the virtual power plant (VPP) as a promising paradigm for managed RES to participate in the power system. The authors have developed a blockchain-based VPP energy management platform to facilitate transactive energy activities among residential users in such a way that users can trade energy for mutual benefits and provide network services, such as feed-in energy, reserve, and demand/response, through the VPP. The conceptual work has been validated by a prototype blockchain network whereby the VPP energy management scheme has been successfully tested with respect to users' energy trading and other network services.

**EFLEX** [70] is a pilot service being carried out in Bulgaria and Romania whose main objective is to demonstrate that trading of services, amongst Transmission Service Operators (TSO), Distribution Service Operators (DSO) and small producers/consumers (prosumers), can be carried out in a transparent, secure, and cost-effective manner by using a blockchain-based flexibility marketplace. The aim is to look for ways to help both DSOs and TSOs to be more directly engaged in managing energy flows on the network by using the same decentralized platform with the aid of specifically designed smart contracts.

**Pop et al.** [71] have written a work on the blockchain-aided grid flexibility in which they investigate the use of decentralized mechanisms for delivering transparent, secure, reliable, and timely energy flexibility, under the

form of energy demand profiles of Distributed Energy Prosumers, to all the stakeholders involved in the flexibility markets (DSOs, retailers, aggregators, etc.). In their approach, a blockchain-based ledger stores the energy exchange information collected from smart metering devices in a tamper-proof manner, while self-enforcing smart contracts programmatically define the expected energy flexibility at the level of each prosumer, the associated rewards or penalties, and the rules for balancing the demand/supply operations at grid level. The devised mechanisms have been validated using a prototype implemented in an Ethereum platform.

**Umar et al.** [72] have studied the integration of distributed battery storage equipped with smart meters by means of a digital platform, in order to improve the overall performance of the microgrid system. The decentralization paradigm is employed to establish a self-sustained community trading energy in the microgrid system. The case study showcases the merits of blockchain technology in providing a secure and effective trading platform for mass users. The DLT-based market help end-users benefit from energy savings and self-sufficiency thanks to the combination of automated decentralized trade and storage flexibility.

Finally, **BloRin**[10] is an academic and industrial initiative that aims to create a blockchain-based technology platform to favor the creation of solar smart communities and to encourage trustworthy interactions between prosumers. The use of the blockchain platform, based on Hyperledger Fabric, manages the accounting of energy flows and the automation of economic transactions [73].

## 3.3 Blockchain design topics

The design of smart contracts can turn out to be quite challenging, relatively to well-known features, such as immutability, execution costs, sequential execution, and so on [74]. Especially on public blockchains, the overall design of smart contracts has a large impact both on its deployment and execution

[10] `https://www.blorin.energy/`

costs. The deployment cost is proportional to the contract size, whereas the execution cost depends on time, space, and message complexity. In consortium blockchains the monetary cost of storing and executing smart contracts is typically not an issue. However, complexity of any kind in smart contract design may become an issue, in that it could affect many aspects of the application throughput as well.

In this section, the surveyed work has been mapped against high-level topics that logically group them. A topic represents a blockchain-related feature that poses specific challenges to application designers in order to mitigate related issues. The related work describes design solutions to these issues, from specific smart contract design patterns to more extensive scalability and security design approaches. Topics, issues and related solutions have been primarily selected in relation to the original decentralized applications presented in this document.

### 3.3.1 Smart contract immutability

***Restricting Pattern***[11] is the most common pattern dealing with the ownership of deployed smart contracts in a context of code immutability. It consists in storing the address of the contract's deployer as its owner. The address can be read from the `msg.sender` global object and set inside an address variable by the constructor method upon contract deployment (see Subsection 2.2.6). In this way, it is easy to let some operations be only performed by the contract's owner.

***Authorization Pattern*** [75] can be considered as an evolution of the Restricting Pattern towards the notion of Access Control List. To perform specific actions, a collection of addresses is stored inside an address mapping table. In this way different users, who might have different privilege levels, can make critical changes in the contract state, or perform operations that other users are not allowed to do.

Because of its immutability, whenever a smart contract needs to be up-

[11] `https://docs.soliditylang.org/en/v0.8.17/common-patterns.html#restricting-access`

graded to fix bugs or add new functionalities, a new smart contract must be deployed with a new address. Hence, it cannot manipulate data embedded in the previous version of the contract, so that importing data from the old contract could be very resource consuming. In these cases, the ***Data Contract***[12] design pattern can help by keeping business logic and data separated in two distinct contracts. The former accesses the required data by exposing read/write functions which call the corresponding operations provided by the data contract. This design pattern allows the replacement of the logic contract, while keeping the data contract alive. A similar pattern, the ***Treasure Manager*** [74], has been devised to cope with the separation of the contract's data from the contract's business logic in payable services that need to transfer the accrued financial reserve to more recent versions, without incurring in the data transfer cost.

Among the blockchain solutions adopting the Data Contract design pattern, **Chronobank**[13] provides employment opportunities and ensures prompt and fair payments. It tokenizes labor and provides a market for professionals that are available to stake their time in a smart contract and receive regular rewards. In its implementation, Chronobank uses a smart contract with a generic data structure that can used by all the other smart contracts implementing the business logic. **Colony**[14] is a platform for running DAOs from scratch in Ethereum. Similarly to Chronobank, it makes use of a Data Contract pattern with a generic data structure for each new organization.

Data Contract (or similar patterns, such as Treasure Manager) needs to be used along with the ***Contract Registry***[15] pattern, which is able to conveniently manage the upgrade of smart contracts, since they will be assigned a new address in case of replacement with a newer version. This pattern maintains a registry that provides the updated version of the (*name, address*) pair. Before invoking a smart contract, a user can look up the

[12] `https://research.csiro.au/blockchainpatterns/general-patterns/contract-structural-patterns/data-contract/`

[13] `https://chrono.tech/`

[14] `https://colony.io/`

[15] `https://research.csiro.au/blockchainpatterns/general-patterns/contract-structural-patterns/contract-registry/`

registry to find its address. Similar outcomes are provided by the ***Proxy*** [76] and ***Migration*** [77] patterns.

Needless to say that the **Ethereum Name Service** (ENS)[16] is the first application that comes to mind in the Contract Registry realm, that is a name service on the Ethereum blockchain implemented as an extensible registry accessible to everyone. It has two principal components: the registry and the resolvers. The ENS registry consists of a single smart contract that maintains a list of all domains and subdomains, along with their owners and their resolvers. The latter are contracts responsible for the actual process of translating names into addresses and they can be replaced when needed. Many other name services are available on different blockchains, as for instance the **Energy Web Name Service** (EWNS)[17], which shares the same pattern as ENS.

### 3.3.2 Smart contract computational cost

The monetary cost of smart contracts execution opens an issue which goes beyond the computational cost that developers have to tackle in traditional programming languages. Therefore, a poor design can lead to expensive and unnecessary execution fees.

Among the several proposals in the arena of execution cost reduction, many of them deal with the optimized usage of variables in Ethereum, which are organized in 256-bit words, in terms of storage occupation and content modification. These two operations have well specified execution costs in the EVM, whose containment is the goal of patterns such as the ones cited by Marchesi et al. [76]: *Packing Variables*, *Packing Booleans*, *Uint* vs Uint256*, *Mapping vs Array*, *Fixed Size*, *Default Value*, *Short Constant Strings*, *Write Values*, and the *Boolean Box* [74], to name a few.

Moreover, automated tools have been developed to detect anti-patterns that the Solidity compiler, in the case of EVM smart contracts, fail to reveal and optimize [78, 79].

[16] `https://ens.domains/`
[17] `https://ens.energyweb.org/`

### 3.3.3 Smart contract storage

Blockchains require full data replication across all participants. Storing large volumes of data within a transaction may be impossible due to the limited size of blocks, such as in Ethereum, where a protocol-defined additional gas limit restricts the computational complexity and data size of the transactions included in them. Therefore, in order to preserve the integrity of those bulk data that cannot be stored on-chain, a few patterns have been envisaged.

Among these, ***Blockchain anchor***[18] uses the blockchain storage to ensure the integrity of an arbitrarily large off-chain dataset. For dynamic data preservation, the pattern can be used to track the integrity of a collection of data by storing the hash of a Merkle trie root, at a particular point in time, in the blockchain.

For instance, **Chainy**[19] is a smart contract running on the public Ethereum blockchain, which stores on-chain the short link to an off-chain file whose immutability is thus guaranteed. **POEX.IO**[20] is instead a service that demonstrates the ownership, the integrity and the proof-of-existence of timestamped documents. This is achieved by letting the document's owner deposit the document cryptographic hash into the Bitcoin blockchain. On top of Bitcoin, **Chainpoint**[21] is an open standard for creating a timestamped proof of any data, file, or process that is used by other blockchain initiatives. Chainpoint links a hash of off-chain data to a proprietary blockchain which returns a timestamped proof. A blockchain node receives hashes which are aggregated together using a Merkle Trie. The root of this trie is then published in a Bitcoin transaction.

A very interesting initiative, the **Dwarna** [23] web portal, connects the different stakeholders of the Malta Biobank in a dynamic "informed consent" logic, which gives individuals control to determine the rules by which their biospecimens and data should be used. The solution covers remarkable as-

[18] `https://research.csiro.au/blockchainpatterns/general-patterns/self-sovereign-identity-patterns/anchoring-to-blockchain/`

[19] `https://chainy.info/`

[20] `https://poex.io/`

[21] `https://chainpoint.org/`

pects regarding the privacy preservation of users' data in the biobank arena, but it is also interesting in some architectural aspects of its implementation. In particular, for security reasons, the back-end storage is split into two components, an off-chain relational database storing the majority of data about users and studies, and a Hyperledger Fabric blockchain that records the "relational table" made up of: anonymized user-ID (the owner of the specimen); study ID; informed consent given to the study-ID by the user-ID. This mixed database design makes it more difficult to access users' sensitive information and prevent pseudonym linkability risks.

### 3.3.4 Security and scalability design

In the wake of the "blockchain trilemma", security and scalability issues are of paramount importance when it comes to making appropriate design choices for the decentralized applications that developers want to build. Security and scalability enhancements involve fine design of smart contracts and blockchain storage, as well as extra features that some (public) blockchain can hardly provide by design. Security and scalability requirements that exceed the scope of blockchains features, and sometimes clash with them, mostly concern access control policies to on-chain, but primarily, off-chain data, according to patterns that have to deal with computational and storage costs, as extensively discussed in the preceding subsections.

Far from deeply discussing the many blockchain-based solutions that deal with the above-mentioned issues, the table in Figure 3.1 lists the design choices, in terms of blockchain type and storage policies, data accessibility features, and extra security add-ons that a few selected decentralized solutions, in the Electronic Health Records (EHR) territory, exhibit [80].

Most systems of the surveyed solutions advocate the use of consortium networks [42, 81, 82, 83, 84, 85, 86] for scalability issues and better protection of medical data from unwanted access. Only two solutions [87, 88] do not limit the blockchain type to either public or consortium, provided that smart contracts are supported, whereas the public type is the choice of only one solution [89]. ''Data accessibility" has to do with data ownership and can

| *EHR Solution* | *Blockchain type* | *Accessibility* | *Privacy support* | *On-chain storage* | *Off-chain storage* |
|---|---|---|---|---|---|
| ACTION-EHR | Consortium (Hyperledger Fabric) | Patients have full control | 3rd party CA | management data | EHR clustered DB |
| UniRec | Consortium (IPFS + Ethereum) | Patients have full control | ACL policy + PGP EHR encryption | hashed data | IPFS-based EHRs |
| Tith et al. | Consortium (Hyperledger Fabric) | Patients have full control | Membership auth + proxy re-encryption | management data | EHR DBs |
| MedRec | Consortium (Ethereum) | Patients have full control | AAA server + PKI authentication | hashed data | EHR centralized DBs |
| Wang et al. | Consortium (Ethereum) | Patients have full control | Proxy re-encryption + EHR encryption | keywords | EHR encrypted cloud DBs |
| FHIRChain | Any blockchain executing contracts | Not addressed for patients, only for authorized providers | PKI DIDs + smart tokens + ciphered EHR | ciphered metadata | EHR DBs |
| Medshare | Any blockchain executing contracts | Not addressed for patients, only for authorized providers | Data provenance + auditing + ACL | management data | EHR cloud DBs |
| MedBlock | Consortium (Hyperledger Fabric) | Patients have partial control | CA + ACL protocol + PKI EHR encryption | EHR summaries | EHR DBs |
| BlocHIE | Public (PoW-based) | Patients have full control IoT devices can write data | Digital signatures + hash functions | EHR verification | EHR DBs |
| Ancile | Consortium (Ethereum) | Patients have full control | ACL smart contracts + proxy re-encryption | management data | EHR DBs |

Figure 3.1: Comparison of key features in selected EHR solutions.

be seen as a derivative property of the adopted "Privacy support", which, in turn, specifies extra access control policies and data encryption measures that are not in the technical provision of blockchains. Most solutions give full ownership and control of medical data to the patients by adding conventional technologies to the solution's back-end, such as Access Control List (ACL), Authentication Authorization Accounting (AAA), and/or cryptographic primitives and protocols, such as digital signatures, hashes, Public Key Infrastructure (PKI), Pretty Good Privacy (PGP). Only one solution resorts to ACLs implemented in on-chain smart contracts [86]. Finally, the on-chain/off-chain design choices follow similar Blockchain Anchor patterns, with a limited amount of information stored on-chain, either in plain or ciphered format (e.g., metadata, summaries, management data, keywords, and so on), whereas the bulk data are stored in external off-chain DBs of different kinds (e.g., centralized, cloud-based, decentralized [82]).

# Chapter 4

# Contact Tracing solution

The **Nausica@DApp** solution was conceived at the time the outbreak of Covid-19 was just beginning to show its disrupting effects in social and working interactions, as, for instance, in teaching and collaborative environments. Its ancestor NausicaApp was part of the wider NAUSICAA (New Approach for a UniverSIty Covid-resilient and Active Again) project at the University of Catania, aimed at addressing Covid-19 related concerns with a multidisciplinary approach, spanning the fields of medicine, law and public space management. The comprehensive solution proposed in this chapter aims at supporting management of critical working conditions in campus environments, thus guaranteeing the continuity of university-level research and teaching in a context of shared social security, in compliance with hygiene and health requirements. It eases compatibility of the in-presence activities of a campus-based organization with the constraints posed by the virus during the pandemic, or at a later endemic stage. This is accomplished throughout several intervention areas, such as *personnel contact tracing*, *overcrowd surveillance*, and *epidemiological monitoring*.

The initiative behind the realization of Nausica@DApp falls in a converging understanding, shared by the healthcare, industrial and academic communities, which pinpoints the development of ad-hoc mobile apps in personal user devices as a means to provide an efficient and reliable response to contact tracing and epidemiological surveillance. Nowadays, although sev-

eral vaccines have entered the scene, epidemiological models say that the viral phenomenon has two possible pathways: a) recurrences or epidemic waves, b) alternate levels of endemic circulation, until the herd immunity will be achieved, if ever. In the meantime, it is essential to maintain well-established practices, such as compliance to social distancing rules, avoidance of overcrowding, preventive swab campaign, forward and bidirectional contact tracing [90]. Digital tools and services, among these best practices, play a central role in the overall game.

However, it is notorious that the level of user acceptance of the national contact-tracing apps has achieved mixed results, especially in the early adoption phase. Centralized solutions cause alarm to users due to the availability of sensitive information to central authorities, for both contract tracing purposes and contagion risk follow-ups [37, 91, 92]. In addition, most of them do not encourage users to trust the veracity of the publicly available aggregate data, particularly about the effects of the pandemic spread (and nowadays of the vaccine campaign) on the number of infected people and deaths. It is however surprising that most decentralized solutions, also if following the DP3T paradigm in which no user information is collected at central level, still make users suspicious about the privacy preserving concerns that such solutions may instill, for reasons more extensively detailed in Section 3.1.

Consequently, in the context of epidemiological surveillance, mobile applications need to implement ***transparency*** measures to guarantee and demonstrate beyond doubt that all user data, including aggregate data, are handled with no threat to their privacy, and are collected and used without counterfeiting them [93, 94].

Therefore, the purpose of the proposed solution consists in ***maximizing trust*** from users in the ***privacy preserving*** and ***data integrity*** aspects, in spite of a restricted number of centralized design choices (which, by the way, play an essential role in the absolute localization of mobile devices). The designated purpose is accomplished by a ***decentralized contact tracing*** scheme carried out by apps running on the users' mobile devices, in which contact data are anonymized by design and immutably authenticated by a ***blockchain***.

In addition, the solution tackles other challenging issues, such as the ***overall performance*** of the prototyped system, especially in terms of ***computing and storage costs*** of the integrated blockchain, which, if not conveniently faced, can undermine the actual effectiveness of the initiative.

The remainder of the chapter highlights both the rationale behind each phase of the project and the specific technical elements that make up the system architecture, with a particular focus on the innovative contributions fed by the research activities carried out over time.

## 4.1 Hybrid decentralized version

In this section, the early version of the developed contact tracing solution, namely **NausicaApp**, is described [95]. This version is a complete app that does not feature any decentralized element in the back-end design, that is, no blockchain is integrated into it. Still a considerable amount of decentralization is provided in the system architecture. This has been, in fact, developed according to a hybrid decentralized design, loosely based on the DP3T proposal (see Subsection 3.1.1), with which it shares the principle that storing and processing of direct contacts are the responsibility of the mobile apps. The system architecture is based on a mixed approach, in which the client-to client dialogue among apps is achieved through the central server mainly acting as a relay and alert system agent, whereas ad-hoc client software is in charge of data collection, storage and computing.

Regardless of the adoption of an administration server in the communication infrastructure, NausicaApp is anyway capable of ensuring advanced features in many areas of users' privacy preserving, thus showing enhancements with respect to existing solutions (see Subsection 3.1.1), as explained hereinafter in Subsection 4.1.6.2. The original choices, which provide for these advanced features, will be presented in this section, since they are also maintained in the architecture of the final version, for which only the added features, coming from the adoption of a blockchain in the back-end design, will be thoroughly explained in Section 4.2. NausicaApp is altogether able to respond to multiple requirements: (1) the management of people flows and

space occupation rates on campus; (2) the booking, with fair rotation, of lectures in attendance; (3) the tracing of direct and indirect contacts for epidemiological surveillance; (4) the aggregation of collected anonymized data, aimed at driving informed campus management decisions, to be shared with local and national health authorities under strict privacy-preserving policies.

These operational requirements, in particular indirect contact tracing and overcrowd monitoring, call for the adoption of an ***absolute device localization*** paradigm, which has been devised on top of the campus WiFi infrastructure, proving to be encouragingly accurate in most cases. Absolute localization, on the other hand, entails a certain amount of server-based centralized operations, as ascertained by other solutions that pursue similar goals (see Subsection 3.1.1), which might affect the preservation of user data privacy. However, NausicaApp features some original choices, in the way localization information is built and potential proximity contacts are detected, that convincingly lessen the unwanted effects of centralized operations, as discussed in what follows.

### 4.1.1 System overview

As briefly introduced, in NausicaApp no data matching processing is performed outside the personal devices, with an administration server responsible for anonymizing and relaying centrally verified information, each time an infected user has authorized the distribution of its presence information to the rest of involved devices. The central server can also use the collected anonymized presence data to detect overcrowd conditions in various spots, emit alarms and send alerts, as well as monitor people flows and occupation rates on the premises, so as to facilitate the smart management of potentially critical situations in a daily prevention routine.

NausicaApp system architecture, depicted in Figure 4.1, shows the information flows between system components, which belong to three dominant areas of communication and processing: (i) device location, (ii) positive detection and notification, and (iii) overcrowd detection and notification. Detailed explanations of these information flows will be given in the next

subsections.

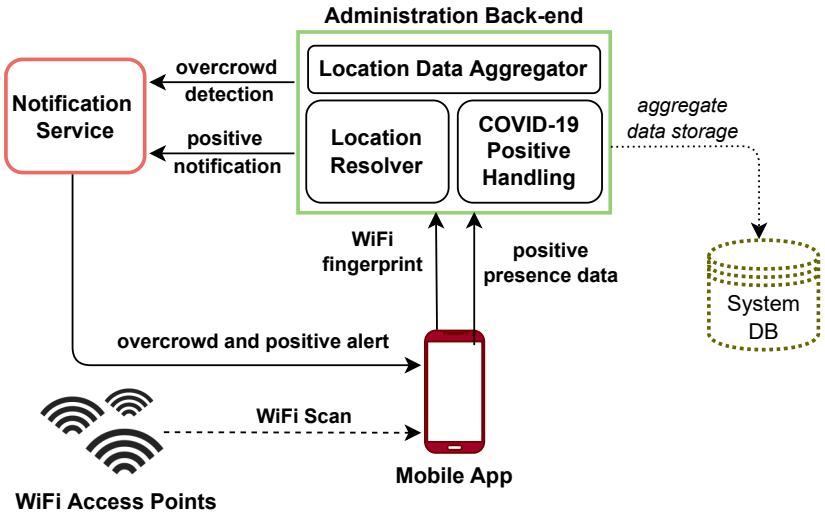


Figure 4.1: NausicaApp system components.

The administration back-end employs a *Notification Service*, which provides a scalable solution to send push notifications to the mobile NausicaApp-enabled devices. This functionality is provided by the Firebase Cloud Messaging (FCM)[1] component of the Firebase platform[2], which is able to broadcast notifications to the registered apps and to receive messages from them.

The solution's back-end may also interface with a *System DB* (whether centralized or decentralized), in which pandemic-related, aggregate data may be stored at the disposal of authorized parties.

Finally, it communicates with the NausicaApp-enabled devices by means of an underlying *Messaging Service*, whose architectural components are shown in Figure 4.2:

– a trusted server environment that supports the Firebase Admin SDK[3], which has the task to configure user authentication and addressing logic, as well as build the notifications that will be used by the FCM back-end. In our prototype system, the Spring Boot[4] framework, de-

[1] `https://firebase.google.com/docs/cloud-messaging`
[2] `https://firebase.google.com`
[3] `https://firebase.google.com/docs/admin/setup`
[4] `https://spring.io`


Figure 4.2: Main components of the messaging service.

ployed on the Heroku Cloud Application Platform[5] is the chosen environment hosting the Firebase Admin SDK;

- the FCM messaging service, which adds transmission functionalities to the Firebase Admin SDK; it is used to distribute notifications and messages to the devices subscribed to the server platform and, conversely, to relay messages from the apps to the administration server. The notification mechanism is based on the enrollment of devices to specific "topics", each of which identifies a distinct Access Point in our application;
- the Android Transport Layer, which supports FCM;
- FCM-enabled Android apps that receive notifications from the FCM messaging service via the above-said transport service; they can also send messages back to the server through the reverse route.

App-to-server communication simply assumes that user devices have access to the Internet (possibly, but not necessarily, leveraging the pre-existent, campus-managed enterprise WiFi infrastructure), in fact making no use of any app-to-app direct channel (e.g., Bluetooth).

### 4.1.2 Device Localization

As already seen, the proposed solution displays, among other features, ***indirect contact tracing*** and ***real-time monitoring*** of premises occupation, which inherently require ***absolute localization*** of devices, that is, positioning them in space within precise time intervals. ***WiFi sensing*** has proven

[5] `https://www.heroku.com`

to be an appropriate choice to fulfil such a requirement, both when it leverages the WiFi enterprise campus infrastructure already in place and also when it relies on external service providers.

Even though scientific literature warns that WiFi indoor localization may be a complex task to accomplish [96], our basic idea is that an Access Point (AP) deployed in a closed environment (e.g., a laboratory) will emerge - most of the times - as the one radiating the most powerful signal strength, to such an extent that the device absolute position can be comfortably determined by the system software. Subsection 4.1.5 will make it experimentally evident that such a choice is compatible with most indoor areas within the campus premises (in the unlucky case of garbled WiFi locations or outdoor spots, NausicaApp is equipped with a feature that lets the localization algorithm reach an external WiFi localization provider[6] through standard RESTful APIs).

The mobile app side features a GPS-driven mechanism which triggers the WiFi fingerprinting mechanism as soon as the device enters the monitored premises (this could be required for compliance with privacy and/or campus admission rules).

The device localization process covers the first two phases of the scheme outlined in Tang's paper [37] on the most outstanding solutions in the Covid-19 contact tracing arena, namely the *Initialization phase* and the *Sensing phase*. The remaining two phases, i.e., *Reporting phase* and *Tracing phase* will be discussed in Subsection 4.1.3.

### Initialization phase

Each active instance of the app, in the initialization phase, does not need to explicitly register with the administration server; instead, it will be transparently associated to a token generated by the FCM component, thus generating an implicit user-ID that uniquely yet anonymously refers to it. The temporal validity of the token should not last longer than 24 hours; it will then be regenerated at the expiration of the established period, in order to

[6] `https://en.wikipedia.org/wiki/Wi-Fi_positioning_system`

contrast "linkability risks" [37] of anonymized identifiers.

### Sensing phase

Figure 4.3 focuses on the device localization process by isolating the system components and the information flows that are related to the sensing phase.

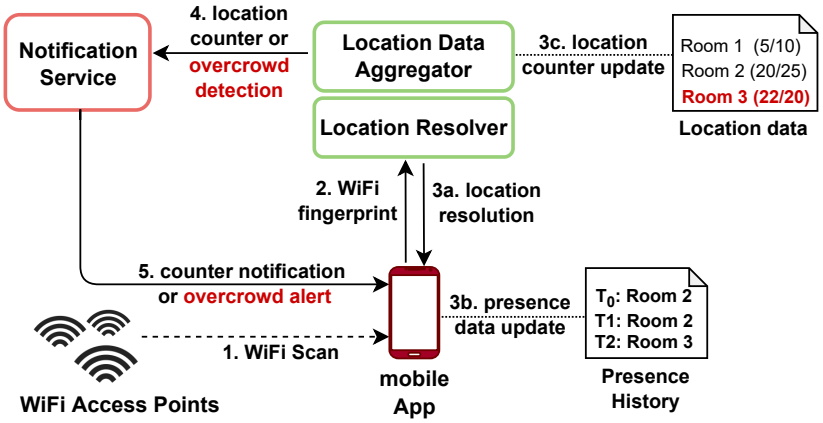


Figure 4.3: Absolute device localization process in normal or overcrowd conditions (red text): information flows among system components.

The mobile app initiates the device localization operations by scanning the surrounding WiFi signals from APs and measuring signal intensity or RSSI[7] (step 1). Then, the app sends the *Location Resolver* service the collected `WiFi_fingerprint` inside the current location (step 2).

The *Location Resolver* then performs the following logical steps:

i. determines the Access Point's MAC/BSSID with the strongest RSSI among four scans;

ii. checks whether the anonymous user-ID has already been associated with the same Access Point, a different one, or none. Since, in the FCM logic, each Access Point's MAC/BSSID is coupled to a distinct "topic", the association or disassociation of an anonymous user-ID implies incrementing or decrementing a topic counter;

[7]Received Signal Strenght Indicator

iii. looks up a pre-loaded table in which indoor locations (e.g., hall, corridor, etc.) correspond to the registered Access Points;

iv. sends the selected location name, along with the corresponding dominant Access Point's MAC/BSSID, to the originating device (step 3a), which, in turn, updates its local presence history records (step 3b).

Following the absolute localization of a device, two more actions occur.

At the server side, a dedicated service, the *Location Data Aggregator*, updates the counters related to the involved Access Points for overcrowd monitoring (step 3c) according to the FCM topics' logic. For each monitored location, if the counter exceeds a configurable threshold (i.e., Room 3's counter) all the active devices will be notified with an overcrowd alert or, conversely, the devices will only be notified with the updated counter (steps 4 and 5).

*Mobile app*-wise, the selected dominant AP's MAC/BSSID will be used to build and temporarily store an internal object, named `Presence Data`, which can be represented, in a loose JSON notation, as:

```
{AP-BSSID, Timestamp, Time-To-Live}
```

to be used in the contact tracing matching phase, as explained in Subsection 4.1.3. The collection of these records makes up the `Presence History`, defined as the temporally ordered sequence of presences stored on a device and not yet expired.

### 4.1.3 Contact tracing

Figure 4.4 shows the contact tracing process, made up of the steps discussed below:

#### Reporting phase

If a positive-tested user wishes to adhere to the containment program, it should send its `Presence History` to the central server provided that the

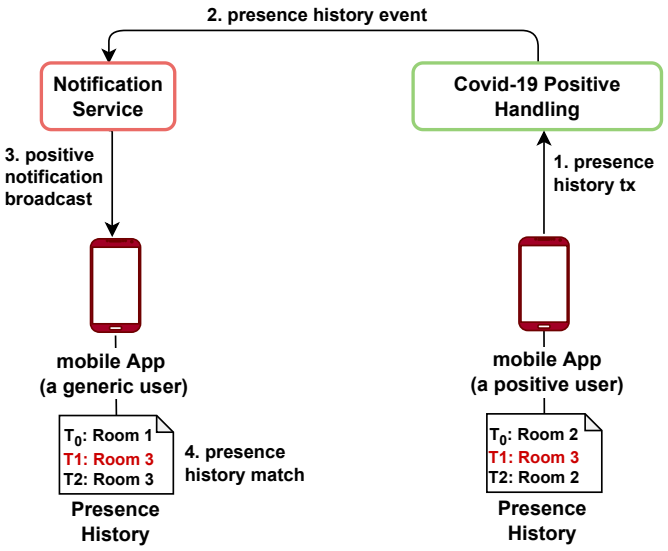


Figure 4.4: Contact tracing with positive matching.

*Mobile App* has been authorized beforehand by means of a code released by an administrator (step 1).

### Tracing phase

The *Covid-19 Positive Handler* will then trigger the *Notification Service* to broadcast a notification, with this `Presence History` as its payload[8], to the rest of the anonymized and uniquely identified active devices (steps 2 and 3). Any such individual (non-aggregate) information at the server side will expire within a predetermined time interval, and will be permanently erased.

Then, each *Mobile App* correlates, according to a given heuristics, places and times of the `Presence Data`, contained in the pulled `Presence History` of the positive user, with its locally stored own, in order to detect direct and indirect contacts (step 4).

[8] `https://firebase.google.com/docs/cloud-messaging/concept-options#notification-messages-with-optional-data-payload`

### 4.1.4 User interface

NausicaApp is a complete solution with back-end and front-end implementation. The mobile front-end has been installed on Android phones to successfully test all the described phases of the contact tracing and overcrowd processes. Figure 4.5 shows a UI screenshot displaying real-time notification of the occupation rate of the lecture hall in which a user has been localized.

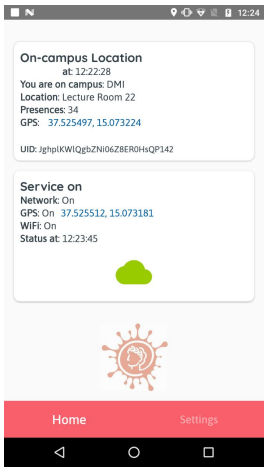


Figure 4.5: NausicaApp screenshot on a mobile Android device.

### 4.1.5 Absolute localization tests

The strategic decision to adopt an absolute localization paradigm, relying on the WiFi campus infrastructure, has been validated by the tests carried out on the NausicaApp system prototype. The most relevant experimental results are described in this subsection.

The adopted localization methodology is based on the idea that, inside any campus indoor location (e.g., classroom or laboratory), the internal WiFi AP will be the one radiating the dominant signal strength or RSSI. Thus, WiFi fingerprints that mobile devices periodically send to the *Location Resolver* service (see Figure 4.3) easily enable it to precisely locate devices using a simple maximum-signal logic. Location information is used by the *Location Data Aggregator* service for both ***occupation rate monitoring***

and ***Covid-19 positive handling***, in order to support direct and indirect contact tracing operations.

The testbed setting involved six people, equipped with NausicaApp-enabled Android devices, wandering within and among lecture halls. Each device creates a `WiFi fingerprint` by performing four consecutive WiFi scan requests and collecting their outcomes in the following JSON scan quadruple:

```
Anonymous-User-ID: {
    Scan-Time-1: "[AP1-RSSI,..,APn-RSSI]",
    Scan-Time-2: "[AP1-RSSI,..,APn-RSSI]",
    Scan-Time-3: "[AP1-RSSI,..,APn-RSSI]",
    Scan-Time-4: "[AP1-RSSI,..,APn-RSSI]"
}
```

which is sent to the *Location Resolver* service in charge of the absolute localization process. This process is repeated about every two minutes.

The experimental setup on a testing client machine consists of a variety of (python, awk, sed, bash) scripts. At the client side, WiFi scan data are first collected, as the server stores them, by a per-user Firebase listener, then suitably processed by a set of filters, and finally fed to gnuplot for real-time visualization.

Figure 4.6 plots the results of a trial pattern in which a single device moves from lecture hall LH3 to LH4 then to LH2, while being monitored by the listening service.

The plotted RSSI strengths (in dBm) of signals from the Access Points, collected over a suitable time interval, show how the user device, as long as it is within a lecture hall, always receives the internal AP's signal as the strongest one, whereas the crossing points between different RSSI plots reveal that the user is moving to a new hall.

The shown figure reports just one of the replicated measurement tests carried out independently by all six devices in various locations within the campus premises. These tests all showed the same reassuring results. Other tests involved positioning the six devices in different spots of the same lecture hall, to verify that at all spots the dominant signal would be that of the hall's internal AP.

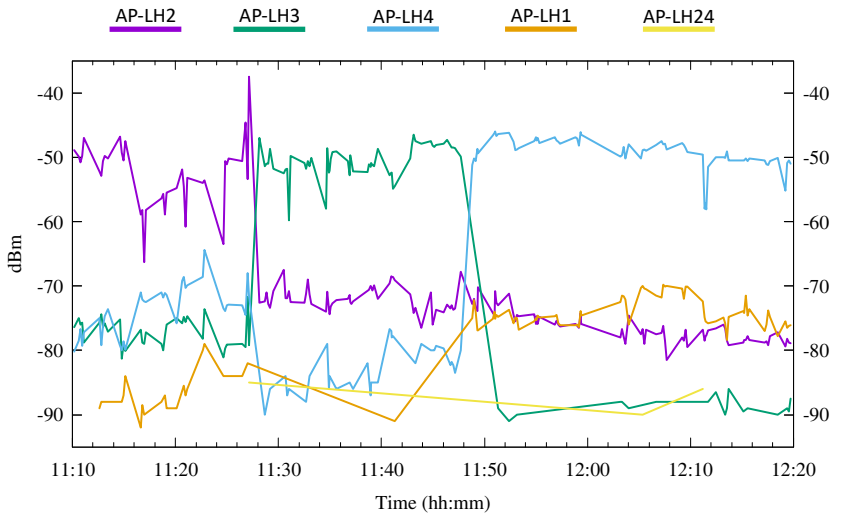


Figure 4.6: Access Point signals (in dBm) detected by a user device wandering across three different lecture halls in a suitable time interval.

These converging experimental results confirmed that it is sensible to implement a maximum-signal logic, within NausicaApp, to identify the "local" AP's MAC to be included in the `Presence Data`.

### 4.1.6 Comparative analysis

Since the outbreak of the Covid-19 pandemic, governments, health authorities and researchers have had to face the sensitiveness of user data, with respect to de-anonymization, detection of private encounters and tracking of individual movements, as they were trying to deliver effective contact tracing solutions. Special operating environments may even pose additional challenges.

In the next two subsections, a comparative overview of the investigated solutions, divided into centralized and decentralized ones, is given with respect to NausicaApp, whose focus is on universities, where protecting the rights of individuals must coexist with a steadfast effort to prevent and counteract the havoc the pandemic wreaks on the academic life.

#### 4.1.6.1 Comparison with centralized solutions

In centralized solutions, the approach is strongly dependant on users fully trusting central and third-party authorities; additionally, none of the work referred in Subsection 3.1.1 integrates blockchain components. Both corporate indoor *(i.e., EPIC, WiFiTrace)* and state outdoor systems, such as the controversial, privacy-wise, South Korea's *Corona-100m* and Singapore's *TraceTogether*, have to face trust concerns about fully reliable central entities.

These centralized solutions share the notion that, for optimal surveillance follow-ups, anonymized information collected at a central site, can be de-anonymized as needed. This approach, in terms of privacy choices, is extremely far from NausicaApp's, whose primary design requirement is to provide anonymity and authenticity to user data with its decentralized app-to-app approach, even though tracing of indirect contacts is obtained centrally in terms of absolute localization data. In fact, the configuration information of the WiFi infrastructure is used to localize the anonymized users within the campus premises, but by no means to track users. Hence, although a decentralized solution, indirect contact tracing can be additionally achieved without affecting the preservation of users' privacy.

#### 4.1.6.2 Comparison with decentralized solutions

NausicaApp, in its hybrid decentralized design, does not make use of the Bluetooth channel for proximity detection of direct contacts, thus bypassing the privacy and security issues typical of those decentralized solutions which, conversely, rely on such a technology [97], as more extensively detailed in the following, where the most common vulnerabilities and attacks, such as the ones pinpointed by Vaudenay's work on DP3T proposal [55], are discussed within the NausicaApp's scope.

##### Channel vulnerabilities and countermeasures

The scientific community seems to have taken for granted that, to ensure privacy preservation in contact tracing initiatives based on apps running on personal devices, it suffices to pursue data origination, storing and processing

following a decentralized paradigm. Quite to the contrary, many vulnerabilities in decentralized systems fail to be sufficiently addressed because of this orthodoxy [37]. Conversely, the approach adopted in our solution, if viewed from the app-to-app channel perspective, is inherently centralized, in that it leverages combined app-to-server and server-to-app channels supported by the campus WiFi infrastructure, and lacks a direct app-to-app channel, which is typically based on Bluetooth in most solutions.

To start with, the absence of a direct Bluetooth channel implies that an adversary cannot passively obtain pieces of personal information from monitoring (extended) Bluetooth beacon broadcasts, such as MAC addresses (when transmitted), explicit user IDs or ephemeral IDs that can jeopardize anonymity. In our system, crucial attention is paid to ensure the deepest protection in the WiFi channel carrying comparable information from apps to apps. Since our solution can rely on either the University's enterprise WiFi infrastructure or the mobile network, such confidentiality targets can be achieved at the minimal cost of leveraging the well-managed message encryption algorithms protecting the WiFi and the mobile network transmission[9].

In particular, in order to prevent malicious sending of critical information by an attacker, such as false infected people's presence data, an additional authorization scheme is needed to upload such data to the authority side. The current solution adopts an external authorization mechanism, built in the app, whereby injecting sensitive information to the system is allowed only by a central authority that releases a unique code following the check of forwarded medical documentation from the user.

As observed in DP3T's documentation [54], data encryption is not strictly necessary, since users' data privacy depends on the very nature of the transmitted data. In our solution, for instance, NausicaApp's data anonymization scheme, achieved by means of the FCM tokens, contributes to preserve users' data privacy. An encryption scheme can be optionally added to the software components of the solution that generate and read sensitive users' data.

[9]However, these security mechanisms require trustworthy servers.

#### Active and passive attacks

When it comes to a large variety of active and passive attacks that can be brought to a typical DP3T-like solution, it has to be said that NausicaApp's architecture itself, by its lack of a direct Bluetooth app-to-app channel, is capable of preventing many of them. For instance, those falling into the category of "False Alert Injection Attacks" [55], such as *Backend Impersonation, False Report, Reply and Relay attacks,* have few chances to succeed, since they are based on the exploitation of the Bluetooth proximity communication at a certain point in time of the attack strategy.

However, threats posed by "collected data linkability risks" [37], cannot be avoided just by stopping injection attacks, because correlations can be inferred in specific circumstances even with limited data sets (e.g., only two users are present in the same spot at a coincident time). The control logic coded in the app counteracts this risk by making decisions on how and when to show the complete set of information relayed by the server to the app users.

### 4.1.7 Solution discussion

This section has presented NausicaApp, a prototyped mobile app for tracing contacts among users within a university campus (or comparable communities, e.g., schools, hospitals, corporate sites, etc.). It improves on existing approaches, as it goes beyond tracing basic person-to-person *direct contacts*, by also revealing *indirect contacts*, i.e., that users happened to be in the same spot within a chosen time interval. Moreover, NausicaApp adopts stricter *privacy and security measures*. User privacy is ensured by avoiding to store at server side tokens identifying apps running on personal mobile devices.

A further advantage of NausicaApp is the adoption of a localization service of mobile devices based on the WiFi campus infrastructure. Absolute localization is obtained by having the devices sense the signal strengths radiating from WiFi Access Points. Scan-based WiFi positioning has indeed proven to be effective, simple to implement and convenient, in the presence of campus sites served by a dominant AP. This approach only requires the

mobile devices' WiFi interface be on, so that users are neither forced to adopt new habits, nor grant additional access permissions to contact tracing apps (potentially undermining their own privacy). Given the relatively low cost of adding some APs to a WiFi campus infrastructure, administrators might even consider to provide every significant site with a dominant AP, so that this may act as a "beacon" for the benefit of enabled devices.

NausicaApp exhibits some features that can be further pushed towards a better system performance and a more emphasized decentralized behaviour. For instance, the localization service could implement more sophisticated algorithms, including AI-based ones, relying on the data collected by the mobile apps in the sensing phase. Moreover, the localization strategies could be easily moved from the centralized administration server to the mobile apps for execution, in order to give the whole solution a higher level of decentralization.

Along the decentralization path, some improvements have been added to the next versions of the contact tracing solution, which concern smooth integration of blockchain features into the current system [98, 99], with a view to distributing anonymized and/or aggregate data to interested parties. This will enable contact tracing, overcrowd monitoring and epidemiological surveillance to be carried out in such a way that decentralization, trustworthiness and transparency are strongly enhanced [60, 93].

## 4.2 Highly decentralized version

The promotion of a shared understanding of how collected data are handled can undoubtedly help, as previously discussed in this chapter, to boost user confidence in the contact tracing solutions based on mobile apps.

The blockchain technology [5], leaning on mutually shared trustworthiness, can open new frontiers in this territory [36]. The World Health Organization, for instance, has been involved with major technology companies and governments in the design and deployment of *MiPasa* [100], a worldwide control and communication platform fuelled by the blockchain technology, which has been employed to enable individuals, state authorities and health

institutions to gather, share and correlate data to determine early detection of Covid-19 carriers and infection hot-spots.

In fact, a significant number of blockchain-based contact tracing applications have been proposed as an alternative to centralized solutions, as described in more detail in Subsection 3.1.2. This alternative and viable approach helps solution designers heighten the trust level that users would acknowledge to the application they are supposed to download and activate in their personal devices [101]. In such a way, a higher degree of adoption rate of contact tracing apps should be consequently achieved, thus allowing the solution to be fully effective.

The topics discussed in the remainder of this section represent a design enhancement to the NausicaApp architecture, which is not only capable of providing the same functionalities but can also effectively help overcome the trust issues by means of new features in a context of trustworthy openness among the various players (e.g., final users, health and governance authorities). With the integration of a blockchain platform into the system design, the new solution, coded under the name of **Nausica@DApp**, can be classified as a full-fledged ***decentralized application (DApp)***.

The DApp enhancements have been introduced in two different steps, with the final version (named *DApp_v2*) drastically overcoming some very annoying performance and cost issues experienced in the first version (named *DApp_v1*) due to a less performing smart contract design. The DApp's final version has eventually shown that its adoption is viable and effective.

The modular approach of Nausica@DApp has allowed to replace the smart contract design from the first to the second version without affecting the overall design of the back-end's core, which remains essentially the same in the two versions. The two incremental versions are presented in two distinct subsections in the remainder of the section.

### 4.2.1 DApp_v1

As previously introduced, Nausica@DApp is a contract tracing DApp that complements the NausicaApp design with the the addition of blockchain com-

ponents, thus extending the back-end design with smart contract business logic and storage. It also shifts the client software towards a decentralized paradigm with built-in blockchain front-end functionalities.

#### 4.2.1.1 Back-end decentralization

A simplified view of the modified system architecture of Nausica@DApp is depicted in Figure 4.7, which highlights the main differences from the NausicaApp one.

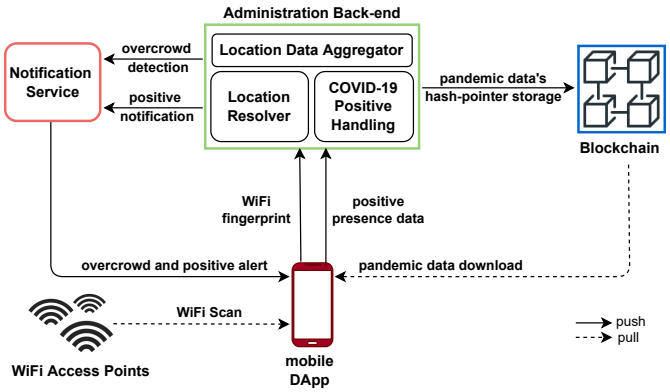


Figure 4.7: Nausica@DApp system components.

The administration back-end now interfaces with ad-hoc blockchain components, incorporated into the system architecture, which enable the back-end server to expose, in a decentralized and unforgeable storage, the anonymized proximity data needed by the mobile DApps to perform the internal matching operations for contact tracing. Furthermore, the blockchain components will allow other involved stakeholders, such as the academic community and the health authorities, to retrieve aggregate data about pandemic trends in a transparent and shared consensus-driven environment.

The administration back-end has been implemented in accordance with a microservice architecture, which is shown in Figure 4.8. The depicted microservices are loosely coupled with the administration server's logical processes shown in Figure 4.7 and thoroughly used in previous subsections for simplicity's sake. From now on the analysis of internal and external back-end

interactions will be carried out according to the microservice server layout.

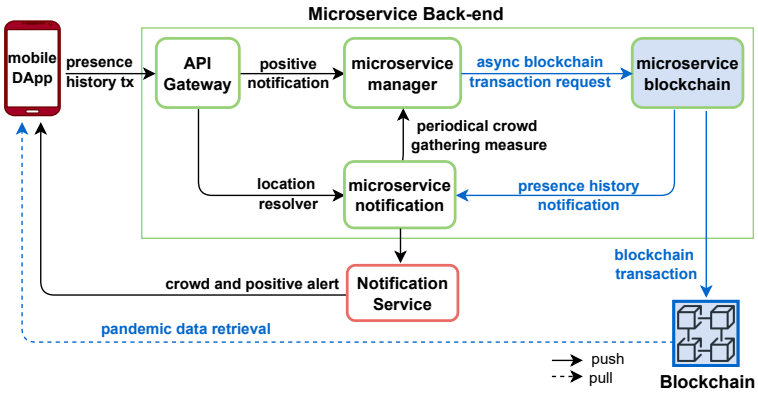


Figure 4.8: Back-end architecture with blockchain integration (blue items).

As previously stated, the well known Spring Java Framework has been used for the development of the administration server, thus providing the microservice portion of the back-end with a reference architecture made up of small and process-independent services.

Three microservices have been implemented:

i. a *microservice-manager* responsible for managing notifications from infected users;

ii. a *microservice-notification* interacting with the *Notification Service* for the issuance of alerts towards the mobile DApp clients;

iii. a *microservice-blockchain*, which adds a back-end component to the decentralized app architecture and is responsible for:

   (a) the sending of pandemic data to the blockchain storage;

   (b) the creation of the blockchain transactions upon asynchronous requests issued by the *microservice-manager*;

   (c) the interaction with the *microservice-notification* to start the alert process towards the mobile Dapp clients.

Since these transactions can take a while to be completed, they are requested asynchronously by the manager using a message queue (by means of the *RabbitMQ* middleware[10]).

#### 4.2.1.2 Blockchain components

The features and functionalities of such blockchain components are depicted in Figure 4.9 and specified in what follows, whereas proper workflow diagrams among software modules are detailed in Subsection 4.2.1.3.

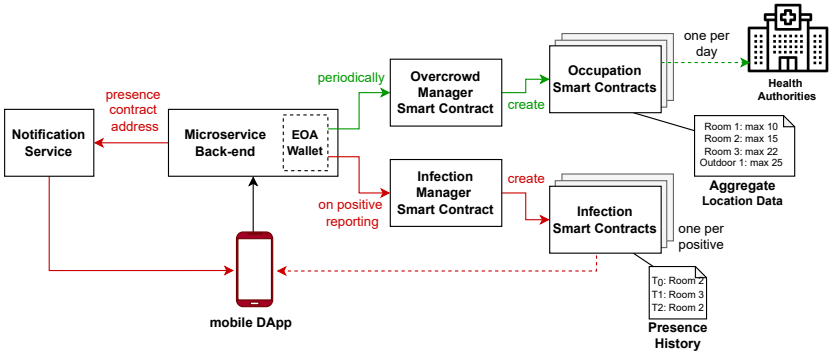


Figure 4.9: Added blockchain components.

1. *Mobile DApp client* (from now on *mobile DApp* or *mDApp*). Extra front-end capabilities are added to the original mobile app design in order to make them download all the information about potential infectious contacts from the blockchain storage. Processing of contact matching is yet again performed locally, namely, in the user devices, once the data are received, exactly as in NausicaApp.

2. *Infection Manager Smart Contract.* This is an Ethereum smart contract deployed by a blockchain account, externally owned by the relevant back-end microservice (see Subsection 4.2.1.3). The *Infection Manager Smart Contract* behaves as follows:

[10] `https://www.rabbitmq.com`

i. each time the server receives the `Presence History` data structure (see Subsection 4.1.3) from the device belonging to the infected individual, the *Infection Manager Smart Contract*'s EOA creates and deploys a new *Infection Smart Contract* and stores the `Presence History` in the latter's storage area; then it passes the address of the newly created *Infection Smart Contract* to the *Notification Service*. This address is then forwarded to all the *mDApps*;

ii. as data published in the storage of the *Infection Smart Contracts* lose time relevance, it triggers their deletion since they are no longer needed.

3. *Infection Smart Contracts*. Each of them corresponds to one infected user; they are created (and eventually deleted) by the *Infection Manager Smart Contract* following the above-described actions ensuing the reception of an infected user's `Presence History` from the server. As noted, each such new contract makes this information temporarily available to *mDApps* for contact matching.

4. *Overcrowd Manager Smart Contract*. This smart contract creates the below *Occupation Smart Contracts* on a daily basis and periodically sends them contagion-related aggregate data, such as exceeding of overcrowd thresholds, making them publicly available to a variety of blockchain users (health authorities, students, press, etc.) which can use them through multiple front-end decentralized apps (e.g. mobile apps, dashboards).

5. *Occupation Smart Contracts*. These are contracts with a limited lifespan, and they are capable of storing and managing the aggregate data objects received periodically (e.g. once/day) by the *Overcrowd Manager Smart Contract*. A typical data object is made up of a `location-ID` and aggregation field values such as `maximum` or `average` number of attendees.

#### 4.2.1.3 Operational workflow

In this section, the workflow of the scenario whereby a user self-reports positive to the system is sketched and discussed with an appropriate level of logical details and system component interactions.

In Figure 4.10, the relevant workflow starts with a positive user's mDApp sending the collected `Presence History` to the specialized *microservice-manager* of the back-end server. An internal write request message, along with the user data, is then sent to the *microservice-blockchain* interfacing with the blockchain, which is the one linked to the back-end's EOA by means of a digital wallet facility, as explained in Subsection 4.2.1.

A create call will be consequently transmitted to the *Infection Manager Smart Contract* deployed on the blockchain back-end, which is in charge of creating and deploying the *Infection Smart Contract* associated with the infected user. Upon deployment of the newly created smart contract on the blockchain, its address will be given back to the *microservice-blockchain*, which forwards it to the *microservice-notification*. Finally, this microservice triggers the *Notification Service* in order to make it broadcast an infected user alert, along with the smart contract's address containing the corresponding `Presence History`, to the rest of the anonymized and uniquely identified active devices.

Time correlations in collected information will make the user devices distinguish between direct and indirect contact tracing events.

#### 4.2.1.4 Blockchain-related software

In the first place, Ethereum has been chosen as the target blockchain platform due to its flexibility, its virtual machine and articulated storage design, as well as the large availability of testnets running diverse consensus algorithms, thus proving to be suitable to enterprise DApps as well. Moreover, Ethereum provides a set of mature tooling for smart contract programming. The smart contract code produced[11] and shown in this subsection has been developed,

[11] `https://github.com/giongion19/thesis-nausicaadapp`

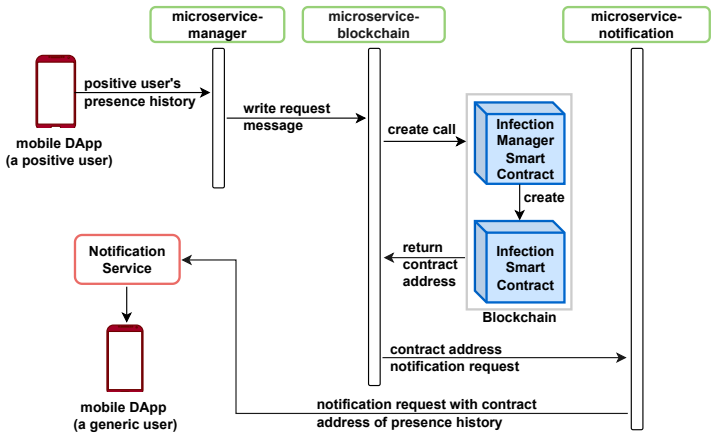


Figure 4.10: Positive user reporting workflow.

tested and deployed with the aid of integrated tools such as Remix[12], Metamask[13], and Truffle[14]. Ropsten[15], historically based on the PoW consensus mechanism and now discontinued, was initially selected as the testnet environment to deploy the developed smart contracts for system testing on public blockchains. More recently, a subset of the same tests have been replicated on Goerli[16], the Ethereum testnet currently based on PoS.

In this smart contract's scenario, a positive user's `Presence History` is sent to the *Infection Manager Smart Contract* from the backend's *microservice-blockchain* as a serialized JSON payload (see Listing 4.1). The received string is permanently written in a newly created *Infection Smart Contract*'s storage with no extra manipulation (see Listing 4.2). The *Infection Smart Contract*'s address is returned to the calling *microservice-blockchain*, according to the workflow sketched in Figure 4.10.

```
pragma solidity ^0.7.2;
import "Infection.sol";
contract InfectionManager {
  address payable private owner; //state variable owner contains owner's
      address
  //initialize contract
  constructor() {
```

[12] https://remix.ethereum.org/
[13] https://metamask.io/
[14] https://www.trufflesuite.com/
[15] https://ropsten.etherscan.io/
[16] https://goerli.etherscan.io/

```
    owner = msg.sender; //store contract owner <-- microservice
  }
  //create a new infection smart contract and return its address to caller
  function newInfection(string memory _payload) external returns (address) {
    Infection infection = new Infection(_payload);
    address added = address(infection);
    return added;
  }
  //destroy an expired smart contract whose address is given by the caller
      <-- microservice
  function removeInfection(address delendus) external {
    Infection history;
    history = Infection(delendus); // this address can be given from caller
    history.destroy();
  }
}
```

Listing 4.1: Infection Manager Smart Contract.

```
pragma solidity ^0.7.2;
contract Infection {
  string private payload; //state variable payload contains json-serialized
      data infection
  address payable private owner; //state variable owner contains owner's
      address
  //initialize contract
  constructor(string memory _payload) {
    owner = msg.sender; //assign ownership to the contract's creator <--
        microservice
    payload = _payload; //write serialized json-serialized string into
        contract's storage
  }
  //PresenceHistory getter
  function getPresenceHistory() public view returns(string memory){
       return presence_history;
  }
  //destroy contract with expired information
  function destroy() external {
       require(msg.sender == owner); // only the microservice can destroy
           the contract
       selfdestruct(owner);
  }
}
```

Listing 4.2: Infection Smart Contract.

The implemented contract pattern, referred to as ***Data Eraser*** keeps things simple and flexible, when it comes to deleting smart contracts' information no longer valid. It belongs to a design pattern family generally applicable where business logic and data must be kept separated. Differently from the *Registry Contract* pattern, it provides a cost-effective mechanism to create and destroy data contract when no longer needed, as in the case of the presence histories of users no longer positive.

The client software functionalities do not differ at all from the Nausi-

caApp's, save for the way a positive user's `Presence History` is downloaded by an *mDApp* to feed the contract tracing routines and perform data matching. In fact, the *mDApps* are only allowed to interact with the blockchain to download the `Presence History` from a smart contract whose address is notified to them by the server's *Notification Service* (see Listing 4.3).

The *mDApp* code has been developed in Java by means of Android Studio [17] platform into which the Web3j library[18] has been imported. By doing so, the *mDApps* are allowed to interface with the JSON-RPC APIs, exposed by the Ethereum nodes, at a higher programming level (i.e., Java) than a native HTTP POST request with JSON-encoded data containing the transaction to be sent (see Subsection 2.3.2).

The `Web3j.build(httpService)` method is used to establish a connection between the client software and an HTTP end-point which, in turn, acts as a Web3 access point to the blockchain network. The endpoint is provided by the Infura infrastructure[19] which can be loosely regarded as a proxy Ethereum node (see 2.3.2).

Another example of how the injected Web3j properties are used is shown in Listing 4.3, where the method `blockchainRead()` is capable of getting the positive user's `Presence History` by invoking the getter function `getPayload()` from the referenced *Infection Smart Contract*. This action returns the `Presence History` of the infected user, which is compared by the *mDApps* software with the locally stored presence data by means of a heuristics that is able to assess if places and times of the compared data can be considered coincidental within established limits.

```
public class Utils {
    Web3ClientVersion web3ClientVersion = null;
    static HttpService httpService = new HttpService(AppConfig.
        BLOCKCHAIN_ENDPOINT_URL);
    static Web3j web3 = Web3j.build(httpService);
    static Credentials cr = Credentials.create(AppConfig.
        BLOCKCHAIN_MAIN_ACCOUNT_SECRET_KEY);
    static BigInteger gasLimit = BigInteger.valueOf(20_000_000_000L);
    static BigInteger gasPrice = BigInteger.valueOf(4300000);
    private static Infection inf = null;
```

[17] https://developer.android.com/studio
[18] https://github.com/web3j/web3j
[19] https://infura.io/

```
    public static Infection instance(String contractAddress) {
        if (inf == null) inf = Infection.load(contractAddress, web3, cr,
            gasLimit, gasPrice);
        return inf;
    }
    public static String blockchainRead(String contractAddress) {
        if (android.os.Build.VERSION.SDK_INT < android.os.Build.
            VERSION_CODES.N) return null;
        Future<String> payload = instance(contractAddress).getPayload().
            sendAsync();
        String returned;
        try {    returned = payload.get();
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }
        return returned;
    }
}
```

Listing 4.3: mDApp code: retrieving information from an Infection Smart Contract.

Server-wise, a portion of the *microservice-blockchain* code is shown in Listing 4.4, in which the same `Web3j.build(httpService)` method is used to establish a connection between the microservice software and an HTTP end-point providing access to the blockchain. The `InfectionManagerBlockchainService()` method is in charge of creating the *Infection Manager Smart Contract* by invoking the `InfectionManager.load()` method. As explained in Subsection 4.2.1, an Externally Owned Account (EOA), managed by a digital wallet facility, needs to be linked to the microservice in order to let it create the *Manager Smart Contract*. The EOA is created, in our testbed, by means of the *MetaMask* browser extension, which manages the EOA balance and digitally signs the transaction generated by the blockchain microservice.

```
public class InfectionManagerBlockchainService {
    private InfectionManager infectionManager;
    public InfectionManagerBlockchainService() {
        HttpService httpService = new HttpService(AppConfig.
            BLOCKCHAIN_ENDPOINT_URL);
        Web3j web3j = Web3j.build(httpService);
        Credentials cred = Credentials.create(AppConfig.
            BLOCKCHAIN_MAIN_ACCOUNT_SECRET_KEY);
        infectionManager = InfectionManager.load(AppConfig.
            INFECTION_CONTRACT_ADDRESS, web3j, cred, Constants.GAS_LIMIT,
            Constants.GAS_PRICE);
    }
}
```

Listing 4.4: microservice-blockchain: creating the Infection Manager Smart Contract.

A similar approach is used for the overcrowd reporting scenario, in that the *Overcrowd Manager Smart Contract* performs the calls to create the *Occupation Smart Contracts'* storage on behalf of the back-end server. As before, for performance and straightforwardness motivations, a unique JSON serialized string, containing all the periodic report data, is transferred and stored in the *Occupation Smart Contracts*. The same *Data Eraser* pattern can be applied whenever aggregate data, no longer needed, are to be deleted from the contracts' Storage Trie.

#### 4.2.1.5 Gas consumption tests

Although a quite refined storage management has been implemented in the first version of the DApp, in that unneeded user or aggregate data can be easily removed from the blockchain by destroying its associated contract (see Subsection 4.2.1.4, the gas used for a typical transaction execution would end up exceeding the block gas limit quite soon, because of the size of stored data. This behaviour is due to a well-known gas-related issue with the Ethereum Virtual Machine, when it comes to storing bulk data in its data structures.

**Experimental tests**

The gas consumption tests have been carried out by creating a test transaction in Remix that simulates the *microservice-blockchain* in the act of calling the *Infection Manager Smart Contract (InfectionMng.sol)* to store the `Presence History` of an infected user in a new contract's storage. In fact, the transaction triggers a contract call cascade in which the the *Infection Manager Smart Contract* creates a new *Infection Smart Contract (Infection.sol)* by providing a string of variable byte-length at each test (i.e., 6B, 200B, 10000B, 20000B, 40000B, 100000B), which simulates the JSON-encoded data sent by the *microservice-blockchain* to the *Infection Manager Smart Contract*.

According to the *gas fee schedule* contained in Ethereum's yellow paper [16], writing data to the *smart contract's storage* is by far more expensive than using, for the same purpose, the *transaction logs*, stored in the

blockchain's Receipt Trie. Therefore, in a later refinement of the smart contracts' code, bulk data have been stored in transaction logs by replacing the contract's state variables with the emission of events, as reported in Listing 4.5 that shows the modified Infection Contract *(InfectionLog.sol)*. Notice that the generated events contain the address of each newly created contract, which can be therefore used as a research index for later enquiries from the DApps' front-end.

```
pragma solidity ^0.7.2;
contract Infection {
    address payable private owner;
    //presenceHistory contains the infected user's presence data in json
        format
    event presenceLog(address indexed presenceContract, string
        presenceHistory);
    //initialize contract and create event
    constructor(string memory _payload) {
        owner = msg.sender;
        emit presenceLog(address(this), _payload);
    }
   function destroy() external {
       require(msg.sender == owner);
       selfdestruct(owner);
    }
}
```

Listing 4.5: Infection Smart Contract storing infection data in the transaction logs.

In either software versions the gas usage would still easily ramp up beyond any sensible limit, as confirmed by the experimental results presented in Figure 4.11 on the dichotomy between the (smart) *contract storage* vs. *transaction logs* solutions. Gas consumption, for both the said storage solutions, are plotted against the amount of data transferred from the server-side by the simulated *microservice-blockchain* in the test environment[20].

Note that the "contract storage" solution runs out of gas as soon as data size exceeds 10,000 Bytes (hence the missing blue bars in the graph), causing the transaction not to be completed by the Ethereum Virtual Machine. The graph shows that storing in "transaction logs" performs better, but data larger than 100,000 Bytes (the graph's upper bound) still prevent transactions from completing.

[20]The smart contracts, used for the said tests, can be retrieved at addresses: `0x158541aFd73F32Ee7b2FfEaF2c51cA7d62BF0be6` *(InfectionMng.sol)* and `0xa1cf4Ce1ed23F84349828C1eac2D4F897Ed82ce1` *(InfectionMngLog.sol)* in Ropsten.

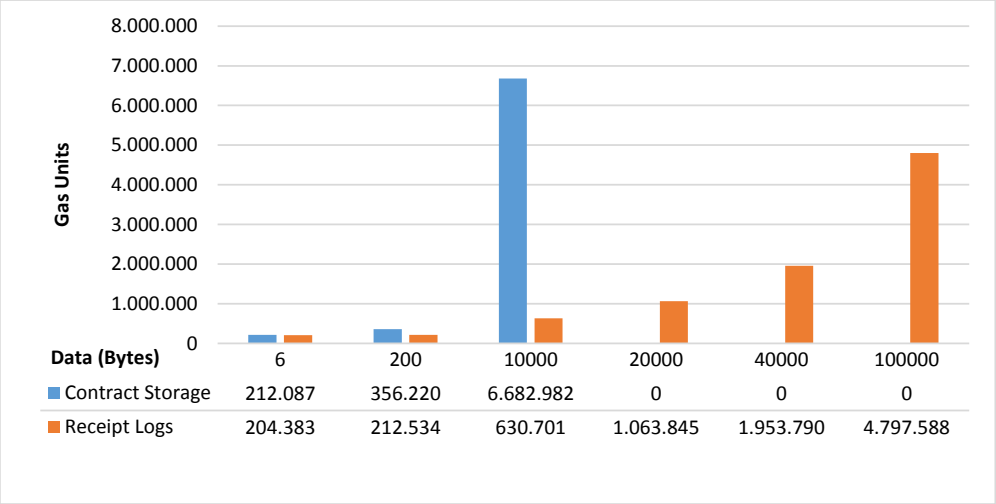


Figure 4.11: Comparison of gas consumption for storage of transaction data: smart contracts vs. transaction logs.

### Transaction costs

In real terms, considering the gas price and the ether market value (the test were first performed at a time the Ethereum was still running on PoW), a transaction storing 10,000 Bytes on-chain would cost (in US $), for each of the two above-mentioned solutions:

***Contract Storage:***

$$6,682,982[gas\ unit]*163[gwei/gas\ unit]*3.496{\cdot}10^{-6}[\$/gwei] = 3,808.28[\$]$$

***Transaction Logs:***

$$630,701[gas\ unit] * 163[gwei/gas\ unit] * 3.496 \cdot 10^{-6}[\$/gwei] = 359.40[\$]$$

With these very high gas price and ether value, the two solutions, although satisfactory designed to meet the project requirements, pay the price of being run in a virtual machine that relies on high transaction costs to secure its operations, especially on PoW networks, which claim high rewards for its miners.

**Further tests**

In a recent recalculation of the same transaction costs, at a time when the Ethereum had already transitioned to PoS, the gas price was much lower than prior to transition, according to a trend that may be associated to the dropping of PoW[21]. Unfortunately these considerations on gas price and ether value fluctuations over time cannot be thoroughly supported with data accrued in such a short period. These updated calculations are reported below.

***Contract Storage:***

$$6,682,982[gas\ unit] * 8[gwei/gas\ unit] * 1.43 \cdot 10^{-6}[\$/gwei] \ = \ 76.45[\$]$$

***Transaction Logs:***

$$630,701[gas\ unit] \ * \ 8[gwei/gas\ unit] \ * \ 1.43 \ \cdot \ 10^{-6}[\$/gwei] \ \ = \ \ 7.22[\$]$$

However, with these new parameters, even the second solution is still quite expensive to legitimate storing bulk data in the blockchain transaction logs. The definitive response to the transaction costs would be in a different design of the smart contract pattern, as described in Subsection 4.2.2. Furthermore, the second solution exhibits an annoying drawback, that is leaving the pandemic data lastingly on-chain even when they are no longer needed, since the transaction logs pertain to the permanent storage of the Ethereum storage, unlike the ephemeral contract storage. A useless and insecure burden abandoned in the network that, in addition to the high transaction costs, calls for further investigation.

As an appendix to the transaction cost tests, a series of trials confirm that transition to PoS has not varied the rules in calculating the gas fee for each EVM operation, Figure 4.12 contains a table in which repetead runs of the *InfectionMngLog.sol* contract, under different input payloads, have been compared in two different execution environment, namely Ropsten and

[21] `https://ycharts.com/indicators/ethereum_average_gas_price`

Goerli[22], which now is the preferred Ethereum testnet based on PoS[23]

| Paylod (B) | Goërli | Ropsten |
|---|---|---|
| 6 | 132.997 | 130.316 |
| 200 | 138.679 | 135.958 |
| 10000 | 429.302 | 424.463 |
| 20000 | 729.173 | 722.246 |
| 40000 | 1.341.678 | 1.330.276 |
| 100000 | 3.273.004 | 3.245.930 |

Figure 4.12: Comparison of gas consumption in different consensus-driven testnets.

In addition to Ropsten, gas consumption tests have also carried out in Q1 2022 on the Kovan testnet[24], which was based, at the time of testing, on PoA consensus mechanism, to appreciate execution costs in a typical enterprise environment. The results reported in Figure 4.13 compare the transaction costs, detected in the two testnets, to invoke the same function belonging to the "overcrowd management" scenario, which is capable of adding a new report to the relevant contract's storage, whose code is listed below.

```
function addReport(string memory _authToken, string memory department,string
     memory hour, string memory _payload) public {
    require(keccak256(abi.encodePacked(authToken))==keccak256(abi.
        encodePacked(_authToken)), "Invalid token");  departments[department
        ][hour] =_payload; }
```

Listing 4.6: Function to store an overcrowd report to the blockchain.

The test results, unfortunately, did not provide a clear indication that the same transaction, run on Kovan, could have changed the gas costs, if compared to a similar execution on Ropsten. At the time of testing the two networks exhibited a different gas limit per block, so that the transaction run on Kovan, in any case, would fail much earlier, thus not enabling a comparison for payload larger than 14,592B (see Figure 4.13). From the few comparable tests, however, running transactions on PoA networks does not affect their execution costs in terms of gas units, as expected. Nowadays, repeating the tests is impossible due to the dismission of any Ethereum testnet

[22] https://goerli.etherscan.io/

[23] The smart contract deployed on Goerli is retrievable at address: 0x45019008192ccb171f8ec1eb9dde3967b2caccdb.

[24] https://kovan.etherscan.io/

based on PoW[25] and PoA[26]. In fact, after a few years of experimentation, the Ethereum foundation is quite exclusively focusing on the development of their PoS-based consensus mechanism, namely *Gasper*[27], *"because it is more secure, less energy-intensive, and better for implementing new scaling solutions compared to the previous proof-of-work architecture".*

| Payload size (bytes) | Gas units | Ether (ETH) |
|---|---|---|
| 16 | 33.821 | 0,005 |
| 304 | 238.372 | 0,0087 |
| 2.432 | 1.409.587 | 0,0137 |
| 4.864 | 1.702.901 | 0,0511 |
| 14.592 | 6.502.832 | 0,195 |
| 22.464 | 5.731.263 | 0,172 |
| 31.168 | 6.608.905 | 0,1983 |
| 40.128 | 7.152.380 | 0,2146 |
| 50.688 | 8.572.256 | 0,2572 |
| **74.416** | **12.499.988** | **0,375** |

(a) Gas consumption in Ropsten

| Payload size (bytes) | Gas units | Ether (ETH) |
|---|---|---|
| 16 | 30.699 | 0,0026 |
| 304 | 239.450 | 0,0103 |
| 2.432 | 1.410.665 | 0,0607 |
| 4.864 | 1.703.980 | 0,0733 |
| **14.592** | **2.000.000** | **0,086** |

(b) Gas consumption in Kovan

Figure 4.13: Gas consumption comparison

### 4.2.2 DApp_v2

As for transaction costs, notice that both software versions tested in DApp_v1 are purely on-chain, storage-wise. The gas consumption tests carried out on DApp_v1 clearly demonstrate that the on-chain approach is nearly impracticable due to the transaction costs in a Ethereum-based blockchain, regardless of the adopted consensus mechanism. Therefore, the final version of Nausica@DApp (DApp_v2) incorporates a ***design pattern*** that provides for an on-chain/off-chain approach, which combines the storage of only essential information in the blockchain and the upload of pandemic-related bulk data in

[25] As of Q3 2022 Ropsten has been dismissed, currently in read-only state.

[26] As of Q3 2022 Kovan has also been dismissed, currently in read-only state.

[27] `https://ethereum.org/en/developers/docs/consensus-mechanisms/pos/gasper/`

a system DB (the specific pattern is discussed in Subsection 4.2.2.5). In addition to performance improvements, this new approach adds extra security features to the whole project, in that the information stored in the blockchain guarantees the integrity of the pandemic data stored in the system DB, which, in turn, may implement either native or external user authentication and data confidentiality features.

The administration back-end now interfaces with a *System DB* (see Figure 4.14), the adopted database (whether centralized or decentralized) in which pandemic-related data are stored at the disposal of authorized parties. These data are hashed and hashes are used as keys to refer to them in on/off-chain correlated operations. For this reason they will be termed *hash-pointers* hereafter. The ad-hoc blockchain components incorporated into the system architecture, in fact, enable the back-end server to expose, in a decentralized and unforgeable storage, the hash-pointers to the anonymized proximity data, which are needed by the mobile DApps to perform the internal matching operations for contact tracing. Furthermore, the blockchain components will allow other involved stakeholders, such as the academic community and the health authorities, to retrieve aggregate data about pandemic trends in a transparent and decentralized consensus-driven environment.

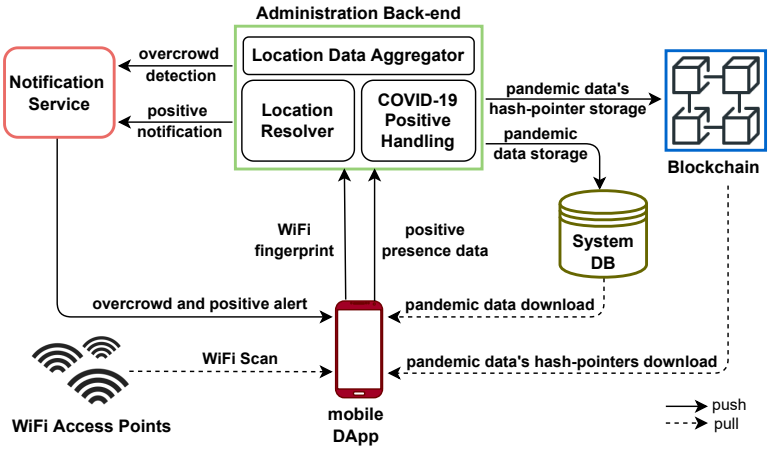


Figure 4.14: Nausica@DApp system components.

The back-end's microservice architecture changes according to the ad-

dition of the system DB, as shown in Figure 4.15. Therefore, pandemic data need be stored and retrieved in two separate steps, involving both the blockchain and the system DB storage (see next Subsection).

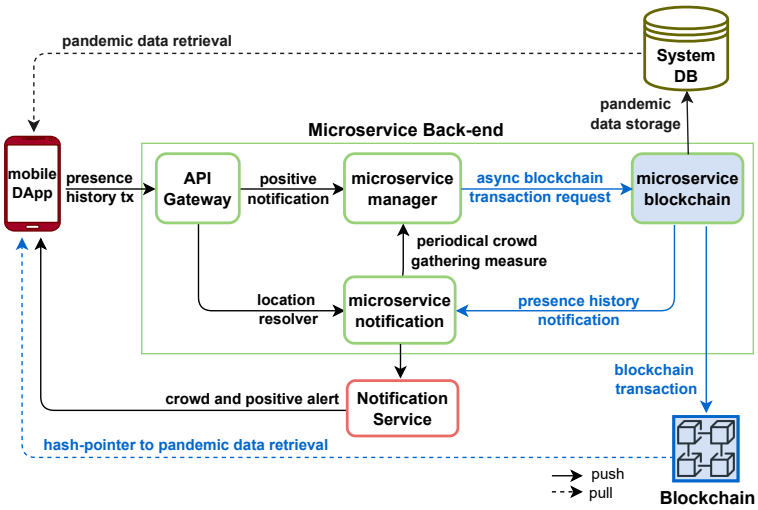


Figure 4.15: Back-end architecture with blockchain integration (blue items).

#### 4.2.2.1 Contact tracing in DApp_v2

In accordance with the new design of the decentralized application, the contact tracing process, shown in Figure 4.16, is modified as discussed below:

i. a positive-tested user sends its `Presence History` to the central server as usual;

ii. the server writes the `Presence History` of the positive user to a newly created DB item (2a) and stores a hash-pointer to the item in a blockchain dedicated data structure (2b);

iii. the indexed references to the stored blockchain information are then transmitted by the server to the *Notification Service* (3);

iv. the *Notification Service* pushes a contagion risk notification to the user apps along with the indexed references to the blockchain data structure,

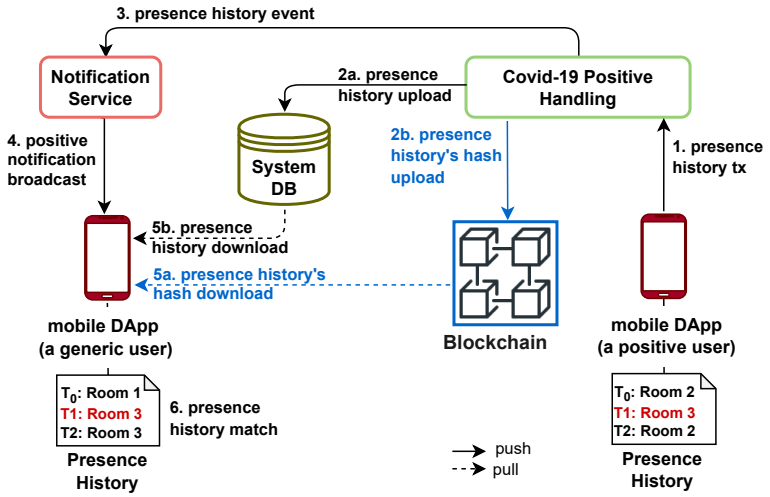


Figure 4.16: Contact tracing process, involving smart contracts (blue text and arrows), with positive matching (red entries in presence histories).

which, in turn, contains the hash-pointer to the `Presence History` of the positive user (4);

v. each notified app pulls the `Presence History` of the new positive user from the relevant DB location (5b), through the hash-pointer previously retrieved from the blockchain (5a);

vi. each app correlates, according to a given heuristics, places and times in the `Presence Data` contained in the pulled `Presence History` of the positive user with its locally stored own, in order to detect direct and indirect contacts (6).

#### 4.2.2.2 Modified blockchain components

In the modified on-chain/off-chain approach, pandemic data can now be collected in either a centralized or distributed storage and made reachable to the involved parties in a transparent, authenticated and unforgeable manner.

Below, the features of the new smart contracts that have been integrated in our solution, and their mutual interactions, are described. An architectural representation is given in Figure 4.17.

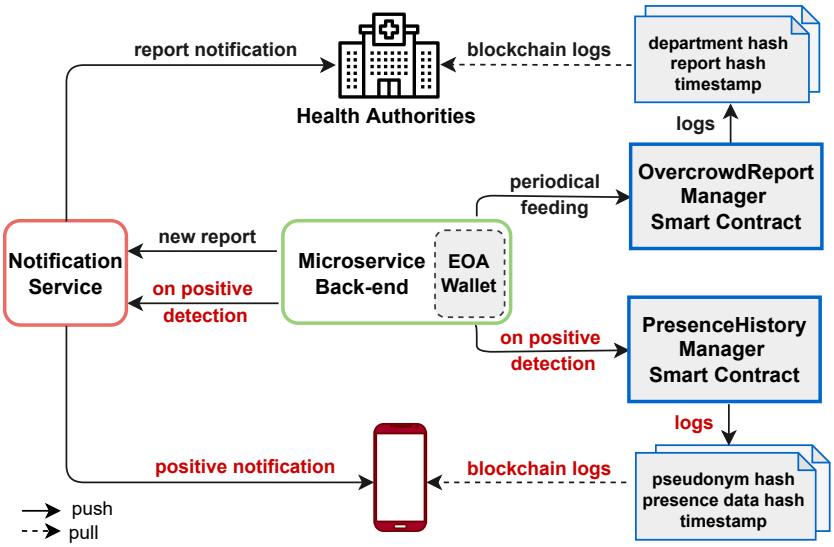


Figure 4.17: Interactions of smart contracts with system components in two scenarios: positive detection (red) and overcrowd reporting (black).

The new design envisages two smart contracts, one for each working scenario, namely "contact tracing" and "overcrowd monitoring".

In the contact tracing scenario a *PresenceHistoryManager* contract is used. It is a single-instance smart contract deployed and interacted with through the relevant blockchain back-end microservice, which, in turn, is associated with an Ethereum's EOA. A simplified workflow of the contact tracing scenario is depicted in Figure 4.18, with regard to how the distributed system services and components interact in the new system design, according to the following operations:

i. each time the server receives a notification from a positive-tested user's device it securely stores the relevant `Presence History` (see Subsection 4.1.3) in a DB item, whose related hash-pointer is now computed;

ii. the server's *microservice-blockchain* sends the hash-pointer and the SHA3 hash of the anonymized user ID to the *PresenceHistoryManager*, by triggering a proper transaction containing the associated function call;

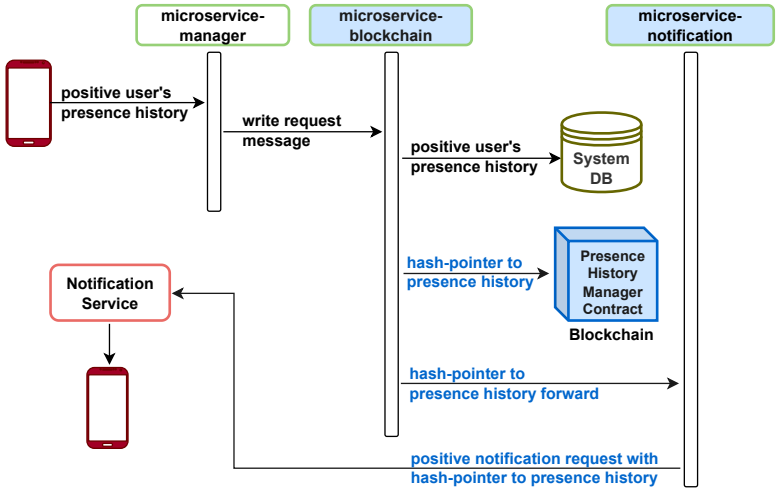


Figure 4.18: Positive user reporting workflow through system components.

iii. as soon as the block containing the above transaction is validated, the *PresenceHistoryManager* creates an event containing the two received hashes and the block timestamp that are included in the blockchain transaction logs for external applications to retrieve them;

iv. upon confirmation of the execution of the triggered transaction, the *microservice-blockchain* sends the *Notification Service* the hash-pointer to the stored `Presence History`; this will be notified to all the enabled mobile devices, so that they can download the `Presence History` from the DB through the received hash-pointer;

v. finally, the mobile devices can perform contact tracing matching with their own locally stored data.

Similar operational processes can be applied to the overcrowd monitoring scenario.

#### 4.2.2.3 Blockchain-related software

This section gives more detailed specifications about the blockchain-related code in DApp_v2[28].

[28] `https://github.com/giongion19/thesis-nausicaadapp`

In the context of the new simplified design of employed smart contracts, there is no change in the *microservice-blockchain* Java code that deploys the two manager smart contracts (see Subsection 4.2.1.4).

Conversely, the smart contracts employed in this new version differ from their predecessors because of a simplified design in which only the two manager smart contracts are, in fact, deployed, namely *PresenceHistoryManager* and *OvercrowdReportManager*, thus avoiding multiple instantiations of new smart contracts every time a new infection or a new report has to be stored in the blockchain storage. Only limited, hashed information is recorded in the contract storage area, as explained in Subsection 4.2.2.

The code in Listing 4.7 refers to the contact tracing scenario in which, as an outcome of the execution of the `newInfection()` function, the *PresenceHistoryManager* emits the event `newHistoryInserted()`, thus storing in the transaction logs, inside the Receipt Trie of the Ethereum blockchain, the following parameters:

- `pseudonymHash`, the infected user's hashed pseudonym, for indexed research of associated events;
- `presenceHistoryHash`, the hash-pointer to the JSON `Presence History` DB object;
- `timestamp`, the block's time registration in the chain.

Once the transaction has been successfully completed, the *microservice-blockchain* can start the `Presence History` notification process, previously discussed in Subsection 4.2.1.2 and depicted in Figure 4.15. This will eventually supply the mobile apps with all the references to the blockchain transaction logs, which contain hash-pointers to DB items storing the anonymized infected user's `Presence History`.

```
contract PresenceHistoryManager {
  address payable public owner;
  //event reporting the infected user's presence history hash and its
      timestamp
  event newHistoryInserted(bytes32 indexed pseudonymHash,bytes32
      presenceHistoryHash,uint timestamp);
  constructor() {
```

```
    owner = msg.sender;
  }
  //function to create a new infection's log entry
  function newInfection(bytes32 pseudoHash, bytes32 presenceHash) external {
      emit newHistoryInserted(pseudoHash, presenceHash, block.timestamp);
  }
  function destroy() external {
    require(msg.sender == owner);
    selfdestruct(owner);
  }
}
```

Listing 4.7: Presence History smart contract.

Listing 4.8 shows a fragment of JavaScript test code mimicking the *micro-service-blockchain* interaction with the above *PresenceHistoryManager* contract.

```
const contractAddress = '0x820813999ad2b1c28415a4daead0d998268b1a48';
//import contract metadata
const metadata = JSON.parse(await remix.call('fileManager','getFile','
    PresenceHistoryManager.json'));
//create a contract instance
let contract = new web3.eth.Contract(metadata.abi, contractAddress);
//send the transaction to the contract - uid and prs (presence) are test
    data
const uid ='0
    xbb27bef7fede14d52825920b5c55650dc753047c9a9f900f9b83a12686b870f8';
const prs ='0
    xd7fbc8b6b0b40750006bcc870088df9c40988fec7a8798545073991a3a2d891d';
const receipt = await contract.methods.newInfection(uid, prs).send({
  from: contract.defaultAccount});
```

Listing 4.8: Test code sending data to a smart contract.

In the script, the transaction call to the contract's `newInfection()` function is fed with test data (`uid`, `prs`), which, in turn, will be stored in the transaction logs through the event specifically emitted by the called function, to be thereafter made available to client apps for contact matching[29].

The **client software** workflow follows a pattern that starts, server-side, from the *Notification Service*, which prompts the client apps to pull, from the blockchain transaction logs, the hash-pointer to either the contact tracing data or the overcrowd report data, as discussed in Subsection 4.2.2.1, and depicted in Figures 4.16. Client apps will thus be able to de-reference the data structures stored in the system DB, possibly after honoring additional DB access rules, and perform data authentication of any DB item by comparison

[29]The script makes use of web3.js (`https://web3js.readthedocs.io`) library methods.

with its unforgeable hash-pointer stored in the blockchain.

The JavaScript test code fragment in Listing 4.9 simulates the essential behavior of a web app pulling previously logged data from the blockchain, by calling the web3.js `getPastEvents()` method. The example code extracts from the blockchain a hash-pointer to the associated `Presence History` items stored in the system DB.

```
const contractAddress = '0x820813999ad2b1c28415a4daead0d998268b1a48';
//import contract metadata
const metadata = JSON.parse(await remix.call('fileManager','getFile','
    PresenceHistoryManager.json'));
//create a contract instance
let contract = new web3.eth.Contract(metadata.abi, contractAddress);
const uid ='0
    xbb27bef7fede14d52825920b5c55650dc753047c9a9f900f9b83a12686b870f8';
//retrieve from the blockchain logs the hash-pointers related to a user-ID
const events = await contract.getPastEvents('newHistoryInserted', {
  filter: { from: contractAddress, pseudonymHash: uid }, fromBlock: 0 })
//print the retrieved information
events.forEach(element => {
  console.log(element.returnValues['presenceHistoryHash'])
  console.log(element.returnValues['timestamp']) })
```

Listing 4.9: Test code to pull transaction logs generated by a smart contract.

#### 4.2.2.4 Gas consumption tests

The storage capabilities of the two Nausica@DApp smart contracts, in the final version DApp_v2, have been limited to the writing of hash-pointers to off-chain bulk data on transaction logs, as a consequence of well-known gas-related issues previously discussed in Subsection 4.2.1.5.

The mixed on-chain/off-chain storage pattern, which stores essential hashed data in the transaction logs, turns out to yield, for a typical transaction, a fixed gas usage of 28,357 gas units[30], whose current price in US dollars is quite reasonable.

***Mixed on-chain/off-chain:***

$$28,357[gas\ unit] * 8[gwei/gas\ unit] * 1.43 \cdot 10^{-6}[\$/gwei] = 0.32[\$]$$

[30]Transaction hash in Ropsten:
0x9df28e24396aed2fb2930afcabc2c0613f6e62bd576a58190c6d17e0a95ad177

The above is an appropriate cost per transaction for the Ethereum PoW ecosystem, dramatically lower than the computed costs of DApp_v1. This reduced cost comes as a result of the combined effect of strictly limiting the amount of data stored in the blockchain and employing the transaction logs as a storage repository. The same transaction, executed on the PoS-based Goerli testbed, has led to similar results (29,184 *gas unit*)[31], once more confirming the transparency of gas fees with respect to the consensus protocol used by the Ethereum platform.

Finally, the mixed on-chain/off-chain pattern does not burden excessively the blockchain storage even when references to the data, which are deleted from the on-chain storage, are left dangling in the transaction logs, due to the limited amount of space occupied by them.

#### 4.2.2.5 Decentralizing the system DB

The mixed on-chain/off-chain design pattern, implemented in DApp_v2, falls in the ***Blockchain Anchor*** family, a simple and robust pattern that uses hashes to ensure the integrity of an arbitrarily large dataset that may not fit directly on the blockchain.

The off-chain bulk data, at the disposal of authorized parties, can be kept according to a few alternative back-end options, namely by: (i) using a centralized database, (ii) integrating a decentralized file system, or (iii) deploying a complementary blockchain.

In this further study, two of the above implementations have been integrated into the DApp_v2[32]. In one case, a centralized solution has been implemented, using the key-value cloud-based MongoDB Atlas database[33], according to the MERN stack reference model[34]. In the second case, a decentralized solution has been created by using the InterPlanetary File System (IPFS)[35] as the peer-to-peer hypermedia protocol for storing and sharing

[31] Transaction hash in Goerli:
`0x659d80e4a25f308817e1cc7aaee14027f1cb6d331134878ee2c95f7dbf3397dd`

[32] `https://github.com/giongion19/ethToIPFS-MongoDB-PHManagerMongoDB`

[33] `https://www.mongodb.com/atlas/database`

[34] `https://www.mongodb.com/mern-stack`

[35] `https://ipfs.tech/`

data in a distributed file system. The integration into the system's back-end of alternative blockchains, such as MultiChain[36] and FileCoin[37], both based on IPFS, or BigchainDB[38], a distributed database with the characteristics of a blockchain, was not experimented in the first place since a native decentralized solution, such as IPFS, was given a higher priority.

Although the implementation includes the interaction with both centralized and decentralized file systems, in the remainder of the subsection only the IPFS-based portion will be discussed, since it contributes to increase the decentralization level of the overall solution, according to the project goals. IPFS aims to become the new Internet solution, in place of the conventional client-server ones, as far as openness, resilience, and efficiency are concerned in storage operations. It is a decentralized, content-addressable global file system based on the DHT technology, to which a file can be added by hashing its content, thus providing a unique fingerprint called a content identifier (CID), which acts as an address and a proof-of-existence of the recorded file. The look-up protocol allows the IPFS peer nodes to retrieve the node which is actually storing the content referenced by the file's CID.

In our solution, a JavaScript front-end part was created, which allows the administrator to perform two different operations:

- uploading the pandemic data through the `PresenceHistoryManager`, according to the smart contract pattern explained in Subsection 4.2.2.3, whose storing workflow is now depicted in Figure 4.19;
- retrieving the infection information by first querying the Ethereum's transactions logs to download the hash-pointer (i.e., the IPFS' CID obtained in the upload phase) to the correlated data content stored in the IPFS nodes.

Thanks to the *ipfs-api* library it is possible to implement the IPFS protocol within JavaScript, making communication with a remote IPFS node

[36] `https://www.multichain.com/`
[37] `https://www.filechain.com/`
[38] `https://www.bigchaindb.com/`

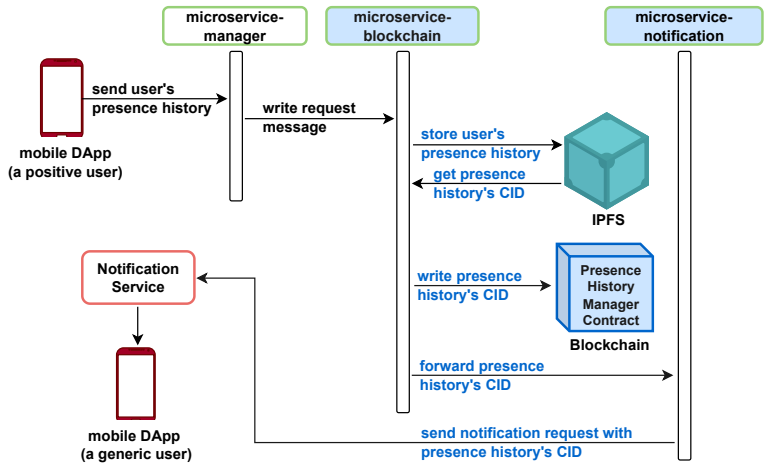


Figure 4.19: Adding a positive user's presence history to IPFS.

possible. In order to correctly save the files on IPFS there are two alternatives: (i) become part of the peer-to-peer network by making your device a full node of the IPFS network; or (ii) use the IPFS remote nodes exposed by the Infura service provider, similarly to what already done to access the Ethereum network by means of a cloud-based remote client. For a simpler deployment of the experimental testbed, the Infura-based solution has been put in place. Listing 4.10 includes the JavaScript module in which the application connects to IPFS through Infura.

```
//using the infura.io node
const IPFS = require('ipfs-api');
const ipfs = new IPFS({ host: 'ipfs.infura.io', port: 5001, protocol: 'https
    ' });
//the ipfs const is exported to including modules
export default ipfs;
```

Listing 4.10: Connecting to IPFS through the Infura service provider.

Data uploading onto IPFS is performed through the `onSubmitData()` function in Listing 4.11, which, first, takes care of storing the data content in IPFS and then returns the file's CID at the disposal of the smart contract (see the `ipfs.add()` function). Then, the `newInfection()` function, contained in the `PresenceHistoryManager` smart contract, is invoked with the user's pseudonym and the file's CID as input parameters, for logging in the transaction receipt.

```
import web3 from './web3';
import ipfs from './ipfs';
import contractPresenceHistoryManager from './contract';
onSubmitData = async (event) => {
  event.preventDefault();
  //bring in user's metamask account address
  const accounts = await web3.eth.getAccounts();
  //obtain contract address from contractPresenceHistoryManager.js
  const ethAddress = await contractPresenceHistoryManager.options.address;
  this.setState({ ethAddress });
  //save document to IPFS,return its hash#, and set hash# to state
  await ipfs.add(this.state.buffer, (err, ipfsHash) => {
    console.log(err, ipfsHash);
    //setState by setting ipfsHash to ipfsHash[0].hash
    this.setState({ ipfsHash: ipfsHash[0].hash });
    // call Ethereum contract method "newInfection" and .send IPFS hash to
        etheruem contract
    //return the transaction hash from the ethereum contract
    contractPresenceHistoryManager.methods.newInfection(web3.utils.
        asciiToHex(this.state.usrPseud),
      this.state.ipfsHash).send({from: accounts[0]}, (error, transactionHash
          ) => {
        this.setState({ transactionHash });
    });
  })
};
```

Listing 4.11: JavaScript module to store data to IPFS and to the blockchain.

Downloading a user's `Presence History` from IPFS is achieved via the `onCheckEvents()` function, which is responsible for retrieving the history's CID by looking up the `newHistoryInserted()` events from the transaction logs. The CID contained in the logs is retrieved by means of the `getPastEvents()` web3.js function (see Listing 4.12), which takes the user pseudonym as the event research index and applies properly tailored filters.

```
import React, { Component } from 'react';
import web3 from './web3';
import ipfs from './ipfs';
onCheckEvents = async () => {
  this.setState({ usrPseud: document.getElementById('insPseudonymHash').
      value })
  var contractAddress = await contractPresenceHistoryManager.options.address
      ;
  var eventName = "newHistoryInserted";
  const pseudonymHash = web3.utils.asciiToHex(this.state.usrPseud);
  const events = await contractPresenceHistoryManager.getPastEvents(
      eventName, {
    filter: {
      from: contractAddress,
      pseudonymHash
    },
    fromBlock: 0,
    toBlock: 'latest'
  })
  if (events.length === 0) {
```

```
    this.setState({ latestPresenceHash: "Cannot find presence with the
        pseudonym indicated" });
    this.setState({ latestPresenceTimestamp: "----" });
  }
  events.forEach(element => {
    var latestPresenceHash = element.returnValues['presenceHistoryHash'];
    var latestPresenceTimestamp = element.returnValues['timestamp'];
    var finalURL = "https://gateway.ipfs.io/ipfs/" + element.returnValues['
        presenceHistoryHash'];
    this.setState({ latestPresenceHash });
    this.setState({ latestPresenceTimestamp });
    this.setState({ finalURL });
  })
};
```

Listing 4.12: JavaScript module to retrieve data from IPFS by first retrieving the data's CID from the blockchain logs.

### 4.2.3 Comparative analysis

In this final subsection, only the enhancements provided by the integration of blockchain components into the decentralized application's system architecture will be discussed against comparable solutions already presented in Subsection 3.1.2. Comprehensive considerations about the entire DApp solution will be made in the final section of the chapter.

As seen, *BeepTrace* [56] is a solution that takes into account many security issues and adopt quite complex mechanisms to address them. However, the drawbacks can be found in the centralized processing of contagion risks and the complexity of the proposed solution in a corporate-wide scenario. In BeepTrace, in fact, the involvement of the blockchain is broadly pervasive from the very beginning, that is, data storage of healthy people, performance and scalability issues may occur, so much that a lightweight consensus protocol, such as Direct Acyclic Graph, is advised.

*Song et al.,*'s solution [57] also suffers from performance and scalability issues due to the heavy data burden on the blockchain. The proposal tends to undermine the performance and the scalability of the whole system, since it stores all location-based and user-based data in the chain for further central processing, thus incurring an issue similar to DApp_v1's. Similar concerns are found in Arifeen et al.,'s proposal [58], since recording all the users' contact lists on the blockchain can heavily stress the system.

With respect to these three decentralized solutions, Nausica@DApp, in its final version, makes a more confined use of the blockchain functionalities; it is in fact designed with the idea in mind that their adoption must be traded off against the overall system performance. Storing of only reference data in the blockchain is a choice that converge towards this target.

*PRONTO-C2* tackles the linkability risks of pseudonyms, emitted by the users' devices in an app-to-app paradigm, by means of a mechanism based on DH protocol. Although an important issue in fully DP3T-like DApps on public blockchains, the linkability risk could be mitigated, in our solution, by further encryption or masquerading techniques provided by the FCM service providing the user-ID tokens. On the other hand, the PRONTO-C2 solution does not provide any localization-based service to the enabled users, as it eschews any centralized intervention from the system, thus also depriving the solution of the possibility to aggregate data for epidemiological surveillance. The lack of this property in DP3T-like applications has instead pushed forward the concepts highlighted in *Micali*'s proposal based on the Algorand blockchain. Such concepts are fully adopted in the Nausica@DApp's rationale, which offers a composite mix of pandemic data both related to users' contacts and to locations' monitoring, the latter not contemplated by Micali's analysis, which does not provide for a localization system.

In this respect, the Proof-of-Location protocol proposed by *Amoretti et al.,* and implemented in *ByChain* could be an interesting add-on to the campus WiFi infrastructure to secure against the injection of forged locations. However, in Nausica@DApp possible forged locations can be easily identified by the Location Service, which makes use of an internal table of enabled WiFi'APs, thus pre-empting the issue.

Finally, the *Marbouh et al.*'s proposal, although exhibiting a smart contract architecture, does not deal at all with contact tracing and crowd gathering monitoring, However, it shows an interesting use of oracles[39] that can be taken into consideration for further development if input of external data from trusted sources should be required.

[39]https://docs.ethhub.io/built-on-ethereum/oracles/what-are-oracles/

## 4.3 Complete solution wrap-up

Nausica@DApp, the complete solution equipped with blockchain components and additional decentralized storage, has added new elements to the decentralization paradigm of the original solution, namely NausicaApp, by specifically leveraging the key features for which the blockchain usage is widely recognized, such as data integrity assurance in a public, consensus-driven environment.

The development of the blockchain-related features has undergone two phases. In the first one, a refined design pattern has been devised to allow the smart contracts to flexibly manage the on-chain pandemic data, without resorting to additional system databases. Unfortunately, it had to face high transaction costs for the heavy payload to record in the contract's storage, which led to theoretical and experimental unpractical results. Consequently, the design of the second version has chosen not to overuse the Ethereum storage capabilities, by selectively employing them as a secured, transparent, immutable shared repository of limited and specific data, which act as integrity and existence proofs for the bulk data, elsewhere stored. In addition, performance and scalability improvements of the second version did not have a dramatic downsizing impact on the decentralization property of the final solution, especially if the tamper-proof InterPlanetary File System is integrated as the decentralized system DB.

Overall, Nausica@Dapp, in its final configuration, leaps forward other existing contact tracing solutions since no extra and costly operations are required on its part, according to a design which favors not only the decentralization approach in most aspects, from the contact tracing matching to the handling of anonymized user information, but also the system performance and scalability, thus avoiding inflated expectations poured out on the pervasive use of the blockchain technology. No purely technology-driven choice has been made. Only effectiveness and efficiency in the target utilization environment, be it a university or a similar context, have been regarded as the sole drivers of the research and prototyping efforts.

The key features exhibited by Nausica@DApp, which make it an advanced

corporate solution, are below recapped:

i. the absolute localization mechanism implemented on top of the campus WiFi infrastructure, which caters for indirect contract tracing and overcrowd monitoring in selected indoor spots;

ii. the decentralized app-to-app paradigm not based on the Bluetooth channel, which enhances the privacy preserving degree with respect to other contact tracing solutions;

iii. the microservice back-end, which is capable of integrating advanced blockchain components into the system within an efficient and modular architecture;

iv. the blockchain's smart contracts and objects, which provide additional decentralized and security features to the application with a design that does not undermine computational costs;

v. the decentralized storage solution, which smoothly fits into the overall architecture and reduces the performance and scalability issues that an uncritical use of the blockchain would instead introduce.

From the decentralization side, the balanced combination of related elements in the devised solution has distanced the consequences of the "blockchain trilemma", in that security, scalability and decentralization can be now pursued with nearly equalled achievements. Additionally, feasibility, performance and accuracy levels have tested to be very reassuring in all the other solution aspects, such as the absolute localization system.

Although user authentication mechanisms and data encryption algorithms for extra network security may be added in the next future, as in similar corporate applications, the successful deployment of the Nausica@DApp, with its ability to transparently exhibit tamper-evident and immutable statistics, should be able to boost the user's level of trust towards the adoption of the novel decentralized application.

Finally, it is worth saying that the Ethereum's Proof of Work consensus mechanism can be excessively energy-intensive for enterprise needs, while

the process-limiting gas mechanism, to be paid in ethers, can be regarded as an unwanted feature. Even with the newly implemented PoS consensus algorithm [102], the transaction gas fees remain unchanged from the protocol point of view (required gas units), whereas gas price and ether value may fluctuate according to supply/demand market logic.

In order to overcome such limitations, ad-hoc implementation of the Ethereum protocol based on less demanding consensus protocols could serve the enterprise requirements more conveniently. The adoption of PoA, for instance, which could do without using a native asset such as ether, may drastically lower the transaction costs. In addition, PoA may as well reduce the time for the creation of a new block, due to the limited number of validating nodes, which, on the other hand, may negatively affect the decentralization dimension of the trilemma [19]. This drawback can be conveniently tolerated by enterprise applications, especially if complementary mechanisms are built aside or on top of the Ethereum platform to provide extra decentralized features (e.g., the integration of IPFS). To support the validity of such approach for enterprise scenarios, the following chapter presents and discusses a few decentralized applications built on top of a PoA-based Ethereum blockchain.

# Chapter 5

# DER management solutions

Distributed Energy Resources (DER) are small power generation units that supply electricity at customer premises, such as solar panels, home batteries, and electric vehicles. Decentralized production of green energy through DERs[1] is deeply affecting the electricity market, which, along with the availability of smart meters capable of measuring and, more productively, regulating energy provision and usage, help new actors and organizations, such as energy producers/consumers (***"prosumers"***) and energy communities, to actively enter the ***"smart grid"*** market, according to the crowd energy concept [103]. In the said scenario, the blockchain advent can provide a cutting-edge solution in the fluctuating demand/response market of green energy production, whose participants may be prone to distrust each other and would then benefit from a technology that handles the monetary transactions in a transparent way by design [104].

The actual participation of DERs in smart grid markets depends on, so far limited, operator capabilities to communicate with these small-scale assets and their digital controllers. In fact, there is no standard and pervasive digital infrastructure to favor such interconnection process, although many initiatives have taken place in recent years to fill this gap.

For instance, *Equigy* Crowd Balancing Platform[2], is a consortium owned by a few European transmission system operators (TSOs), which aims to

[1] Typically the term DER also implies "renewability", as in RES

[2] `https://equigy.com/`

set cross-industry standards to support a reliable and cost-effective power system that is dependent on renewable energy sources. The driving idea is to provide crowd energy players, such as prosumers, with a trusted data exchange platform to effectively participate in the grid flexibility. One of the underlying ideas is the introduction of a European standard for a new role in the flexibility market, namely the ***aggregator***, a distributed operator facilitating the sale of excess electricity and optimising the pool of flexible energy resources [105]. Unfortunately, Equigy is not an open-source initiative, being headed by leading European TSOs, and very little is known about specific objectives and actual achievements outside the project enclosure.

Conversely, *Energy Web*[3] is an open community that shares the technical and application findings with no access restrictions, in such a way to create a global, common understanding of the direction that the green transition should take in the renewables market. Energy Web is structured as an international non-profit organization accelerating a low-carbon electricity system through the provision of utilities and services, which can help grid operators and solution developers manage the DERs involved in prosumer activities.

According to the above concepts, the drivers of our initiative share the principles of open ***smart communities*** where many scattered actors can take part in the upcoming radical market transition with the utmost level of awareness and transparency of the contractualized energy exchanges [106]. The objective of the study is to evaluate the benefits of introducing the blockchain technology in support of the *aggregator*'s activities, by replacing in a convenient and extensive way the components of a traditional architecture (databases, application servers, etc.), to produce improvement, automation, useful innovation and reduction of operating costs. Therefore, the devised decentralized applications lean on two founding keystones for the provision of an effective, transparent and secured solution: (i) the notion of ***aggregator***; (ii) the adoption of a ***blockchain*** in the overall design [107].

The chapter is structured in three sections. Section 5.1 introduces the dedicated blockchain environment selected to develop the devised decentralized applications. Section 5.2 describes the implemented blockchain-based

[3] `https://www.energyweb.org/`

solutions in the smart grid scenario. Section 5.3 draws the final conclusions on the developed work.

## 5.1 Blockchain environment

Before digging into the design specifications and implementation details of the devised DApp, the current section provides an overview of the application environment in which it has been developed. It is based on a native permissionless blockchain (i.e., Ethereum), which, however, has been specifically tailored for consortium decentralized applications, in that it adopts a consensus algorithm which shifts the access paradigm towards a permissioned one, with selected partners providing the validator nodes. Once again, the adoption of the Ethereum technology from the Energy Web organization demonstrates the flexibility of such a platform with respect to diverse application contexts.

### 5.1.1 Energy Web Decentralized Operating System

The Energy Web Decentralized Operating System (EW-DOS) is an open-source, three-layer digital infrastructure (see Figure 5.1), aimed at helping the development of decentralized applications for a pervasive and competitive decarbonized energy system. EW-DOS is based on the adoption, at the foundational level, of a Proof-of-Authority (PoA) Ethereum blockchain[4], named Energy Web Chain (EWC), which is carried out by a set of designated validator nodes that process transactions and seal new blocks to the EWC in a round-robin fashion. A specific utility native token, the Energy Web Token (EWT), is used to reward validators for creating new blocks and charge users for the operations performed on utility services.

The EWC mainnet and its complementary Volta testnet are publicly accessible through the MetaMask browser's extension with very minimal effort[5].

[4] `https://openethereum.github.io/Aura`

[5] The transactions are visible, respectively, at `https://explorer.energyweb.org/` and `https://volta-explorer.energyweb.org/`

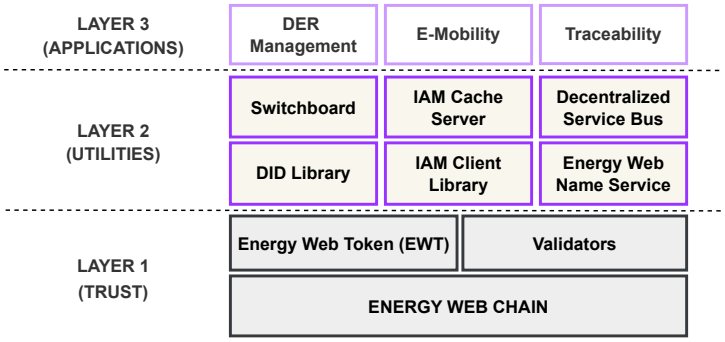


Figure 5.1: The Energy Web Decentralized Operating System layered architecture.

EWC integrates, in its core, a dedicated smart contract mechanism to manage the decentralized identities (DIDs) and the verifiable credentials (VCs) of the entities involved in the applications, according to the W3C recommendation on decentralized identifiers (DID)[6].

Furthermore, the EW-DOS community has implemented and deployed, at the intermediate layer of the EW-DOS stack, a collection of utility packages that enable application developers to share a common protocol for identity handling and information exchange. Such in-built solutions facilitate the integration of clean energy assets, customers and marketplaces within the Energy Web environment.

Energy assets play a pivotal role in the EW-DOS ecosystem. They represent physical or virtual devices with a digital representation (i.e., a DID) whose digital owners manage their participation in decentralized marketplace activities properly designed by EW-DOS developers. An asset can be transferred from an owner to another, with the entire chain of custody being anchored to the blockchain. Current and formerly owned user assets can be retrieved by using ad-hoc utilities of the EW-DOS's intermediate layer, which refer to identity smart contracts permanently deployed in the bottom layer of the EW-DOS stack.

In addition, the EW-DOS's intermediate layer provides a management utility (i.e., the Switchboard) to support Distributed Energy Resources (DER) coordination through identity and access management (IAM) built-in mech-

[6] `https://www.w3.org/TR/did-core/`

anisms, which specifically support decentralized application developers to design and deploy solutions with reduced coding. The purpose of the Switchboard is to allow the developers to easily define what users can do within an organization or an application.

Finally, the Applications layer, in addition to several complete solutions provided by the EW-DOS technical partners and partly discussed in Subsection 5.1.2, contains, in the Traceability box, software development kits (SDK) and toolkits facilitating the exchange of "green" certificates and the tracking of assets in decentralized marketplaces.

The goal of the EW-DOS traceability utilities is to increase transparency and interoperability in the green energy procurement process by means of a decentralized technology. For instance, **24/7** is a software development toolkit for tracking and matching renewable electricity provision and consumption at customizable intervals.

Another toolkit, **Origin**, deals with the Energy Attribute Certifications (EACs). These are documents that account for the "greenness" of purchased electricity and allow the green units of energy to be traded between parties, unfortunately under similar, but loosely compatible, international standards, which, in addition, are managed by central authorities. Conversely, Origin offers customizable modules to create automated cross-standard issuance processes of digital EACs in a decentralized fashion [108].

### 5.1.2 Energy Web applications

Recently, many solutions have been added by affiliated technical partners of the Energy Web community to the upper layer of EW-DOS in the DER Management and e-Mobility sectors.

Some of the DER management applications will be briefly presented below, as they fall in the "grid flexibility" territory, as much as our proposed original DApps, presented in the dedicated Section 5.2 of this chapter, do.

**EDGE**[7] is an Australian initiative in which grid distribution and transmission operators and aggregators can dynamically balance grid services

[7] `https://www.energyweb.org/aemo/`

making use of solar electricity stored in household batteries, which are managed by means of the IAM facilities of the EW-DOS. The Energy Web platform, in turn, integrates with external technology systems, such as market intelligence software and cloud computing resources, to complete the digital infrastructure that makes the remote 'prosumer' systems communicate with the institutional partners.

The EW-DOS technology has recently entered the **Flex Alert** program[8], promoted by a Californian independent Transmission Service Operator (TSO), to maintain grid balance in peak demand hours. The original Flex Alert system was unidirectional: the operator would send an alert to the final recipients to save energy in specific hours. Unfortunately, the operator could not have any confirmation about who would adhere the energy conservation alerts and where the participating consumers resided, thus strongly hindering the effectiveness of the program. Besides making the communication bi-directional, the operator has also integrated the digital identification of participants, based on Energy Web's DIDs, in the enhanced system. The enrolled customers can also choose to respond to the flexibility demand by providing their zip code, thus giving the operator more awareness of the amount of preserved energy, as well as the areas where the flexibility supply is coming from.

**EasyBat**[9] is a Belgian initiative, headed by a grid operator and battery waste company, built on the EW-DOS technology which focuses on the entire battery lifecycle. All the actors involved in the process, from OEMs to installers can certificate every relevant asset transaction by means of a shared, decentralized ecosystem where IoT and blockchain technologies meet to provide reliable answers to issues related to the battery waste process.

## 5.2 Smart Aggregator DApps

In the evolving electricity market also a large number of smaller actors will contribute to the generation of green energy by using their small-sized plants.

[8] `https://flexalert.org/`
[9] `https://www.energyweb.org/easybat/`

However, such power plants are subject to well-known production fluctuations due to weather conditions and other factors, such as malfunctioning, or the need to consume part of the produced energy inside the plant. Hence, a single producer cannot give guarantees for the actual amount of energy that will flow out of the plant and to the distribution power grid. The fluctuations of energy production levels have to be properly handled and compensated upon entering the distribution network to make the energy production rate more reliable.

Balancing the power fluctuations introduced by so many DERs is one of the reasons why the introduction of an ***aggregator***, which assumes the role of an operational and administrative interface among grid operators (TSOs and DSOs) and the prosumers in the renewables scenario, is highly advisable [105].

An aggregator can perform various tasks, as described in the following.

- *DER administration*: negotiation and contractualization of "prosuming" conditions with private energy sources, dealing with flexibility baselines of produced and/or consumed energy, as well as economic management of payments and remuneration.

- *Energy profiling*: diversification of energy sources by nature, produced power, time availability and power flexibility.

- *Measure collection*: secured storage of power production and consumption measures coming from the controlled DERs' smart meters.

- *Real-time monitoring*: continuous monitoring of the controlled energy sources' performances in order to promptly react to any unexpected changes in the contractualized power provision.

- *Grid flexibility*: implementation of commands to timely adjust the amount of the "prosumed" energy that should enter the distribution network at a given time, thus producing an active contribution to balancing operations of energy flows required by national TSOs and/or local DSOs.

– *Prosumer marketplace*: provision of a shared digital environment to allow prosumers to place energy demands and offers and aggregators to match them up.

However, since the aggregator is not a central authority trusted by all, we need to have an immutable and counterfeit-resistant record of data that characterizes the implementation of the tasks listed above. For instance, for the *Measure collection* task to be effective and reliable, each small energy producer needs a guarantee that the provided energy, as recorded by smart meters, will be paid without risk of dispute. As in many other areas, the blockchain technology, given its ability to store data and ensure that they are immutable, is a proper foundation to provide the required chain of trust to a a distributed community, embracing a large amount of actors, that does not rely on a central authority [64, 65, 66]. Through the adoption of the blockchain, the authenticity of data stored on-chain and off-chain can be easily checked by the involved parties.

In this emerging application context, which is related to introduction of ***aggregators in the prosumers scenario***, the envisaged decentralized application pursues the provision of a distributed solution that caters for some of the above-mentioned tasks.

The release of the final solution has passed through a couple of preliminary experiments in which discrete features have been implemented in separate decentralized applications, designed for:

– the creation of a flexible demand/supply marketplace of DERs, orchestrated by the aggregators, able to match producers' offers with consumers' requests;

– the recording of counterfeit-resistant data, from accredited aggregators, which characterize energy produced and consumed by prosumers, in such a way that balancing of monetary accounts can be transparently achieved.

In a later work, the study has been redesigned to allow the *aggregator* to perform more complex and articulated energy management tasks in the

power fluctuation arena, according to its primary role. The realization of these tasks are enriched with decentralized automated features that are peculiar to the adoption of a blockchain platform specifically designed for the management of DERs, such as EW-DOS. The details of this study are discussed in Subsection 5.2.3.

The remainder of the section is as follows. Subsection 5.2.1 gives an overview of the DApp envisaged to deliver a web-based platform for the prosumers marketplace orchestrated by the aggregator. Subsection 5.2.2 describes the DApp initially designed to handle a secure and trustworthy mechanism in which the aggregator collects the actual production/consumption data from prosumers to credit/debit them accordingly. Subsection 5.2.3 provides a thorough description of the DApp that delivers a comprehensive solution to the power fluctuation of green energy sources in a smart grid.

### 5.2.1 DER marketplace DApp

The first experimentation carried out on top of the EW-DOS utilities has been about building a set of functionalities that enable prosumers to create energy demands and offers and aggregators to match them up.

#### 5.2.1.1 Application description

The developed DApp demo exhibits a front-end that drives the operations to create a distributed energy marketplace, which, in turn, is a customized replication of the *Flex Alert* platform included in the Energy Web's open-source technology stack (see Subsection 5.1.2). It includes interfaces for the three key roles identified in the system:

- the *producer* interface, i.e., the owner of many DERs, who produces and sells energy (Owner tab);
- the *consumer* interface, i.e., the consumer interested in purchasing the energy they need (Buyer tab);

– the *aggregator* interface, where multiple available supplies can be selected to cumulatively match a given price-compatible demand (Aggregator tab).

The procedure to create a match between producers' supply offers and consumers' demands consists of the following steps:

i. a *selling* user posts one or more selling offers, which specify the available volume of energy and the required price for the DERs it owns; the offer can be modified over time or withdrawn, if not yet accepted;

ii. the *buying* user posts one or more requests specifying the amount of energy (kW) needed and the maximum price (ct/KWh) it is willing to pay; such requests can be modified or withdrawn later on;

iii. the *aggregator* collects and matches offers and demands to select those that form a valid combination, which is locked by registering it as a proposed deal (from this moment on, supply and demand can no longer be changed until the proposed deal is either accepted or rejected);

iv. once accepted, the combination takes effect and a closed deal is permanently registered in the blockchain ledger; otherwise, at any time, both the buyer and the seller can remove the pending combination from the marketplace.

#### 5.2.1.2 Demand/offer matching

Figure 5.2 shows a screenshot of the DApp web interface for the *owner* role.

In the shown example, a producer has logged in and claimed ownership of two digital assets, corresponding to two different DERs that it owns and manages. By means of this interface, the producer issues an energy selling offer for each of its DERs, separately specifying the quantity of available energy and the minimum accepted price per energy unit. The DApp also includes an interface to access many components of the Energy Web platform architecture, that is, the *Energy Web Naming Service*, *Digital ID* and the

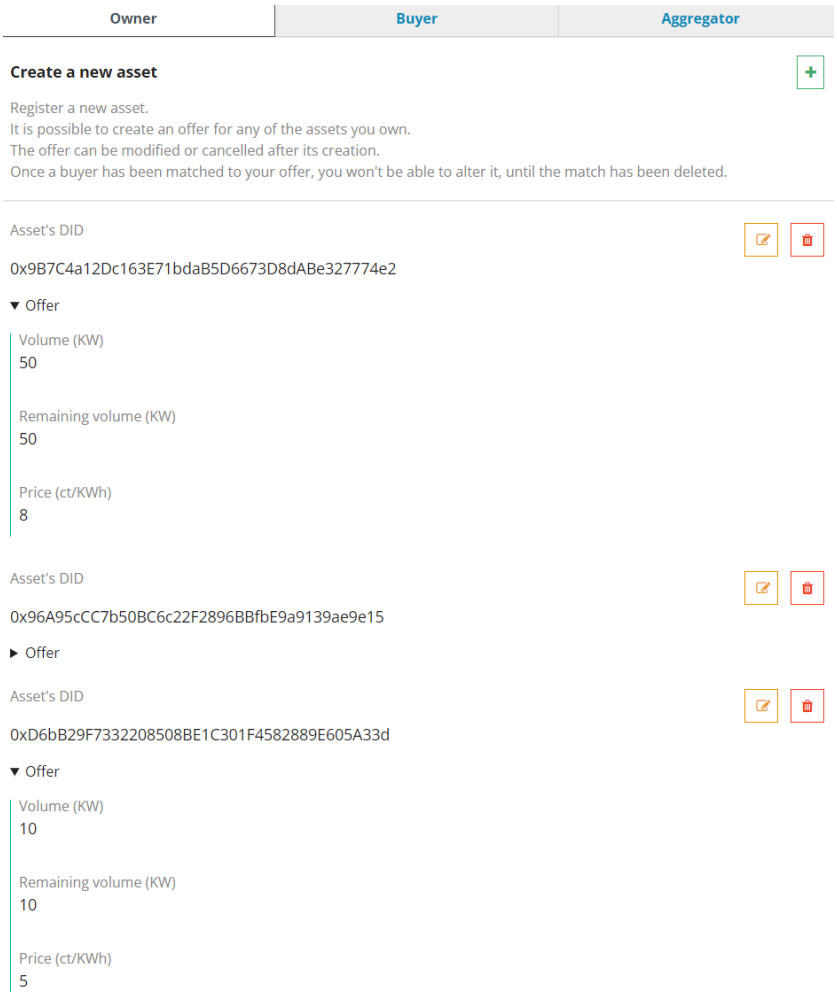


Figure 5.2: Producer interface tab of the implemented demo DApp.

*Identity and Access Management* libraries, made publicly available by the EW-DOS framework.

Figure 5.3 shows a screenshot of the aggregator interface, listing a set of energy offers and energy demands, available to be manually selected to create a proposed match, as described in Subsection 5.2.1.1.

The DApp implementing the marketplace employs two smart contracts deployed on the Volta testnet at the following addresses:

– Identity Manager smart contract:
(0x84d0c7284A869213CB047595d34d6044d9a7E14A),

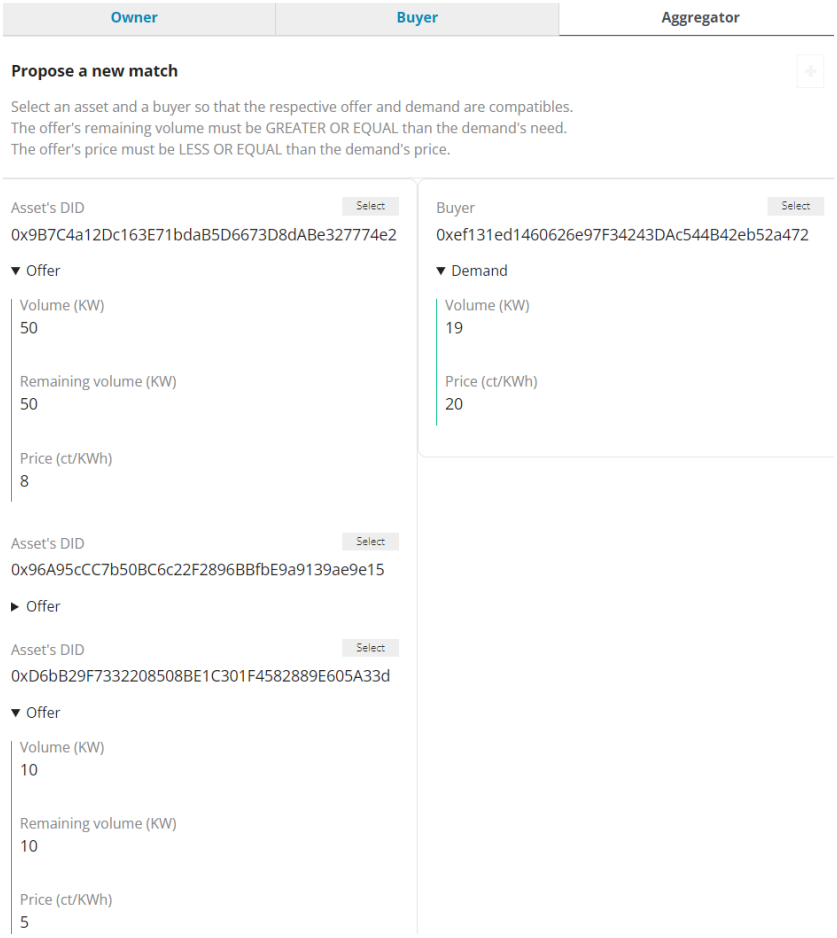


Figure 5.3: In the aggregator tab, offerings and demands can be selected and combined in order to create a new deal proposal.

– Marketplace smart contract:
(0x37dfeF9b9c56A81927Dfa73994E2fb23c3dd4b37)

Code and documentation of the complete DApp are publicly available in the GitHub repository at `https://github.com/TendTo/EW-showcase`.

### 5.2.2 Smart metering DApp

This subsection describes the DApp originally developed to help the aggregator handle payments and charges following the collection of real data coming from the managed DERs.

The scenario sketched in Figure 5.4 represents a smart energy distribution system, in which there are two distinguishable flows: on one hand, the flow of energy that is handled by a power distribution grid; on the other hand, the flow of data that is handled by the ubiquitous internet.

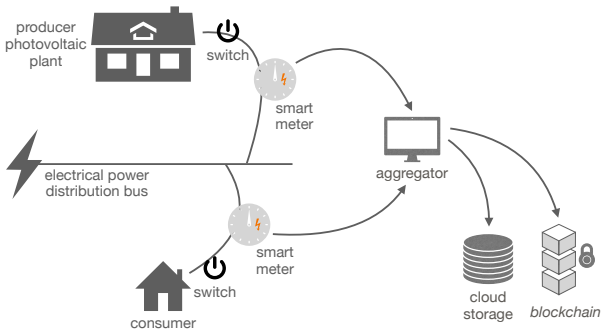


Figure 5.4: Main components of the proposed "smart" system.

As for the energy flow, each producer/consumer is attached to an electrical power distribution bus. To control the flow of electricity, switches are to be remotely configured, i.e., each producer controls a switch to determine whether electricity can flow out of its power plant, and each consumer controls the switch to let the electricity flow in. Other switches are spread over the power distribution grid to isolate portions or avoid energy fluctuations.

The decentralized applications presented in this chapter do not deal directly with the energy flow, since the interactions between the solution components only concern the data network portion of the smart grid. The way the outcomes of the logic developed in the applications are reflected into the command chain of the energy switches is beyond the scope of this work.

#### 5.2.2.1 Application overview

In the proposed systems several agents interact with various roles: an arbitrary number of *DERs* under the responsibility of one of several *prosumers*, which in turn refer to an *aggregator*. From this point of view, DERs can be identified as rather simple *IoT* devices connected to the global network and using the *http* protocol to provide a continuous stream of metering data.

Any DER is equipped with a smart meter that measures both the outgoing flow of electrical power leaving the plant towards the power distribution bus and the incoming flow of the electrical power from the distributor. Such a smart meter is connected to the Internet to periodically send the measures taken on the power line (see Figure 5.4). Each measure is labelled with the digital identifier (DID) of the smart meter, hence it is possible to distinguish the various originating prosumers.

Figure 5.5 shows the components of the proposed system architecture and their respective dependencies.

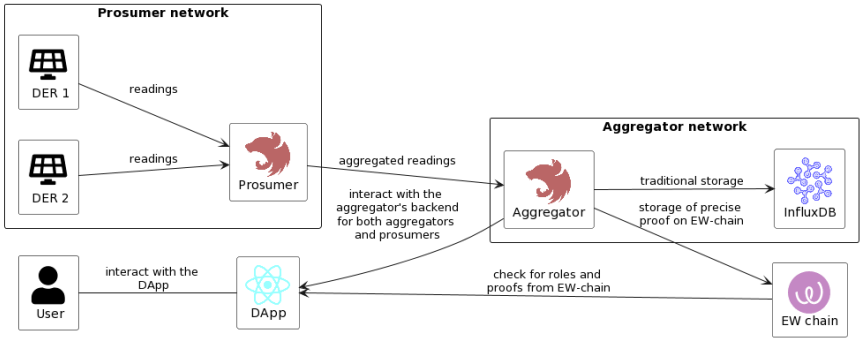


Figure 5.5: Components of the decentralized solution.

In the devised solution, the aggregator handles the incoming data (typically every minute) and performs authentication checks before transferring them to cloud storage at a selected time intervals (e.g., every hour) for temporary or permanent recordkeeping, according to needs. Finally it stores specific hashed information to the blockchain, needed to ensure provenance and integrity of the received data, by invoking a specific function of an ad-hoc smart contract.

#### 5.2.2.2 Data collection and storage workflow

The off-chain/on-chain storage scheme provides that data characterising production and consumption be recorded in a cloud storage, while their signatures (or hashes) are to be written in the blockchain storage, according to the following dataflow, which is also depicted in Figure 5.6.

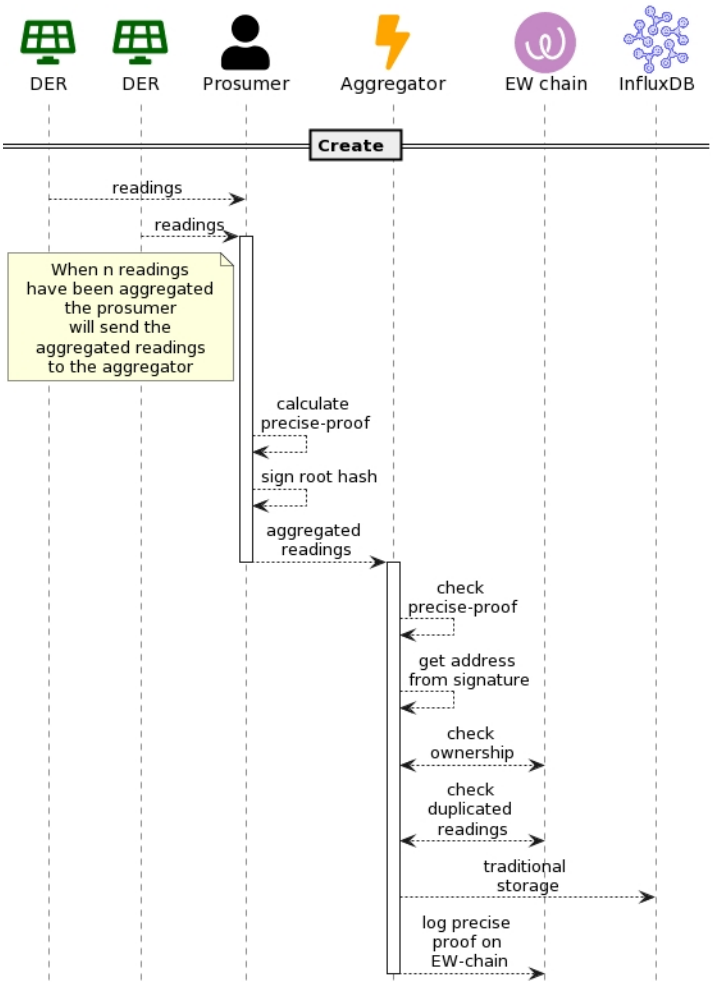


Figure 5.6: Dataflow to create and store list of readings.

For the generation of the list of meter readings, each *prosumer* is responsible for a certain number of actions:

i. the creation of the *precise proof*, which is the root hash of the Merkle trie created on the smart meter readings over an established period of time (e.g., every minute);

ii. the computation of the *digital signature* on the precise-proof with its private key: this provides the aggregator with a guaranteed origin and at the same time protects the precise-proof, and its entire underlying payload, from malicious or accidental alterations.

iii. the packing of the *message payload* containing a list of all the readings from the DERs and the calculated value of the precise-proof; a list of *salt values* (used both in the creation of the trie and the digital signature applied to the precise-proof) and some convenient info, such as the first and last reading *timestamps*.

With a precise-proof schema based on a Merkle trie, if a proof is required that a specific reading has been really carried out and stored, only the single contested data will be needed in clear, together with a bunch of hashes required to correctly rebuild the trie. This significantly reduces the amount of potentially sensitive data to be shared. Incidentally, the data reduction of the on-chain portion prevents from incurring unmanageable expenses in the execution of the smart contract that performswrite operations to the blockchain storage [78, 99], as thoroughly discussed in the Contact Tracing solution.

Once the data have been received, the *aggregator* performs the following flow of actions on the data storage side:

i. collects the data it receives from the authorized *prosumer* and verifies that they are valid; this is done by first verifying the digital signature of the precise-proof;

ii. reconstructs the Merkle trie starting from the readings and the salts, so that it is verified that the precise-proof corresponds;

iii. checks if the sending *prosumer* is authorized to participate in this application, and that it is also the owner of all the *DERs*, associated to the incoming data, by means of specific *EW-DOS* utilities, such as the ready-to-use IAM functionalities;

iv. saves the information in a *database* capable of storing time series of data (e.g. InfluxDB [10]);

v. invokes the `store()` method of the deployed smart contract purposely designed and deployed on the blockchain, whose aim is to issue a new

[10] `https://www.influxdata.com/`

transaction, which, in turn, stores, in the blockchain permanent logs, the DID of the *aggregator* with the precise-proof associated with its last validated group of reads (see Listing 5.1)[11].

Once the log is stored into the blockchain, i.e., the entire writing flow has been successful completed, it is guaranteed that the processed data will remain permanently recorded, undisputed and available.

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.4;
contract ReadingsNotary{
    /** A new metered reading has been emitted */
    event NewMeterReading (address indexed operator, bytes indexed proof);
    /** Store a new reading @param _proof Merkle root of the
     * tree of merkle proofs of the aggregated readings */
    function store(bytes calldata _proof) external{
            emit NewMeterReading (msg.sender, _proof);
    }
}
```

Listing 5.1: Smart contract `ReadingsNotary`.

At a later stage, in order to check that the measured data stored have been neither corrupted nor tampered with, the *prosumer* follows the steps shown in Figure 5.7:

i. meets a cryptographic challenge launched by the *aggregator*;

ii. gathers a set of data by querying the cloud database to extract the data pertaining to its own associated DIDs for a certain time slot;

iii. accesses the blockchain logs to download the precise-proof related to the list of readings;

iv. computes the precise-proof on the data and matches against the hash retrieved from the blockchain to ensure that data have not been tampered with.

The *prosumer* is enabled to access the stored data though an API REST interface provided by the *aggregator*. In fact, a *prosumer* has access to data

[11]The contract is running on Volta at `https://volta-explorer.energyweb.org/address/0xe574fDD8C3148f2E883612A9C6CDA7b-9C12d1566/transactions`.

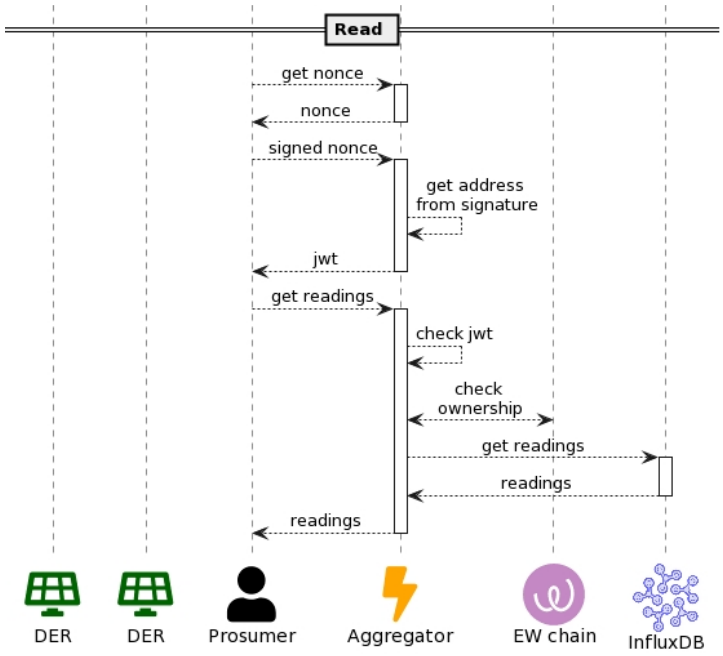


Figure 5.7: Dataflow to retrieve a list of readings for checking.

relating only to the *DID* it owns, while the *aggregator* can access all data stored in the database.

Code and documentation of the complete DApp are publicly available in the GitHub repository at `https://github.com/TendTo/EW-DER-API`.

### 5.2.3 Grid flexibility DApp

The envisaged decentralized application, presented in this subsection, provides a blockchain-based energy management strategy that enables the *aggregator* to handle several energy sources as a single virtual entity in order to counteract fluctuations of the power grid.

In this regard, the project has been designed in order to include support for complex energy management strategies, such as:

i. negotiation and writing of adjustable contracts with energy sources diversified by power and flexibility profiles;

ii. handling of payments and penalties according to the fulfilment of flexibility agreements;

iii. real-time monitoring (and characterization) of the contractualized resources;

iv. implementation of commands for the controlled DERs to modulate their energy production.

By acting on prosumer flows within the limits of contractualized variations, the aggregator is able to respond to a change request, received from the transport operator to cope with a temporary fluctuation of load in the national network, by offering the resulting aggregate power from flexible resources. Therefore, the aggregator assumes the role of a virtual power plant (VPP) with the function of a balancing service provider. The complete scenario envisages the presence of various aggregators located throughout the territory managed by the transport operator.

To implement the blockchain paradigm, the prosumers' DERs need to perform an additional task in the devised system, that is the capacity of joining the blockchain peer-to-peer network to enforce its related mechanisms, from the block creation process to the execution of deployed smart contracts implementing the system business logic. The blockchain is used not only as a ledger to seal the energy provision agreements and payments among the parties but also as an *asynchronous communication infrastructure* to receive balancing commands from the grid flexibility operator. The complexity of the required equipment performing such tasks are tailored both to the DERs' size and the blockchain-based system's overall performance, as discussed in more detail in Subsection 5.2.3.5.

#### DER administration

In the smart grid flexibility scenario, a propaedeutic management task, carried out by the aggregator, is the implementation of smart contracts for the automatic updating of the working parameters of the controlled prosumers, within negotiated boundaries. This task is particularly useful in case of unpredictable events, not only due to external conditions (e.g. unexpected climatic changes heavily impacting the expected energy production) but also

to random internal issues of the prosumer plant (e.g., temporary unavailability for maintenance or faults, higher internal requirements of the produced power, etc.).

In real terms, these renegotiable parameters consist of:

- *baseline*, that is the maximum reference power, if consumed, or the minimum reference power, if produced;
- *flexibility percentage*, that is the fraction of the baseline the prosumer is willing to modulate;
- *time base*, that is the set of available time intervals to provide flexibility.

In this administrative scenario, the use of the blockchain can bring a few benefits:

i. reduction of the technical complexity of the aggregator's centralized back-end, since the operations for the contractual renegotiation can be decentralized as smart contracts in the blockchain. This approach has undoubtedly a positive impact on the scalability issues of the overall system, since the number of managed energy resources can scale up in the decentralized environment without a significant increase in the availability requirements of the aggregator's back-end;

ii. transparency of actual renegotiation conditions for the prosumer, which does not have to trust either the fairness of the central operator or the aggregator's ability to detect errors or abuses in the applied contractual clauses;

iii. internal rewarding and accounting mechanisms by means of the virtual blockchain currency, which makes it a particularly interesting technological solution to support immediate and guaranteed payments of debit and credit fees associated with the actions performed by DERs, without margins of dispute.

The extra technological costs charged to the prosumers, which have to be equipped with adequate hardware and software to act as validators in the

blockchain ecosystem, thus contributing to its overall solidity, could be compensated by protocol rewards, directly paid through the blockchain utility token as billing discounts. Alternatively, the IoT devices that cannot or do not want to participate as validator nodes, must be equipped with minimal hardware and software to interface with an external blockchain access node through an HTTP or Web Socket channel, without affecting the interaction semantic with the aggregator.

### Real-time monitoring

Continuous monitoring from the aggregator of the actual energy production/consumption data provided by the DERs is an ancillary task for the proposed solution, which consists of an uninterrupted and frequent flow of tiny payloads per transmitted DER packet, made up of its global identification parameter (e.g., a DID) and the instantaneous power value measured by the associated smart IoT device. Such live data are particularly useful to determine the actual power supply of each DER, bearing in mind the difficulty that arises in the case of inconstant or unpredictable energy sources, whose contributions cannot be analytically predetermined but only estimated in the contractualization phase. Prosumers can then take advantage of the chance to renegotiate the terms of power and flexibility of their contracts with the aggregator, according to changing climatic conditions and specific internal circumstances.

For this task, the centralization of these data flows from DERs to the aggregator is the only sensible choice to be implemented (although quite demanding on the hardware scalability requirements), since significant drawbacks emerge from a deep analysis of the alternative approach, based on the collection of real-time data through blockchain-based mechanisms. Even in an ideal situation, in which each DER implements a blockchain peer client (node) to increase the scalability factor for such a consisting number of transactions per time unit, there is no guarantee of ensuring a correct temporal sequence to the transaction writings in the blockchain logs, due to the inner features of the decentralized protocol for validating transactions. Therefore,

since sorting of real-time data from independent sources is essential in a such a task, their centralized collection is not only a mere technical choice, but also the most consistent one to implement.

In the chosen centralized approach, the aggregator may process the real-time data from each DER in the following ways:

- stores all received data in an off-chain system database, optimized for storing time series, for a time interval useful to take immediate or near-future decisions on modulation interventions based on the profiled energy sources;
- compresses them as lossless data in blocks that are better suited to permanent archiving, if deemed necessary for future disputes.

The potential use of blockchain storage of hashed data to cope with future disputes has been dealt with in the Smart Metering DApp (see Subsection 5.2.2). However, these functionalities may be redundant, since the blockchain already logs significant system operation events that concern the interactions among DERs and aggregator, which are agreed-upon, transparent and no longer disputable by design, as better explained in the next subsection. Therefore, there is no reason to keep the collected data if not provisionally until all transactions, settling a potential dispute, are confirmed by the blockchain protocol.

### Balancing actions

A fundamental task assigned to the aggregator relates to supporting the transport and distribution operators in balancing actions of the overall grid load, in response to unforeseen unbalancing events or planned changes. These occurrences force the aggregator to react to a load change requirement by disseminating flexibility request commands to the associated DERs, in order to obtain the aggregate load change corresponding to the balancing needs.

The requests from the operator (e.g., a TSO) to set the flexibility parameters (e.g., extent of the power modulation, date and time of the planned intervention) must be conveyed through a communication interface between

a TSO agent and the aggregator, which is beyond the scope of the present study. Conversely, the technical arrangements supporting the implementation of the flexibility commands, from the aggregator to its affiliated DERs, is a matter of investigation, in that blockchain-based mechanisms are compared with traditional client-server ones.

In general terms, the modulation commands can be diversified, for each DER, according not only to their contractualized flexibility profiles, but also to their dynamic "prosuming" conditions detected by real-time monitoring activities from the aggregator. In order to achieve optimal results in the implementation of the load change request, the aggregator needs to associate further parameters to each DER's digital identity, such as the reliability of the source in previous actions, and the real availability of modulation power of the source with respect to the time interval of the flexibility request, which can be constantly renegotiated and estimated on the basis of the DER profiling carried out over time. Based on this information, the aggregator should process an intervention profile on how to spread a portion of the aggregate requested variation on each DER. If implemented via a centralized approach, these requirements can be quite resource demanding.

Conversely, the implementation of a grid balancing mechanism relying on the blockchain makes the involvement of DERs more scalable and reliable, and it ensures that working and flexibility conditions, aggregator requests and DER actions cannot be disputed or counterfeited. The use of the blockchain decentralizes the response logic towards the blockchain-enabled DERs through an automated and transparent policy, immutably written in specialized smart contracts, thus unburdening the aggregator from heavily demanding computations at each load change request coming from the grid operator. In this case, the aggregator has only the task to invoke these smart contracts with a cycle of transactions, which contain the same information: the entity and the time window of the aggregate intervention requested. This policy is based on the assumption that the modulation action is always executed by the involved DERs proportionally to their respective contractualized baselines, with the only possibility to select in/out the managed DERs case by case. The selection is based on the compatibility of DERs' time availabil-

ity and power flexibility profiles, either contractualized or estimated, with duration and extent of aggregate modulations required by central operators.

An alternative approach provides that the aggregator can have a greater control on implemented commands by means of pre-emptive algorithms, which can sort all DERs by reliability and amplitude of the maximum possible contribution, thus reaching the requested aggregate contribution by leaving out more volatile or simply smaller partners, according to a "staking" policy. On the other hand, relying on fewer large subjects may amplify the effects of unexpected defection events and compromise the correct execution of the aggregate order. Therefore, a broader distribution of modulation commands towards small entities can represent an equally valid alternative strategy, provided that the involved DERs are included in a mechanism that prevents fraudulent or malicious behaviours.

Notice that catching the responses of the DERs to the aggregator commands is an integral part of the real-time monitoring of DER activities, which determines compliance with conditions in force at the time of issued commands for billing, payments and profiling.

#### 5.2.3.1 DApp overview

To experimentally evaluate the feasibility and scalability limits of the described solution, a prototype has been created on which experiments have been carried out to measure and analyze the blockchain subsystem performance under different stress conditions, so as to draw appropriate conclusions on the adequacy of blockchain technology to support the application scenario envisaged and described in this solution. The prototype consists of a real blockchain and of a simulator that models other system devices (i.e. DERs) in such a way to realistically mimic their functionalities.

The application has been created using the Electron framework[12], thus making it possible to create a multi-OS application.

The structure of the project has two distinct parts:

i. a web-based front-end for the configuration of system parameters (e.g.,

[12] `https://www.electronjs.org/`

number and type of involved DERs, blockchain endpoints, time information), and the simulated interaction with system components (e.g., aggregator and DERs);

ii. a back-end which is a set of TypeScript classes and smart contracts created to interact with the blockchain.

A large portion of the front-end page is reserved to the simulation dashboard, with graphs and results of current and past tests. Figure 5.8 shows a screenshot of the page during the system initialization phase.

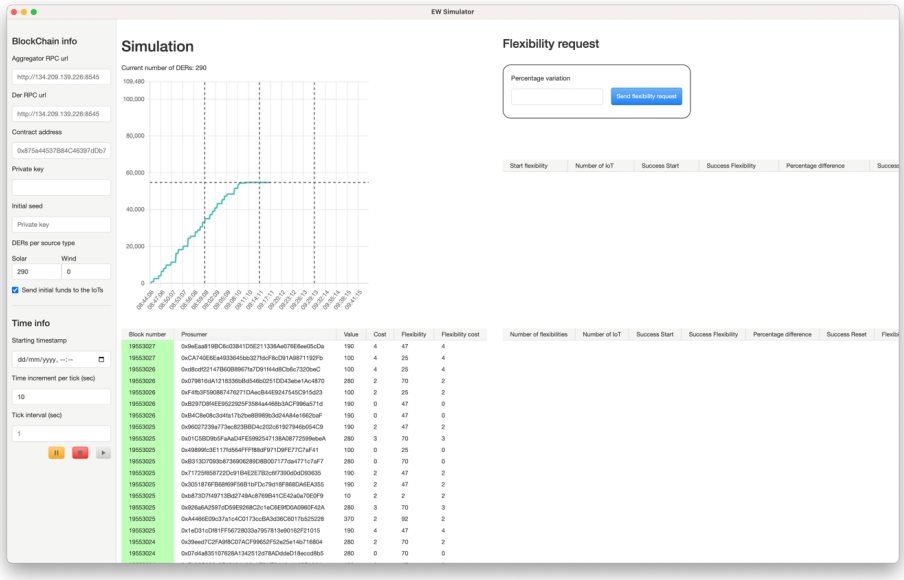


Figure 5.8: Front-end page of the DApp simulator in the set-up phase.

### DER simulation

Two types of renewable energy sources are modeled, i.e., solar and wind.

For the former, an oscillating data model with a double periodicity function is used, which represents the daily solar cycle and the seasonal variation of inclination of the sun. It is also taken into account that these systems are equipped with accumulators so that the daily fluctuations can turn out to be as flattened as possible, to comply with the 5% tolerance in the contractualized reference baseline. The wind source is instead characterized by a

rather impulsive and unpredictable trend with lower stabilizing effects from the accumulators. These random situations of "off-grid" and lower production are also present in the model of solar energy production, but in such a case due to adverse weather conditions or unpredictable failure situations. In both cases, the simulator models these random events with a probability distribution of Poisson whose characteristic parameter is adjustable in the user interface prior to simulation sessions. However, in the reported test sessions, the occurrence probability of these adverse events has been set to zero, in order not to interfere with the evaluation of the uncompleted response percentage due to performance reasons in the blockchain, which is the real focus of the study.

DERs are implemented in the simulator as software components that interact with the aggregator exclusively through the blockchain. Each DER is in fact identified by a dedicated account in the Energy Web Chain, thus interacting with it through the EW-DOS APIs. It also interacts with the simulator interface through RPCs over the HTTP protocol. At the same time, a corresponding module, named "IoT", is instantiated for each DER, which generates real-time data (see Subsection 5.2.3 and sends them to the aggregator via the internet, but outside the blockchain communication channel, for real-time monitoring of the source production. These IoT modules generate data with an algorithm corresponding to one of the two types of sources described above, and with a reading frequency which can be adjusted in a parametric way from the simulator interface. DERs are also shaped in such a way that the flexibility they offer may also be negative, referring to household consumer DERs. In general, the contractual flexibility of the prosumer is modelled with the following parameters: percentage change (+/-) of the baseline, maximum duration supported, time of implementation of the variations.

DERs are instantiated on the Energy Web Chain in the simulation set-up phase, by generating their accounts through a pseudo-random algorithm of the "Hierarchical Key Generation" type, which is based on a seed and a deterministic mechanism for pseudo-casual generation, thus allowing the simulator to reuse, with the same seed, always the same accounts of previous

simulations, in this way providing historical transaction logs to the same DERs' accounts for simulation comparisons.

### Aggregator

The modelled aggregator component represents the administration reference point for prosumers' DERs; the aggregator's tasks consist of:

- off-chain activities to monitor the readings from the associated DERs' IoT devices in real-time;
- on-chain activities to immutably and transparently record administrative and operational events (see Subsection 5.2.3).

The readings from the IoT devices allows the aggregator to have a real-time overview of the power grid's state under its control, for example making supply forecasts based on the source used for energy production. IoT readings can be collected and stored in an off-chain database for a limited, configurable time frame in case of possible disputes on the actual readings prior to payments. When a modulation command has been completed according to the conditions written in DER smart contracts, the agreed transfers of funds are then recorded on the blockchain logs, which are cumulatively used to store the followings events:

- *contract registration* of a new DER;
- *temporary modification* of the contractual parameters of a DER;
- *modulation request* of the supplied power from the aggregator to the DERs
- *accounting* of the contribution of individual DERs to the modification request.

The accounting of credits and debits to prosumers is managed automatically, transparently and securely by the smart contract that has predefined features to assign ERC20-like tokens to prosumers according to the parameters negotiated in the stipulated contracts. These tokens can then be redeemed

for appropriate compensations. The mechanism of calculation of token remuneration is set within the smart contract that regulates the functioning of the system, which is defined, owned and deployed by the aggregator (see Subsection 5.2.3.2). To operate on the blockchain, all DERs must have a minimum token credit to pay the (negligible) operational cost of executing transactions. For this, a special transaction is launched in the simulation set-up phase, which distributes a basic quantity of funds to DERs' accounts. The said procedure is obviously not at all necessary in real systems.

#### 5.2.3.2 Blockchain implementation

As mentioned before, the Energy Web Decentralized Operating System (EWDOS) has been selected as the experimental blockchain testbed for the scalability investigations carried out in the study. In the deployed setting, the aggregator is equipped with a Energy Web Chain light node (EWC client), implemented on the public code of the Nethermind client[13], through which it broadcasts transactions to prosumers in order to implement aggregate modulation actions. The aggregator's EWC client can also act as a Web3 Provider.

However, a second EWC client is introduced into the system in order to act as the common access point for DERs' IoT devices. This choice is a lightweight solution in place of installing an EWC client in each DER, which would require around 200GB space and demanding computational resources (that is, a dedicated platform for each DER). However, this solution is perfectly suited for the simulated DERs in the demo environment, which can thus remotely access the EWC through the dedicated client.

The introduction of a separate node to provide Web3 access to DERs prevent the transaction process from bypassing the EWC network. In fact, a single node, both for the aggregator and the DERs, would represent a direct channel for their communication, in which transactions issued by a part would be written in a data block that could be read by the other part without waiting a realistic propagation delay in the EWC. Such a circumstance would have interfered with a consistent and reliable assessment of the response times

[13] `https://nethermind.io/`

and the scalability boundaries of the implemented prototype.

### Smart contract design

The aggregator node, in its set-up phase, takes care of deploying the smart contract, named `AggegatorContract()`, on the EWC network, through which it regulates the interaction with DERs. The aggregator is, in fact, the account's owner of this smart contract, which exhibits the following functionalities:

i. registration and updating of the contractual conditions of each DER;

ii. transmission of modulation commands to DERs;

iii. accounting of the implemented modulations on each DER's account.

The contractual agreements take into account the energy source type, the amount and price of the produced energy, and the flexibility that the prosumer is able to provide. A modulation request from the aggregator is characterized by the following values:

- *flexibility:* modulation that the network must produce with respect to the baseline;
- *start:* deadline by which all DERs must reach the required flexibility value;
- *end:* indication of the moment from which the DERs have fifteen minutes to return at the baseline.

As for on-chain data, the smart contract uses both its associated storage, for storing the ephemeral variables representing the contract's state, and the transaction logs, for a persistent and much cheaper recordkeeping of public information related to the occurrence of triggered events.

The smart contract storage, which is only used for the state variables it needs to carry out the function executions, contains the following data structures:

```
address public immutable aggregator; //aggregator's account
int256 public energyBalance; //instant aggregate power
mapping(address=>Prosumer) public prosumers; //prosumers' states
mapping(address=>Agreement) public agreements; //prosumers' contracts
address[] public prosumerList; //prosumers' accounts
FlexibilityRequest public flexibilityRequest; //requested aggregate
    modulation
mapping(address=>int256) public flexibilityResults; //prosumers' modulation
    outcomes
```

The events emittable by the smart contract's functions, whose content is stored in the transaction logs, are below listed:

```
event RegisterAgreement(...);
event ReviseAgreement(...);
event CancelAgreement(...);
event RequestFlexibility(...);
event EndRequestFlexibility(...);
event FlexibilityProvisioningSuccess(...);
event FlexibilityProvisioningError(...);
event RewardProduction(...);
```

The event mechanism is used by the aggregator to notify listening prosumers about flexibility requests.

The way the smart contract's functions interact with state variables in the contract administration operations is concisely described in the following. Three different functions, `registerAgreement()`, `reviseAgreement()`, and `cancelAgreement()` are invoked to create an agreement, to later alter its terms or to delete it. Specifically, after checking that the invoked operation is legal, the data structure `agreements`, containing the registered contracts, is updated. Following the execution of `registerAgreement()` and `cancelAgreement()` a prosumer is added or removed from the `prosumerList` and `prosumers` data structures.

The actions performed by `AggegatorContract()`, in response to the invocation of its functions concerning the modulation request to DERs, can be summarized as follows:

i. upon receipt of an aggregate modulation request, characterized by the flexibility amount and two timestamps, corresponding to the beginning and end of its duration, the aggregator first saves it in the `flexibilityRequestdata` data structure and then transfers such a request to the selected DERs by invoking the `requestFlexibility()` function;

ii. upon response to the request, the aggregator emits the corresponding `RequestFlexibility()` event through a dedicated transaction, which causes its publication on the EWC logs for the DERs' IoT devices to intercept it through a specific event watcher[14] implemented in the smart contract;

iii. upon event notification, each DER calls the appropriate smart contract's `ProvideFlexibility()` function, through which they register their modulation activities, which, potentially, could turn out to be either compliant or non-compliant with the contractual terms;

iv. the aggregator, by monitoring the real-time data from the IoTs, understands whether the action of each DER is successful or not and, fifteen minutes after the end of the flexibility request, it invokes the `endFlexibilityRequest()` function, with the list of registered results;

v. after validating the flexibility event, the smart contract saves the received results in the `flexibilityResults` data structure and emits the `EndFlexibilityRequest()` event to notify the recorded results to each DER;

vi. on event notification, the DERs invoke the `provideFlexibilityFair()` function to claim their reward in a non-repudiable way, if they agree with the aggregator on the registered outcomes of the modulation. This reward is added to the `prosumers` data structure, which also contains their financial statements, as long as the claimed value is consistent with the calculation of the corresponding compensation from the aggregator, which, in fact, emits either the `FlexibilityProvisioningSuccess()` event upon successful check, or the `FlexibilityProvisioningError()` event otherwise.

[14] An event watcher is a specific Web3 functionality that allows a requester to be notified when a filtered event occurs in a blockchain's smart contract.

### Smart contract deployment

The development of the smart contract has been carried out using the Hardhat framework[15], a development environment for smart contracts that enables operations such as debugging, compilation, and testing. Furthermore, Hardhat includes the functionalities to deploy the smart contract on EWC through a simple script, by providing a private key associated with an account with sufficient credit and an RPC-API node to connect to.

```
import { ethers } from "hardhat";
async function main() {
  const contractFactory = await ethers.getContractFactory("
      AggregatorContract");
  const contract = await contractFactory.deploy({ maxPriorityFeePerGas: 7 })
      ;
  await contract.deployed();
}
main().catch((error) => {
  console.error(error);
  process.exitCode = 1;
});
```

Listing 5.2: Script to deploy the aggregator's smart contract on EWC.

### Smart contract testing

The Hardhat framework also provides a series of libraries for quick and easy testing of smart contracts internally to a simulated blockchain environment, so as to validate the expected behaviour of the contract's functionalities. Tests include checking the return value of the invoked function, verifying the correct alteration of the status of the contract, ensuring that a due exception is produced in case of invalid inputs. Putting all these tests together, considerable coverage of the smart code was achieved[16].

The following listing provides an example of a test snippet simulating the invocation of the smart contract's `requestFlexibility()` function:

```
describe("requestFlexibility", function () {
  it("create a new flexibility request", async function () {
    await contract.requestFlexibility(start, end, gridFlexibility);
```

[15] `https://hardhat.org/`

[16] Test coverage results can be found at `https://app.codecov.io/gh/TendTo/EW-DER-Simulator`

```
    const request = await contract.flexibilityRequest();
    expect(request.start.toNumber()).to.equal(start);
    expect(request.end.toNumber()).to.equal(end);
    expect(request.gridFlexibility.toNumber()).to.equal(gridFlexibility);
  });
  it("revert on unauthorized use with 'UnauthorizedAggregatorError(msg.
      sender)'", async function () {
    await expect(iot1Contract.requestFlexibility(start, end, gridFlexibility
        ))
    .to.be.revertedWithCustomError(contract, ContractError.
        UnauthorizedAggregatorError)
    .withArgs(iot1Addr);
  });
});
```

Listing 5.3: Test snippet to simulate a smart contract's function call.

#### 5.2.3.3 Back-end architecture

The decentralized application's back-end is responsible for simulating the DERs' behaviour, performing the aggregator's actions and managing the interactions among the parties (see the diagram of the classes in Figure 5.9). As previously stated, some interactions go through the blockchain, according to an asynchronous paradigm that consists of function calls and corresponding event emissions.

The simulated time passing is beaten by the `Clock` class, which is tick-configurable. On simulation start, the aggregator and the Clock are initialized with user-specified parameters. The aggregator initializes the correct number of IoTs of each type and provides them with their own blockchain identities, characterized by an account with their own (*private-key, public-key*) pairs. Each IoT will be assigned the same identity at each simulation run for comparison purposes.

The simulation set-up phase also involves interacting with the blockchain. After a few preliminary actions, the aggregator resets the smart contract and sends the funds to each created DER. Upon successful set-up completion, the graph starts showing the aggregate data coming from the DERs (see Figure 5.8), which, at the same time, begin to register their agreements with the smart contract. When the aggregator receives the registration event, the corresponding IoT production is added to the aggregate one. The DER recording period has a variable delay proportional to the number of DERs

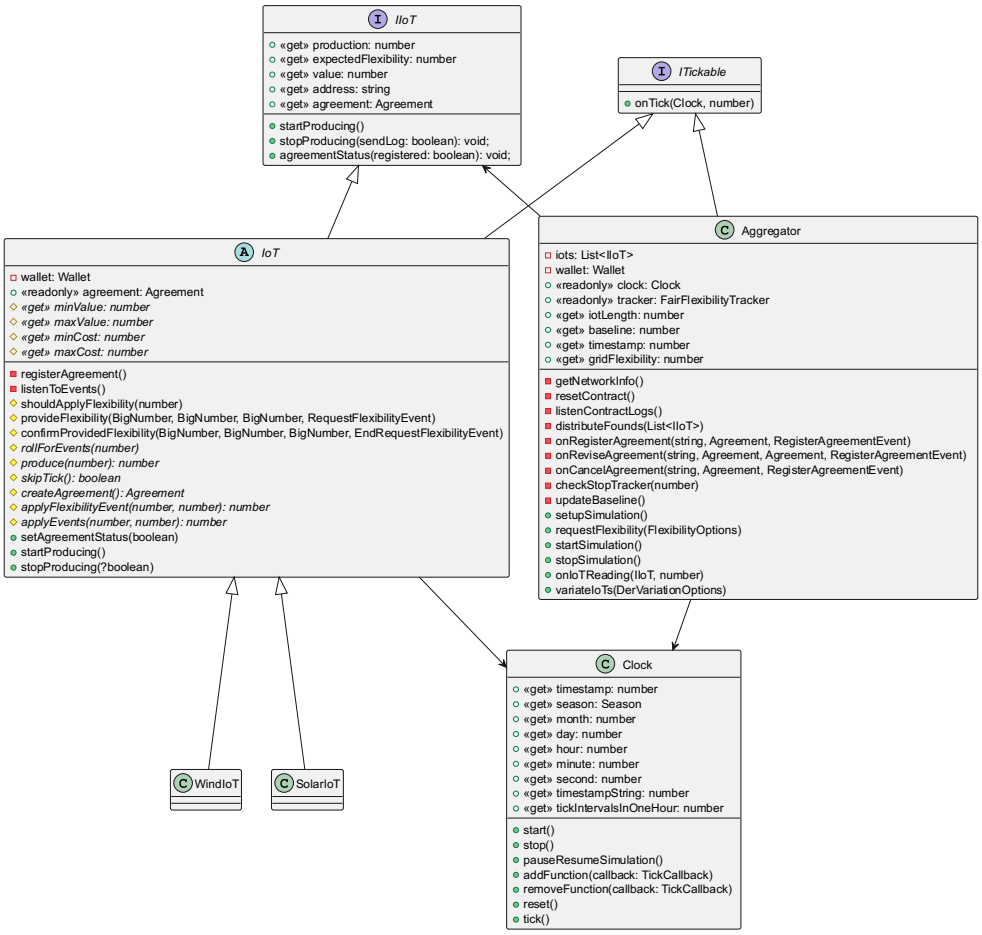


Figure 5.9: Back-end's class diagram.

used in the simulation.

The simulation allows the user (e.g., a TSO operator) to send a flexibility request command specifying the baseline percentage to be altered. The system reaction to such a command has been thoroughly explained in Subsection 5.2.3.2 and it is shown in the sequence diagram sketched in Figure 5.10, as far as the flexibility request to the IoT devices is concerned.

#### 5.2.3.4 Experimental assumptions

In a plausible future scenario, the aggregator will stipulate contracts with prosumers of renewables sources of variable size, from large industrial sites pro-

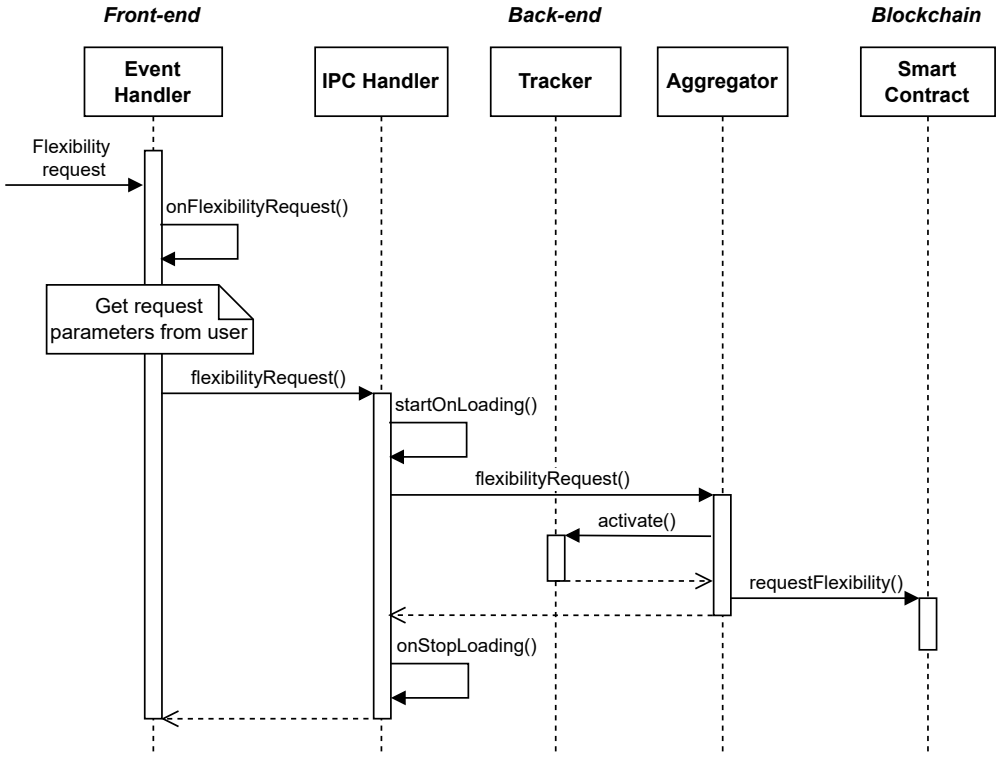


Figure 5.10: Sequence diagram upon flexibility request command.

ducing power for the grid exceeding the MW threshold, to private medium-sized producers, able to provide power in the range of tenths of KW. Furthermore, a comprehensive case study needs to include a large number of domestic prosumers equipped with either small solar systems, wind turbines or commercial batteries capable of delivering electric power in the range of a few KW. The presence of flexible passive users, with a chance to temporarily reduce the maximum consumed power, should also be taken into account in the overall scenario.

The purpose of the tests carried out in the simulation is to determine a few experimental outcomes, namely:

i. the number of DERs per aggregator that is possible to manage without introducing unacceptable delays in the system;

ii. technological constraints of the implementation blockchain that can affect the scalability of the entire system;

iii. response times of the DERs to modulation change commands.

In order to carry out verifiable and comparable tests, some assumptions on the modelled system have to be made beforehand, as explained in the following subsections.

### DER modelling

In order to come up with a cost-effective strategy, the aggregator needs to classify the managed industrial or residential sources in terms of a few parameters such as: (i) the produced power size; (2) the type of source (e.g., wind, solar, batteries, etc.), which can affect its overall reliability and availability; (3) the level of flexibility offered as a percentage of the contractual base power; (4) the reference time profile, that is, the daily time slot(s) in which the power variation can occur.

In the investigated case study, we envisage an evolving scenario where the number of managed DERs can grow from a few units to hundreds, for a total aggregate power of hundreds of megawatts, statistically distributed by contractualized energy baseline according to a Poisson curve, for which we have few large-sized DERs and many decreasing small-sized DERs to make the flow of monitored data as realistic as possible. It is also assumed the presence of DERs of different types based on the energy source type. Presently, only solar and wind sources are modelled in the case study, as the complexity increase determined by the modelling of other source types would not bring any specific contribution to the experiment, which is instead centred on the effectiveness of the blockchain adoption in the management system.

The total number of managed DERs, optionally differentiated according to their nature, will be the main configurable parameter for the evaluation of the prototyped solution's scalability. For instance, two reference scenarios could be configured as follows:

i. *basic scenario,* in which the aggregator manages a total of 690 KW, divided into three solar industrial plants of 30 KW and ten of 10 KW, plus one hundred prosumers of 5 KW;

ii. *evolved scenario,* in which the aggregator manages a total of 1.75 MW, divided into five 30 KW solar industrial plants and ten of 10 KW, one hundred prosumers with solar systems of 5 KW, one hundred prosumers with 3 KW batteries and two hundred passive users with a 3 KW contract.

In real terms, the simulating scenario follows the incremental management of ten DERs at a time, with an upper boundary determined by the detection of technical limits in the prototype implementing the case study, with particular regard to blockchain scalability performances.

### DER readings

As for real-time aspects, the reading flows frequency from DERs must be properly tuned to provide significant evidences of the DERs' instant state and, on the other side, to allow the aggregator to react promptly to any unexpected change. A specific reference range is between one reading per second and one reading every sixty seconds, which represents a sensible choice to balance performance and accuracy. The frequency parameter must be also combined with the varying number of DERs to achieve acceptable scalability responses from the experimentation.

### Balancing scenario

In the prototyped DApp, a simplified version of the decentralized approach, described in Subsection 5.2.3, has been specified for the execution of the devised feasibility and performance tests. This working hypothesis assumes that the modulation action be spread in a fair manner on each DER, that is with the same proportionality with respect to the aggregate request, possibly selecting the envolved DERs on the basis of matching flexibility criteria of the pursued action.

In a blockchain-based implementation, where the transactions cannot be run in parallel with the same node, the commands can be conveniently executed in a sequential order, so that the negative outcome of a transaction can be immediately managed by redefining the amount of modulation on the

remaining DERs, or by cycling it on the previously involved DERs a second time around. In more general terms, it is assumed that the aggregate variation command is applied to all DERs in the same time window , so no new similar commands can be received in time windows that even just overlap.

The time window consists of {`start:datetime`} of the requested modulation and {`end:datetime`} for completion. Acceptable intervals can range from a minimum value of fifteen minutes up to a few hours, with steps of fifteen minutes. Note that the intervention can be immediate or future-oriented (e.g., within the next twelve or twenty-four hours).

In the feasibility tests described in the next section, a few thresholds have been set to define in which conditions the aggregate modulation can be considered satisfactory fulfilled. In particular, this conditions are detected when the aggregator verifies that 95% of the DERs hit the following targets:

- within the initial transient the DER baseline is modified with a maximum error of 5%;
- within the final transient the DER baseline is restored with a maximum error of 5%;
- between the two instants above, the average value measured for the DER corresponds to that modulated with a maximum error of 5%.

#### 5.2.3.5 Experimental results

As repeatedly said, the prototyped system has been specifically developed to experimentally evaluate the advantages and drawbacks in the use of the blockchain in such a specific industrial case, namely an energy balancing operator. In this respect, numerous operating tests have been performed to profile the behavior of the blockchain network and to identify the *scalability limits* of the current implementation and the strategies to overcome them. The number of DERs involved in the performed functional tests ranges from ten to three hundred, with varying results depending on the prototype version. Beyond the upper limit, the network performance are unsatisfactorily and/or unreliable, for the reasons examined throughout this subsection.

The experiments deal with the two following scenarios: (i) DER administration, and (ii) grid balancing, which are served by the same smart contract by design. Hence, the upper scalability limit is a figure determined by the worst detected performance in both scenarios[17].

### Transaction confirmation time

For the understanding of some experimental outcomes, it is useful to highlight some inner blockchain mechanisms that affect them. First, any performing blockchain should guarantee, under conditions of "healthy state" of the nodes[18], a response time to *transaction validation* not higher than five or six seconds. The transaction validation is performed locally at the blockchain's entry node and it must be clearly distinguished from *transaction confirmation*, which indicates the inclusion of a validated transaction in a newly created block.

The confirmation time is variable over time, since it relies on the gas fee mechanism, inherent in the transaction confirmation process of Ethereum-like platforms [16]. Each transaction can in fact freely offer to pay an established gas price to obtain confirmation, thus incentivizing the mining nodes to process the transactions as quickly as possible. High offered gas prices (e.g., 30 GWei in the Volta testnet[19]) can therefore make the network become very selective with transactions willing to pay a fairer and lower price. Furthermore, this mechanism has also to combine with the momentary load of the blockchain network, since more congested networks require higher gas fee to complete the transaction confirmation and vice versa. As a result of these two factors, it is generally possible that a transaction is confirmed after a few seconds, a few minutes or a few months.

In our prototype, we then set the gas price to be always competitive, even in congestion situations, thus making it unlikely that validated trans-

[17]Note that the solutions gradually identified during the study have been correspondingly integrated into the prototype up to the final version, in which it works correctly in all its phases and interacts reliably with the testnet blockchain (i.e., Volta).

[18]Healthy state is a condition in which each node is in sync with the general state of the blockchain, having adequate computing and network resources.

[19]Wei is a measure of the utility currency.

actions can excessively be long pending before confirmation (in the order of minutes in worst cases). In its current version, the blockchain integrated in the prototype responds to all modulation commands well within the limits of fifteen minutes, which has been set as the maximum accepted transient of the transition intervals between state changes (e.g., network set-up, modulation requests).

In particular, in the largest possible scenario, i.e. with 290 DERs (see Subsection 5.2.3.5 for the determination of this limit), the initial activation of all IoTs in the set-up phase typically takes about three to five minutes to complete (see Figure 5.8 in Subsection 5.2.3.1). The time-span of such activity is configured at the application level to avoid the generation of too an intense burst of transactions (see Subsection 5.2.3.5). On the other hand, the reaction times to the modulation commands exhibited by the DERs range from a few seconds to about a minute or two for each DER; these times obviously vary in proportion to the managed DERs.

Figure 5.11 shows the system behavior at maximum DER load in the execution of a modulation command, exhibited at the final stage of a testing period of about two months, during which several experimental configurations have been tuned up[20]. Modulations are carried out following an approach that requires an equal contribution from all contracted DERs.

In the following, we briefly summarize the main issues related to the use of the blockchain that have been tackled.

### Gas limit

Some transactions may occasionally fail by exhibiting a "gas limit exceeded" error code and then rejected, if their processing cost exceeds a threshold set, i.e., the gas limit (8M gas units for Volta), which is a prescriptive feature of Ethereum-like blockchains, introduced in the decentralized protocol for security reasons [17].

The transaction cost, in terms of gas units, is not always predictable, but it can only be estimated because it is connected not only to the complexity

[20]Tests can be performed by downloading the DApp software available at `https://github.com/TendTo/EW-DER-Simulator`.

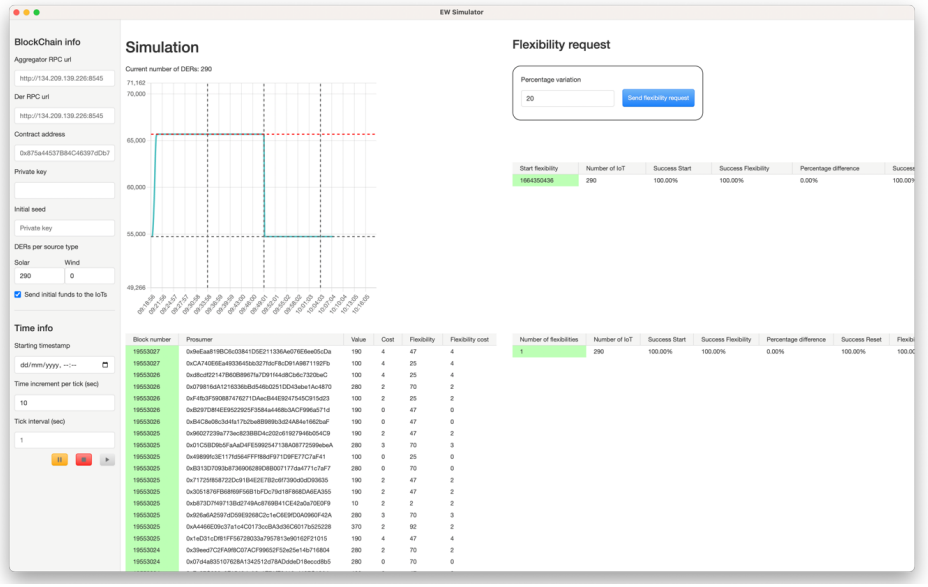


Figure 5.11: DER responses to a modulation request command.

of the transaction itself, but also to the actual number and size of the input parameters, which, in turn, can heavily affect the storage operations. In our case study, for instance, the number of DERs concurrently managed in a single transaction can cause this faulty situation.

The gas limit exception represents a sturdy obstacle to overcome when the number of manageable DERs scales beyond an upper threshold, which is, on the other hand, difficult to determine in all working conditions of the newtork. This limit is also directly attributable to the current implementation of the smart contract, where there are some transactions that suffer from this problem. For instance, if the transaction that simulates the uploading of funds of IoTs' accounts is launched for a number of devices higher than 290, it experiences an execution fail and consequently the system cannot activate the IOTs. However, this will not be a problem at all in a field implementation where IOTs are real, and where owning prosumers will be in charge of refunding their accounts, thus making the transaction unnecessary and the problem immaterial.

In broad terms, the gas limit exception requires a fine design of the smart contract patterns, which favors as few state variables as possible in the smart

contract's account storage, since the transactional cost to create, update and delete them are far more expensive than using the transaction logs. Furthermore, no smart contract should have functions that handle a large numbers of data structures of any kind at a time. Since the functions and the data structures of the DER administration scenario can be ideally kept separate from the ones of the balancing scenario, a split of the aggregator's smart contract in at least two pieces would also contribute to a more refined and cost-effective design of the software project.

Alternatively, one of the most interesting utilities of the EW-DOS stack, namely the "Decentralized System Bus"[21], designed to provide a messaging system to make a more functional and reliable use of transactions among the actors of the system, could be experimented to unburden the communication paradigm between the aggregator and the prosumers.

Pending an updated design of the smart contract pattern, the scalability limit of 290 DERs has been currently kept as the test boundary not only for further functionalities of the DER administration scenario, but also for the grid balancing trials, as already seen in Subsection 5.2.3.5.

#### Network congestion

Congestion conditions in the blockchain network have proven to change rapidly and unpredictably, thus affecting the performance of our application in a significant way. As previously mentioned, the *transaction confirmation* times are highly susceptible to network overloads not only for obvious delays in the network components, but also for a demand/supply mechanism that makes the gas price rise, thus causing the overall gas cost of transactions to follow suit. This circumstance has two major consequences: on one hand, the transactions emitted with a lower gas price than the current average one, determined by the congestion conditions, risk not to be confirmed by the block-creation protocol within acceptable time frames; on the other hand, a higher transaction cost can undermine the balance of blockchain accounts or even prevent them from transacting, in case of insufficient funds.

[21] `https://www.energyweb.org/tech/`

A massive reduction in gas price volatility and transaction costs may be introduced by permissioned blockchain technologies, where the network access can be regulated by selected nodes, so as to prevent intolerable congestion situations. Better scalability and security performances in permissioned blockchains are obviously obtained at the detriment of the decentralization dimension, as stated by the "blockchain trilemma", a circumstance that, in enterprise environments, is not regarded as critical as in public blockchains.

Up to now the network congestion issues have been treated as if the blockchain network were a real, meshed peer-to-peer network. Unfortunately, this is mostly untrue, since it relies on the internet structure which has a more hierarchical infrastructure architecture. For instance, in public networks, the inevitable crossing of a certain number of routers causes packet scrambling, which, in turn, increases the likelihood that the transactions will then be delivered in a very different order from the sending sequence. In spite of the application logic expecting a rapid and sequential transaction processing, the system may experience a disordered and discontinuous response, based on congestion conditions of the validator nodes and in particular, in our prototype, of the RPC interface node. In worst cases, crossing public networks may determine the occurrence of situations in which some transitions, already in the transaction pool for confirmation in a definitive block, may randomly remain "pending" for an indefinitely long time without any feedback to the sender. To limit the above problem, it can be convenient to connect the aggregator, which generates transaction bursts by design, to the blockchain network with more reliable mechanisms, such as an internal interprocess communication (IPC) channel, a private connection or an IP tunnel (i.e., a VPN). This relieves the problem, as the packets can be received by the blockchain network sequentially.

### Overloaded RPC node

When the computing resources of the RPC node are overloaded by the incoming call flow, this starts dropping the excess calls, thus discarding the packets containing the transactions. A too intense burst of packets can cause a very

significant loss of part of them, thus compromising the functioning of the system. The problem is technically a consequence of the lack of a synchronization mechanism in the flow of calls between the source (application) and destination (RPC node), which is a single logical sequence at the application level, but it is managed by the network as a sequence of independent TCP connections. Although the RPC node can be configured to increase the limit of the maximum number of concurrent TCP connections, there is an insurmountable constraint given by the actual number of independent CPU and, consequently, of effectively parallel threads, that the node is able to run simultaneously to open and process data on these connections. This situation is reported to the application as a request timeout or network error.

A plausible solution could be the artificial reduction of the intensity of transaction bursts, for example by introducing micro-delays in the application logic at safety levels with respect to the processing resources of the RPC node, thus bringing the system back to sustainable operational conditions.

A further issue with a congested node is the inability to keep up with the continuous updating of the status of the blockchain, thus losing synchronism and not being able to respond correctly to remote calls. When the issue occurs, it is detected as an anomalous error, which indicates that a received transaction request has been marked with the same identification number as a previous one. Having jammed the synchronization mechanism with the global state of the blockchain, the node stops processing blocks and no longer has a consistent internal state. A consequence of this situation is that the ordinal count of the transactions also hangs, so that the same value of "nonce" is used for all the transactions, which, by definition, should instead be a unique guaranteed sequential identifier[22].

Unfortunately, if a node goes out-of-sync, the only practical chance to recover from this adverse situation is to reinstall it in a better performing hardware platform.

[22] As known, the "nonce" is the order of acceptance of the transactions originating from the same account.

## 5.3 Solution discussion

The solutions developed in this chapter have examined the feasibility and the performance of a blockchain-based management system in a smart grid scenario, where a virtual operator, the *aggregator*, administers an amount of distributed energy resources on behalf of transmission or distribution operators. The implemented DApps have been tested against increasingly complex management scenarios, from a simple marketplace of energy demands and offers to a sophisticated platform at the disposal of a virtual power plant, capable of dynamically adapting to ever changing energy needs.

The capabilities provided by smart contracts allow the automated handling, in a trustworthy and efficient way, of the whole innovative process, including power generation and distribution, contract handling, instant billing, rewarding and payments.

The investigation has demonstrated the feasibility of the devised services built on top of a blockchain and pointed out its technological benefits in support of the aggregator's despite some experienced issues, which, overall, do not undermine the accrued competitive edge with respect a traditional client-server architecture, based on application servers and internal databases. The use of the blockchain has in fact introduced significant simplifications in the back-end system requirements, since the decentralized infrastructure can now take over the management of the contractual and historical data of the DERs and the automated implementation of the business logic in the form of specialized smart contracts.

In addition to a reduction of operating costs in the examined cases, the beneficial innovation also relies on the property that the blockchain, by definition, ensures about the total correctness and transparency of the interactions between the involved parties, which do not have to trust a centralized service provider. Finally, the massive test production has assured that the scalability levels and the response times of the services, as provided in the different solutions, are adequate to the application needs within practical working conditions.

In some operations the blockchain network has also been used as an asyn-

chronous communication infrastructure to a command/response mechanism among the parties to seal the energy provision agreements and payments. Furthermore, the native token mechanism allows to resort to an automated implementation of the remuneration process of the prosumers based on smart contracts, thus providing, in such a delicate aspect, more transparent assurances on the operations performed by the aggregator. The complexity of the required equipment to perform such tasks are tailored both to the DERs' size and the blockchain-based system's overall performance.

Finally, scalability and performance issues exhibited by the decentralized applications in certain overload conditions, along with high transaction costs, may suggest to release the service on a permissioned blockchain, with the implementation costs shared among aggregator and prosumers. The latter can compensate these extra costs for securing the blockchain by earning tokens associated to the block validation process. At the same time, in a further development of the project, the above issues can be efficiently tackled with the design of more cost-effective patterns for the developed smart contracts.

Notice that the blockchain technology is subject to a fast and continuous evolution, so that it would be unwise to assume that the investigation carried out in the various experiments has managed to make an exhaustive list of all the benefits and issues associated with its use. It is also expected a step forward in the underlying communication protocols that can provide the logically meshed peer-to-peer model, on which the blockchain is based, with a more reliable mapping of the related communication mechanisms in the public network.

# Chapter 6

# Conclusions

Beyond the cryptocurrency universe, the decentralized blockchain paradigm aims at establishing the backbone for a new type of public internet, in which every participant can transact directly with any other participant without the intermediation of a third-party to validate and secure transactions. In fact, since the advent of Ethereum, many blockchains have allowed for the deployment of "smart contracts", which are programmed to automatically run when specified conditions are met, without the intervention of a central application server and/or database.

The blockchain technology has been originally released to serve a typical permissionless network, meaning that the governance is totally decentralized, since anyone can participate in the process of block verification and creation, typically through a highly energy demanding mechanism. It offers great transparency and decentralization, but unfortunately faces scalability and computational cost challenges, which can undermine its large adoption for enterprise applications besides the monetary/financial ones. However, regardless of their open technological issues, blockchains, due to their decentralized consensus mechanisms and cryptographic secured nature, offer unmissable features to a new audience of stakeholders that are looking for new means of securing networks and increasing transparency in distributed applications, such as checking the authenticity of on-chain data in a totally disintermediated fashion.

The primary objective of this thesis is to demonstrate that the Ethereum protocol can be flexibly and adaptively used to build enterprise decentralized applications in fundamental sectors, such as healthcare and green energy, since it still represents the most widespread, robust and interoperable blockchain technology in the smart platform scenario, ranging from totally public and permissionless implementations (i.e., the native PoW-based Ethereum) to more enterprise-focused ones with permissioned characteristics (e.g., Energy Web Chain).

The presented solutions have been devised by introducing appropriate system designs that balance, case by base, the decentralization aspects of the blockchain technology with overall system performance and scalability. The design choices, supported by numerous experimental results, have primarily taken into account not to undermine the effectiveness and efficiency of the solutions in the selected applications fields, as compared to traditional centralized approaches, thus preventing an uncritical utilization of the blockchain technology. Actually, in many situations the adoption of an extensive decentralized paradigm, inclusive of off-chain decentralized storage systems, have proven to generate more sustainable distributed systems.

Furthermore, many peripheral actors can be enabled to contribute to the implementation of the decentralized infrastructure, being rewarded via mechanisms that are intrinsic to the blockchain protocol. For instance, in the energy market sector, a new value chain can be unlocked in which also household prosumers may run their own nodes to secure the peer-to-peer infrastructure, while earning rewards for their effort.

In each of the presented solutions, specific technological aspects have been looked after to fulfil the enterprise application requirements, so as to provide extra features of security, performance, cost and decentralization. For instance, the content tracing solution has managed to enhance its scalability level by enriching the decentralized app-to-app contact matching mechanism with customized back-end and blockchain design choices, such as finely tuned smart contract patterns, essential on-chain storage, decentralized off-chain file systems (i.e. IPFS), thus making the adoption of the novel decentralized application both secure and performing.

On the other hand, the energy management solution has deeply investigated and experimentally tested some blockchain specific performance issues under heavy transaction loads, showing reassuring outcomes even in presence of a non-optimal design of the developed smart contract, which can be appropriately improved in a further refinement of the solution, as already done for the contact tracing one.

Moreover, an extensive use of the decentralized identity management of digital assets has been experimented in the energy management solution, by directly or indirectly interfacing the developed DApps with the dedicated EWC's smart contract, as far as the enrollment, updating and revocation of digital identities and verifiable credentials are concerned.

Overall, the applied research carried out during the doctorate placement has successfully shown that the Ethereum technology can represent a satisfactory platform to fulfil typical enterprise requirements, especially if an appropriate combinations of tailored features are adopted, case by case, to ensure balanced achievements in security, scalability and decentralization.

The investigation carried out meets the global vision of the Ethereum organization, which aims at making its platform the most pervasive blockchain technology, even for enterprise applications, by enhancing the scalability and security levels, yet not undermining the degree of decentralization, which remains, in the blockchain context, the custodian of *"neutrality, censorship resistance, openness, data ownership and near-unbreakable security"*[1].

From its launch in 2015, the Ethereum community has grown high expectations on some upgrades that would unlock Ethereum's full potential, by mitigating the constraints of the *blockchain trilemma* and some inherent drawbacks, such as the high transaction fees, the demanding computation and storing requirements to run an Ethereum node, the predictable network congestion, and the big environmental impact of the PoW consensus algorithm.

With the *Paris* hard fork (aka *The Merge*) deployed at block 15537393 on 15th September 2022, the transition from PoW to PoS consensus layer (aka 'Eth2') has been completed. According to the Ethereum foundation,

[1] `https://ethereum.org/en/enterprise/`

PoS introduces a *"more secure, more decentralized, less energy consuming consensus mechanism (-99.95% as compared to PoW), which is also more suitable for implementing new scalability solutions"*. Scalability-wise, another ongoing development, the so-called Ethereum's Layer 2[2], is addressing specific challenges that have driven enterprise developers to choose private chains in the past. This comprehensive terminology includes all the off-chain solutions designed to help scale the developed applications, such as, for instance, the off-chain storage mechanism implemented via IPFS in Nausica@DApp, yet leveraging the robust decentralized and secure model of the mainnet (Layer 1). The Layer 2 initiative, in the long run, should promote the adoption of Ethereum as the reference platform for enterprise applications, a vision that our experimental outcomes encouragingly support.

[2] `https://ethereum.org/en/layer-2/`

# Bibliography


[1] S. Haber and W. S. Stornetta, "How to time-stamp a digital document," in *Conference on the Theory and Application of Cryptography*. Springer, 1990, pp. 437–455.

[2] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," *Decentralized Business Review*, p. 21260, 2008.

[3] E. A. Brewer, "Towards robust distributed systems," in *PODC*, vol. 7, no. 10.1145. Portland, OR, 2000, pp. 343 477–343 502.

[4] S. Gilbert and N. Lynch, "Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services," *Acm Sigact News*, vol. 33, no. 2, pp. 51–59, 2002.

[5] I. Bashir, *Mastering Blockchain: Distributed ledger technology, decentralization, and smart contracts explained*. Packt Publishing Ltd, 2018.

[6] P. Baran, "On distributed communications networks," *IEEE transactions on Communications Systems*, vol. 12, no. 1, pp. 1–9, 1964.

[7] L. Lamport, "The weak byzantine generals problem," *Journal of the ACM (JACM)*, vol. 30, no. 3, pp. 668–676, 1983.

[8] M. Castro, B. Liskov *et al.*, "Practical byzantine fault tolerance," in *OsDI*, vol. 99, no. 1999, 1999, pp. 173–186.

[9] P. Akunne, "A guide to blockchain consensus protocols," 2021. [Online]. Available: https://blog.logrocket.com/guide-blockchain-consensus-protocols/

[10] R. Awati, "Consensus Algorithm," 2022. [Online]. Available: https://www.techtarget.com/whatis/definition/consensus-algorithm

[11] D. Chaum, "Blind signatures for untraceable payments," in *Advances in cryptology*. Springer, 1983, pp. 199–203.

[12] A. Back *et al.*, "Hashcash-a denial of service counter-measure," 2002.

[13] W. Dai, "b-money, 1998," 1998. [Online]. Available: http://www.weidai.com/bmoney.txt

[14] N. Szabo, "The idea of smart contracts," *Nick Szabo's papers and concise tutorials*, vol. 6, no. 1, p. 199, 1997.

[15] H. Finney, "Reusable proofs of work (rpow)," 2004. [Online]. Available: https://nakamotoinstitute.org/finney/rpow/index.html

[16] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum project yellow paper*, vol. 151, no. 2014, pp. 1–32, 2014.

[17] A. M. Antonopoulos and G. Wood, *Mastering Ethereum: building smart contracts and dapps*. O'Reilly Media, 2018.

[18] J. Woods and J. Iyengar, *Enterprise Blockchain Has Arrived: Real Deployments. Real Value.* Jorden Woods & Radhika Iyengar, 2019.

[19] Coinhouse, "What is Proof of Authority?" [Online]. Available: https://www.coinhouse.com/what-is-proof-of-authority/

[20] S. Falkon, "The story of the DAO-its history and consequences," *Medium*, 2017. [Online]. Available: https://medium.com/swlh/the-story-of-the-dao-its-historyand-consequences-71e6a8a551ee

[21] S. Wang, W. Ding, J. Li, Y. Yuan, L. Ouyang, and F.-Y. Wang, "Decentralized autonomous organizations: Concept, model, and applications," *IEEE Transactions on Computational Social Systems*, vol. 6, no. 5, pp. 870–878, 2019.

[22] "KSI Blockchain - e-Estonia." [Online]. Available: https://e-estonia.com/solutions/cyber-security/ksi-blockchain/

[23] N. Mamo, G. M. Martin, M. Desira, B. Ellul, and J.-P. Ebejer, "Dwarna: a blockchain solution for dynamic consent in biobanking," *European Journal of Human Genetics*, vol. 28, no. 5, pp. 609–626, 2020.

[24] F. Yiannas, "A new era of food transparency powered by blockchain," *Innovations: Technology, Governance, Globalization*, vol. 12, no. 1-2, pp. 46–56, 2018.

[25] J.-H. Tseng, Y.-C. Liao, B. Chong, and S.-w. Liao, "Governance on the drug supply chain via gcoin blockchain," *International journal of environmental research and public health*, vol. 15, no. 6, p. 1055, 2018.

[26] E. Tijan, S. Aksentijević, K. Ivanić, and M. Jardas, "Blockchain technology implementation in logistics," *Sustainability*, vol. 11, no. 4, p. 1185, 2019.

[27] M. Das, X. Tao, Y. Liu, and J. C. Cheng, "A blockchain-based integrated document management framework for construction applications," *Automation in Construction*, vol. 133, p. 104001, 2022.

[28] A. M. Antonopoulos, *Mastering Bitcoin: unlocking digital cryptocurrencies*. O'Reilly Media, 2014.

[29] D. Mingxiao, M. Xiaofeng, Z. Zhe, W. Xiangwei, and C. Qijun, "A review on consensus algorithm of blockchain," in *IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, 2017, pp. 2567–2572.

[30] S. M. S. Saad and R. Z. R. M. Radzi, "Comparative review of the blockchain consensus algorithm between proof of stake (pos) and delegated proof of stake (dpos)," *International Journal of Innovative Computing*, vol. 10, no. 2, 2020.

[31] I. Barinov, V. Baranov, and P. Khahulin, "POA network white paper," 2018. [Online]. Available: https://github.com/poanetwork/wiki/wiki/POA-Network-Whitepaper

[32] Q. Zhou, H. Huang, Z. Zheng, and J. Bian, "Solutions to scalability of blockchain: A survey," *IEEE Access*, vol. 8, pp. 16 440–16 455, 2020.

[33] Coinhouse, "The blockchain trilemma: Fast, secure, and scalable networks." [Online]. Available: https://www.gemini.com/cryptopedia/blockchain-trilemma-decentralization-scalability-definition

[34] J. Xie, F. R. Yu, T. Huang, R. Xie, J. Liu, and Y. Liu, "A survey on the scalability of blockchain systems," *IEEE Network*, vol. 33, no. 5, pp. 166–173, 2019.

[35] M. Bowman, D. Das, A. Mandal, and H. Montgomery, "On elapsed time consensus protocols," in *Progress in Cryptology – INDOCRYPT 2021*, A. Adhikari, R. Küsters, and B. Preneel, Eds. Cham: Springer International Publishing, 2021, pp. 559–583.

[36] L. Ricci, D. D. F. Maesa, A. Favenza, and E. Ferro, "Blockchains for Covid-19 contact tracing and vaccine support: A systematic review," *IEEE Access*, vol. 9, pp. 37 936–37 950, 2021.

[37] Q. Tang, "Privacy-preserving contact tracing: current solutions and open questions," *arXiv preprint arXiv:2004.06818*, 2020.

[38] L. Luu, J. Teutsch, R. Kulkarni, and P. Saxena, "Demystifying incentives in the consensus computer," in *Proc. of ACM SIGSAC Conference on Computer and Communications Security*, 2015, pp. 706–719.

[39] Zuidhoorn, Maarten, "The magic of digital signatures on Ethereum," 2020. [Online]. Available: https://medium.com/mycrypto/the-magic-of-digital-signatures-on-ethereum-98fe184dc9c7

[40] McCallum, Timothy, "Diving into Ethereum's world state," 2018. [Online]. Available: https://medium.com/cybermiles/diving-into-ethereums-world-state-c893102030ed

[41] L. Luu, D.-H. Chu, H. Olickel, P. Saxena, and A. Hobor, "Making smart contracts smarter," in *Proc. of ACM SIGSAC conference on computer and communications security*, 2016, pp. 254–269.

[42] Y. Wang, A. Zhang, P. Zhang, and H. Wang, "Cloud-assisted ehr sharing with security and privacy preservation via consortium blockchain," *Ieee Access*, vol. 7, pp. 136 704–136 719, 2019.

[43] N. Prusty, *Building blockchain projects*. Packt Publishing Ltd, 2017.

[44] P. Windley, "How blockchain makes self-sovereign identities possible," *Computerworld*, vol. 10, 2018.

[45] A. Tobin and D. Reed, "The inevitable rise of self-sovereign identity," 2016. [Online]. Available: https://sovrin.org/

[46] M. S. Ferdous, F. Chowdhury, and M. O. Alassafi, "In search of self-sovereign identity leveraging blockchain technology," *IEEE Access*, vol. 7, pp. 103 059–103 079, 2019.

[47] M. Zastrow, "South Korea is reporting intimate details of COVID-19 cases: has it helped?" *Nature*, 2020.

[48] H. Cho, D. Ippolito, and Y. W. Yu, "Contact tracing mobile apps for COVID-19: Privacy considerations and related trade-offs," *arXiv preprint arXiv:2003.11511*, 2020.

[49] T. Altuwaiyan, M. Hadian, and X. Liang, "EPIC: efficient privacy-preserving contact tracing for infection detection," in *Proc. of IEEE International Conference on Communications (ICC)*, 2018, pp. 1–6.

[50] A. Trivedi, C. Zakaria, R. Balan, and P. Shenoy, "WiFiTrace: Network-based contact tracing for infectious diseases using passive WiFi sensing," *arXiv preprint arXiv:2005.12045*, 2020.

[51] G. Li, S. Hu, S. Zhong, W. L. Tsui, and S.-H. G. Chan, "vContact: Private WiFi-based IoT contact tracing with virus lifespan," *IEEE Internet of Things Journal*, vol. 9, no. 5, pp. 3465–3480, 2021.

[52] J. Bay, J. Kek, A. Tan, C. S. Hau, L. Yongquan, J. Tan, and T. A. Quy, "BlueTrace: A privacy-preserving protocol for community-driven contact tracing across borders," *Government Technology Agency-Singapore, Tech. Rep*, 2020.

[53] Yonap, "S. Korea to run system to better detect virus patients' routes," *The Korea Herald*, 2020. [Online]. Available: http://www.koreaherald.com/view.php?ud=20200311000132

[54] C. Troncoso, M. Payer *et al.*, "Decentralized privacy-preserving proximity tracing," *arXiv preprint arXiv:2005.12273*, 2020.

[55] S. Vaudenay, "Analysis of DP3T," *IACR Cryptol. ePrint Arch.*, vol. 2020, p. 399, 2020.

[56] H. Xu, L. Zhang, O. Onireti, Y. Fang, W. J. Buchanan, and M. A. Imran, "BeepTrace: blockchain-enabled privacy-preserving contact tracing for COVID-19 pandemic and beyond," *IEEE Internet of Things Journal*, 2020.

[57] J. Song, T. Gu, X. Feng, Y. Ge, and P. Mohapatra, "Blockchain meets COVID-19: A framework for contact information sharing and risk notification system," *arXiv preprint arXiv:2007.10529*, 2020.

[58] M. M. Arifeen, A. Al Mamun, M. S. Kaiser, and M. Mahmud, "Blockchain-enable contact tracing for preserving user privacy during COVID-19 outbreak," 2020.

[59] G. Avitabile, V. Botta, V. Iovino, and I. Visconti, "Towards defeating mass surveillance and Sars-Cov-2: The PRONTO-C2 fully decentralized automatic contact tracing system," *Cryptology ePrint Archive*, 2020.

[60] S. Micali, "Algorand's approach to covid-19 tracing," 2020. [Online]. Available: https://www.algorand.com/resources/algorand-announcements/silvio-micali-approach-to-covid-19

[61] J. Chen and S. Micali, "Algorand," *arXiv preprint arXiv:1607.01341*, 2016.

[62] M. Amoretti, G. Brambilla, F. Medioli, and F. Zanichelli, "Blockchain-based proof of location," in *2018 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C)*. IEEE, 2018, pp. 146–153.

[63] D. Marbouh, T. Abbasi, F. Maasmi, I. A. Omar, M. S. Debe, K. Salah, R. Jayaraman, and S. Ellahham, "Blockchain for COVID-19: Review, opportunities, and a trusted tracking system," *Arabian Journal for Science and Engineering*, pp. 1–17, 2020.

[64] F. Casino, T. K. Dasaklis, and C. Patsakis, "A systematic literature review of blockchain-based applications: Current status, classification and open issues," *Telematics and informatics*, vol. 36, pp. 55–81, 2019.

[65] M. Andoni, V. Robu, D. Flynn, S. Abram, D. Geach, D. Jenkins, P. McCallum, and A. Peacock, "Blockchain technology in the energy sector: A systematic review of challenges and opportunities," *Renewable and sustainable energy reviews*, vol. 100, pp. 143–174, 2019.

[66] M. B. Mollah, J. Zhao, D. Niyato, K.-Y. Lam, X. Zhang, A. M. Ghias, L. H. Koh, and L. Yang, "Blockchain for future smart grid: A comprehensive survey," *IEEE Internet of Things Journal*, vol. 8, no. 1, pp. 18–43, 2020.

[67] D. Kirli, B. Couraud, V. Robu, M. Salgado-Bravo, S. Norbu, M. Andoni, I. Antonopoulos, M. Negrete-Pincetic, D. Flynn, and A. Kiprakis, "Smart contracts in energy systems: A systematic review of fundamental approaches and implementations," *Renewable and Sustainable Energy Reviews*, vol. 158, p. 112013, 2022.

[68] C. DeCusatis and K. Lotay, "Secure, decentralized energy resource management using the ethereum blockchain," in *2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*. IEEE, 2018, pp. 1907–1913.

[69] Q. Yang, H. Wang, T. Wang, S. Zhang, X. Wu, and H. Wang, "Blockchain-based decentralized energy management platform for residential distributed energy resources in a virtual power plant," *Applied Energy*, vol. 294, p. 117026, 2021.

[70] V. Mladenov, V. Chobanov, G. C. Seritan, R. F. Porumb, B.-A. Enache, V. Vita, M. Stănculescu, T. Vu Van, and D. Bargiotas, "A flexibility market platform for electricity system operators using blockchain technology," *Energies*, vol. 15, no. 2, 2022. [Online]. Available: https://www.mdpi.com/1996-1073/15/2/539

[71] C. Pop, T. Cioara, M. Antal, I. Anghel, I. Salomie, and M. Bertoncini, "Blockchain based decentralized management of demand response programs in smart energy grids," *Sensors*, vol. 18, no. 1, p. 162, 2018.

[72] A. Umar, D. Kumar, and T. Ghose, "Blockchain-based decentralized energy intra-trading with battery storage flexibility in a community microgrid system," *Applied Energy*, vol. 322, p. 119544, 2022.

[73] G. Sciumè, E. Riva Sanseverino, P. Gallo, A. Augello, G. Sciabica, and M. Tornatore, *Blorin Blockchain Platform.* Springer International Publishing, 2022, pp. 139–170. [Online]. Available: https://doi.org/10.1007/978-3-030-96607-2_6

[74] V. Mandarino, G. Pappalardo, and E. Tramontana, "Some blockchain design patterns for overcoming immutability, chain-boundedness, and gas fees," in *The 3rd Asia Conference on Computers and Communications, Shangai, China.* ACCC, 2022.

[75] M. Bartoletti and L. Pompianu, "An empirical analysis of smart contracts: platforms, applications, and design patterns," in *International conference on financial cryptography and data security.* Springer, 2017, pp. 494–509.

[76] L. Marchesi, M. Marchesi, G. Destefanis, G. Barabino, and D. Tigano, "Design patterns for gas optimization in ethereum," in *2020 IEEE International Workshop on Blockchain Oriented Software Engineering (IWBOSE).* IEEE, 2020, pp. 9–15.

[77] C. R. Worley and A. Skjellum, "Opportunities, challenges, and future extensions for smart-contract design patterns," in *International Conference on Business Information Systems.* Springer, 2018, pp. 264–276.

[78] T. Chen, X. Li, X. Luo, and X. Zhang, "Under-optimized smart contracts devour your money," in *2017 IEEE 24th international conference on software analysis, evolution and reengineering (SANER)*. IEEE, 2017, pp. 442–446.

[79] T. Chen, Y. Feng, Z. Li, H. Zhou, X. Luo, X. Li, X. Xiao, J. Chen, and X. Zhang, "Gaschecker: Scalable analysis for discovering gas-inefficient smart contracts," *IEEE Transactions on Emerging Topics in Computing*, vol. 9, no. 3, pp. 1433–1448, 2020.

[80] V. Mandarino, G. Marotta, G. Pappalardo, and E. Tramontana, "Issues related to ehr blockchain applications," in *2021 2nd Asia Conference on Computers and Communications (ACCC)*. IEEE, 2021, pp. 132–137.

[81] A. Dubovitskaya, F. Baig, Z. Xu, R. Shukla, P. S. Zambani, A. Swaminathan, M. M. Jahangir, K. Chowdhry, R. Lachhani, N. Idnani *et al.*, "ACTION-EHR: patient-centric blockchain-based electronic health record data management for cancer care," *Journal of medical Internet research*, vol. 22, no. 8, p. e13598, 2020.

[82] T. Quaini, A. Roehrs, C. A. da Costa, and R. da Rosa Righi, "A model for blockchain-based distributed electronic health records." *IADIS International Journal on WWW/Internet*, vol. 16, no. 2, 2018.

[83] D. Tith, J.-S. Lee, H. Suzuki, W. Wijesundara, N. Taira, T. Obi, and N. Ohyama, "Application of blockchain to maintaining patient records in electronic health record for enhanced privacy, scalability, and availability," *Healthcare informatics research*, vol. 26, no. 1, pp. 3–12, 2020.

[84] A. Ekblaw, A. Azaria, J. D. Halamka, and A. Lippman, "A case study for blockchain in healthcare:"MedRec" prototype for electronic health records and medical research data," in *Proceedings of IEEE open & big data conference*, vol. 13, 2016, p. 13.

[85] K. Fan, S. Wang, Y. Ren, H. Li, and Y. Yang, "Medblock: Efficient and secure medical data sharing via blockchain," *Journal of medical systems*, vol. 42, no. 8, pp. 1–11, 2018.

[86] G. G. Dagher, J. Mohler, M. Milojkovic, and P. B. Marella, "Ancile: Privacy-preserving framework for access control and interoperability of electronic health records using blockchain technology," *Sustainable cities and society*, vol. 39, pp. 283–297, 2018.

[87] P. Zhang, J. White, D. C. Schmidt, G. Lenz, and S. T. Rosenbloom, "FHIRChain: applying blockchain to securely and scalably share clinical data," *Computational and structural biotechnology journal*, vol. 16, pp. 267–278, 2018.

[88] Q. Xia, E. B. Sifah, K. O. Asamoah, J. Gao, X. Du, and M. Guizani, "MeDShare: Trust-less medical data sharing among cloud service providers via blockchain," *IEEE access*, vol. 5, pp. 14 757–14 767, 2017.

[89] S. Jiang, J. Cao, H. Wu, Y. Yang, M. Ma, and J. He, "Blochie: a blockchain-based platform for healthcare information exchange," in *2018 ieee international conference on smart computing (smartcomp)*. IEEE, 2018, pp. 49–56.

[90] W. J. Bradshaw, E. C. Alley, J. H. Huggins, A. L. Lloyd, and K. M. Esvelt, "Bidirectional contact tracing could dramatically improve COVID-19 control," *Nature communications*, vol. 12, no. 1, pp. 1–9, 2021.

[91] Q. Feng, D. He, S. Zeadally, M. K. Khan, and N. Kumar, "A survey on privacy protection in blockchain system," *Journal of Network and Computer Applications*, vol. 126, pp. 45–58, 2019.

[92] L. Garg, E. Chukwu, N. Nasser, C. Chakraborty, and G. Garg, "Anonymity preserving IoT-based COVID-19 and other infectious disease contact tracing model," *IEEE Access*, vol. 8, pp. 159 402–159 414, 2020.

[93] A. Khurshid, "Applying blockchain technology to address the crisis of trust during the COVID-19 pandemic," *JMIR Medical Informatics*, vol. 8, no. 9, p. e20477, 2020.

[94] T. McGhin, K.-K. R. Choo, C. Z. Liu, and D. He, "Blockchain in healthcare applications: Research challenges and opportunities," *Journal of Network and Computer Applications*, vol. 135, pp. 62–75, 2019.

[95] G. Marotta, F. Billeci, G. Criscione, F. Merola, G. Pappalardo, and E. Tramontana, "Nausicaapp: A hybrid decentralized approach to managing Covid-19 pandemic at campus premises," in *Proc. of Asia Conference on Computers and Communications (ACCC)*. IEEE, 2020, pp. 124–129.

[96] D. Jaisinghani, R. K. Balan, V. Naik, A. Mirsa, and Y. Lee, "Experiences & challenges with server-side wifi indoor localization using

existing infrastructure," in *Proc. of EAI Intern. Conf. on Mobile and Ubiquitous Systems: Computing, Networking and Services*, 2018, pp. 226–235.

[97] G. Kwon, J. Kim, J. Noh, and S. Cho, "Bluetooth low energy security vulnerability and improvement method," in *Proc. of IEEE International Conference on Consumer Electronics-Asia (ICCE-Asia)*, 2016, pp. 1–4.

[98] G. Marotta, A. Fornaia, A. Moschitta, G. Pappalardo, and E. Tramontana, "Nausichain: a mobile decentralized app ensuring service continuity to university life in Covid-19 emergency times," in *Proc. of International Conference on Software Engineering and Information Management*, 2021, pp. 74–81.

[99] A. Fornaia, G. Marotta, G. Pappalardo, and E. Tramontana, "A decentralized solution for epidemiological surveillance in campus scenarios," *IEEE Access*, vol. 10, pp. 103 806–103 818, 2022.

[100] G. Singh and J. Levi, "MiPasa project and IBM Blockchain team on open data platform to support Covid-19 response," *IBM, Armonk, NY, Mar*, vol. 27, 2020.

[101] A. Chawla and S. Ro, "Coronavirus (COVID-19)–is blockchain a true savior in this pandemic crisis," *Available at SSRN 3655337*, 2020.

[102] V. Sagar and P. Kaushik, "Ethereum 2.0 blockchain in healthcare and healthcare based internet-of-things devices," in *Proceedings of the International Conference on Paradigms of Computing, Communication and Data Sciences*. Springer, 2021, pp. 225–233.

[103] S. Teufel and B. Teufel, "The crowd energy concept," *Journal of electronic science and technology*, vol. 12, no. 3, pp. 263–269, 2014.

[104] B. Teufel, A. Sentic, and M. Barmet, "Blockchain energy: Blockchain in future energy systems," *Journal of Electronic Science and Technology*, vol. 17, no. 4, p. 100011, 2019.

[105] IRENA, "Aggregators; Innovation Landscape Brief," 2019. [Online]. Available: https://www.irena.org/-/media/Files/IRENA/Agency/Publication/2019/Feb/IRENA_Innovation_Aggregators_2019.PDF

[106] O. Juszczyk and K. Shahzad, "Blockchain technology for renewable energy: Principles, applications and prospects," *Energies*, vol. 15, no. 13, p. 4603, 2022.

[107] IRENA, "Blockchain; innovation landscape brief," 2019. [Online]. Available: https://www.irena.org/publications/2019/Sep/Blockchain

[108] D. Miller, "PJM-EIS UPDATE: Modernizing a legacy U.S. REC tracking system with blockchain-based technology," July 2021. [Online]. Available: https://medium.com/energy-web-insights/pjm-eis-update-modernizing-a-legacy-u-s-rec-tracking-system-with\-blockchain-based-technology-db0ad5a4f924